PhyreStation User's Guide

Release 2.5.0

© 2009 Sony Computer Entertainment Inc.
All Rights Reserved.
SCE Confidential

Table of Contents

Using This Guide

The purpose of this guide is to show you around PhyreStation and to describe how to use PhyreStation to inspect PhyreEngine™ data or objects, write and execute scripts to automate repetitive tasks, and use workspaces to consolidate your assets and work in a project type manner.

If you are new to PhyreStation, we recommend that you read the information in this guide in the order in which it is presented, as you may need to understand information in earlier chapters in order to understand subsequent chapters.

If you are already familiar with the application, you can use the guide for reference purposes.

Audience

The guide is intended for all users of the software, ranging from artists and support staff through to technical developers and programmers.

We assume that you are familiar with GUI software, therefore general terms such as 'drag', 'drop', 'minimize', and so on, are not explained.

Conventions

Hyperlinks

Hyperlinks are used to help you navigate around the document. To allow you to return to where you clicked a hyperlink, enable the Adobe® Acrobat® Reader **Previous View** and **Next View** buttons by selecting **View > Toolbars > More Tools...** from the Adobe® Acrobat® Reader main menu.

Hints

A GUI shortcut or other useful tip for gaining maximum use from the software is presented as a 'hint' surrounded by a box. For example:

> **Hint:** An externally linked database failure can occur if a linked database file cannot be found.

Notes

Additional advice or tangible, related information to help maximize the use of the application or the *PhyreStation User's Guide* is presented as a 'note' surrounded by a box. For example:

> **Note:** PhyreStation source code is not available to third parties.

Text

- Named application windows or GUI features are formatted in a bold sans-serif font. For example, the **Texture Viewer**, the **Log** window, the **Save** button.
- Names of keyboard functions or keys are formatted in a bold serif font. For example, **Ctrl**, **Delete**, **F9.**
- File names, source code and command-line text are formatted in a fixed-width font. For example:

```
[PhyreStation_root_directory]/Preferences
```

- Menu options are separated with chevrons. For example, **File > Exit** refers to the **Exit** option on the **File** menu.

Graphics

All screenshots used in this guide are taken from the Windows version of PhyreStation. The GUI may differ slightly depending on your desktop style and the version of Windows you are using.

Errata

Any updates or amendments to this guide, including new features added to PhyreStation after this guide was published, can be found in `[PhyreStation_root_directory]/Readme_e.txt`.

1 PhyreStation Overview

This chapter provides a brief overview of the PhyreStation application.

Introduction

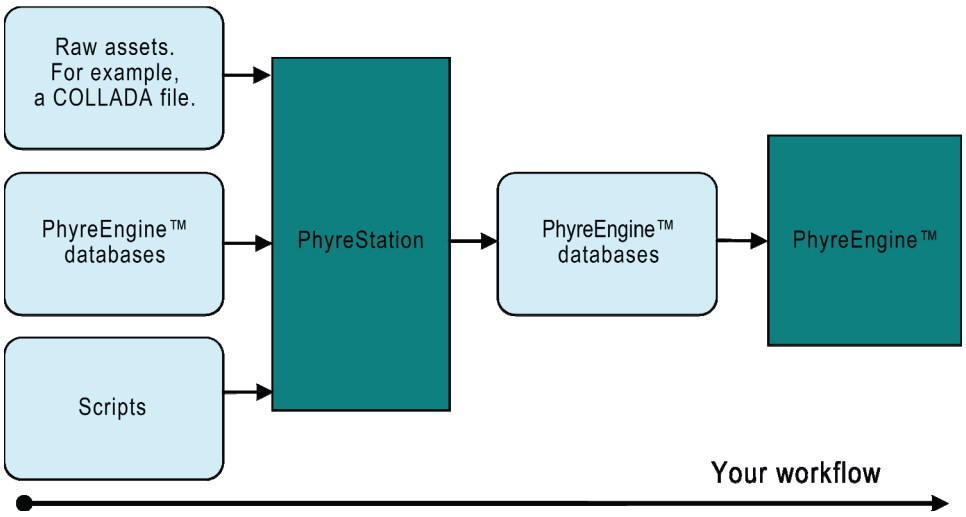
PhyreStation is a GUI tool that allows you to view, edit, process, and tune digital art assets for use in development projects. Using the various functions provided by PhyreStation, you can inspect the relationships between different resources and manipulate them.

PhyreStation supports PhyreEngine™ (`.hier`, `.PSSG`) and COLLADA (`.xml`, `.dae`) database file formats.

You use PhyreStation to create projects (called 'workspaces') that contain databases and are associated with scripts for automating work.

PhyreStation can operate as a stand-alone tool or it can operate in batch mode (no GUI). In batch mode, PhyreStation can be executed as part of a larger batch process, executing scripts on your raw assets. The scripts convert or condense the raw assets to form PhyreEngine™ databases or PhyreEngine™ objects within those databases. In this way, PhyreStation helps you to achieve your goal of consolidating all your game assets into a final solution that is efficient and compact.

Figure 1 PhyreStation Workflow



PhyreStation can help you inspect the contents of PhyreEngine™ databases from a file or from a live PhyreEngine™ instance. It also allows you to inspect individual PhyreEngine™ objects.

Typically, a programmer uses PhyreStation to monitor, debug, and optimize the PhyreEngine™ databases, manage objects, and edit object attributes. Some object attributes can be manipulated by using either the GUI or one of the predefined commands provided or a custom command created by the programmer. Most of the predefined commands or custom commands can be used from within a script. By continuous iteration, the process of workflow creation can be fine-tuned, as well as the data. The scripting language used is Lua.

An artist may use PhyreStation to import art assets from an art package into a PhyreStation workspace, run an appropriate script, and view the results either in PhyreStation or on the target platform. The artist can also experiment by changing PhyreEngine™ objects' parameters and then immediately viewing the changes as they would appear on the target platform. There is no need to re-export to the target platform, thus saving time.

This guide provides workflows and exercises to help you get the best out of PhyreStation.

Main Concepts

This section provides an overview of the main concepts of PhyreStation.

Workspaces

A workspace is the main container element in PhyreStation. It is a type of project file. A workspace can contain any of the following 'workspace objects': PhyreEngine™ databases, scripts, and custom groups of objects. Most of these objects can be shared amongst many different workspaces.

In a workspace, you can view any PhyreEngine™ object in any PhyreEngine™ database. You can search the workspace for a set of objects by name or type. For more information, see *Chapter 3, Workspaces*.

Databases

A PhyreEngine™ database (referred to hereafter as a 'database') is a collection of PhyreEngine™ objects or user data. A database's objects can be manipulated to share resources by referencing resources in other databases. Databases that contain shared resources are called 'linked databases'. For more information, see *Chapter 4, Databases*.

Objects

A PhyreEngine™ object is a data asset derived from PObject class instance. A PhyreEngine™ object can be any predefined PhyreEngine™ object type or a custom type defined by you. To create custom objects, see *Chapter 5, PhyreEngine™ Objects*.

Commands

There are two types of PhyreStation commands:

- Internal PhyreStation commands. These are commands defined internally to PhyreStation and cannot be changed. These types of commands normally work on a workspace or database level. Internal commands include all the workspace commands, such as `OpenWorkspace()`, and many of the database commands, such as `ResolveLinks()`. These commands are automatically registered with PhyreStation.
- PhyreStationDLL commands. These are commands that can work at an object or database level. You can change commands found in the PhyreStationDLL and you can create new commands. To group together your custom command and object class code, you create your own PhyreEngine™ User Utility extension. PhyreStationDLL commands must be registered with PhyreStation to be available for the various methods of execution: from a GUI context menu, in a drag-drop operation, or from a script.

For more information, see *Chapter 6, PhyreStation Commands*.

Scripts

One of the primary purposes of PhyreStation is to facilitate the automation of processing data. A script file is a simple ASCII text file that can contain any valid PhyreStation-registered commands combined with Lua commands. The PhyreStation script interpreter is based on the Lua script engine. For more information, see *Chapter 7, Scripts*.

Running PhyreStation

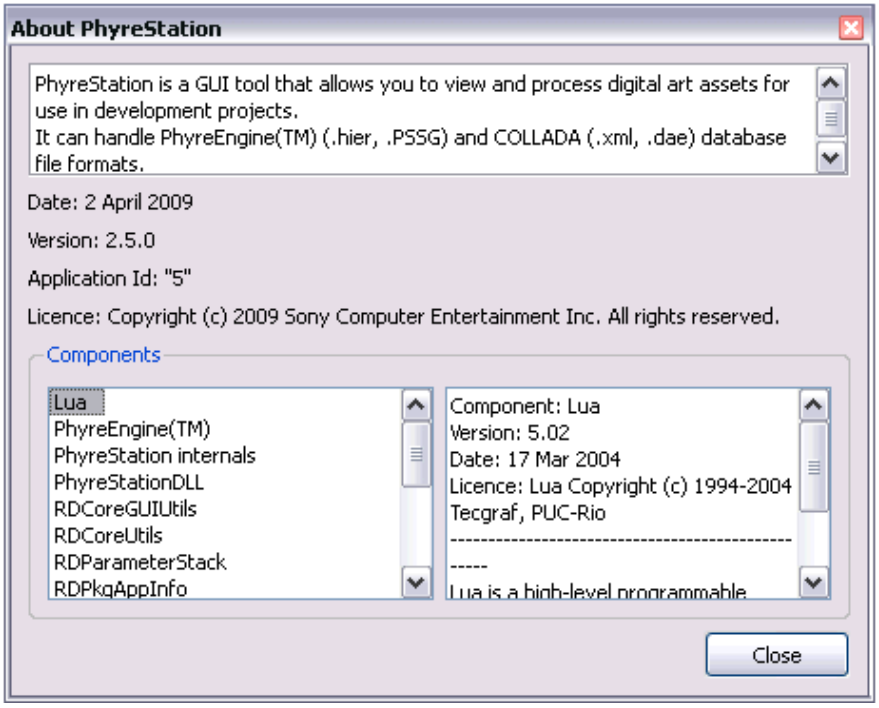
PhyreStation is suitable for the Microsoft Windows XP and Vista platforms only. Run PhyreStation from the DOS command prompt as follows:

```
[PhyreStation_root_directory]> PhyreStation.exe
```

About PhyreStation

To obtain version information about PhyreStation and its components, select **Help > About PhyreStation...** from the main menu. The **About PhyreStation** dialog box is displayed.

Figure 2 About PhyreStation Dialog Box



PhyreStation is composed of packages (code modules) and code libraries, which are listed in the **Components** section of the **About PhyreStation** dialog. The information that is displayed in the right-hand group box when either **PhyreEngine™** or **PhyreStationDLL** is selected is especially useful. The right-hand group box displays the date, version number, and the types of PhyreEngine™ objects and PhyreStationDLL commands that are currently registered with PhyreStation. If you add any new commands or PhyreEngine™ object types, you can see them listed here.

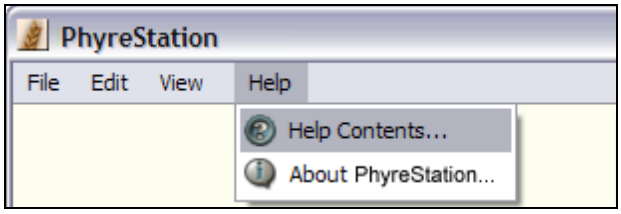
The Help Viewer

The **Help Viewer** window displays information about all PhyreStation internal and registered PhyreStationDLL commands, and all PhyreEngine™ object types registered through the PhyreStationDLL. The information displayed on these pages is updated each time PhyreStation starts.

Displaying the Help Viewer

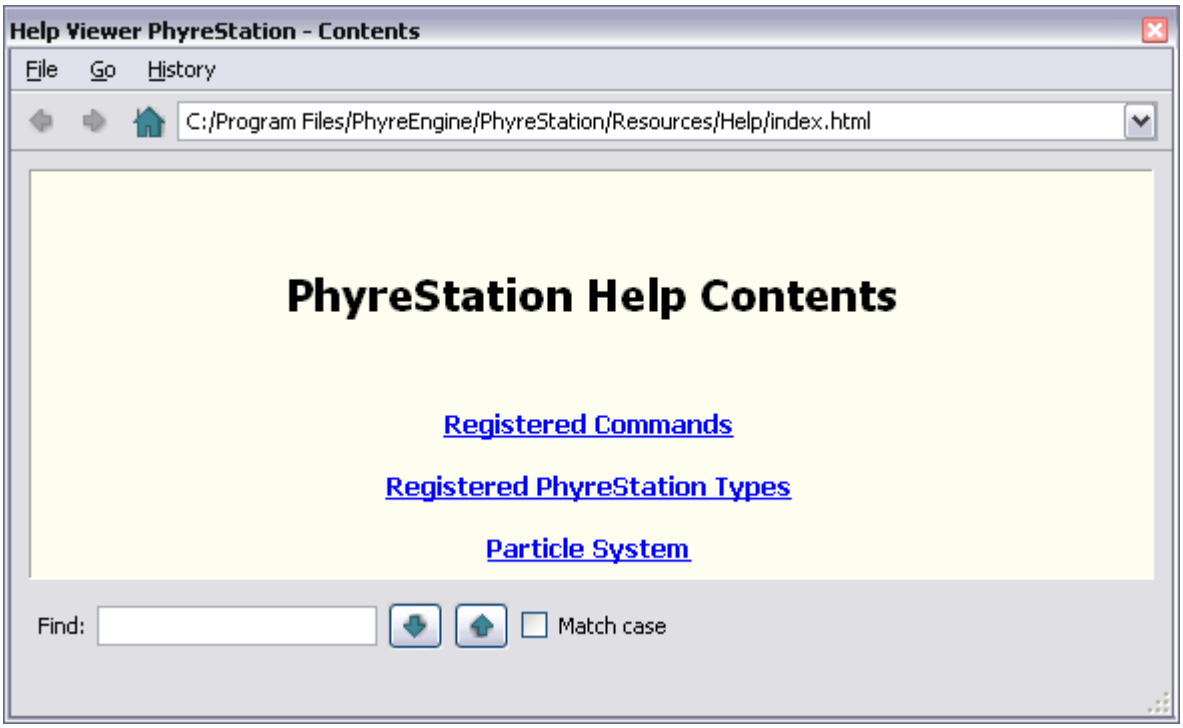
To open the **Help Viewer**, select **Help > Help Contents...** from the main window's menu bar.

Figure 3 Help Menu



The **Help Viewer Contents** page opens.

Figure 4 Help Viewer Contents Page



To view and search information about all registered commands, click **Registered Commands**. To view and search information about all registered PhyreEngine™ object types, click **Registered PhyreStation Types**.

Figure 5 Help Viewer Commands Page

Navigation

You can navigate through the **Help Viewer** pages as follows:

- To return to the index page, click the **Home** button or select **Go > Home** from the window's menu.
- To move forward and backward through the pages already visited, use the arrow buttons or select from the **Go** menu.
- To revisit a page, select the page from the drop-down list in the toolbar, or select **History** from the window's menu.

Find

The **Help Viewer** provides Find functionality for the currently displayed page.

- To search within the current document, for example `commands.html`, enter text into the **Find** field and press **Enter**.
- To search forward through the document, press **Enter** or click the **Down arrow** button.
- To search backwards from the current position in the document, click the **Up arrow** button.
- The **Match case** checkbox toggles the case-sensitivity of the search.

Adding Pages to the Help Viewer

You can add additional custom information pages to the **Help Viewer**, which is useful in situations where users process custom workflows.

To add additional pages, put your external help files in the PhyreStation `Resources/Help` directory of the PhyreStation installation. The **Help Viewer** automatically displays the additional Help pages the next time PhyreStation is started and the browser is activated. The **Help Viewer** supports the display of files with both `.txt` and `.html` extensions (if their contents are valid).

2 Workflows

This chapter includes some typical user workflows and provides links to the detailed descriptions of how to perform the tasks described in each workflow.

Overview

For the purposes of this chapter, a workflow is defined as a description of work carried out by a programmer or artist that may involve PhyreStation, primarily to manage data. A workflow is a descriptive overview of how to perform tasks to achieve a desired goal. A workflow may also describe at a high level how to operate PhyreStation to achieve a task.

In addition to the workflows provided in this chapter, you can find a series of exercises in *Appendix A: Exercises*

These exercises describe in detail how to perform the tasks to complete the workflows.

Artist's Workflows

The overall goal of PhyreStation is to automate the process of turning raw art assets into PhyreEngine™ objects that have a structure specified by the programmer. PhyreStation helps artists provide artwork in the required structure, without restricting their workflow.

Artists are free to produce raw art assets according to their own working practices. The only condition when using PhyreStation is that art packages must be able to export to the COLLADA file format.

The appearance of artwork often depends significantly on the rendering system, that is, on the art package and platform used by the artist. Because PhyreStation uses PhyreEngine™, artists can use PhyreStation for viewing art assets consistently, without having to worry about subtle differences introduced by viewing assets under different packages and on different platforms. PhyreStation may also be more efficient at rendering complex scenes than a comprehensive art package.

Game Templates

PhyreEngine™ game templates are examples of how to achieve the best from PhyreEngine™ for a particular genre of game. They also demonstrate typical workflows one would set up when developing software using PhyreEngine™.

The game templates can be found in the `<PhyreEngine>\Samples\Applications` directory.

In each game template directory, there is a white paper that describes the merits of the game template. The white papers describe from a technical programmer's point of view how best to use PhyreEngine™ for a particular technical solution.

Planning a Workflow

You will probably form your own customized workflows. Use the workflows featured in this chapter to help you define your workflows. When planning a workflow, consider the following:

- Determine how PhyreEngine™ data should be structured to be most efficient.
- Consider any dependencies between PhyreEngine™ objects.
- Determine how PhyreEngine™ objects will be loaded and used in the final application.
- Establish how the art team will use their artwork in the workflow.
- To process the raw assets into optimized assets required by the final application, construct a PhyreStation Workspace.
- Establish a standard sequence of operations (a sequence of commands) to prove the required steps work, and then code a *script* from these commands to automate the process.

- Possibly extend PhyreStation to work with new PhyreEngine™ objects you have created or will create. Select **Help > About PhyreStation** to see the current list of registered PhyreEngine™ PObject derived types.
- Possibly extend PhyreStation to work with new commands that are unique to your workflow, either to work on your own custom PhyreEngine™ objects or to perform a specific task during processing. Select **Help > About PhyreStation** to see the current set of commands available for use.
- Create and use a set of scripts to work with the PhyreStation application. These scripts represent the automated parts of the programmer's workflow and can be operated by other members of the team, for example, artists.

Workflows

The workflows in this section provide an overview of some of the processes that can be carried out using PhyreStation. They are:

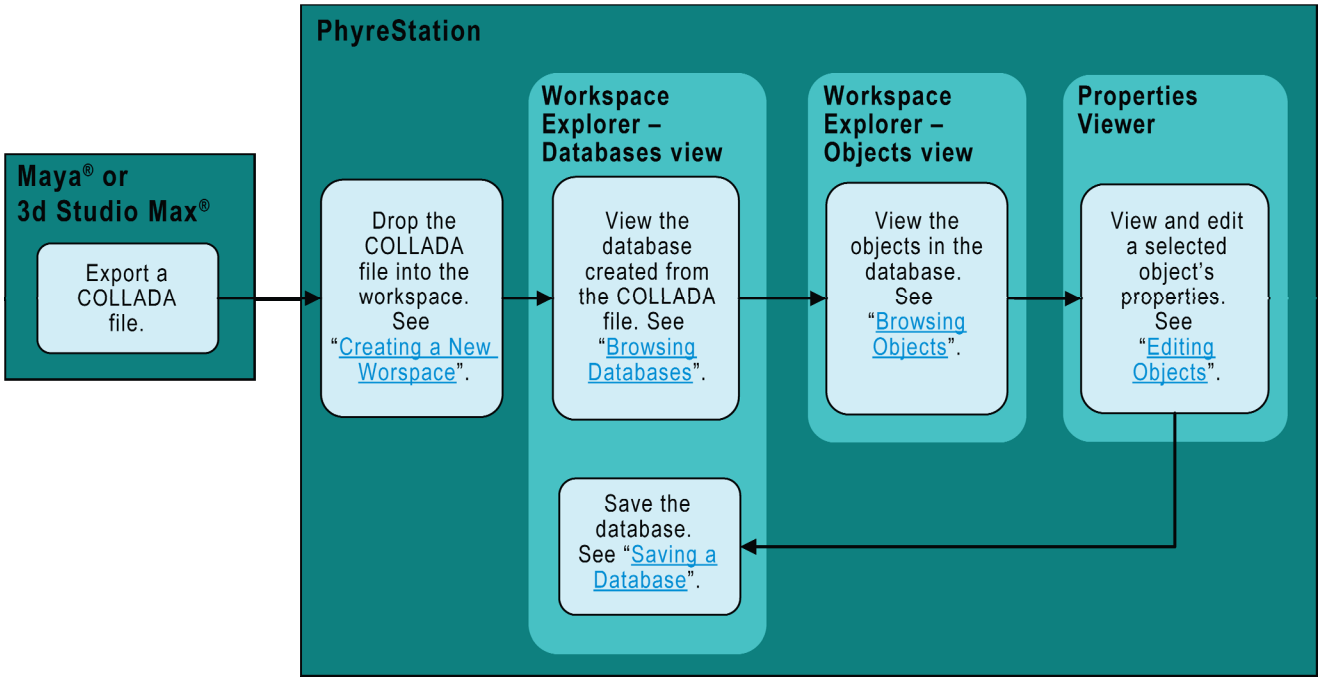
- Getting Started
- Viewing a Scene
- Customizing PhyreStation
- Converting a `.cgfx` to `.pssg` file, then Debug
- Debugging a Remote Database
- Consolidating Assets from Multiple Databases
- Creating a Particle System

Hint: To allow you to return to the workflow diagram after navigating to a hyperlink, enable Adobe® Acrobat® Reader's **Previous View** and **Next View** buttons by selecting **View > Toolbars > More Tools...** from the Acrobat® Reader main menu.

Workflow: Getting Started

This workflow shows how to start PhyreStation and then view the contents of a COLLADA file, the PhyreEngine™ objects, and the object's properties.

Figure 6 Getting Started

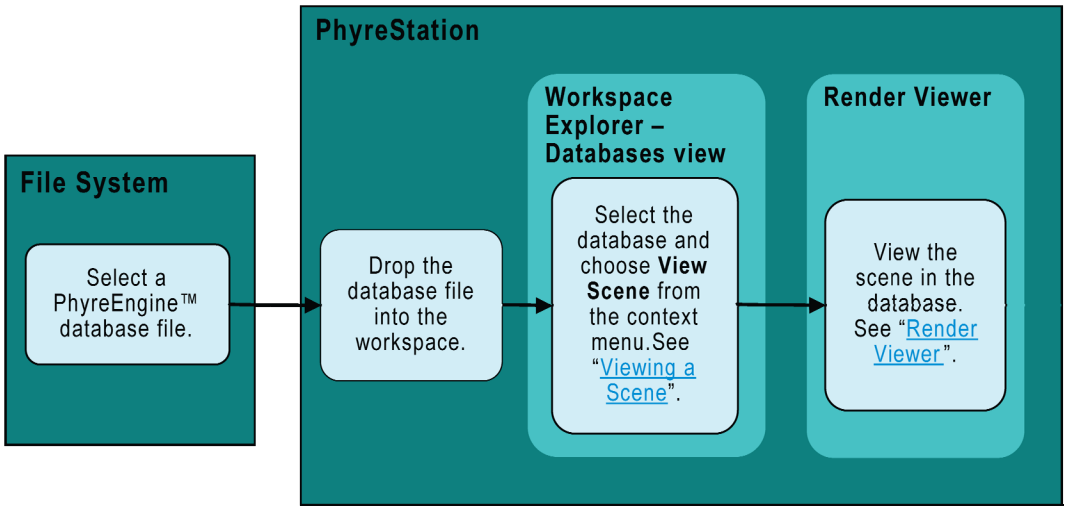


Workflow: Viewing a Scene

This workflow describes a 'fast track' to viewing the objects in a database scene. PhyreStation creates temporary camera and light objects, and then the scene is displayed automatically in the **Render Viewer**.

A scene (PhyreEngine™ root node) must exist in the database.

Figure 7 Viewing a Scene in a Database



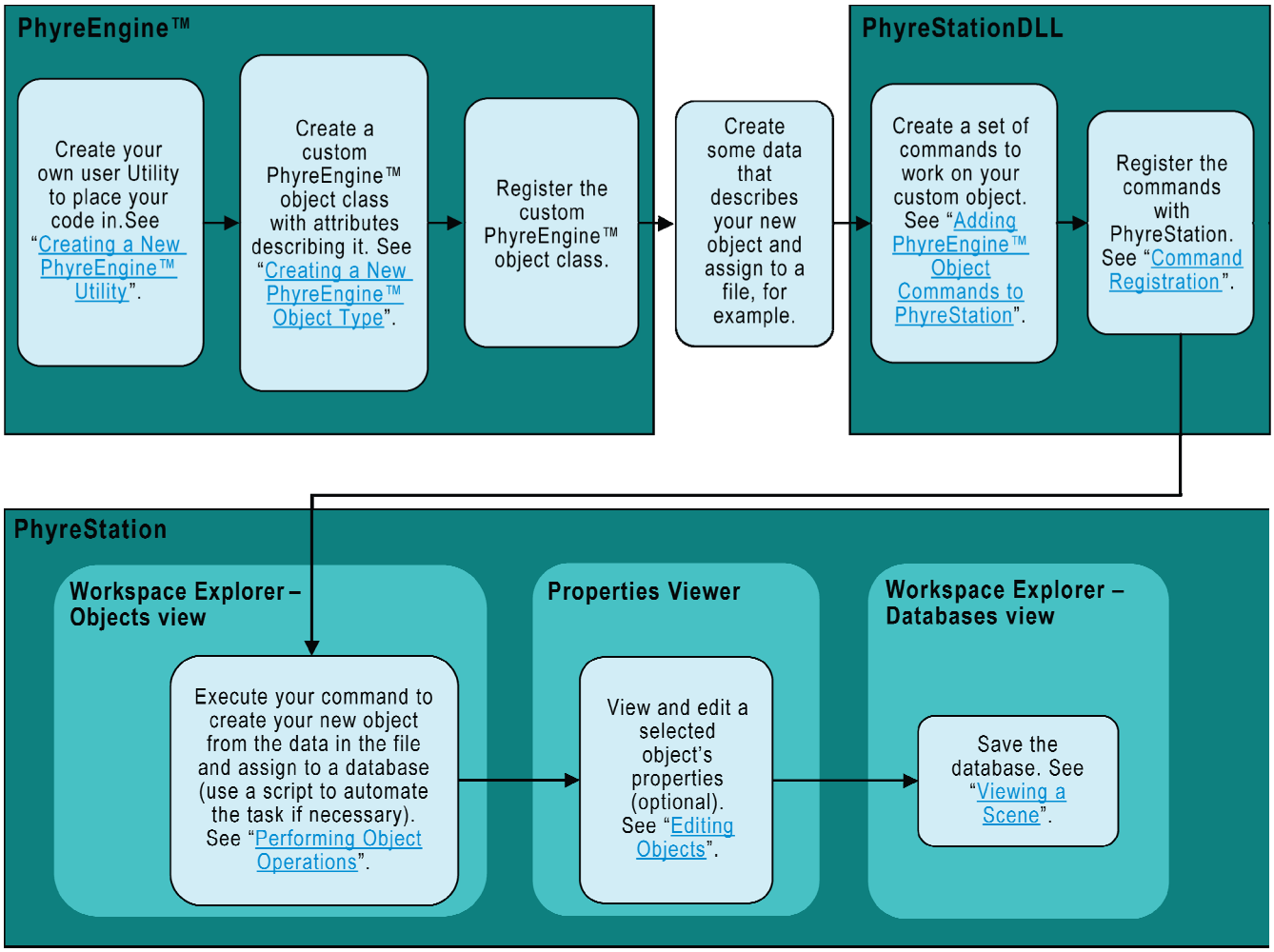
For more information about temporary objects, see the *Temporary Objects* section in *Chapter 5, PhyreEngine™ Objects*.

Workflow: Customizing PhyreStation

This workflow describes a typical customization of PhyreStation to work with your objects and commands.

PhyreStation is extendable to satisfy your data management requirements. By extending PhyreStation and PhyreEngine™, you can create your own commands to work on your custom PhyreEngine™ objects. For full details, see *Chapter 8, Customizing PhyreStation*.

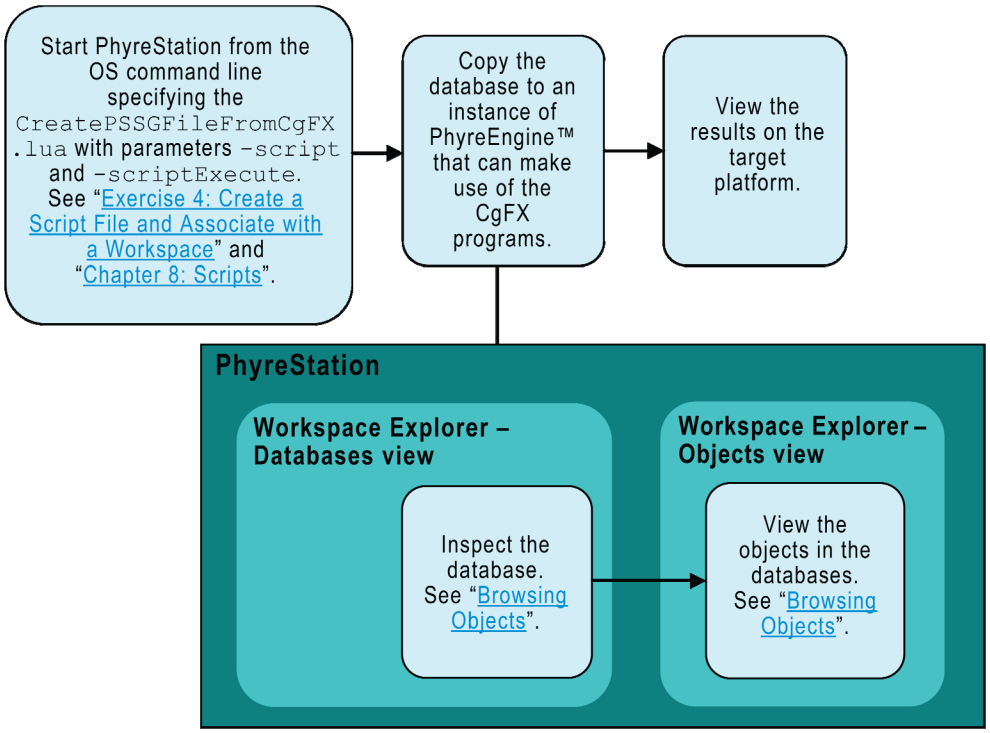
Figure 8 Customizing PhyreStation



Workflow: Converting a .cgfx to .pssg File, then Debugging

Available in PhyreEngine™'s build directory, the script `CreatePSSGFileFromCgFX.lua` demonstrates how to convert a CgFX file type into a format that PhyreEngine™ can use on either a Sony PLAYSTATION®3 or PC target.

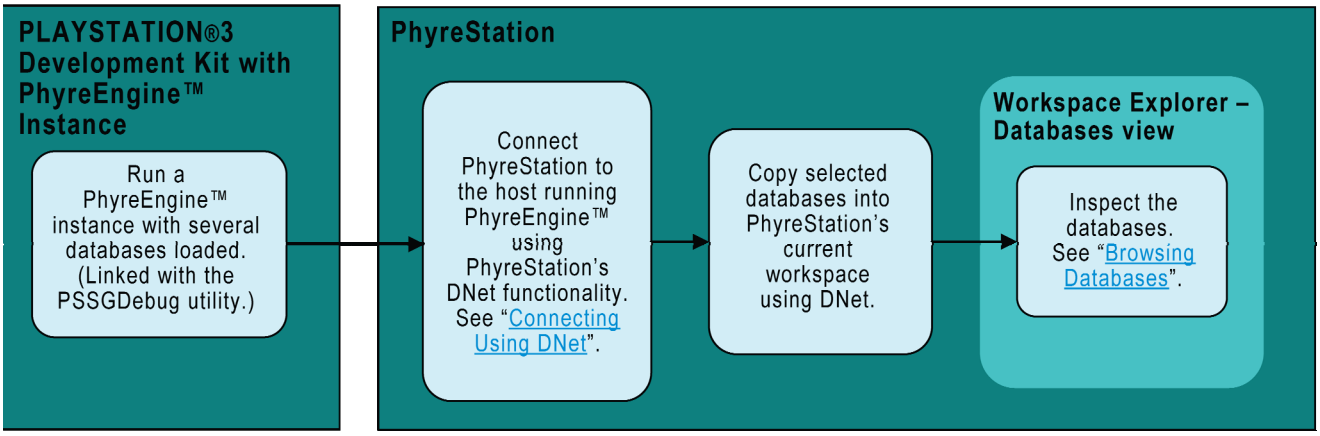
Figure 9 Debugging a .cgfx File



Workflow: Debugging a Remote Database

This workflow shows how to use PhyreStation to inspect PhyreEngine™ databases currently being used by an instance of PhyreEngine™ executing on a target platform.

Figure 10 Debugging a Remote Database



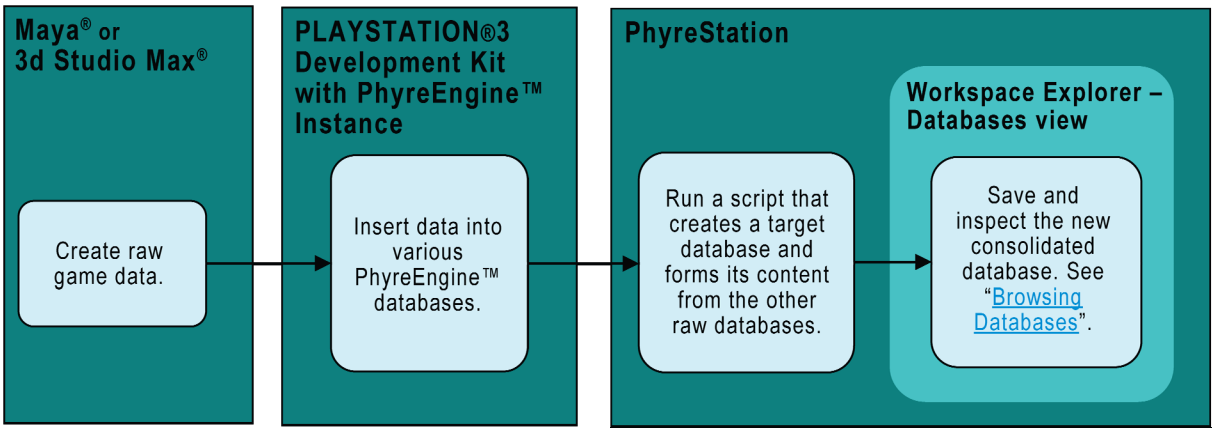
Workflow: Consolidating Assets from Multiple Databases

PhyreStation can be used to automate the often long and repetitive work of reducing a game's raw data by removing duplicate or unused data, and consolidating the data into one PhyreEngine™ database. The script driving the automation can also be used to insert custom data if you wish.

This workflow demonstrates how to take a set of 'raw' PhyreEngine™ databases and combine them into one database suitable for the target platform. The resultant database contains only assets that are relevant; any duplicates are removed. Finally, the database is compressed.

Before starting this workflow, see *Planning a Workflow* in *Chapter 2, Workflows*. For a detailed example of consolidating assets, see *Exercise 5: Consolidate PhyreEngine™ Databases* in *Appendix A: Exercises*.

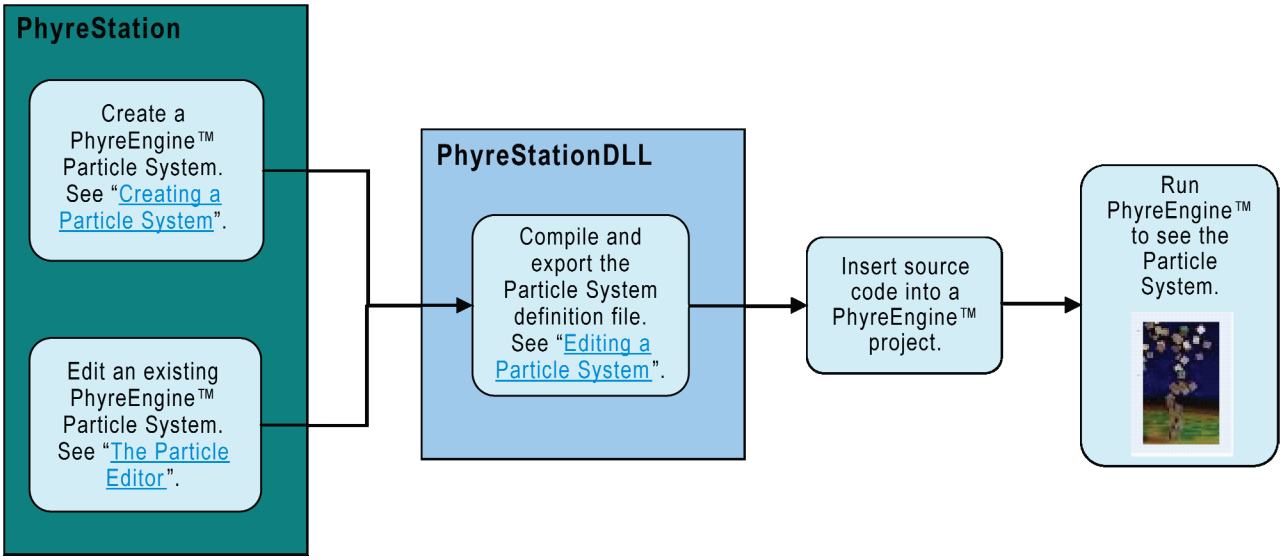
Figure 11 Consolidating Assets



Workflow: Creating a Particle System

This workflow shows how to create a PhyreEngine™ Particle System. For a detailed description of how to set up an example Particle System, see *Exercise 6: Set up a Particle System* in *Appendix A: Exercises*.

Figure 12 Creating a Particle System



3 Workspaces

This chapter describes the concept of the PhyreStation workspace. It explains how to manage workspaces and workspace files, and how to access the objects contained in a workspace.

Overview

The working environment in PhyreStation is defined as a workspace. The concept of a workspace is analogous to that of a project in other GUI applications. The workspace is a container for all the elements (called "workspace objects") in the development project.

When a workspace file is opened, it will probably contain one or more databases. The workspace may also have some scripts and custom groups associated with it, though this is not always the case.

You navigate the workspace using the **Workspace Explorer**. Each **Workspace Explorer** filter view has a different way to display either information or certain objects in the workspace.

If databases in the workspace rely on resources in other databases, they will often need to be resolved. When PhyreStation resolves a database, it collects from PhyreEngine™ all the objects contained in the database, and in other databases, if required. During this process, PhyreStation creates *representations* of the objects in the database. It is these representations of objects that you interact with in the workspace.

After the workspace has been populated with the objects from the databases in the workspace, you can carry out work on those objects or databases. For example, you can execute commands from an object's context menu or from the **Command** window. When a command or a script containing commands is executed, the workspace is automatically updated.

Whenever a workspace is open and at least one database is present (this does not include the PhyreEngine™ internal database), PhyreEngine™ objects can be created, edited, or deleted in the GUI.

A workspace can represent several thousand PhyreEngine™ objects. PhyreStation allows you to search for a type of object or search for an object by a partial name.

You can select an object and then display the object's attributes and attribute values. In some cases, you can edit these values. Depending on the object type, you can open other windows to display more information about an object.

Workspace Objects

A workspace object can be one of the following:

- Database – a container for PhyreEngine™ objects. See *Chapter 4, Databases*.
- PhyreEngine™ object – any of the many different objects held in the databases; for example, a shader program or a light node. See *Chapter 5, PhyreEngine™ Objects*.
- Custom Group – a user-defined group of workspace objects. See *Chapter 9, Custom Groups*.
- Script – See *Chapter 7, Scripts*.

Note: Although PhyreStation treats PhyreEngine™ databases as workspace objects, a PhyreEngine™ database is actually a file that holds PhyreEngine™ objects. From a PhyreEngine™ point of view, PhyreEngine™ databases are not PhyreEngine™ objects.

Workspace Files

Workspaces are stored as workspace files. When you save a workspace, references to the workspace data are saved in the workspace file. A workspace file is a relatively small XML file and has the same name as the workspace name you specify in PhyreStation, for example `MyWorkspace.xml`.

You can find sample workspace files in the following directory:

`[PhyreStation_root_directory]/PhyreStationWorkspaces`

Locations of database files and script files are recorded as relative paths in the workspace file. During loading, these paths are translated to absolute paths internally.

The Current Workspace

Only one workspace may be open in PhyreStation at any time; this is known as the current workspace. The name of the current workspace is displayed in the application title bar. If the current workspace has been changed but not saved, the name is suffixed by an asterisk.

Figure 13 Modified Workspace

The workspace is marked as changed when any of the following occur:

- The workspace name is changed.
- A workspace object (database, script, or custom group) is changed, added, or deleted.
- A PhyreEngine™ object is created or added to a database, or deleted from a database.
- A PhyreEngine™ object's attributes are changed.

The asterisk remains in the title bar until all changes have been saved. Saving the workspace does not automatically save the databases in the workspace. They must be saved separately.

> **Note:** The current workspace name may contain unexpected characters if the regional settings on the platform differ from the locale used when the workspace file was created.

By default, PhyreStation starts up with an empty workspace called `Untitled.xml`. To change this behavior, select **Edit > User Preferences > PhyreStation** and set the **Blank workspace** preference.

Workspace Data

Workspace files contain two types of data:

- User-defined data such as PhyreEngine™ databases, scripts, and custom groups.
- Workspace data such as windows and their positions, and workspace preference data. When you re-open a workspace, PhyreStation uses this data to restore the workspace windows to the state they were in when the workspace last closed. Workspace data is always saved to the workspace file when the workspace is closed, whether or not the workspace is marked as modified.

To change the way workspaces are saved and restored, and to set the default workspace directory, select **Edit > User Preferences > Workspaces** and set the appropriate preferences.

Workspace files also contain version information that PhyreStation reads first when attempting to open the files. If the version number of the file is not supported by a version of PhyreStation, the load aborts with a warning message.

> **Hint:** If the workspace file version is newer than the version of PhyreStation that you are using, upgrade to a newer version of PhyreStation. If the file is obsolete, locate all objects associated with the old workspace and create a new workspace containing those objects.

Creating a New Workspace

To create a new workspace:

(1) Open the **Workspace Explorer** by clicking the tool bar button, and then do one of the following:

- Select **File > Workspaces > New** from the main menu.
- Save the default `Untitled.xml` with a different name by selecting **File > Workspaces > Save As...** from the main menu.
- Right-click the white space in the **Workspace Explorer** window and select **New workspace...** from the context menu.

(2) Enter a name for the workspace, for example 'MyWorkspace', and save.

After creating the new workspace, you can populate it with PhyreEngine™ databases and associate scripts with it to automate the processing of objects in that workspace. See *Chapter 4, Databases* and *Chapter 7, Scripts* for more details.

Creating a new workspace automatically closes the current workspace session.

Opening an Existing Workspace

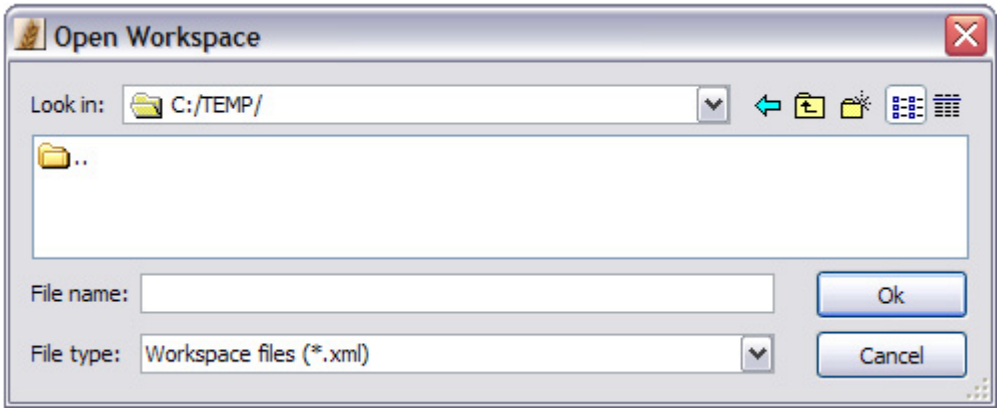
When you open an existing workspace file, PhyreStation automatically loads the workspace's databases and prepares the workspace's scripts for use. If a workspace is already open, it will be closed when you open another workspace.

To open a workspace, do one of the following:

- Select **File > Workspaces > Open...** from the main menu.
- Right-click the white space in the **Workspace Explorer** window, and then select **Open workspace...** from the context menu.
- Drag the workspace file into the PhyreStation main window from the OS file browser.

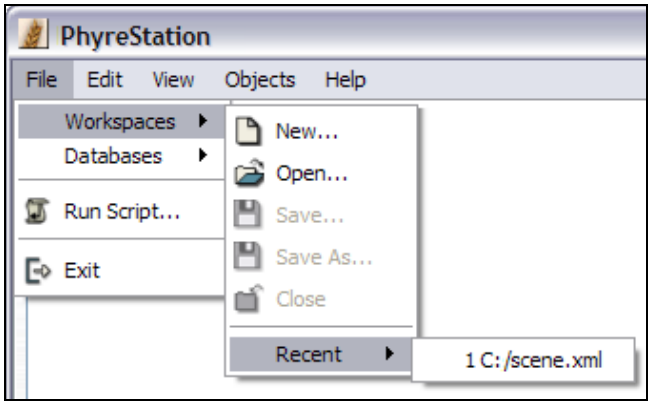
In the first two cases, the **Open Workspace** dialog box appears.

Figure 14 Open Workspace Dialog Box



You can re-open recently viewed workspaces by selecting **File > Workspaces > Recent** from the main menu.

Figure 15 Loading Recent Workspaces



Hint: To specify the number of recent workspace files that appear in this list, select **Edit > User Preferences > PhyreStation** and set the **Recent Workspaces** preference.

The Workspace Explorer

You can access all the objects in the current workspace using the **Workspace Explorer**. More than one **Workspace Explorer** can be open at one time. To open a **Workspace Explorer**, click the button on the main toolbar.

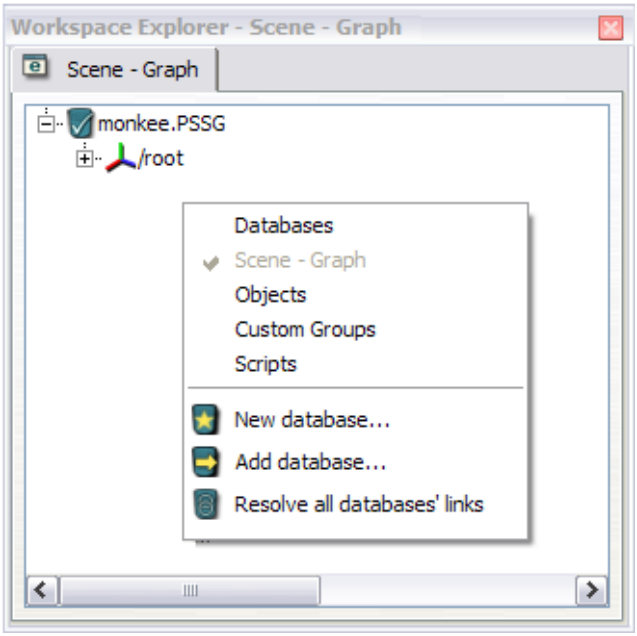
Figure 16 Workspace Explorer Toolbar Button

Workspace Explorer Views

The **Workspace Explorer** has several filter views available. Each filter view displays a specific type of workspace object or subset of objects. Each filter view displays information in a specific and concise manner and has its own unique set of menu options.

To select a filter view, right-click on the whitespace of the **Workspace Explorer** view (where there are no items) to bring up the view's context menu. From this menu, a filter view can be selected.

Figure 17 Workspace Explorer Context Menu



Each **Workspace Explorer** filter view allows you to select one or more objects, and then select an action to be carried out against the object(s) from the object's context menu.

Note: If you press **Shift** while selecting a set of node objects, some of which are collapsed, the child nodes of the collapsed nodes will also be selected. These child nodes will also have any commands executed on them, even though they are not visible; for example, Delete. To avoid this, expand any collapsed nodes using the **Ctrl** key and deselect the child nodes.

Note: PhyreStation allows you to perform operations that can put the stability of PhyreEngine™, upon which PhyreStation depends, in jeopardy. PhyreStation warns you before you perform those operations, but does not prevent you from carrying them out. If you proceed to carry them out, the reliability of both PhyreEngine™ and PhyreStation may be compromised.

Scene-Graph View

The **Workspace Explorer – Scene-Graph** view displays only PhyreEngine™ PNode-derived objects in the workspace, grouped by database.

Each scene-graph view is a hierarchy of PhyreEngine™ PNode-derived objects (cameras, lights, joints, and so on). When these objects are connected together, they form the physical or logical structure of a graphical scene. You can derive specialized node object types to perform specific functions when rendering the scene; for example, adding lights or geometry to the scene.

If a database contains at least one root node, any child nodes with a parent node are displayed in a hierarchy. Otherwise, the nodes or nodes without a parent are displayed as a flat list under the database item.

Objects View

The **Workspace Explorer – Objects** view displays all the PhyreEngine™ objects in all the databases that have been link-resolved or had their objects collected in the workspace. For more information about this view, see *Browsing Objects* in *Chapter 5, PhyreEngine™ Objects*.

Databases View

The **Workspace Explorer – Databases** view displays the top-level PhyreEngine™ databases in the workspace and the first level of linked databases. For more information about this view, see *Browsing Databases* in *Chapter 4, Databases*.

Custom Groups View

The **Workspace Explorer – Custom Groups** view displays user-defined groups of workspace objects. For more information about this view, see *Browsing Custom Groups* in *Chapter 9, Custom Groups*.

Scripts View

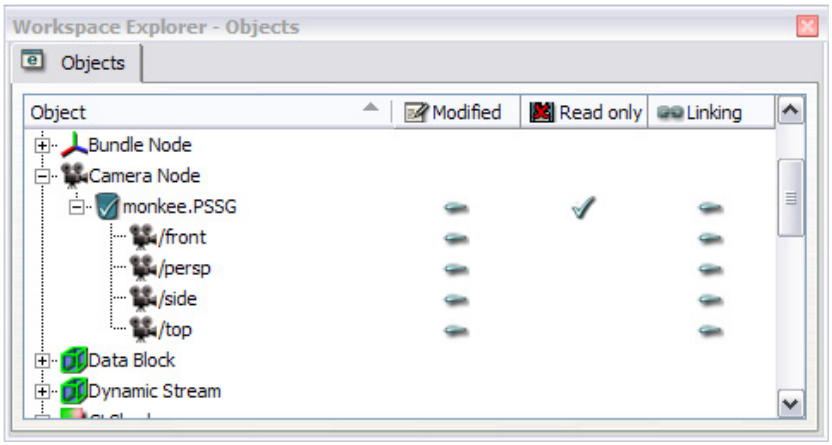
The **Workspace Explorer – Scripts** view displays the script files associated with the current workspace. For more information about this view, see *Browsing Scripts* in *Chapter 7, Scripts*.

Workspace Explorer Filter Properties

The **Workspace Explorer** can display additional properties of a workspace object. To display these properties, select **Edit > User Preferences > Workspace**, and select the **Enable workspace explorer filter properties** checkbox.

Figure 18 shows the **Workspace Explorer** with this option enabled.

Figure 18 Workspace Explorer Filter Properties



Hint: The filter properties preference displays basic information about the workspace object. To display all the attributes of a PhyreEngine™ object, select **Open Properties window...** from the object's context menu. See *Editing Objects* in *Chapter 5, PhyreEngine™ Objects*.

The filter properties are:

Modified – Whether or not the object has been changed since the last database save.

Read only – Whether or not the PhyreEngine™ database file is a read-only file. Applies only to database objects.

Linking – Whether or not the PhyreEngine™ object is referenced by another PhyreEngine™ object. If the object is a database, the property indicates whether another database is linked to this database.

A column can display the following:

The property is true.

The property is false.

The specific PhyreEngine™ database file cannot be found. It may not exist, or is currently unavailable.

Additional Features

- To change the alphabetical order of the object types to descending or ascending, click on the **Object** label in the properties header.
- PhyreStation automatically updates the filter properties every five minutes. To update the properties at any time, select **Refresh** from the **Workspace Explorer** context menu or press the **F5** key.
- To hide the titles from the properties' headers, select **Hide Text** from the context menu or press the **F4** key.

Hint: Before resolving PhyreEngine™ databases, switch off the filter properties to speed up the process.

Dropping External Files into a Workspace

You can load data into PhyreStation by dragging an external file onto any of the following:

- The workspace (the blank work area)
- A **Workspace Explorer** view (object, database, scene)
- A PhyreEngine™ database
- A PhyreEngine™ object

When you drop a file, PhyreStation iterates a set of PhyreStationDLL file handlers, which determine if the file can be handled, depending on where the file has been dropped. If the file can be handled, it is handled and the drop is completed. If there are no suitable file handlers, PhyreStation assumes that the file is a database file and adds it to the current workspace as a database. If the file is not a database file, PhyreStation creates a database containing a binary object to represent the item.

PhyreStation currently supports three types of file handlers: audio, CgFx, and texture images.

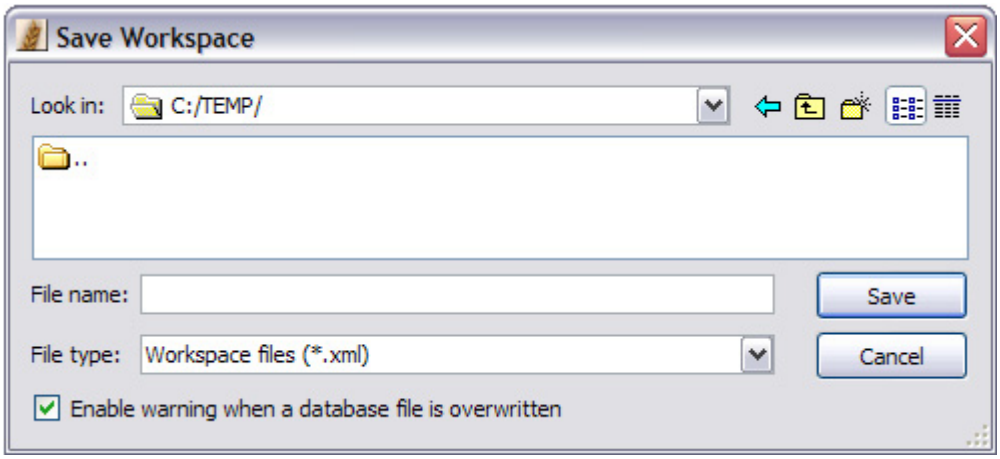
For scripting, this functionality is provided by the script command `AddExternalObject`. See *The Help Viewer* in *Chapter 1, PhyreStation Overview* for a list of available commands.

Saving a Workspace

The **File > Workspaces > Save** option is enabled when a workspace file is writeable and modified. This option is not available if the current workspace is the default workspace `Untitled.xml`.

You can save a workspace file at any time by selecting **File > Workspaces > Save As...** from the main menu. The **Save Workspace** dialog box is displayed.

Figure 19 Save Workspace Dialog Box



Automatically Saving a Workspace

To save the workspace automatically at specified intervals, select **Edit > User Preferences > Workspace** and set the **Auto save workspace** preference (the default is 'off'). This preference instructs PhyreStation to save the current list of databases in the workspace, the workspace preferences, and the window layout settings. PhyreStation does not save any PhyreEngine™ databases if they have been changed. If there are databases modified in the workspace, the workspace is saved but still marked as modified (an asterisk is displayed in the title bar).

If the current workspace is the default `Untitled.xml` workspace, PhyreStation does not perform automatic saves until the name of the workspace has been changed.

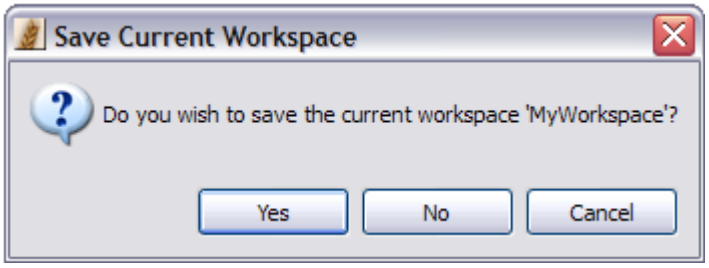
If the workspace file is write-protected, PhyreStation does not perform automatic saves. No warning is given.

Note: While the application is executing a script, the automatic save function is not carried out even if it is enabled.

Closing a Workspace

When you close a workspace session that has unsaved changes, a message box prompts you to save the workspace. It also warns you if any databases in the workspace have been modified.

Figure 20 Save Current Workspace Message Box



4 Databases

This chapter describes PhyreEngine™ databases and database files. It also explains how to perform operations on databases.

Overview

A PhyreEngine™ database (referred to here as a 'database') is a collection of PhyreEngine™ objects.

To add PhyreEngine™ objects to a database, you first create a database or load an existing database into the workspace. Normally, an existing database is a file on some media somewhere. However, a database can also be loaded from a 'live' PhyreEngine™ instance.

When you open an existing workspace, the databases associated with that workspace are automatically loaded, though not always resolved (resolving means collecting the objects to populate the workspace).

You can view and access the databases in the current workspace by opening a **Workspace Explorer – Databases** view.

Database Files

Database files contain PhyreEngine™ object data and information about object relationships. PhyreStation supports four database file formats:

Table 1 Database Files

File Format	Description	Action
PhyreEngine™ `.PSSG` files	Files with the '`.PSSG`' extension are binary data files. This is the default format for PhyreStation.	Import or export.
PhyreEngine™ `.hier` files	Files with the '`.hier`' extension are human-readable XML versions of PhyreEngine™ files.	Import or export.
PhyreEngine™ `.gz` files	Files with the '`.gz`' extension are compressed binary data files.	Import or export.
COLLADA files (`.xml` or `.dae`)	COLLADA is a popular standard file format for representing art assets.	Import only.

PhyreStation will accept files with any name or file extension, as well as those shown in *Table 1*.

Note: A PhyreEngine™ database file cannot contain the character '`#`'. PhyreStation will not accept the character '`#`' as part of an object's name.

Third-Party Data

A database file may contain data that is used by third-party software, but is not meaningful to the PhyreEngine™ engine. This data is identified by the PhyreEngine™ engine, wrapped inside a binary object, and then ignored by PhyreStation (unless a command is written to handle the binary object). Therefore, it is possible to load any type of `.xml` file into PhyreStation and see what looks like an empty database in the **Workspace Explorer**. This behavior is normal and does not indicate a bug. All non-PhyreEngine™ data is retained in the database file and can be accessed by the appropriate third-party software. A binary object can be viewed in a **Workspace Explorer – Objects** view; however, the data that the object represents is not visible.

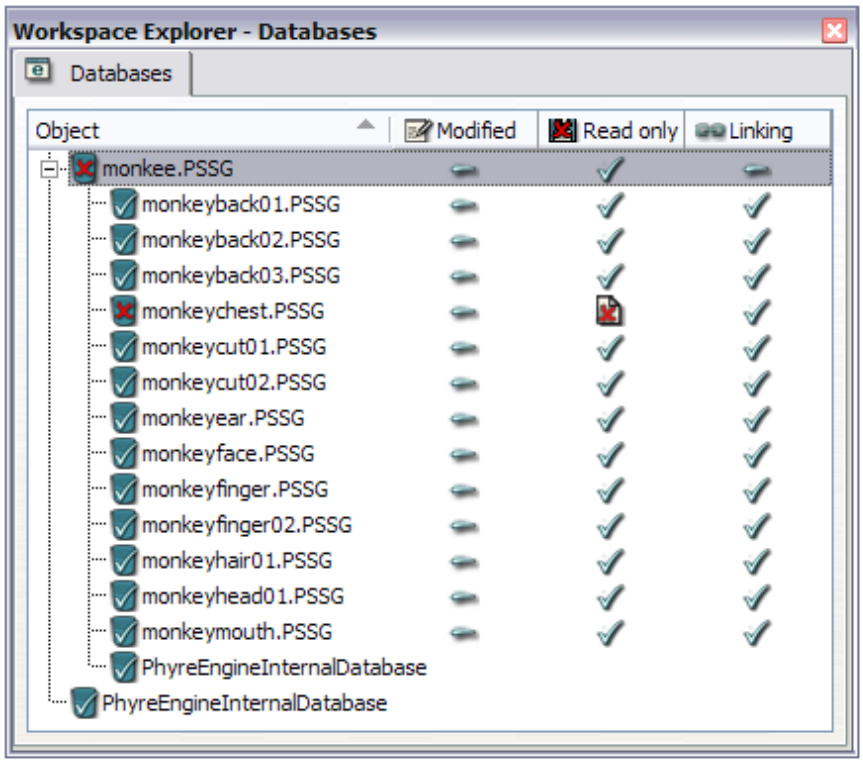
Browsing Databases

To view the databases in the current workspace:

(1) Open the **Workspace Explorer** using the tool bar button (see *Figure 16*).

(2) Right-click in the **Workspace Explorer** and select **Databases** from the context menu. The **Workspace Explorer – Databases** view displays all the top-level databases in the current workspace.

Figure 21 Workspace Explorer – Databases View



Database Icons

In the **Workspace Explorer – Databases** view (*Figure 21*), the database icon denotes state.

Loaded		The default state of all new databases added to the current workspace.
Link-Resolved		All the database's referenced elements, including its dependencies, have been successfully verified. To resolve a database's links, select **Resolve Links** from the database's context menu.
Link-Unresolved		A database is *link-unresolved* or broken if one or more of its referenced elements, or dependencies, cannot be successfully verified. A red cross on a top-level or Linked database's icon indicates that PhyreEngine™ failed to resolve all the resource links for that database. If any external database indicates a resource failure, the top-level database is also marked as not being resolved. You should fix any resolution failure problems before continuing. **Hint:** A linked database failure can occur if a linked database file cannot be found.

Objects Collected		All the objects contained in the selected database have been collected and loaded into the workspace. No linked objects (or linked databases) have been loaded into the workspace.
Unloaded		Unloading a database causes all its objects to be removed from both PhyreEngine™ and the workspace. Unloading databases is useful for making valuable system resources available. When you unload a database, any linked databases are also unloaded by PhyreEngine™ and the linked database representation is removed from the workspace where possible. However, if any linked databases are linked and loaded from *other* loaded databases, they remain loaded in memory. To unload a database, select **Unload** from the database's context menu.

Linked Databases

A linked database is a PhyreEngine™ database referenced by another PhyreEngine™ database. It is visible in a **Workspace Explorer – Databases** view when a database has been resolved and PhyreEngine™ has determined a resource connection.

Links are required when objects belonging to one database have dependencies on objects from one or more other databases. Linked databases are shown as child items of the database referencing them (top-level databases). A linked database can also be added to the current workspace as a top-level database.

Linked Databases and PhyreEngine™

PhyreEngine™ does not support the concept of top-level databases and linked databases. All databases are the same. The **Workspace Explorer – Databases** view reveals a resource link relationship between PhyreEngine™ databases that is otherwise transparent in other PhyreStation operations.

Hint: The only other way to establish if a database is linked to another database is to look in the object **Properties** view; an object may reference objects in other databases as dependencies or dependants. See *Editing Objects* in *Chapter 5, PhyreEngine™ Objects*.

The **Workspace Explorer – Databases** view shows only the first level of resource links to a potentially complex relationship between databases.

PhyreEngine™ databases cannot be added to the workspace as linked databases, only as top-level databases.

PhyreEngine™ treats each database as atomic, and does not support the partial loading of a database. PhyreEngine™ does not allow rendering of a database resource that has unresolved links, nor one that depends on a database with unresolved links.

Internally, PhyreEngine™ makes the link between the databases that is required to obtain the information needed to complete the resolution of the top-level database. After the resolution of a database is complete, PhyreEngine™ objects that are referenced by other PhyreEngine™ objects cannot be easily deleted or moved from the GUI. This ensures that PhyreEngine™ has all the necessary items available when it needs to perform any work.

If you delete a top-level database, which also happens to be a linked database, from a workspace, PhyreEngine™ continues to use the linked database. PhyreEngine™ would use the database even if you had not added it to the workspace.

Note:

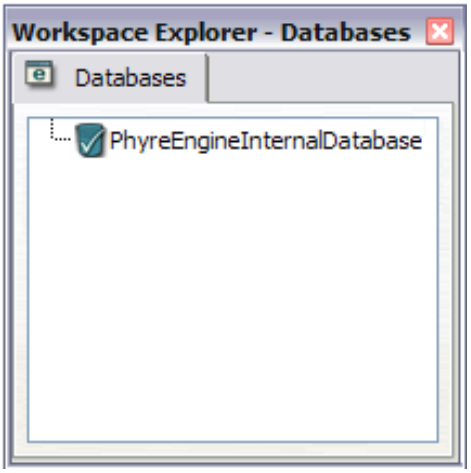
- Link/load states are not saved in workspace files. When loaded from a workspace file, a database is always put in the loaded or resolved state.
- Although a top-level database may have a complex interrelationship of resource dependencies, for example, a database linked to a database linked to another database, only first-level databases are shown in the **Workspace Explorer – Databases** view (see *Figure 21*).

The PhyreEngine™ Internal Database

The PhyreEngine™ internal database is a special database that is always present in every workspace. This database provides commonly used PhyreEngine™ objects that are required by most development projects.

When you create a new workspace, the PhyreEngine™ internal database is automatically added. This database is always visible in the **Workspace Explorer – Databases** view.

Figure 22 PhyreEngine™ Internal Database



Although you can change the PhyreEngine™ internal database, it is recommended that you do not permanently remove any of its original pre-installed objects.

Performing Database Operations

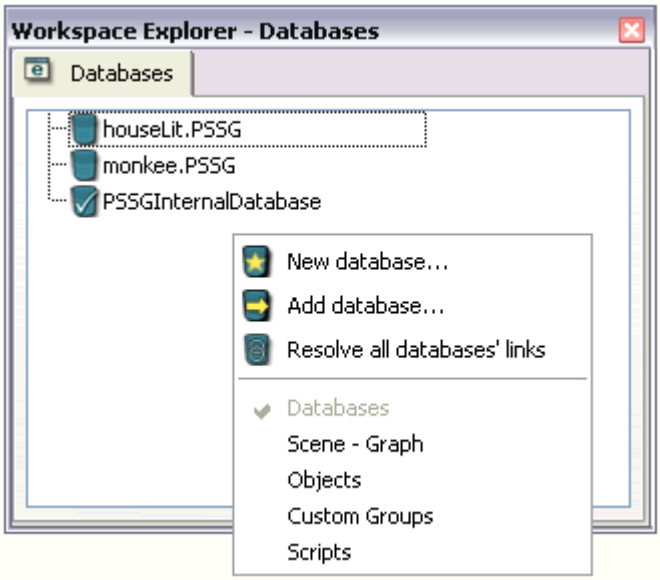
You can perform operations on databases using:

- The general context menu that is displayed when you right-click on the white space in the **Workspace Explorer – Databases** view.
- The specific context menu that is displayed when you right-click on a database name in the **Workspace Explorer – Databases** view.
- Scripts – See *Chapter 7, Scripts*

General Context Menu

To perform general database operations, right-click on the white space in the **Workspace Explorer – Databases** view.

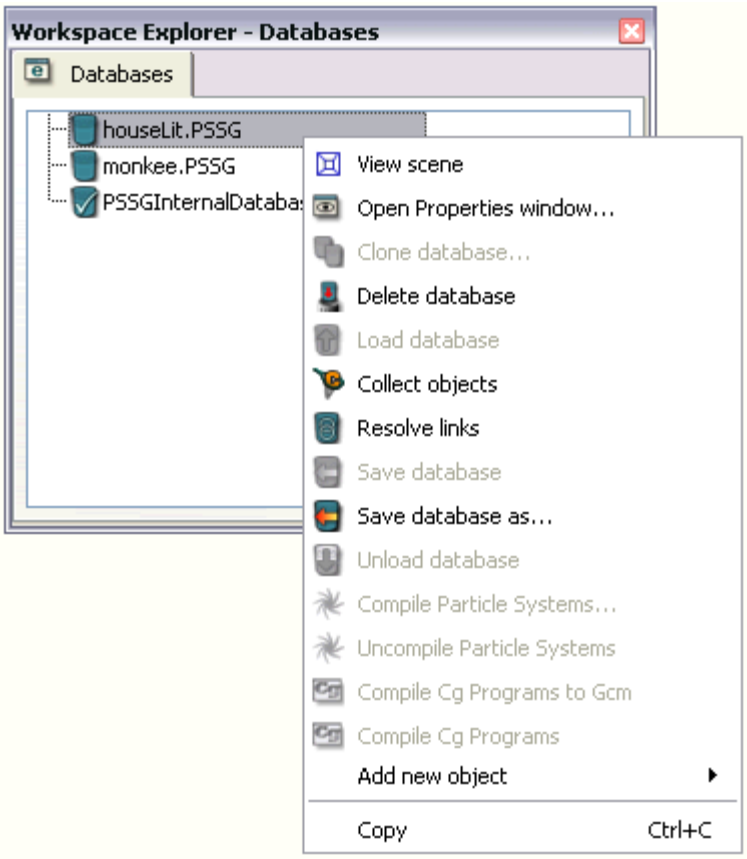
Figure 23 General Database Context Menu



Database Context Menu

To perform operations on a database, right-click on a top-level database.

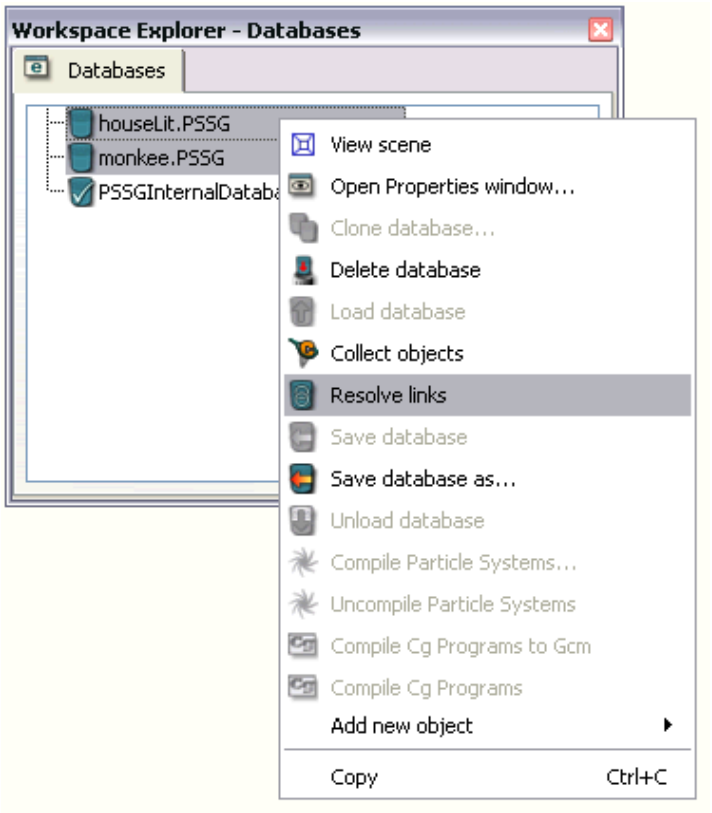
Figure 24 Specific Database Operations



You can perform operations on multiple databases at one time, by selecting the required databases and then right-clicking on the selection to display the context menu.

For example, in *Figure 25*, two databases are selected and resolved in a single step.

Figure 25 Resolving Multiple PhyreEngine™ Databases



Database Operations

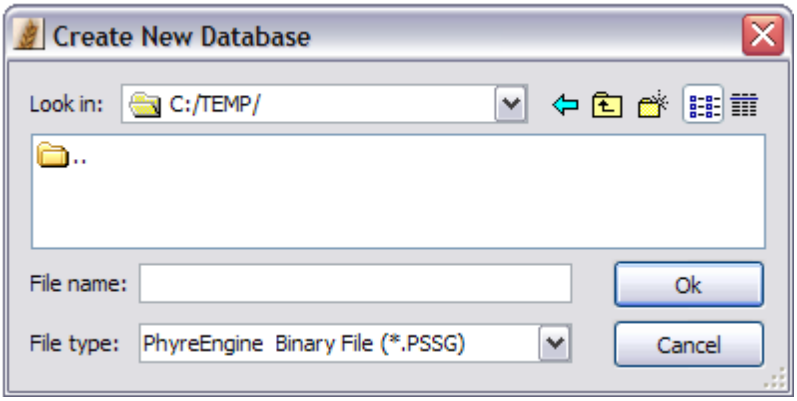
Creating a New Database

To create a new database, do one of the following:

- Select **File > Databases > New database...** from the main menu.
- Right-click the white space in the **Workspace Explorer – Databases** view, and select **New database...** from the context menu.

The **Create New Database** dialog box is displayed. Enter a name for the new database.

Figure 26 Create New Database Dialog Box



The type of database that is created is denoted by the filename extension:

- `*.PSSG` denotes a binary PhyreEngine™ database.
- `*.hier` denotes a textual PhyreEngine™ database.

For example, `MyFile.hier` is a PhyreEngine™ format database, saved as text. If no extension is entered, `*.hier` is added by default.

Note: Extensions are not case sensitive.

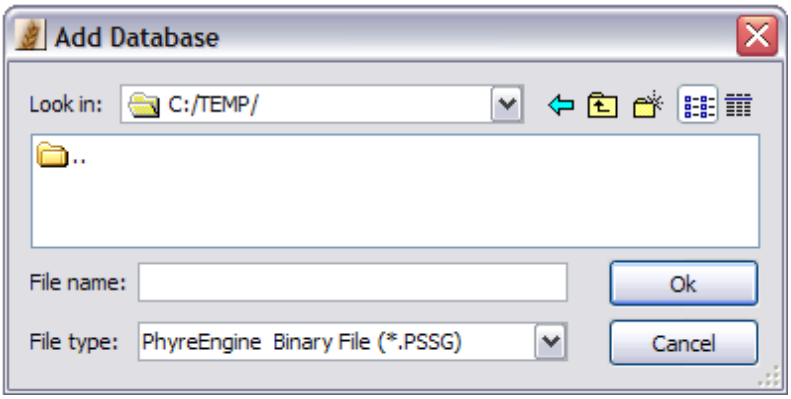
Adding an Existing Database

To add an existing database to the current workspace, do one of the following:

- Drag a PhyreEngine™ database file from an OS file browser into either the PhyreStation main window or the **Workspace Explorer – Databases** view. Invalid files are ignored without comment.
- Right-click the white space in the **Workspace Explorer – Databases** view, and select **Add database...** from the context menu.
- Select **File > Databases > Add database...** from the main menu.

In the latter two cases, the **Add Database** dialog box appears.

Figure 27 Add Database Dialog Box



Link-Resolving a Database

If a database has just been added to a workspace, the database will need PhyreEngine™ to resolve its dependencies. To resolve a database's links, right-click the database and select **Resolve Links** from its context menu.

To link-resolve all loaded databases in the current workspace in a single step, do one of the following:

- Right-click the white space in the **Workspace Explorer – Databases** view, and select **Resolve all databases' links** from the context menu.
- Select **File > Databases > Resolve all databases' links** from the main menu.

Note: Databases without any links to other PhyreEngine™ databases are automatically resolved.

Hint: To resolve links automatically when loading databases, select **Edit > User Preferences > Workspaces**, and set the **Auto-resolve databases when loading a new workspace** preference.

Link-resolving involves a comprehensive iteration of all referenced elements within the database, whether internal or external (that is, residing in separate databases).

The link-resolving algorithm performs the following steps:

- Tests the validity of resources that are requested by the current database, but found in other databases.
- All internal PhyreEngine™ objects are link-resolved.
- If any object depends on another object, the algorithm tests whether all appropriate resources are present and obtainable.
- Where appropriate, the algorithm loads the linked resources into memory, if they are not already present from previous operations.

Collecting Database Objects

To populate the workspace with the objects in the selected database only, right-click the database and select **Collect Objects** from its context menu.

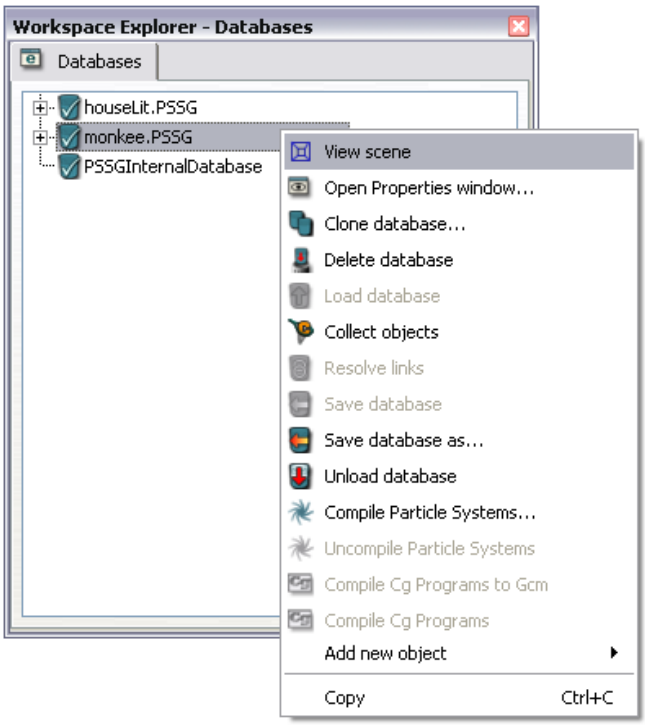
This is an alternative command to **Resolve Links**, which collects objects from the selected database and all of its linked databases. You can use **Collect Objects** whenever a database is loaded, irrespective of the resolve state of the database.

For scripting, this functionality is provided by the `DBCollectObjects` script command.

Viewing a Scene

To view the whole scene in a database, right-click the database and select **View Scene** from its context menu.

Figure 28 Viewing a Scene



Note:

- The **View Scene** menu option is available only if the database contains a scene (PhyreEngine™ root node).
- You may not be able to see your scene if you are 'inside' the model. Zoom out by moving the camera until you can see the scene.

PhyreStation creates temporary workspace PhyreEngine™ objects that represent a camera node and a light node (see *Temporary Objects* in *Chapter 5, PhyreEngine™ Objects*), irrespective of whether that database already contains a camera or light object. The **Render View** automatically opens, and displays the scene in the database using the temporary camera and light objects. No other objects in the database are affected. The database and the workspace are not marked as dirty when this command is used.

Saving a Database

To save a writable database that has been modified, right-click the database and select **Save database** from the context menu.

To save a database with a new file path or name, right-click the database and select **Save database as...** from the context menu**.** The **Save Database As** dialog box is displayed. Saving a database under a new name automatically updates its links to other databases.

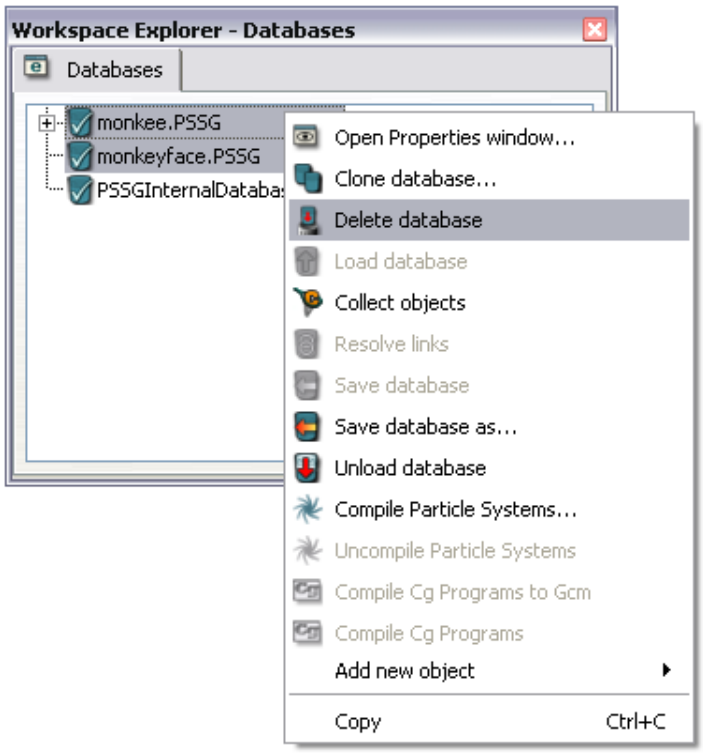
Note:

- PhyreStation does not support exporting data to the COLLADA file format.
- Depending on how PhyreEngine™ has implemented its 'save data' mechanism, the size of the database file or file content structure may be different from when it was loaded, even though the name of the database has not changed.
- PhyreStation does not save temporary workspace PhyreEngine™ objects.

Deleting a Database

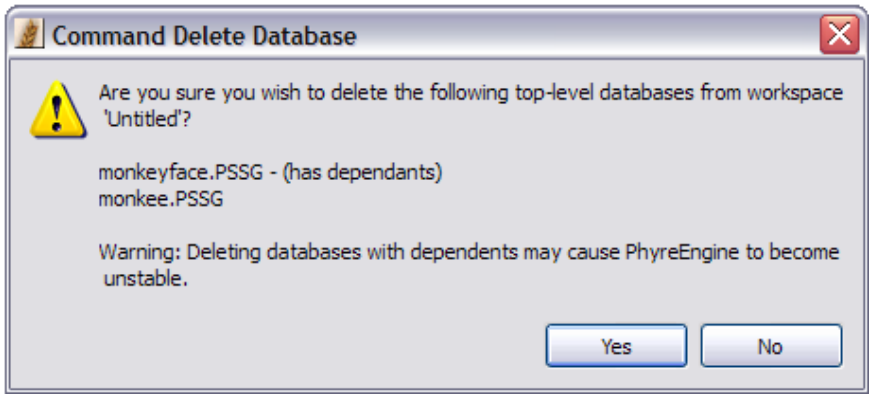
To delete one or more databases, select the database(s) and then right-click and select **Delete database** from the context menu.

Figure 29 Deleting Multiple Databases (linked)



A dialog box is displayed that asks you to confirm the deletion.

Figure 30 Delete Database Confirmation Dialog



The message "has dependants" or "not saved" is displayed next to each database listed for deletion.

- "Has dependants" means the database is referenced by, or linked to, one or more other databases. Deleting the database breaks the reference to objects.
- "Not saved" means that you will lose changes made to the database since it was last saved.

Note: If the deleted database has link-resolved databases associated with it (dependants), the dependants are also removed from memory. Be careful when deleting databases with dependants, as this causes PhyreEngine™ to unload the database and may break links to resources that it still requires. This can cause PhyreEngine™ to become unstable.

Unloading and Reloading a Database

You can temporarily unload a database if it is not referenced by other databases. Unloading a database is useful to recoup system memory, to re-resolve a database, or if the database is broken. To unload a database, right-click the database and select **Unload database** from the context menu.

To reload a database that has been temporarily unloaded, right-click the database and select **Load database** from the context menu.

Cloning a Database

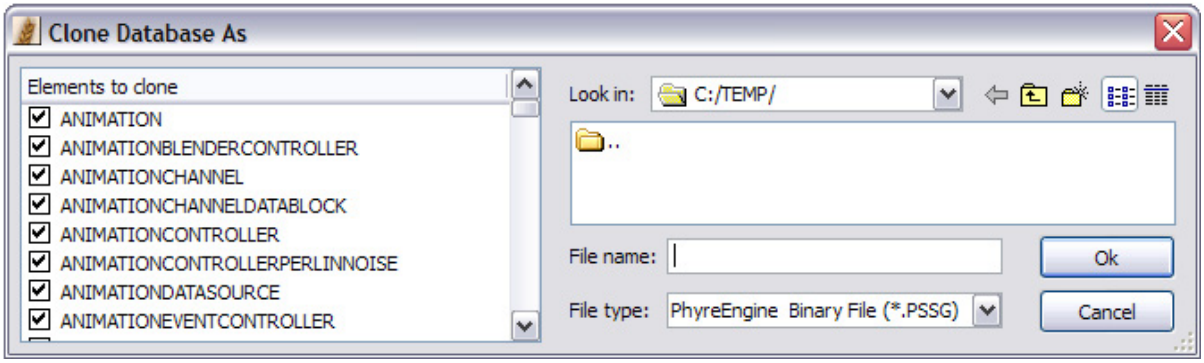
You can clone a database but only when it has been resolved successfully.

To clone a database:

(1) Right-click on the database and select **Clone database...** from the context menu. The **Clone Database As** dialog box is displayed.

(2) Select the PhyreEngine™ object types that you wish to clone.

Figure 31 Clone Database Dialog Box



By default, all PhyreEngine™ object types are selected for cloning; that is, the entire database is cloned exactly.

The clone database operation does not create a PhyreEngine™ database file even though you can enter a file path if you wish. The file path is used by the **Save database** command and the script version of the command, `SaveDatabase`.

Viewing Database Properties Using the Properties Viewer

To view the properties of a database, select the database and do one of the following:

- Right-click on the database and select **Open Properties window...** from the context menu. This opens a static **Properties Viewer** for the database.
- Click the **Show the Dynamic Property Viewer** button from the main menu. The dynamic **Properties Viewer** displays the properties for the currently selected object. Select the database you are interested in to see its properties in the **Properties Viewer**.

5 PhyreEngine™ Objects

Overview

A PhyreEngine™ object is the representation of an art or data asset. All PhyreEngine™ objects are derived from the PhyreEngine™ class PObject. PhyreEngine™ objects are a subset of workspace objects.

In PhyreStation you can browse, create, edit, clone, move, and delete PhyreEngine™ objects in the current workspace.

To see a list of all supported PhyreEngine™ object types, select **Help > Help Contents** from the main menu, and then click **Registered PhyreStation Types**. You can also see a list of available object types by selecting **Help > About PhyreStation** and selecting the PhyreStationDLL component.

You can create custom object types, and then create custom commands to manipulate objects of those types. You must register custom object types and custom commands in the PhyreStationDLL to be able to use them. For more information, see *Chapter 8, Customizing PhyreStation*.

Temporary Objects

In certain situations, PhyreStation creates temporary objects to perform a function; for example, when the user selects the **View Scene** database command. See *Viewing a Scene* in *Chapter 4, Databases*.

A temporary object is a PhyreEngine™ object. It exists in a database and is represented in the workspace just like any other object. However, temporary PhyreEngine™ objects are removed from the database just before any `Save`, `SaveAs`, or `Clone Database` commands are carried out, and are reinserted after these commands are finished. Temporary objects can be identified by the prefix `_PhyreStationPSSGTmpObj` in their name.

Note:

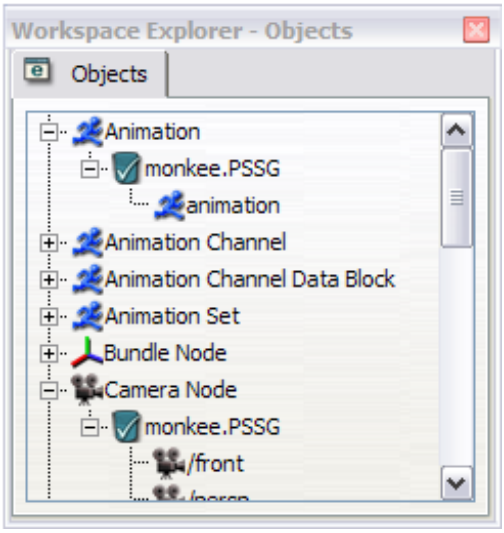
- Temporary PhyreEngine™ objects are added to the database object count. From a PhyreStationDLL point of view, they are not distinguishable from any other PhyreEngine™ object (apart from their name).
- Temporary PhyreEngine™ objects can contaminate a database if a command or action from the PhyreStationDLL side does not handle them correctly. Temporary objects can be removed by using the script command `RemoveUnusedObjects()` or code that can identify them. If temporary PhyreEngine™ objects are deleted, they are automatically created again when needed.
- PhyreStationDLL commands cannot operate on temporary PhyreEngine™ objects from within the PhyreStation's GUI environment.

Browsing Objects

Before you can access PhyreEngine™ objects, they must be present in the workspace; that is, the workspace must contain a database with one or more PhyreEngine™ objects and that database must be either resolved or collected.

To browse PhyreEngine™ objects, open the **Workspace Explorer** using the tool bar button, then right-click in the **Workspace Explorer** and select **Objects** from the context menu. The **Workspace Explorer – Objects** view displays all the PhyreEngine™ objects in all link-resolved PhyreEngine™ databases in the workspace.

Figure 32 Workspace Explorer – Objects View



By default, the objects are grouped first by PhyreEngine™ object type, and then by database. To display all categories, including those that contain no objects, select **Show Empty Categories** from the view's context menu.

To change the view so that the objects are grouped by database and then by PhyreEngine™ object type, select **View By Database** from the view's context menu.

In the **Workspace Explorer**, you can also view PhyreEngine™ objects that are part of a scene graph or custom group, by selecting either **Scene-Graph** or **Custom Groups** from the **Workspace Explorer** context menu.

Accessing Objects Using Viewers and Editors

You can access all PhyreEngine™ objects using the **Workspace Explorer – Objects** view as described in the previous section. In addition, some types of PhyreEngine™ objects possess special attributes that you can view and edit using dedicated PhyreStation windows known as Viewers and Editors.

Note:

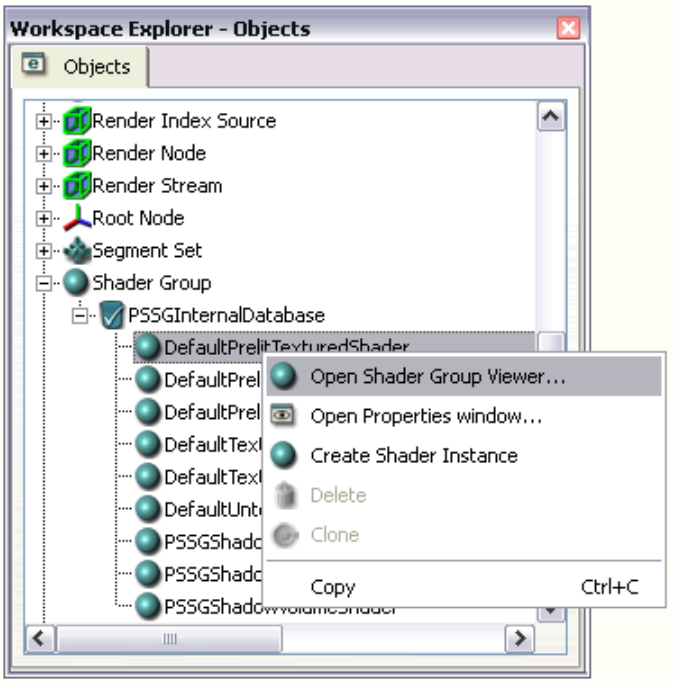
- You can also view and edit an object's properties using the **Properties Viewer**. See *Editing Objects* further on in this chapter.
- Some PhyreEngine™ objects may not be locked until PhyreEngine™ determines they are required at the time an operation is carried out. A PhyreEngine™ Animation Set is an example; you can delete it (this is not advised) but not after an animation has been played.
- Some PhyreEngine™ objects may not be present until PhyreEngine™ determines it needs to create those objects to continue.

Objects	Viewer/Editor	Reference
Camera node objects	Render Viewer	See *Render Viewer* in *Chapter 11, Viewers*.
Segment set objects	Segment Set Viewer	See *Segment Set Viewer* in *Chapter 11, Viewers*.
Shader instance objects	Shader Instance Viewer or Shader Instances Viewer	See *Shader Instance(s) Viewer* in *Chapter 11, Viewers*.
Texture objects	Texture Viewer	See *Texture Viewer* in *Chapter 11, Viewers*.
Shader program objects	Shader Program Viewer	See *Shader Program Viewer* in *Chapter 11, Viewers*.
Shader group objects	Shader Group Viewer	See *Shader Group Viewer* in *Chapter 11, Viewers*.
Animation objects	Animation Editor	See *Chapter 13, Animation Editor*.

Animation target blender objects	Graph Editor	See *Target Blender Editor* in *Chapter 12, Graph Editors*.
Modifier network objects	Graph Editor	See *Modifier Network Editor* in *Chapter 12, Graph Editors*.
Modifier network Instance objects	Graph Editor	See *Modifier Network Instance Editor* in *Chapter 12, Graph Editors*.
Particle emitter node objects	Particle Editor	See *Chapter 14, Particle Editor*.

To display the dedicated viewer or editor for one of the special objects listed above, right-click on the object in the **Workspace Explorer – Objects** view and select **Open <object type> Viewer** or **Editor** from the context menu.

Figure 33 Displaying an Object Viewer



Performing Object Operations

Performing an object operation means executing a command for a selected object or objects.

- You can perform an operation on an object by right-clicking the object in the **Workspace Explorer – Objects** view and selecting the operation from the context menu.
- The operations on the context menu can be PhyreStation internal commands or commands developed and registered by a third party in the PhyreStationDLL.
- You can carry out operations on multiple objects. You can select multiple objects from one window or from multiple windows. You can also perform operations on objects by executing a script.
- You can drag-and-drop objects between windows.

Hint:

- A drag-and-drop operation or a context menu command may not be valid for every object in a multiple selection. Check the **Log** window for confirmation of the operation. It is recommended that you invoke object-specific commands on objects of the same type.
- If you press the **Shift** key while selecting a set of objects in a tree hierarchy, for example nodes in the **Scene-Graph** view, any collapsed child nodes (that is, they are not visible) will also be selected. Performing an operation on the selection will also be performed on the hidden child nodes. To avoid this, expand any collapsed nodes and, using the **Ctrl** key, deselect the child nodes.
- If you are making a selection using the **Shift** key, and you have selected a collapsed tree view item, select the next item down in order to select all the hidden child items of the collapsed item, and then deselect the next item down. Otherwise only the items that are highlighted are selected and not the hidden child items.

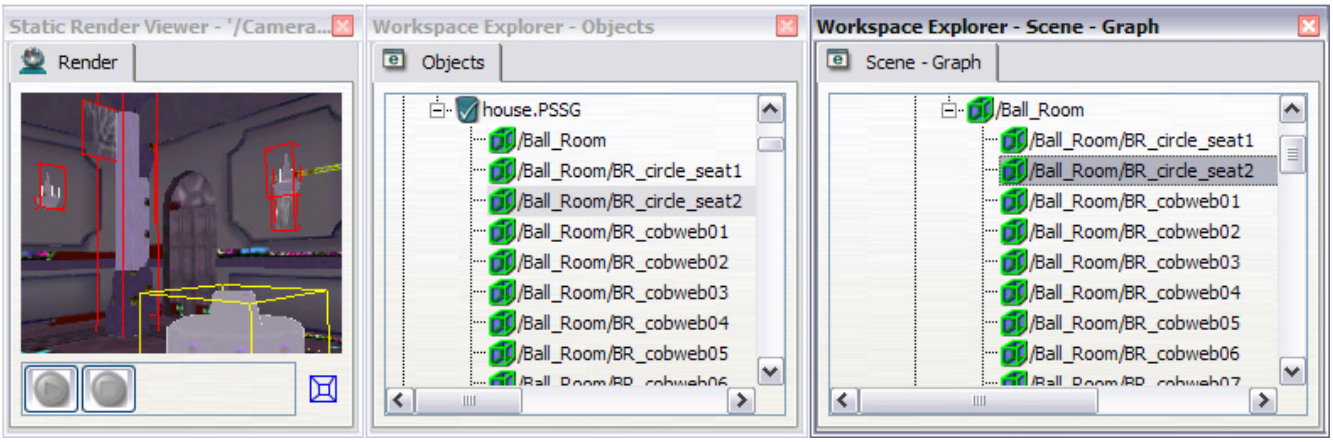
Selection from Multiple Windows

Selecting a visible object in one window automatically highlights the object as selected in any other window containing that object, even if the object is not currently visible in that window.

Objects are highlighted as follows:

- Blue: indicates selected items in the active window (the window with focus).
- Grey: indicates selected items in other window(s).
- Light blue: indicates a parent item that has a hidden, selected item(s).
- Yellow: indicates selected items in the **Render** view (the bounding box).

Figure 34 Selection from the Render Viewer



Note: If the selected item is not visible, the item's closest visible parent will be highlighted with a light blue color.

Hint:

- If you are selecting a large number of items, close any unwanted **Workspace Explorers.** This will speed up the selection or deselection of items, as there are fewer views for PhyreStation to update**.**
- To see selected objects in the **Render** view, enable the **Bounding Boxes** or **Highlight Selection** option on the context menu. Highlighting colors do not apply to this view.

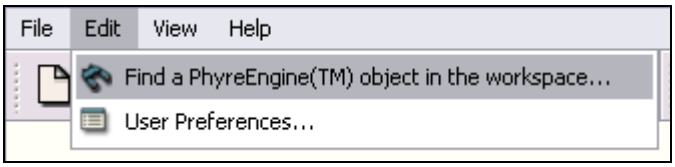
Object Operations

Finding Objects in the Workspace

A workspace may contain several thousands of PhyreEngine™ objects. The **Find** dialog allows you to search the current workspace and display a list of PhyreEngine™ objects that match particular search criteria.

To display the **Find** dialog, select **Edit > Find a PhyreEngine(TM) object in the workspace...** from the main menu or click the **Find** button on the main toolbar.

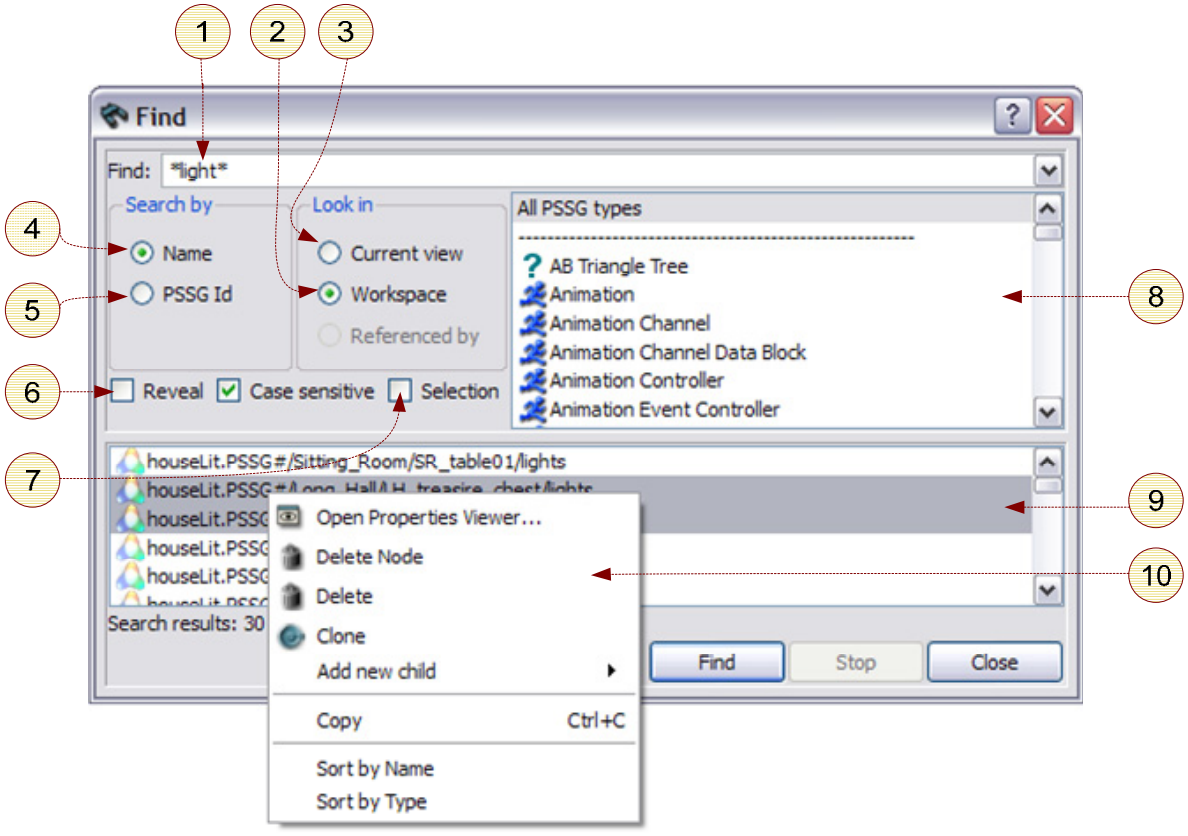
Figure 35 Display Find Dialog



From the **Find** dialog, you can either:

- Reveal the object, then expand and position the object in a **Workspace Explorer** window so that it is visible (item 6).
- Display a context menu to carry out operations on all, or part, of the search result (item 10).

Figure 36 Find Dialog Box



(1) Enter text search criteria or drag-and-drop PhyreEngine™ objects from any **Workspace Explorer** onto the edit line. Alternatively, right-click on the edit line and select **Paste from views...** from the menu. You can then edit the text to search for similarly named objects. You can use wildcard characters.

(2) Search all the PhyreEngine™ objects in the workspace.
(3) Search only for the objects in the current window with focus.
(4) Search the workspace by PhyreEngine™ object name as seen in the **Workspace Explorer** window; this does not include the database name.
(5) Search the workspace by PhyreEngine™ object link ID (*databaseName#objectName)*.
(6) Reveal the workspace objects that were found in a search in any open **Workspace Explorer** windows.
(7) Add items chosen from the search results list (item 9) to the overall object selection in the workspace.
(8) Restrict the search further by only searching the selected PhyreEngine™ object types.
(9) A list of PhyreEngine™ objects that were found using the last search criteria (item 1).
(10) Command context menu for all, or a selection of, the search results.

To search for an object:

(1) Enter text search criteria into the **Find** box (item 1). The search criterion can be a PhyreEngine™ object name or a combination of name and search key symbols.
(2) Click the **Find** button to start a search. A list of objects matching the search is displayed along with the actual number of items found (item 9). You can make further selections from the items listed. Right-click on the selection to display the objects' context menu (item 10).

The **Find** dialog automatically stores the last search criteria text in a history buffer so that it can be reused later.

> **Note:** The context menu displays a list of commands for all the PhyreEngine™ object types selected. As a result, when you select a command from a context menu, it may fail for some PhyreEngine™ objects because the command may not be appropriate for that object.

Advanced Search

A more advanced way to filter the results is to create expressions using the symbols in *Table 2*:

Table 2 Search Symbols

`*`	This matches zero or more of any characters.
`?`	This matches any single character.
`[...]`	Sets of comma-separated characters can be represented in square brackets.

Table 3 gives some examples of expressions that use these symbols. Assume your workspace has three objects: `Texture1` in database `DB1`, and `texture2` and `_textureOld` in `DB2`.

Table 3 Expressions Using the Search Symbols

Search by label	
`*exture*`	Returns all the textures in the workspace
`?texture?`	Returns `DB1#Texture1` and `DB2#texture2`
`*`	Returns all the objects in the workspace
Search the workspace by PhyreEngine™ object ID	
`*#?[T,t]exture?`	Returns `DB1#Texture1` and `DB2#texture2`
`DB?#*`	Returns all the objects in the two databases
`DB?#[T,t]exture[1-2]`	Returns `DB1#Texture1` and `DB2#texture2`

The length of the expressions is unlimited. Any extra spaces or consecutive "`*`" symbols are ignored. A single "`*`"retrieves all the objects.

Creating Objects

PhyreStation allows you to create new PhyreEngine™ objects in various ways:

- Add a new object to a database by selecting **Add New Object** from the database's context menu.
- Drag an external object file into PhyreStation, for example a texture file. (Note that this method does not work for all object types; a suitable PhyreStationDLL file handler must be present).
- Add a new child node to a node by selecting **Add New Child** from the node's context menu in the **Workspace Explorer – Scene-Graph** view.
- Add a particle system by creating a Particle Emitter Node in a selected database and then choose **Create Particle System** from the node's context menu.
- Execute a `NewObject` command from the **Command** window or from within a script.

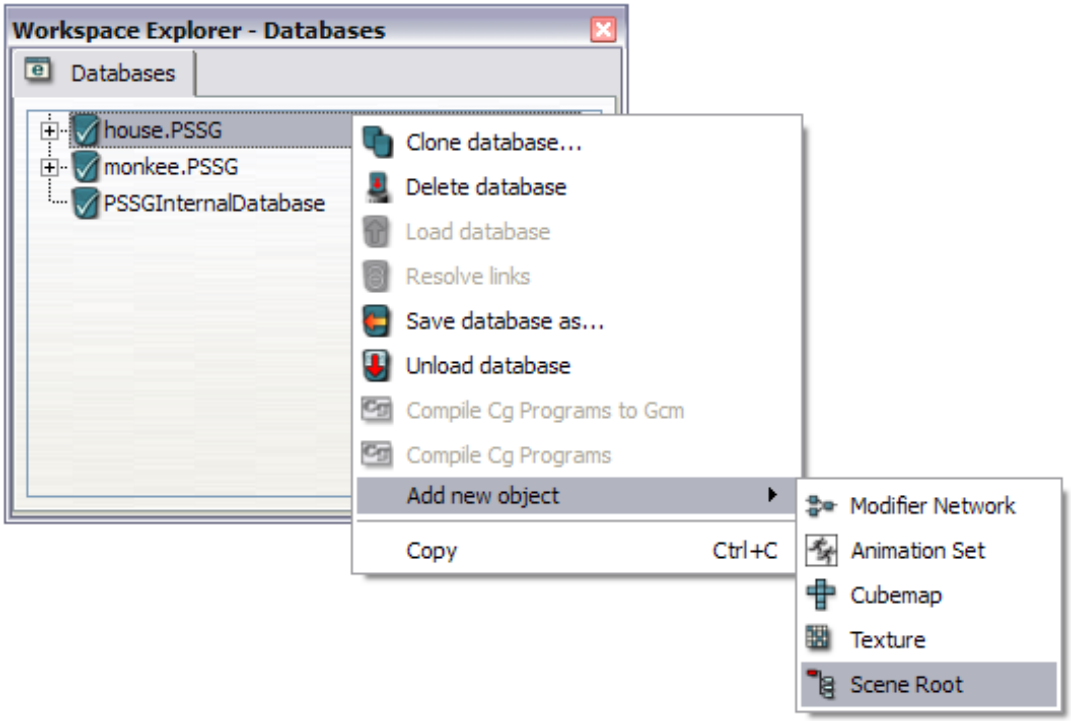
The first two methods are described below.

Adding an Object to a Database

To create a new PhyreEngine™ object and add it to a database, do one of the following:

- Select an object in another database and drag it to the target database, then select **Clone** from the context menu. See *Cloning Objects* in *Chapter 5, PhyreEngine™ Objects*.
- Right-click the database in the **Workspace Explorer**, and then select **Add new object > [Object type]** from the context menu.

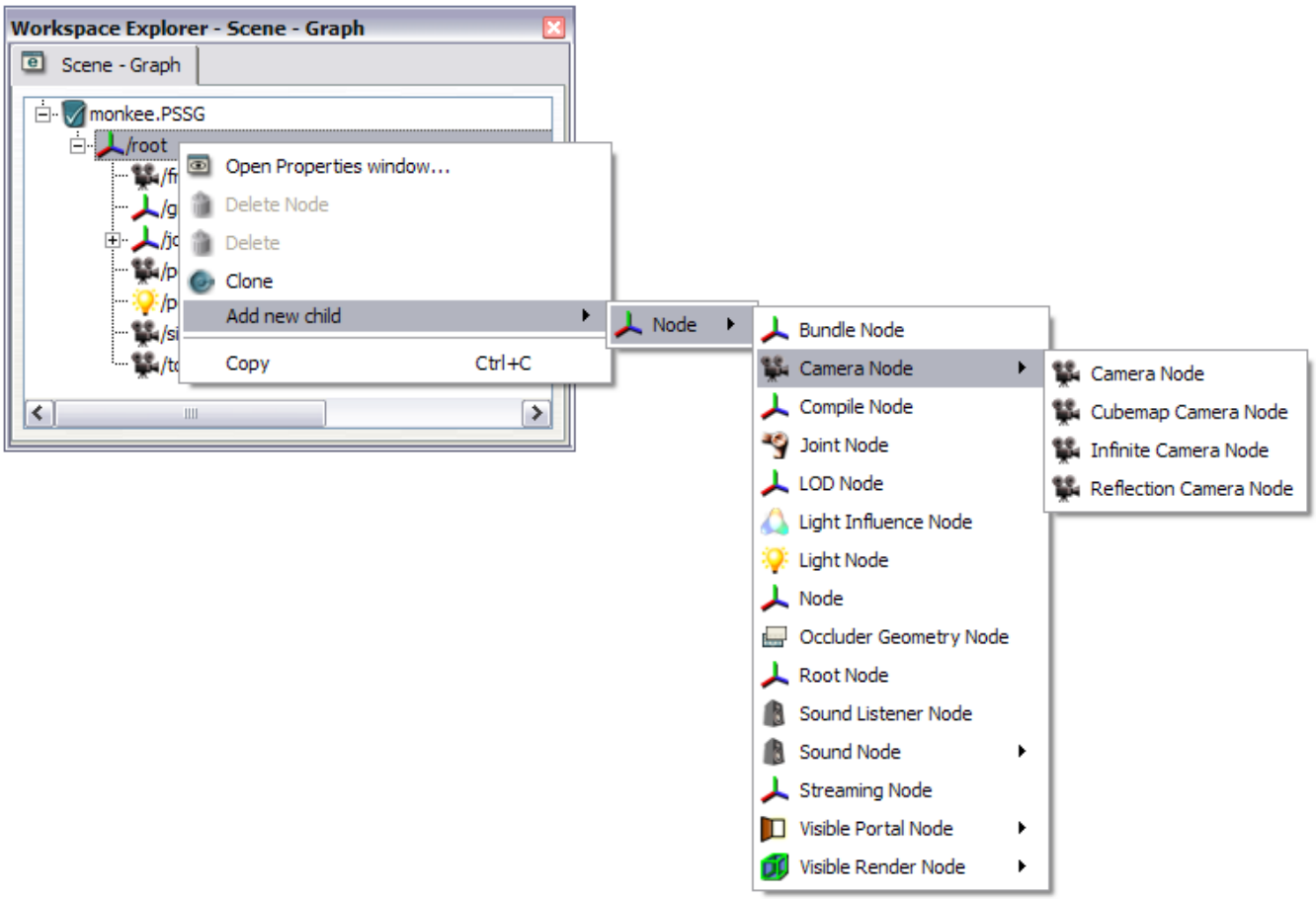
Figure 37 Adding a New PhyreEngine™ Object to a Database



Adding an Object to a Scene-Graph

To add a new node object to a scene-graph, right-click the parent node icon in the **Workspace Explorer – Scene-Graph** view, and then select **Add new child > Node > [Node type]** from the context menu.

Figure 38 Adding a New PhyreEngine™ Node Object

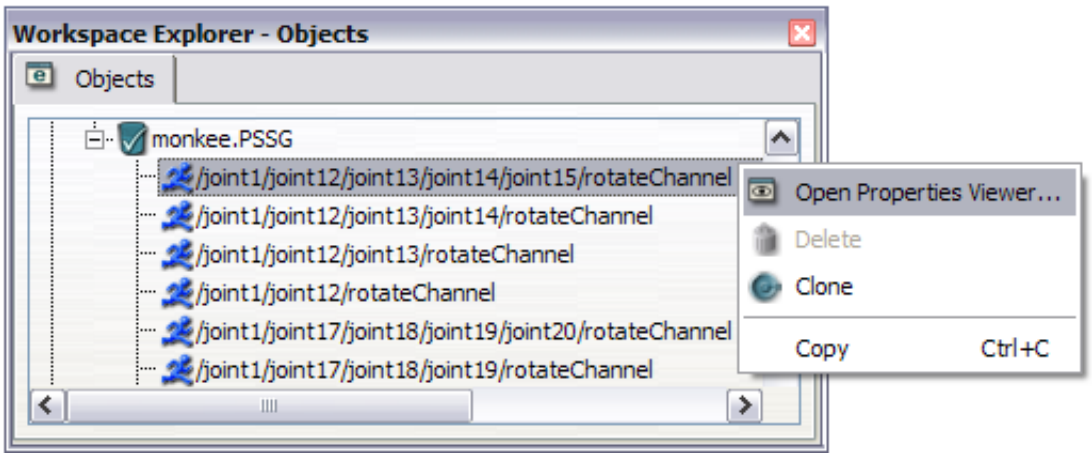


Editing Objects

To view and edit the properties of a PhyreEngine™ object, select the object in any **Workspace Explorer – Objects** or **Scene-Graph** filter view and do one of the following:

- Right-click on the object and select **Open Properties Viewer...** from the context menu. This opens a static **Properties Viewer** for the object.
- Click the **Show the Dynamic Property Viewer** button on the main toolbar. A dynamic **Properties Viewer** displays the properties for the currently selected object. Select the object you are interested in to see its properties in the **Properties Viewer**.

Figure 39 Opening a Properties Viewer Window (Static)



For information about dynamic and static modes, see *Dynamic and Static Modes* further on in this chapter.

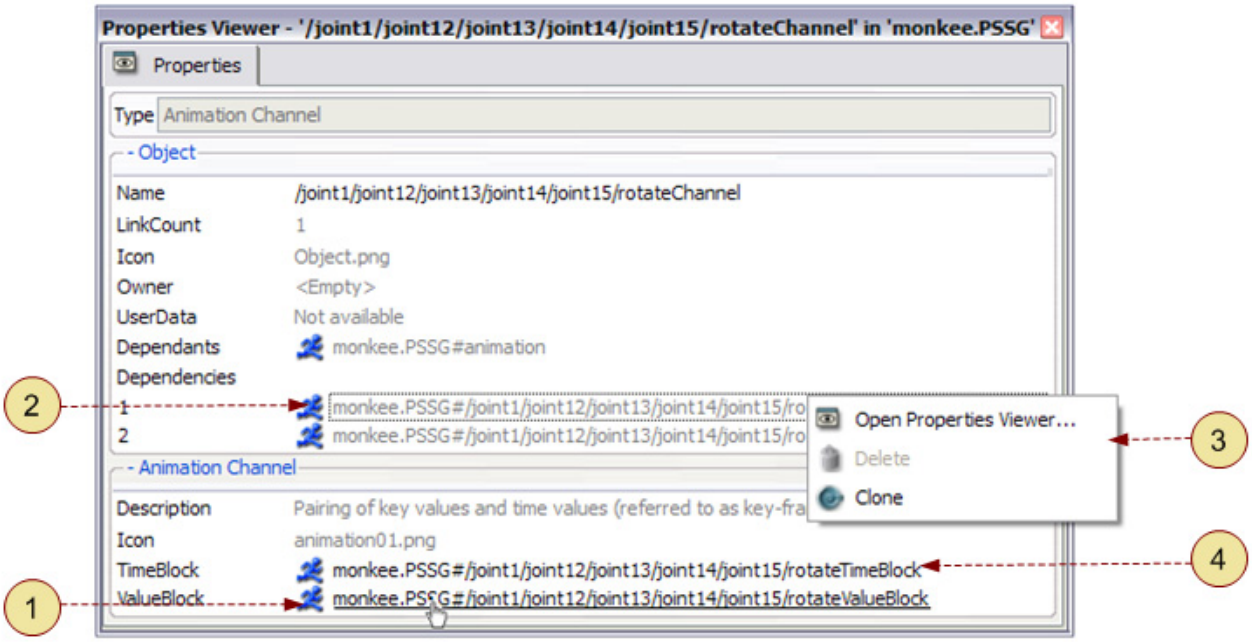
You can also edit a PhyreEngine™ object's properties by executing a script that changes the object either directly or by some other related change.

When one or more attributes of an object have been changed, the object, the database it belongs to, and the current workspace are marked as dirty. The database must be saved in order for the changes to become permanent.

The Object Properties Viewer

The **Properties Viewer** displays the value of an object's attributes. Some of the attributes' values can be edited directly either by entering new values or choosing from a list of options.

Figure 40 Properties Viewer Window



(1) Move the cursor over a PhyreEngine™ object to reveal it as a hyperlink. Click the link once to change the **Properties Viewer** to show the object specified by the link.

(2) Items in the table that are grayed are disabled. They cannot be edited or changed.

(3) Right-click a child PhyreEngine™ object to display the context menu.

(4) A PhyreEngine™ object name, which is a secondary object to this view.

> **Hint:** It is recommended you close any unwanted **Properties Viewer** windows. PhyreStation uses resources to keep these windows up-to-date, and so may become slow to respond.

Some of the properties displayed in the **Properties Viewer** may be references to other PhyreEngine™ objects (see *Figure 40*). The **Properties Viewer** allows you to select and execute commands on these secondary PhyreEngine™ objects.

The properties displayed for a PhyreEngine™ object are governed by the current PhyreEngine™ object's properties' classes and static functions compiled in the PhyreEngine™ utility Extra source code with the PhyreStationDLL. You can customize this. The PhyreEngine™ objects' attributes are accessed using the `SetObjectAttribute` and `GetObjectAttribute` functions. If an attribute field is not available, or there is an error in the PhyreStationDLL, PhyreStation displays "Unhandled PhyreEngine Attribute type" in place of the parameter.

Dynamic and Static Modes

The **Properties Viewer** window has two modes of operation: a *static* mode and a *dynamic* mode. In the static mode, the subject of the view remains constant, even if you subsequently select other objects in the workspace. The dynamic mode view changes to display the latest object you have selected.

To distinguish between the two modes, the background color of the tab changes to indicate the mode of operation. A grey or neutral background indicates that the window is operating in *static* mode. A blue background indicates that the window is operating in *dynamic* mode.

The *dynamic* mode version is displayed by clicking on the **Properties Viewer** window icon in the toolbar. The *static* mode version is displayed by selecting the **Open Properties Viewer...** option from the object's context menu.

Object Properties History Navigation

The **Properties Viewer** operates a subject **History Navigation Toolbar** (see *Window History* in *Chapter 15, The GUI*). This can be useful when navigating objects that have a parent-child relationship or that are referenced by other objects. The history buffer operates differently depending on the mode of the view.

- In dynamic mode, the toolbar operates two separate history buffers or lists. One list displays all objects that were selected since the window was opened. The other list records the objects navigated by moving through a parent child relationship (*Figure 38*, item 4).
- In static mode, the history feature is usually disabled because typically only one object is displayed. However, if you change the subject of a static viewer by dragging and dropping objects into it, the history feature is enabled.

Changing an Object's Attribute Values

Some of an object's attributes may be editable. If you do edit the value of an object's attribute and PhyreEngine™ determines that the new value is not valid given the object's context, PhyreEngine™ will not set the new value and the value in the **Properties Viewer** will revert to the value prior to the edit. A status message for the operation may appear in the **Log** window.

Item 4 in *Figure 40* represents a child PhyreEngine™ object that is referenced by the PhyreEngine™ object that is the subject of the **Properties Viewer**. You can change the child object reference by selecting another PhyreEngine™ object in a **Workspace Explorer** and dragging it onto the child object in the view. PhyreEngine™ validates the change.

Updating Referenced PhyreEngine™ Objects

Changing the properties of a PhyreEngine™ object in a **Properties Viewer** (dynamic or static) can affect the state of secondary objects that rely on the changed primary object.

Ideally, a change to a primary object should automatically update any secondary object windows. However, by default PhyreStation does not do this because it affects the performance of the tool when updating other windows. To enable this functionality, select **Edit > User Preferences...** from the main menu, then select the **PhyreStation** tab and set the **PhyreEngine™ objects referenced by other PhyreEngine™ objects** preference.

Note: The relational information between PhyreEngine™ objects that PhyreStation requires to update secondary object windows is gathered during the loading and resolving of a PhyreEngine™ database, therefore it takes longer to perform these operations.

Viewing Database Statistics

The **Properties Viewer** reflects the state of each database as it does objects' attributes. The database attributes or statistics that are available in the **Properties Viewer** are determined by the current PhyreEngine™ database's properties' classes and static functions compiled in the PhyreEngine™ utility Extra source code with the PhyreStationDLL. You can customize this to suit your requirements.

Cloning Objects

To clone one or more PhyreEngine™ objects, right-click the object(s) in the **Workspace Explorer – Objects** view and select **Clone** from the context menu. A copy of the selected object is made and a new object is created from that copy.

Hint: To clone all or selected objects in a database, see *Cloning a Database* in *Chapter 4, Databases*.

Moving or Deleting Objects

To delete one or more PhyreEngine™ objects, do one of the following:

- Right-click on the selected objects and choose the required delete action from the context menu. You may not be able to delete one or more objects if they are referenced by another PhyreEngine™ object.
- Drag an object from one database to another. This deletes the object from the source database and recreates it in the destination database.

To move an object, do one of the following:

- Drop an object onto a destination database and select **Move** from the context menu. The **Move** operation is not available if the object is referenced by another PhyreEngine™ object. Objects of type PhyreEngine™ PNode cannot be moved.
- Select and drag an object to another database.
- Select and drag a node to a new parent node or database.
- Perform a deep clone of the object and then delete the original.

A move is a Delete and an Add operation combined. When added, some of the object's attributes remain the same while other attributes are changed or re-referenced to other objects, for example a new parent object. A move may not be allowed from the GUI if there is a link to this object. However, you can still perform a move using a script command.

Hint: Use the **Properties Viewer** to display the **Link Count** attribute for a given PhyreEngine™ object. If this attribute is not zero, the PhyreEngine™ object is referenced by another object and you may not be able to delete the object from the GUI. If this is the case, the object can be deleted using the script command `DeleteObject()`.

6 PhyreStation Commands

This chapter describes the various PhyreStation command types, and the different ways of issuing them.

Overview

Every non-trivial operation performed by PhyreStation involves the execution of a command. PhyreStation commands may originate either from the PhyreStationDLL component, or from within PhyreStation itself.

In most cases, a command performs only one task. However, a command can call other commands or a command can work on multiple objects (only from the GUI).

The command system has been designed to allow developers to extend PhyreStation with new commands. New commands are either developed in a PhyreEngine™ User utility (for your own work) or put into the PhyreStationDLL (for individual stand-alone commands).

PhyreStation Internal Commands

PhyreStation is supplied with a small number of internal commands. These include all workspace commands (such as `OpenWorkspace()`) and many of the database commands (such as `ResolveLinks()`). Because PhyreStation source code is not supplied, PhyreStation internal commands cannot be changed.

PhyreStationDLL Commands

Most of PhyreStation's commands originate within the PhyreStationDLL and operate mainly on PhyreEngine™ objects. These commands can either be invoked from the GUI by selecting an object and then an action, or by issuing the command from within a script. A typical example is the `CloneObject()` command.

Clients may customize PhyreStation with new commands, thereby extending the PhyreStation command base. You must register custom commands in PhyreStation before you can use them. For more information, see *Chapter 8, Customizing PhyreStation*.

Viewing Available Commands

To see all available commands and their syntax, select **Help > Help Contents...** from the main menu, and then click on **Registered Commands**.

For more information, see *The Help Viewer* in *Chapter 1, PhyreStation Overview*.

Methods of Executing Commands

You can execute a PhyreStation command in the following ways:

- Enter a PhyreStationDLL registered script command in the **Command** window. See *The Command Window* section below.
- Select a command from the main window's menu bar, for example **File > Databases > New database**.
- Click a button in the main window's toolbar, for example the **New Workspace** button.
- Right-click on a workspace object (or the white space) in the **Workspace Explorer**, and then select the command from the context menu, for example **Delete**.
- Drag-and-drop an object within the **Workspace Explorer**, and then select the command from a pop-up menu, for example **Clone**.

- Run a script file by selecting a script object in the **Workspace Explorer – Scripts** view and choosing **Run** from the context menu. For more information about scripts, see *Chapter 7, Scripts*.

Some commands may be executed by only one of these methods; other commands can be executed in multiple ways. For example, `NewWorkspace()` is a script command, therefore you can run it from the **Command** window and also invoke it from the main menu and toolbar.

The Command Window

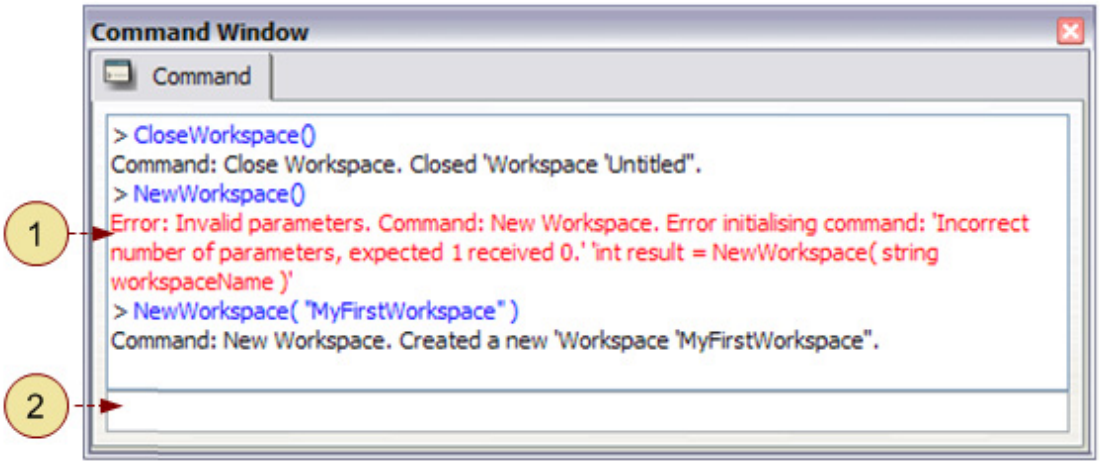
Overview

The **Command** window is used to enter and execute script commands. These commands have been specifically registered as scriptable in the PhyreStationDLL. For more information about forming scriptable commands see *Adding PhyreEngine™ Object Commands to PhyreStation* in *Chapter 8, Customizing PhyreStation*. For information about scripting in general, see *Chapter 7, Scripts*.

The **Command** window allows you to execute commands that are either unavailable to the PhyreStation GUI or blocked (disabled on the context menu in the GUI). Executing commands from the **Command** window is more versatile and less restrictive, but some operations are blocked for a reason and therefore you should be careful not to destabilize PhyreEngine™.

To open the **Command** window, select **View > Command** window from the main menu.

Figure 41 Command Window



The window contains two regions: a *history pane* (item 1) and an *input field* (item 2)

History Pane

When you enter a command in the input field, it is displayed in the history pane above. The history pane buffer contains all script commands entered from the start of the current PhyreStation session (including invalid or failed commands). Entered text is colored blue. All error and warning messages are colored red. You can select and copy text in the history pane.

Input Field

The input field is case-sensitive. You can manually enter Lua or script-registered commands in this field.

If the syntax is recognized and understood, the command is immediately executed when you press **Return** or **Enter** on the keyboard. Any entered text is colored blue. If a syntax error occurs, a description of the required syntax is displayed in the history pane.

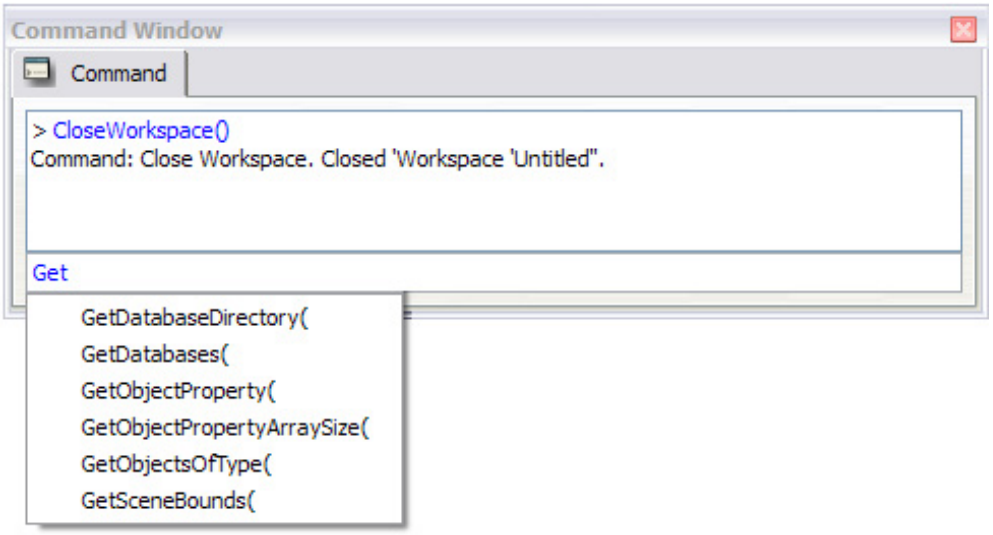
History Buffer

The input field has a history buffer. To display previous entries, press the **Up** key. To display all entries in chronological order, press the **Down** key.

Auto-completer

The input field has an auto-completer. If you enter text that matches one or more command names, the command names are displayed in a drop-down menu. You can select a command from the menu using the cursor keys.

Figure 42 Input Field Auto-completer



When the input field is empty, you can right-click on it to display a pop-up list of all script-registered commands. Select a command from this list to automatically copy the command into the input field. After editing the command, press the **Return** or **Enter** key as you normally would.

Hint:

- The command interpreter accepts both forward slash ('/') and pairs of reverse slash ('\') characters for delimiting file paths. For example, `C:/Docs/PhyreEngine` and `C:\\Docs\\PhyreEngine` are equivalent.
- You can drag script files into the input field from some OS file browsers (for example, Windows Explorer). PhyreStation will try to execute the script. Script files can also be executed from the input field using the Lua command `dofile`.

Note: For the **Tab** key to operate as expected in the context of the **Command** window described here (and not move keyboard focus to another window or widget), place the cursor in any part of the **Command** window.

PhyreEngine™ Object ID

Some commands require a PhyreEngine™ object ID as one of their parameters. A PhyreEngine™ object ID is a string with one of the following two formats:

- `database.extension#object`
- `database.extension`

The following rules apply to IDs:

- Object name must be defined, that is, `database.extension#` is invalid.
- When you are dealing with database objects, database names must be followed by a recognized PhyreEngine™ file extension, that is, `.hier` or `.pssg`.
- IDs are case-sensitive.

Log Information from Commands

When a command is executed, it can produce a status message, which PhyreStation directs to the **Log** window or a log file.

Typically, messages contain information about the workspace name, the PhyreEngine™ object name and the type of work carried out. Unless performance is an issue, it is recommended that all commands should report status information to allow a user to monitor the success of a command or script. For more information, see *The Log Window* in *Chapter 17, PhyreStation Log, User Preferences, and Error Handling*.

7 Scripts

This chapter describes PhyreStation's scripting mechanism. It explains how to run script commands and manage script files.

Overview

One of the primary objectives of PhyreStation is to execute scripts to automate the processing of your data. Scripts provide an alternative to entering individual commands in the **Command** window. You can specify a script file from an OS command-line prompt when you pass PhyreStation script parameters as part of the application's execution statement, or you can execute a script from the PhyreStation GUI.

PhyreStation has a built-in command interpreter (or *scripter*) for automating script commands. The command interpreter is built on the Lua scripting engine, and has been extended to deal with some extra data types and user-defined commands. For a general overview, refer to the relevant Lua documentation. See also http://www.Lua.org.

Script commands must be specifically registered as scriptable in the PhyreStationDLL so that PhyreStation and Lua can recognize and operate the commands. For more information about commands, see *Chapter 6, PhyreStation Commands*.

To see a list of commands registered for operations by PhyreStation, including those that are scriptable, select **Help > Help Contents...** from the main menu.

Script Files

A PhyreStation script file is an ASCII text file you can compose using a normal text editor. A script file may contain any valid combination of script commands and Lua instructions.

Associating Scripts with a Workspace

Script files exist independently of workspace files. However, you can associate a script file with a workspace by adding the script to the workspace. A reference to the script file is saved to the workspace file using a relative file path. One script file may be associated with multiple workspaces (workspace files). Script files associated with a workspace are represented in the workspace as script objects.

To associate a script file with a workspace, do one of the following:

- Select a script file using the OS file browser and drop it into the **Workspace Explorer – Scripts** view.
- Select **Add Script** from the **Workspace Explorer – Scripts** view's context menu.

To associate an empty script file with a workspace, select **Create Script** from the view's context menu. If you have not specified a default editor, PhyreStation prompts you to browse for the editor that you want to associate with script Lua text files. Your preferred editor then opens automatically and you can edit the empty script file.

Note: On Windows, the script file and workspace file must share the same drive.

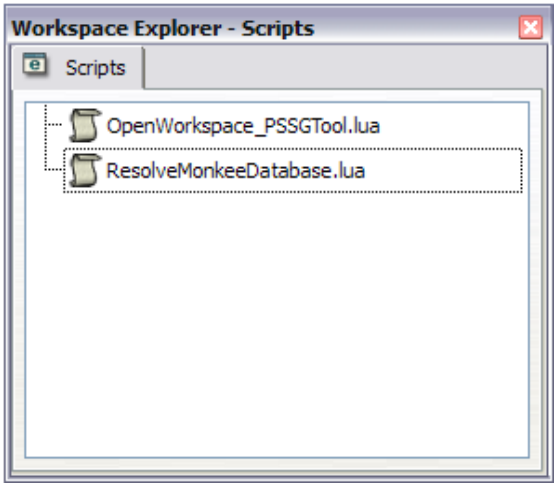
Changing the Script Editor

To change your preferred editor, select **Edit > User Preferences > PhyreStation** from the main menu and set the **Default script editor** preference.

Browsing Scripts

You can view script objects in the **Workspace Explorer – Scripts** view. If a workspace cannot locate a script file that is associated with it, the script object is grayed out.

Figure 43 Workspace Explorer – Scripts View



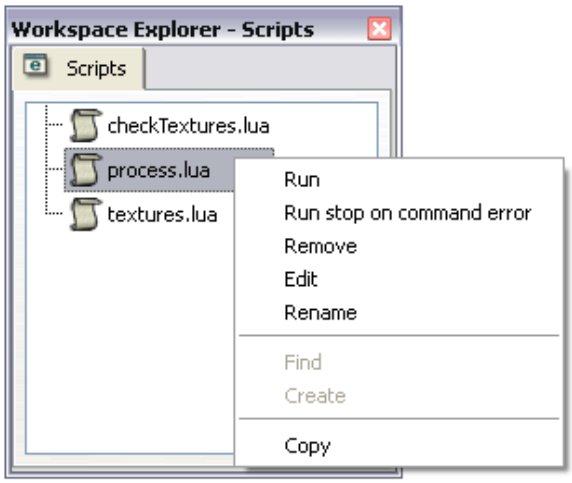
You can find some sample script files in the following directory:
`[PhyreStation_root_directory]/PhyreStationScripts`

GUI Script Object Operations

You can right-click a script object in the **Workspace Explorer – Scripts** view to display a context menu with the following options:

- **Run** – Execute the script until the end or a Lua error is encountered.
- **Run stop on command error** – Execute the script until either a Lua error or a command error is encountered, or the end of the script is reached.
- **Remove** – Remove the script object from the workspace (break the association).
- **Edit** – Send the script to your chosen editor for viewing or amending.
- **Rename** – Rename the script file.
- **Find** – Locate the missing script file and reassociate it with the workspace.
- **Create** – Recreate the missing script file.
- **Copy** – Copy the name of the script file onto the OS clipboard.

Figure 44 Workspace Explorer – Scripts View



Executing Scripts

PhyreStation can operate in one of three modes depending on how you start PhyreStation:

- GUI mode. Click on the PhyreStation icon or run from the OS command line with no parameters.
- A script GUI mode. Run from the OS command line with additional parameters, including the parameter `-scriptQuit="false"` specified. PhyreStation executes the script and then continues as a GUI application. You can specify a workspace from a script given on the command line.
- Batch mode. Run from the OS command line. PhyreStation executes a script and immediately quits.

From the GUI, you can load and run a script file by doing one of the following:

- Select **File > Run script from file...** from the main menu. This option is disabled if a workspace is open.
- Click the **Run script from file** button  on the main toolbar. This button is disabled if a workspace is open.
- Run a script from the **Command** window's input field using Lua's `dofile()` command.
- When running PhyreStation from the OS command line.
- Load and run a script from another script file, using Lua's `dofile()` command.
- Drag a script file from your OS file browser and drop it into the **Script Manager** dialog box or the **Command** window's input field.

If PhyreStation is in GUI mode and executing a script, PhyreStation disables the GUI and updates the GUI only when the execution of the script has finished. During the execution of a script, PhyreStation directs any command status messages to either the **Log** window or an optional script file.

Executing Scripts from the Command Line

Overview

You can execute PhyreStation from the OS command line, either with or without command-line parameters.

- If no parameters are specified as part of the command-line entry for the PhyreStation executable, the application starts in GUI mode in the same way as if launched from the desktop icon.
- You can set parameters to determine how PhyreStation starts. Some of these parameters are script-type parameters that enable you to execute scripts from the command line. All script-specific parameters are prefixed with the word "`script`". All parameters are case-sensitive. Some script parameters cause PhyreStation to start up in a minimized mode with no splash screen or any evidence of the application running on the OS task bar.

When PhyreStation is executed from the OS command line, the command prompt returns immediately.

Application Instance ID

The application instance ID identifies the instance of the PhyreStation application that is currently running. PhyreStation predetermines this ID when it starts up in GUI mode. You can find the instance ID by selecting **Help > About PhyreStation** from the main menu.

However, if you start PhyreStation from the command line, you can specify the ID using the `-appId` parameter. This parameter enables you to target a specific instance of the application to carry out a new task. For example:

```
-appId="OvernightJob05"
```

From the command line, you can have several instances of the application running at the same time and use just one OS command-line window to send different jobs to specific instances of PhyreStation.

Parameters

-help

To see all the possible command-line options available and their instructions, use the `-help` parameter option.

-script

To run a Lua script file automatically at PhyreStation start-up, add the `-script` option to the OS command line. For example:

```
-script="C:/user/idave/projects/setup.lua"
```

This loads and executes the Lua script file from the specified file directory.

- If the parameter `-workspace` is used in conjunction with this parameter, the `-script` parameter only needs to specify the name of a Lua script file that is contained within the specified workspace. The parameter `-workspace` takes the name and directory path of an existing PhyreStation workspace file. For example:

```
PhyreStation -workspace="C:/user/idave/projects/AWorkspace.xml"
-script="AScript.lua"
```

- If the script cannot be found in the workspace, PhyreStation attempts to load and run the script from a file with the same name.
- You can use environmental variables as part of the file path. For example:

```
PhyreStation -workspace="$(MY_ENV_VAR)/AWorkspace.xml"
-script="AScript.lua"

PhyreStation -workspace="%MY_ENV_VAR%/AWorkspace.xml"
-script="AScript.lua"
```

- The parameter `-workspace` can be used only with the `-script` parameter and cannot be targeted against an instance of the application that has already run.
- If the parameter `-workspace` is not specified but the preference **Load workspace on startup** is set, a default blank workspace is created before the script executes. If the `NewWorkspace` or `OpenWorkspace` command is executed in a script, it overrides the default workspace.

Note:

- When specifying either the workspace filename or script name, you must include a file extension (such as `.lua)`; otherwise, the parameter will be rejected.
- Some commands do not work unless there is a workspace session open in PhyreStation.

-scriptQuit

To instruct PhyreStation whether to shut down after running the script file or command, use the `-scriptQuit` parameter. For example:

```
-scriptQuit=true
```

By default, this option is set to "`false`", which causes PhyreStation to continue to run after loading and executing the script file.

If this option is set to "`true`", PhyreStation is operating in batch mode. The application runs in a minimized mode, which has the following effect:

- The application does not start up or update any GUI systems; therefore, the application runs faster.
- No application or command messages appear in the OS command-line window as a result of executing the application. Use the `-scriptLog` parameter to export messages to a text file.
- The application instance starts up but does not appear, and is not visible as running on any task bar.

- The `-scriptQuit` parameter will be ignored if it is targeted against an existing running instance of the application.

-workspace

See the `-script` parameter.

-scriptExecute

You can pass an additional Lua script on the command line using the `-scriptExecute` parameter in conjunction with the `-script` parameter. The additional script is executed first, and then the script specified by the `-script` parameter is executed.

This makes it possible to pass parameters from outside to an existing script. For example:

```
-scriptExecute="MyDbName=\"a.PSSG\"; MyDbDir=\"C:/user/idave/databases\""
-script="C:/user/idave/projects/readfiles.lua"
```

If a Lua script is read from file and it contains and uses these variables, the values entered on the command line are used within the execution of that script.

Lua or PhyreStationDLL registered script commands can also be executed. For example:

```
-scriptExecute="dofile(\"C:/user/idave/projects/setup.lua\")"
-scriptExecute="AddDatabase(\"C:user/idave\", \"monkee.pssg\")"
```

The `-script` parameter does not have to be used with the `-scriptExecute` parameter.

Note: Some commands will not work unless there is a workspace session open in PhyreStation.

-scriptLog

To instruct the application to output all messages to a specified file, use the `-scriptLog` parameter. The messages you normally see in the **Log** window appear in the file instead. The file produced is a text file. For example:

```
-scriptLog="BatchJob01Results.txt"
```

If the file exists already, it is over-written with a new file that has the same name.

This parameter is ignored unless the `-scriptQuit` parameter is also specified.

Note: The `scriptLog` only reports errors. It does not mirror the contents of the **Log** window as this allows the script to execute at its fastest. It is recommended that you first run a script from the application to check that it runs successfully.

-scriptStopCmdError

If this parameter is used as follows:

```
-scriptStopCmdError=true
```

the execution of the script will stop on the first command that returns a failure, that is, a return result that does not equal "completed task ok". The default value for this parameter is false.

Note: The scripting system stops when a Lua error is encountered, regardless of this parameter.

PhyreStation Execution Return Results

PhyreStation returns an integer number after the execution of the application is complete. *Table 4* lists the return results.

Table 4 PhyreStation Execution Results

Status Code	Description
0	Application successfully completed task.
1	PhyreStation does not support the OS window system.

Status Code	Description
2	PhyreStation failed during initialization.
3	PhyreStation encountered a terminal code failure.
-1	Command-line script error occurred.
-2	PhyreStation internal usage.

Extra Lua Data Types

PhyreStation's implementation of Lua has been extended to deal with some extra data types that are important for PhyreEngine™. New commands have been added to create and operate on these new types.

Vector

A `vector` stores a vector of four numbers. Vectors can be created with the function `vector(a,b,c,d)`, which returns a new vector.

`a`, `b`, `c`, and `d` are optional parameters that set initial values of the vector's components. If a component is not specified, the value 0 is assumed. For example:

```
v1 = vector()
v2 = vector( 0, 1, 0, 3 )
```

The components of a vector can be accessed either through an array index, or by name. `v[0]` is equivalent to `v.x`, `v[1]` is equivalent to `v.y`, `v[2]` is equivalent to `v.z`, and `v[3]` is equivalent to `v.w`. For example:

```
v3[0] = v1.y // equivalent to v3[0] = v1[1]
```

Vectors are automatically converted to a string when attempting to perform a string operation such as printing or concatenation. For example:

```
v3 = vector( 1, 2, 3, 4 )
print( v1 )
```

Vector Operations

For the following, assume `v1` and `v2` are vectors, `m1` is a matrix, and `n1` is a number.

Table 5 Vector Operations

Vector Addition	For example: `r = v1 + v2`	Returns a new vector that is the sum of `v1` and `v2`.
Vector Subtraction	For example: `r = v1 - v2`	Returns a new vector that is the result of subtracting `v2` from `v1`.
Vector – Matrix Product	For example: `r = v1 * m1`	Returns a new vector that is the result of applying `v1` to `m1`.
Vector – Vector Product	For example: `r = v1 * v2`	Returns a new vector whose fields are the products of the respective fields in `v1` and `v2`.

Vector – Number Product	For example:
	`r = v1 * n1`
	is equivalent to:
	`r = n1 * v1`
		Returns a new vector that is equal to `v1` scaled by the value of `n1`.

Vector – Vector Cross-product	For example: `r = v1 ^ v2`	Returns a new vector that is the cross-product of `v1` and `v2`, treating both as `vector-3`s. The fourth component of the result is set to 0.

Vector Normalization	For example: `r = v1:normalise()`	Returns a new vector that is a normalized form of `v1`.
Vector Length	For example: `r = v1:length()`	Returns a number that is the length of `v1`.

Matrix

A matrix stores a 4x4 matrix of numbers.

Matrices can be created with the function `matrix()`, which returns a new matrix. The new matrix is set to the identity matrix. For example:

```
m1 = matrix()
```

The components of a matrix can be accessed through array indices. `m[0]` is the first element in the first row of the matrix, `m[1]` is the second element in the first row of the matrix, `m[4]` is the first element in the second row of the matrix, and so on. For example:

```
m1[0] = 2.0
m2[5] = m2[5] * 3.0
```

Matrices are automatically converted to a string when attempting to perform a string operation such as printing or concatenation. For example:

```
m1 = matrix()
print( m1 )
```

Quaternion

A quaternion stores four numbers for use as an orientation.

Quaternions can be created with the function `quaternion(a,b,c,d)`, which returns a new quaternion. `a`, `b`, `c`, and `d` are optional parameters that set initial values of the quaternion's components. If a component is not specified, the value 0 is assumed. For example:

```
q1 = quaternion()
q2 = vector( 3, 2, 1, 1 )
```

The components of a quaternion can be accessed either through an array index, or by name. `q[0]` is equivalent to `q.x`, `q[1]` is the same as `q.y`, `q[2]` is the same as `q.z`, and `q[3]` is the same as `q.w`. For example:

```
q3[0] = q1.z // equivalent to q3[0] = q1[2]
q2.z = q2[1] * q2[0]
```

Quaternions will automatically be converted to a string when attempting to perform a string operation such as printing or concatenation. For example:

```
q1 = quaternion( 0, 0, 0, 1 )
print( q1 )
```

8 Customizing PhyreStation

This chapter gives detailed information about customizing the PhyreStationDLL.

Overview

You can extend PhyreStation to accommodate new types of PhyreEngine™ objects. To enable PhyreStation to manipulate these custom object types, you can also extend the PhyreStation's internal command set by adding new commands to PhyreStationDLL. PhyreEngine™ object commands are commands solely associated (or registered) with PhyreEngine™ objects.

PhyreStationDLL source code has two parts:

Core Functionality

The core functionality part of the source code should not be modified. This part of PhyreStationDLL contains foundation classes and interfaces that PhyreStation needs to operate properly, such as visitor classes and operation interfaces. It also contains base classes to extend PhyreStation's command system.

> **Note:** Programmers must be careful not to alter those PhyreStationDLL interface and visitor classes that PhyreStation already relies on.

Extension Utilities

The PhyreStationDLL extension utilities comprise new code that use PhyreStationDLL's core functionality to extend the system. These libraries are linked to the PhyreStationDLL build project.

Programmers should use the extension utilities architecture to manage separate areas of code dedicated to specific tasks, whilst maintaining the integrity of PhyreStationDLL's core functionality.

To customize the PhyreStationDLL, follow these steps:

(1) Create a new PhyreEngine™ utility.
(2) Create a new PhyreEngine™ object type.
(3) Register the new PhyreEngine™ object type.
(4) Compile the PhyreStationDLL, which in turn compiles and links your PhyreEngine™ Utility.
(5) Create a new PhyreEngine™ object command.
(6) Register the new PhyreEngine™ object command.
(7) Optional: make the new command available in PhyreStation, for example on context menus. Create a new binding class, if required.
(8) Compile the PhyreStation DLL again.
(9) Run the new command from PhyreStation.

Creating a New PhyreEngine™ Utility

To keep new classes separate from PhyreEngine™ Core, it is recommended that you implement a new PhyreEngine™ utility class for creating a new PhyreEngine™ object.

To create a new PhyreEngine™ utility, see the *PhyreEngine™ Programming Guide,* "PhyreEngine™ Utilities" chapter. For more information about creating a PhyreEngine™ utility, see the `readme.txt` file in the PhyreEngine™ Utilities directory.

Adding New PhyreEngine™ Object Types to PhyreStation

This section explains how to extend PhyreStation by creating and registering new PhyreEngine™ object types.

Overview

A powerful feature of PhyreEngine™ is that you can adapt it to suit your requirements by implementing your own types of objects, each with a unique set of attributes. When you have created your new object types and registered them with PhyreEngine™, PhyreStation automatically recognises them and can display them (when present in a database). The attributes you have assigned to your new objects are visible in the object **Properties Viewer** for your inspection.

PhyreEngine™ provides an extensive range of default object types. To see the set of currently registered object types, run PhyreStation, select **Help > About PhyreStation**, and then select the PhyreStationDLL component in the left hand group box. The right hand group box displays a list of all the commands and object types currently registered.

Supplied Object Types

It is recommended that you derive any new objects from the supplied PhyreEngine™ object type classes, thereby not changing the supplied types. Otherwise, any changes that you make (including adding your own object type classes to the original files) will be lost when you install the next release of PhyreEngine™.

Your Custom Object Types

It is recommended that you add any new object type class code using the PhyreEngine™ Utilities or PhyreStationDLLUser files. This keeps your code separate from the shipped PhyreEngine™ and main PhyreStationDLL code.

To create your own commands to operate on your new objects, see *Adding PhyreEngine™ Object Commands to PhyreStation* in *Chapter 8, Customizing PhyreStation*.

Creating a New PhyreEngine™ Object Type

Any new object you create must be derived from PhyreEngine™'s base object type class. PhyreStation can display all the types of object that are currently registered with PhyreEngine™ as follows: open a **Workspace Explorer – Objects** view and select **Show Empty Categories** from the view's context menu. The view displays all the types currently available.

For PhyreStation to display an object that you have created, either the object must be present in a PhyreEngine™ database that is loaded into the workspace, or you must run a command that creates a new object of your type and adds it to a database in the workspace. When the object is present in the workspace, visible under your object category, you can select the object, view its attributes in the object **Properties Viewer**, make changes to it, or save the database.

To help you create a new PhyreEngine™ object:

- See the *PhyreEngine™ Programming Guide* chapters entitled "PhyreEngine™ Objects" and "Adding New PhyreEngine™ Object Types".
- Refer to the PhyreEngine™ code example '`newObject`' in `[PhyreStation_root_directory]/Samples/Intermediate/newObject`. Use the code from this example in your own PhyreEngine™ utility.
- See the PhyreEngine™ code `PSSG::PSSGInit()`.

Compiling, Linking, and Debugging

(1) When the code for creating the new PhyreEngine™ object is complete, compile the PhyreStationDLL, which in turn compiles and links your PhyreEngine™ Utility.

(2) Optional: When the code for executing and registering the new command is complete, compile PhyreStationDLL again. The command is then ready for use from PhyreStation.

(3) Copy the new PhyreStationDLL file to the PhyreStation directory.

> **Note:** For Microsoft Visual Studio, the PhyreStationDLL will be a Release type build. In order to debug your PhyreStationDLL library, you must turn on debugging and turn off all compiler optimizations via the PhyreStationDLL project setting.

Object Icon

An optional but recommended part of creating a new object type is to create an icon for the object type. The icon is a 16x16 PNG image. To assign an icon, create it and place it in the directory `[PhyreStation_root_directory]/Resources/Icons`. When a PhyreEngine™ accessor function for your type is called, referring to the icon by its file name, the PhyreStation views display the icon along side your new object type.

Adding PhyreEngine™ Object Commands to PhyreStation

This section explains how to extend PhyreStation's set of commands to better fit your requirements.

Overview

The PhyreStationDLL, the DLL for which you have the source code, can be extended to incorporate your commands or change existing commands.

A command is a function wrapped in a PhyreStation command pattern API. The function can perform any processing you require. For example, you can create a command to open a file and parse it, or to operate on a PhyreEngine™ database or object.

A command can be executed from the PhyreStation GUI (normally bound to a type of PhyreEngine™ object and executed from a context menu) or from within a Lua script. Note that not all commands that are registered to operate from within a script must be registered with PhyreStation's GUI or vice versa.

The PhyreStationDLL source code has three parts:

Core Functionality

The core functionality part of the source code should not be modified. This part of PhyreStationDLL contains foundation classes and interfaces that PhyreStation needs to operate properly, such as visitor classes and operation interfaces.

> **Note:** Programmers must be careful not to alter those PhyreStationDLL interface and visitor classes that PhyreStation relies on.

Supplied Commands

Most of the commands that PhyreStation is shipped with can also be found in the PhyreStationDLL. See PhyreStation's **About PhyreStation** dialog to see the supplied commands. Some of the commands found in the DLL are PhyreStation's functional commands (named commands), which should not be altered.

The commands listed in the **About PhyreStation** dialog can be changed if you wish, but be careful. The supplied commands can be altered but they will be overwritten in the next release unless you record any changes you have made. Similarly, you can put your own command here alongside the existing command, but this is not recommended.

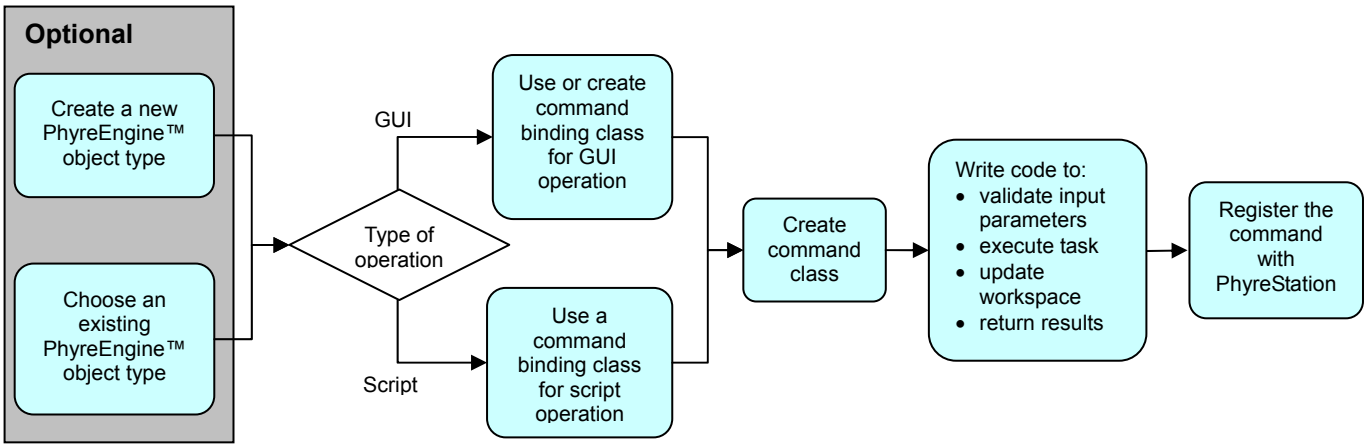
Extension Utilities

The best way to extend PhyreStation is to put your code into the PhyreStationDLL user files. An alternative method, depending on your requirements, is to use PhyreEngine™'s Utilities. These libraries are linked to the PhyreStationDLL build project.

It is recommended that you use the extension utilities architecture to manage separate areas of code dedicated to specific tasks, while maintaining the integrity of PhyreStationDLL's core functionality.

If you wish to create your own objects, as well as new commands to work with them, see *Adding New PhyreEngine™ Object Types to PhyreStation* earlier in this chapter and create your objects first.

Figure 45 Creating a New Command



Command Availability Control – The Command Binding Class

This section is valid only if you wish to operate a command from the PhyreStation's GUI, for example from a context menu or as part of a mouse drag drop action. (A binding is also used to register a command for use within a script, but that type of binding is not used to control availability.) A GUI command binding controls the following:

- Dictates if a command's name should appear in a context menu.
- Dictates which (if any) type of PhyreEngine™ object the command should be associated with.
- If the command appears in a context menu, dictates if the command is enabled in that context menu.

As part of the GUI interaction with the user, the GUI issues queries to all the registered commands' binding objects so that it can present options or actions to the user. The information above represents the type of answers that a binding must supply.

When the conditions or rules that you have specified are satisfied, the GUI instructs the binding class to create an instance of the command class. This command class will carry out the required task on a selected workspace object when PhyreStation calls the command class's `Execute()` function.

Finally, when the binding object has attempted to create a command object, it informs PhyreStation of the status of its success by setting its success status member variable `m_status` and status message variable `m_statusMsg`.

The PhyreStationDLL provides a context menu base class `PDllContextCmdBindingBase` that you can use or derive from, to suit your purposes. Similarly, for any drag drop operation, you can use or extend from `PDllDragCmdBindingBase` class. The base binding class interface specifies the following set of functions, which must be implemented in your binding class or your class's parent binding class:

- `IsCmdValid()`
- `IsCmdEnabled()`
- `CreateCmd()`
- `IsCmdValidForSrc()` (Drag-drop binding only)

Note that the PhyreStationDLL command binding classes are limited in how they can ask the user any questions to either satisfy the conditions required for each rule or obtain more information so that it can be passed on to the command during its creation. A basic Yes/No dialog interface is provided to use PhyreStation's message dialog system using PhyreStationDLL interface `PhyreStationDLLInterfaceAskUser`.

Template classes are also derived off these command binding base classes. The template classes are based on the type of command that the binding creates when satisfied.

Examples of PhyreStationDLL command binding classes are:

- `PDllDragCmdBindingLinksAndInternalDB`

  Defines drag commands, unless the item has links to other objects or is an internal database.

- `PDllContextCmdBindingLinksResolved`

  Defines context menu commands for only when the database is resolved successfully.

- `PDllContextCmdBindingDisableForPhyreEngineInternalDatabase`

  Defines context menu commands for any object that does not belong to the PhyreEngine™ internal database.

- `PDllContextNodeCmdBinding`

  Defines context menu commands for objects derived from `PNode`. The set of `PNode` objects, for which the command is valid, is passed to the binding class constructor.

An instance of a binding class is passed to PhyreStation as part of the command's registration process. It is recommended that any instances of the binding classes are static and referenced at some point in the DLL as the compiler may dead strip the binding class, which can cause command registration problems.

The Command Class

All the commands that communicate with PhyreStation are derived from the base command class `PDLLCmd`, found in the PhyreStationDLL. This also applies to any command you create, whether for use with the GUI or from within a script. The PhyreStationDLL contains various base classes derived off the `PDLLCmd` base class. These derived classes are more specialized in the type of PhyreEngine™ object they operate on or the type of task they carry out, for example `SaveDatabaseCompressed()`.

Some template type classes are included with the various bases classes. Again, you can use these or create your own.

A further classification of the command class is whether the command handles only single objects or can handle multiple objects (that is, when the user has selected many objects in the GUI). Commands that handle multiple objects are not suitable for use as script commands.

Note: The `PDLLCmd` class is derived from `BBaseCmd` class. Do not change the `BBaseCmd` as it is fundamental to the operation of PhyreStation.

The Command ID

The class member variable `m_cmdId` holds a unique text string to identify the command from any other command ID. It is used by PhyreStation to keep track of individual commands. It is not normally seen by the user.

The Command Name

The class member variable `m_cmdName` holds a text string for the command's name normally taken from the name of the type of object the work is to be done on, for example 'Database'. This text is seen by the user. It is the command noun.

The Command Context

The class member variable `m_cmdContext` holds a text string for the command's action. It normally describes what will be done, for example 'Save Compressed'. This text is seen by the user. It is the command verb. PhyreStation concatenates the command name to the command context to form a description for the user, for example 'Save Compressed Database'. The description appears in PhyreStation Log reports; it can also appear in the GUI as an item in a context menu, therefore the description must be short.

The Command Description

All commands can store information about themselves. This extra information can be displayed by PhyreStation to help the user to understand and use the command. The information is split into the following categories:

- Syntax – text describing the format of the command.
- Syntax description – text information about input and output parameter types.
- Description – text explaining to the user what the command does and the context in which it would be used.

The information is stored as metadata. PhyreStation accesses this information through the command API function `GetMetaData(...)`.

PhyreStation uses command information in the following ways:

- To form online help pages in the PhyreStation **Help Viewer Registered Commands** page. See *The Help Viewer* in *Chapter 1,* PhyreStation Overview.
- To form tool-tips.
- To provide command syntax information.

The Commands' Arguments

A command often needs extra data or information at run time to be able to carry out its task. It may also need to return information to PhyreStation or the script when it has completed its task.

The command's input data is sent to it via the command's parameters (specified between its brackets). The command's output data returns by equating to the left value if present. A PhyreStation command can return more than one data value. The command uses a variant value FIFO stack for passing both input and return data.

When a command is used on a specific object, the first input parameter is always a PhyreEngine™ object ID or link name (*database*#*name*). This can be easily seen when using a command from within a script. Other values vary depending on the command. The command's parameters can be validated in either the command class constructor or the `Execute()` function. If a parameter is found to be invalid, return the command status code `PE_CMDRESULT_CMDOKBUTBADPARAMSIN` via the `Execute()` function before performing any part of the task.

The first value on the command's return stack must be the command's PhyreEngine™ error code. The `PDLLCmd::m_result` is assigned an error code in the command's `Execute()` function and then automatically put on the return results stack when the `PDLLCmd::GetReturnsStack()` is called by PhyreStation. PhyreStation may use this result code to assess the success of the command just executed. If the command is used in a script, the result code is returned to the script . Other subsequent return values vary depending on the command.

`BRedoUndoParamStackBase` is the variant type stack class used to pass parameters. To add data (command parameter data) to the stack, use its `AddParameter()` function. The function `GetParameters()` is used by PhyreStation to pop the values you have put on to it when preparing the returns stack during the command's execution.

The Command's Execute () Function

A command carries out its task in the scope of the Execute() function. When PhyreStation calls this function, it should have all the data or information it requires to carry out the task. If the command is to be executed in PhyreStation's GUI, the command should not prompt the user for more information. All necessary information should be requested by the command's binding class.

Command Status Messages

Although it is not mandatory to form a command status message, it is recommended that you do so to provide feedback to the user on the success or failure of the command's task. For example, if the task

succeeds, the user might want to know what object the command used or where the command has put a new object. If the task fails, the command status message can provide advice. Any messages returned are also sent to the PhyreStation log file.

A command can only send back one message, using either of the command's functions `SetLogReport()` or `SetCmdErrorMsg()`.

> **Note:** Any text messages that report the success or failure of the command's task are formed using the UTF8 format. See *Localization Support for Commands* in *Chapter 17, PhyreStation Log, User Preferences, and Error Handling*.

The Command's Success Status Code

The command passes back a PhyreEngine™ error code (via the command's return stack) to PhyreStation. This error code is passed on to either the script or the GUI.

In addition, a command returns a command status code. The command status code is used by PhyreStation to update its internal state, carry out appropriate actions, and create reports for the user. A command status code must be returned when leaving the scope of the command's `Execute()` function.

Table 6 lists the possible command status return codes.

Table 6 Command Status Return Codes

Status Code	Description	Feedback function to use
`PE_CMDRESULT_COMPLETEDTASKOK`	Command successfully completed its task.	`SetLogReport()`
`PE_CMDRESULT_USERCANCEL`	Command was canceled or interrupted by user.	`SetLogReport()`
`PE_CMDRESULT_CMDOKBUTTASKFAILED`	Some external factor caused the task to fail, for example attempting to write to a read-only file.	`SetCmdErrorMsg()`
`PE_CMDRESULT_CMDOKBUTBADPARAMSIN`	One or more command parameters are either missing, or the wrong type, or otherwise invalid.	`SetCmdErrorMsg()`
`PE_CMDRESULT_CMDFAILED`	Command aborted; caused by failure in its internal integrity checking code.	`SetCmdErrorMsg()`
`PE_CMDRESULT_UNDOFNNOTIMPLEMENTED`	Return from a command's `UnExecute()` member function, if `IsUnExecuteable()` returns `true`.	`SetCmdErrorMsg()`

The return code `PE_CMDRESULT_CMDFAILED` is also used by PhyreStation if a command fails catastrophically (usually when an exception is thrown and caught).

The Command's UnExecute() Function

The command class provides an `UnExecute()` function. This function provides the opposite functionality to that of the command's `Execute()` function.

In this release, PhyreStation does not actively support undoing a command's `Execute()` function. Instead, you should implement the command's `IsUnExecuteable()` and return `false`. The `UnExecute()` function is not executed.

Updating the Workspace

Except when PhyreStation executes a script in batch mode (runs, executes a script, and quits immediately), a command must update the workspace by informing it of any objects that the command has created, changed, or deleted. In the scope of the command's `Execute()` function, a command informs

PhyreStation of the objects it has dealt with. When the command has finished executing, PhyreStation updates the workspace based on the list of objects that have been submitted to it. If commands are executed in a script, the workspace is updated at the end of the script's execution.

In order for a command to inform the workspace of any pending changes, it must call one or more of the following command's base class functions:

- `void PDLLCmd::addObject( ... )`
- `void PDLLCmd::refreshObject( ... )`
- `void PDDLCmd::deleteObject( ... )`

The functions can be called any number of times and can be called repeatedly on the same object. If a second function is called after a previously updated object, the new function overrides the previous function. However, calling the refresh function after a delete function has no effect.

When a command has finished making changes, it must finally call the command's base class function `commandCompleted(...)`. This function flushes the pending changes to PhyreStation; PhyreStation then updates the workspace. If you derived your command class from an existing command class in the PhyreStationDLL, this function may already be called.

Note: PhyreStation updates the workspace only for the objects you specify using one of the three functions mentioned above. Objects that are changed due to changes on the specified objects are not automatically updated; therefore, you must manually specify them too. This strategy is used to optimize the performance of refreshing the contents of the workspace.

Command GUI Icon

If the command is to be used in PhyreStation's GUI, it is recommended that the command has an icon to represent it. An icon is a 16x16 pixel image PNG type (.png) image. The icon is specified as part of the command's registration process in the PhyreStationDLL.

Icons for commands reside in PhyreStation's Resource directory Icons. To use a new icon, you must register it with PhyreStation in the PhyreStationDLL().

Command Registration

During its initialization or start up process, PhyreStation must be informed of any additional commands it can use. The type of registration determines where the command can be used. The types of registration are:

- Context – GUI context menus
- Script – Lua script or **Command** window execution
- Drag – GUI object selection drag-and-drop operations
- Named – For PhyreStation's internal use only

For each type of registration, you use the appropriate command binding class (either an existing binding or one that you created).

The majority of commands are registered in the PhyreStationDLL function `PPSSGDll::registerDllCommands()`. It is recommended you use the same approach as shown in function `PPSSGDll::registerDllCommands()`, but place your code in the `PPSSGDll::registerDllUser()` function. This maintains the separation between your code and the code provided.

If you have created a command associated with a particular object type, but no objects of that type exist yet in the workspace, you can check if registration was successful as follows: open PhyreStation's **About PhyreStation** dialog and select the PhyreStationDLL component in the left hand group box. The right hand group box displays a list of all the commands and object types currently registered.

Reference Material for PhyreEngine™ Object Commands

This section refers you to extra information that you may find useful when creating your own commands for PhyreStation.

Creating Commands

See the PhyreStationDLL source code; for example, `PhyreStationDLLObjectCmds.h`, `PObjectGetAttributeCmd` class.

Compiling, Linking, and Debugging

(1) Optional: When the code for creating the new PhyreEngine™ object is complete, compile the PhyreStationDLL, which in turn compiles and links your PhyreEngine™ Utility.

(2) When the code for executing and registering the new command is complete, compile PhyreStationDLL again. The command is then ready for use from PhyreStation.

(3) Copy the new PhyreStationDLL file to the PhyreStation directory.

Note: For Microsoft Visual Studio, the PhyreStationDLL will be Release type build. In order to debug your PhyreStationDLL library, you need to turn on debugging and turn off all compiler optimizations via the PhyreStationDLL project setting.

9 Custom Groups

This chapter describes how to use custom groups in PhyreStation.

Overview

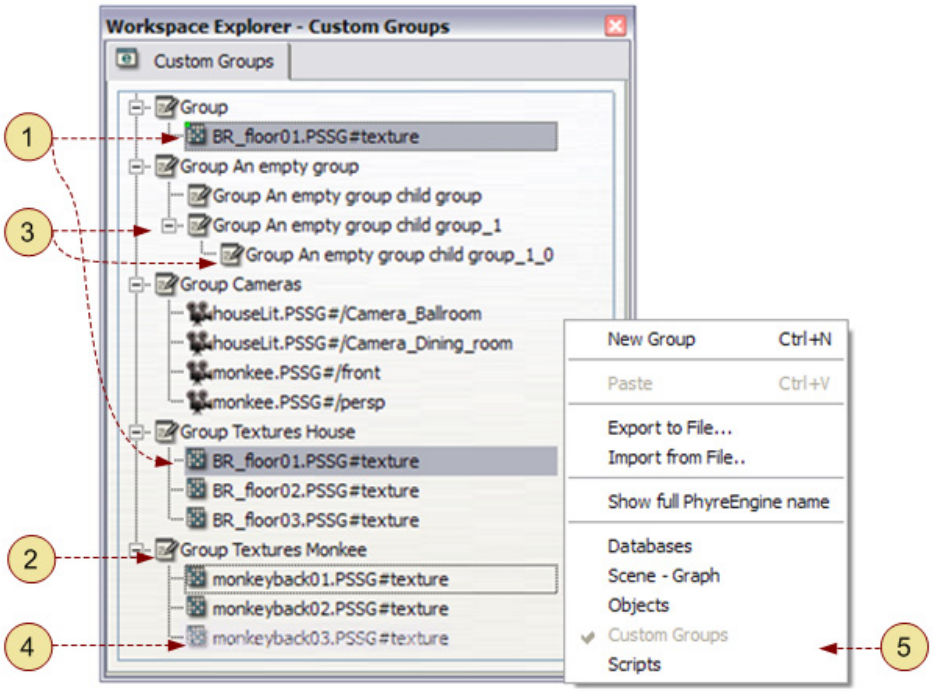
Custom groups are user definable groups of workspace objects. You can assign different types of objects (databases, PhyreEngine™ objects, and scripts) to a group and give the group a name. This allows you to work on a specific subset of objects in the workspace without repeatedly navigating the workspace to find those objects. By grouping the objects that you are interested in, it is easier to manage those objects.

- A workspace can contain an unlimited number of groups.
- A group can be a child of another group.
- A workspace object can be represented in multiple groups.
- When you assign a workspace object to a group, the group contains a *representation* of the object, and not the object itself. The representation of the workspace object in the group looks the same as the real object. You can carry out work on the representation (by using commands) as if the object were in its own **Workspace Explorer** view.
- An object in a group can be copied, moved, or deleted within the **Workspace Explorer – Custom Group** view, leaving the 'real' object unchanged.
- Groups can be exported from the workspace and imported into another workspace.
- Groups are part of the current workspace and can be saved in the workspace by setting the appropriate preference.
- Custom group preferences are available in the **User Preferences** dialog.

Browsing Custom Groups

To display the custom groups in the current workspace, open a **Workspace Explorer** window and select **Custom Groups** from the view's context menu. The **Workspace Explorer – Custom Groups** view opens.

Figure 46 Workspace Explorer – Custom Groups View



(1) Representation objects are highlighted when user selects the real object or duplicate representation objects. The **green dot** points to the originally selected object.
(2) User-defined group.
(3) User-defined child groups.
(4) A ghosted representation object (the real object does not exist in the current workspace).
(5) View context menu.

Multiple objects can be selected, and objects chosen in a **Workspace Explorer – Custom Groups** view can be part of a larger object selection set made up from multiple views (item 1).

The **Workspace Explorer – Custom Groups** view can display basic object properties alongside the object. To display these properties, select **Edit > User Preferences > Workspace** and set the **Enable workspace explorer filter properties** preference. For more information, see *Workspace Explorer Filter Properties* in *Chapter 3, Workspaces*.

Identifying PhyreEngine™ Objects in Groups

Most workspace objects that are assigned to custom groups are PhyreEngine™ objects. A PhyreEngine™ object may be represented in more than one group. The representation object uses the same name as the real object, in this case the PhyreEngine™ object name. This can make it difficult to identify the object and the PhyreEngine™ database it belongs to, especially if there are duplicate real objects with the same name in different PhyreEngine™ databases.

The **Workspace Explorer – Custom Groups** view can display the full PhyreEngine™ object link ID for a PhyreEngine™ object, in the format *databaseName*#*objectName*. To enable this option, select **Show full PhyreEngine™ name** from the **Workspace Explorer – Custom Groups** view context menu.

Custom Group Operations

You can perform all custom group operations by selecting either from the **Workspace Explorer – Custom Groups** view context menu, or from the context menu of an object in a group, depending on the command you wish to execute.

Creating a Group

To create a custom group, do one of the following:

- Right click in the **Workspace Explorer – Custom Groups** view and select **New Group** from the context menu.
- Use keyboard entry **Ctrl-N**. The **Workspace Explorer – Custom Groups** view must have focus.
- Drag an object from another **Workspace Explorer** view into the **Workspace Explorer – Custom Groups** view.

When you perform any of the above actions, a new group appears with a default name. You can rename the group by either clicking on its label or choosing **Rename Group** from the group's context menu.

Saving a Group to the Workspace

By default, custom groups are not saved in the workspace. To save your custom groups and restore them when you next open the workspace, select **Edit > User Preferences > Workspace** from the main menu and select the **Remember and restore the custom groups** preference.

Adding an Object to a Group

To add an object to a custom group, do one of the following:

- Drag an object from another **Workspace Explorer** view into the group.
- Cut (or copy) and paste a representation object into the group from another group.

If the group already contains an object with the same name as the one you are trying to add, the operation terminates.

Adding a Group to a Group

To add a group as a child of another group, do one of the following:

- Right-click a group and select **Add Group** from the context menu.
- Drag the child group onto the parent group.
- Cut or copy a group, choose a parent group and paste the child onto it.

Detaching a Child Group from a Group

To detach a child group from a parent group, select **Detach Branch** from the context menu of the child group. This moves the child group (with all its objects) to become a top-level group. The names of the groups and any objects are not changed. This is useful if there is no white space left in the view to drag and drop a group.

Copying and Moving Objects between Groups

An object in the **Workspace Explorer – Custom Groups** view can be duplicated in multiple groups, but the duplicated representations still represent only one object in the workspace. You cannot have two representations of the same object in one group.

To create another representation of the same object, do one of the following:

- Drag the representation object to another group and choose **Copy** from the pop-up menu.
- Drag the same object from another **Workspace Explorer** view and drop it onto the target group.

You can move representation objects among groups by dragging a representation object from one group to another.

Renaming a Group

To rename a group, do one of the following:

- Click on its name and edit the name.
- Select **Rename Group** from the group's context menu.

The **Rename Branch** option in the context menu is enabled when a group contains at least one child group. This option allows you to rename the selected group and its child groups.

Deleting a Group

To delete a group, do one of the following:

- Select the group and press the **Delete** key.
- Right-click the group and select **Cut** from the context menu.
- Uncheck the **Remember and restore custom groups** preference setting and close the workspace. No custom groups will be saved to the workspace.

Deleting a group also deletes its child items. The real objects are not deleted from the workspace. Duplicate representations of the real objects in other groups are not deleted.

Deleting an Object from a Group

To delete an object from a group, do one of the following:

- Select an object and press the **Delete** key.
- Right-click an object and select **Cut** from its context menu.

When you cut an object from a group, the real object is not deleted from the workspace. However, if you select **Delete** from the context menu, the real object is deleted.

Deleting an Object from the Workspace

When you delete an object from the workspace, and there is a representation of that object in a custom group, one of two things can happen:

- All representation objects are turned into *ghosts*, that is, they are grayed out. If the real object is reintroduced back into the current workspace, each ghost representation reverts back to a 'solid' representation. When a representation is a ghost, commands cannot be carried out on it.
- All representation objects are removed from the **Workspace Explorer – Custom Groups** view.

To define this behaviour, select **Edit > User Preferences > Workspace** from the main menu and set the **Create ghost item(s) when a workspace object is deleted** preference.

Importing and Exporting Groups

You can export all the custom groups in the current workspace to a file so that they may be imported into another workspace session later. To export the groups, select **Export to File...** from the **Workspace Explorer – Custom Groups** view context menu.

To import a group file, select **Import from File...** from the view context menu. Importing a **Custom Groups** file replaces all the groups in the workspace with the groups in the file.

10 DNet Communication

This chapter describes PhyreStation's DNet communication mechanism. It describes how to connect to a remote PhyreEngine™-derived application over a network connection to interrogate, monitor, or deliver PhyreEngine™ data to the remote application.

Overview

Using DNet communication, PhyreStation can:

- Connect to a PhyreEngine™-derived application (such as the PhyreEngine™ example program COLLADA Viewer) and pass PhyreEngine™ camera scene information to be viewed in the application.
- Connect remotely to a PhyreEngine™ active session on a Sony PLAYSTATION®3 to import PhyreEngine™ databases into a PhyreStation workspace as if loaded from a file.

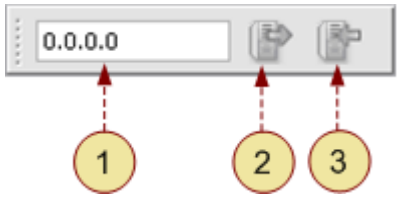
This chapter assumes that the PhyreEngine™ COLLADA Viewer is the application used to enable DNet communication. If compiled to do so, the PhyreEngine™ COLLADA Viewer can operate on either a Microsoft Windows platform or the Sony PLAYSTATION®3 Reference Tool.

DNet Communication Toolbar

The **DNet Communication** toolbar is used to make a connection to a PhyreEngine™-derived application over a network connection. The toolbar contains a text box for the host IP address, a **Connect** button and a **Load Remote Database** button.

To show the **DNet Communication** toolbar, select **View > Toolbars > DNet** from the main menu.

Figure 47 DNet Communication Toolbar



1. Edit box to specify:
   - a host name i.e. "localhost"
   - an IP address
2. Connect button to establish a remote connection or break a connection
   - Establish a connection button icon
   - Break connect icon
3. Load remote databases button

Connecting Using DNet

To make a connection using DNet communication:

(1) When you enter a host name or a valid IP address, the connect button becomes enabled.

(2) Click the connect button. When the connection has been established, the connect button changes to a disconnect button. This may take a few seconds while PhyreStation establishes a connection.

(3) When you have finished with the remote connection, click the connect button again.

Note: A PhyreEngine™ session must have the PhyreEngine™ PUtilityDebug library linked into it at compile time in order for the DNet communication to operate.

Hint:

- A PhyreEngine™-derived application must be running before you try to connect to it with PhyreStation.
- On a Microsoft Windows OS platform, the PhyreEngine™-derived application must have the default local port. The application will get the default port if it is run before PhyreStation; this is the case, for example, for the Microsoft Windows compiled version of the PhyreEngine™ COLLADA Viewer.
- If you are using a Sony PLAYSTATION®3 as a host with SN ProDG Target Manager software, look at its log output for the text reporting the current TTY connection IP address. Use this IP address to connect to the host.

Viewing Camera Objects over DNet

To view the projected scene for a selected camera object over DNet:

- Establish a connection as described above. After you have made a valid connection to a host, the **View Over DNet** option is enabled on the PhyreEngine™ camera object's context menu.
- Select **View Over DNet** to transfer the necessary data to the host and view the scene. In the case of the PhyreEngine™ COLLADA Viewer, after the data has been received, it displays the scene as shown in the PhyreStation **Render Viewer** for that selected camera.

Figure 48 View Over DNet Command

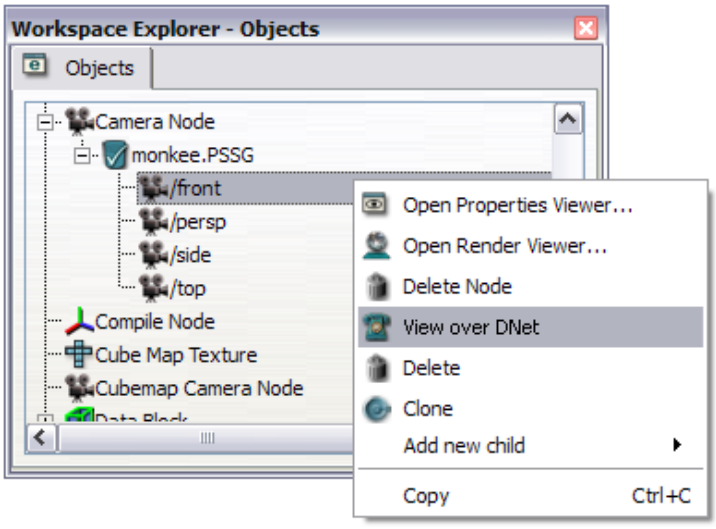
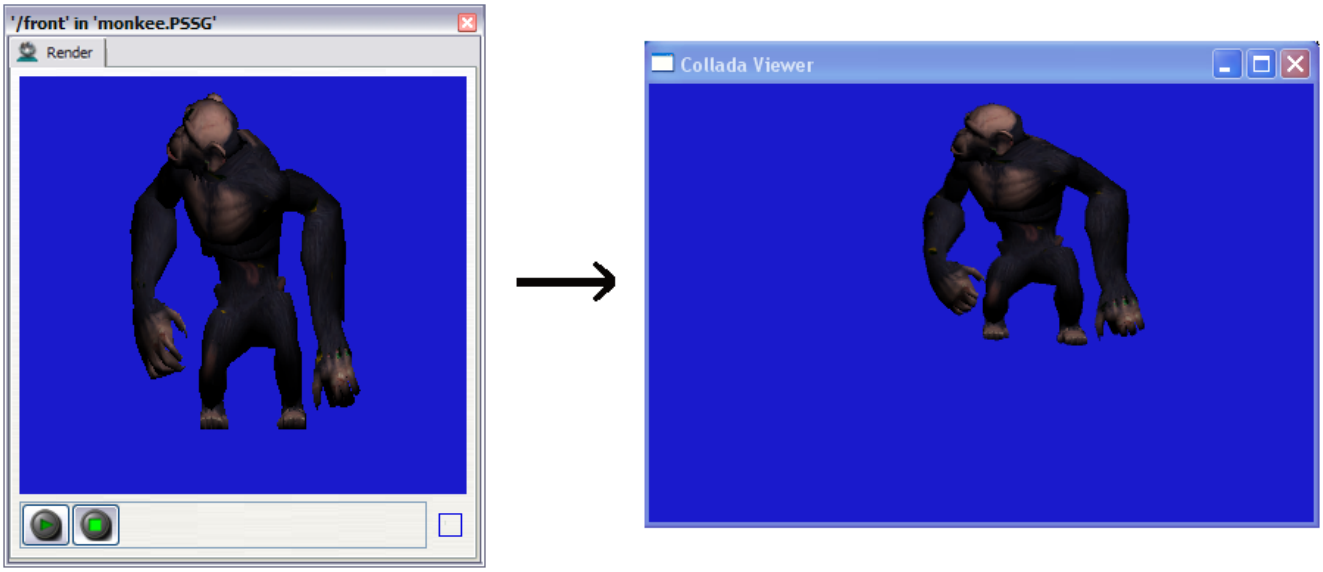


Figure 49 demonstrates how the PhyreEngine™ COLLADA Viewer reflects the same view of a scene through a camera as selected and shown in a PhyreStation **Render Viewer**.

Figure 49 View Over DNet Command Transfer

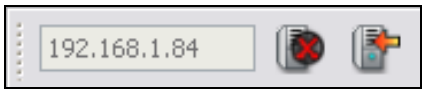


Loading a Remote Database over DNet

To load a remote database:

- Establish a connection as described above. The load remote database button is enabled when a connection has been established.

Figure 50 Load Remote Database Button



- Click the load remote database button to display a list of PhyreEngine™ databases found on the host.
- Select the required database and load it into the current workspace. Databases that are shown in red cannot be selected, because a database by that name already exists in the current workspace.

An imported database is identical or almost identical to a database selected from the host. These are some limitations when imported database from a host:

- You cannot import a database if a database with the same name already exists in the workspace.
- You cannot resolve a database if it is disconnected from the host and the database needs resources that are available only over the remote connection. The database will be shown as a broken database in the **Workspace Explorer – Databases** view.
- Host applications may have discarded data to minimize memory usage. This data will not be available in the workspace or to PhyreEngine™. In some cases, this can affect how some scenes are rendered.
- When rendering a scene using PhyreStation's **Render Viewer**, the scene may also appear different to that on the host. This can be due to the host dynamically creating data as it runs. Such data can be:
  - Textures
  - Shadows
  - Text

  Only static data is transferred from the host.
- Shaders can also cause differences in how some scenes are rendered. Shaders can be specific to a platform and so may not be compatible with the platform PhyreStation is running on.

When resolving any databases that have been remotely loaded into the workspace, PhyreEngine™ first looks locally for any link databases, before searching the remote connection.

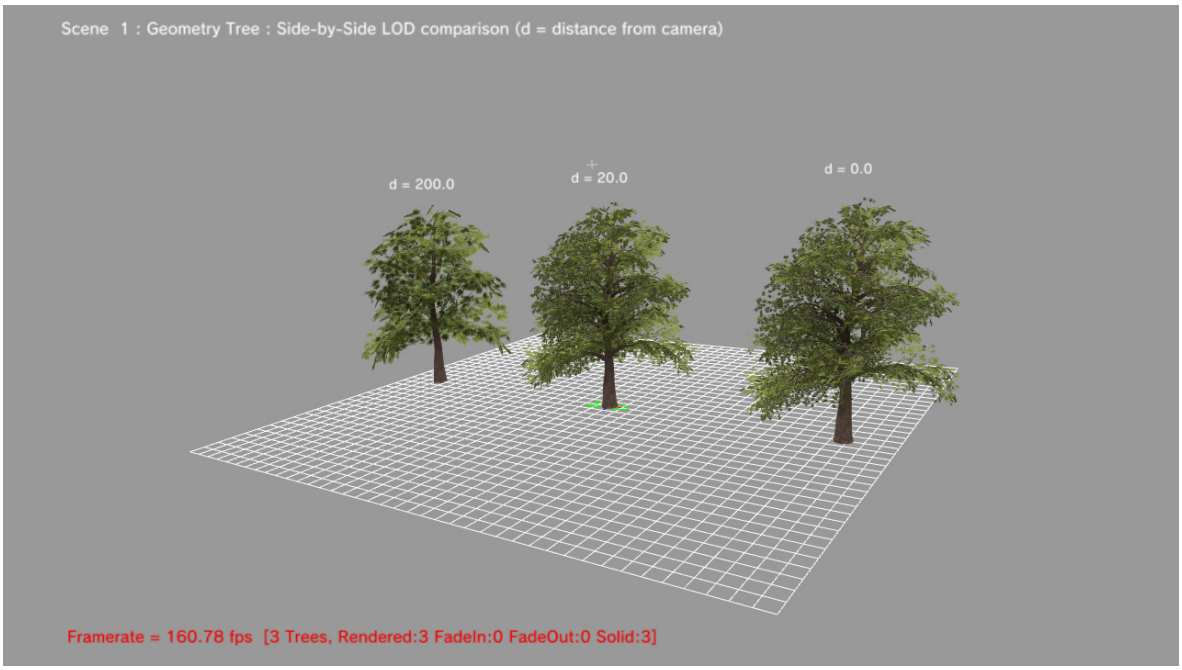
Example: Remote Database Usage

This example demonstrates how to load a remote database using DNet.

PhyreStation is connected to a Sony PLAYSTATION®3 Reference Tool that is running the PhyreEngine™ sample FoliageRendering (`[PhyreEngine]\Samples\Advances\FoliageRendering`).

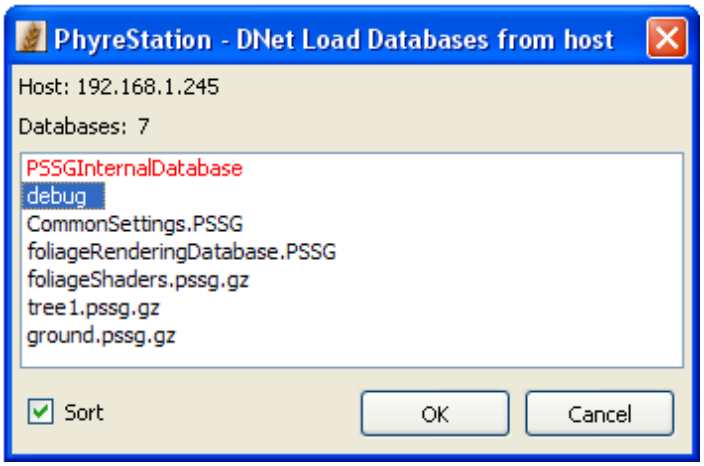
Figure 51 shows the FoliageRendering sample running on a host.

Figure 51 PhyreEngine™ Sample Running in Host



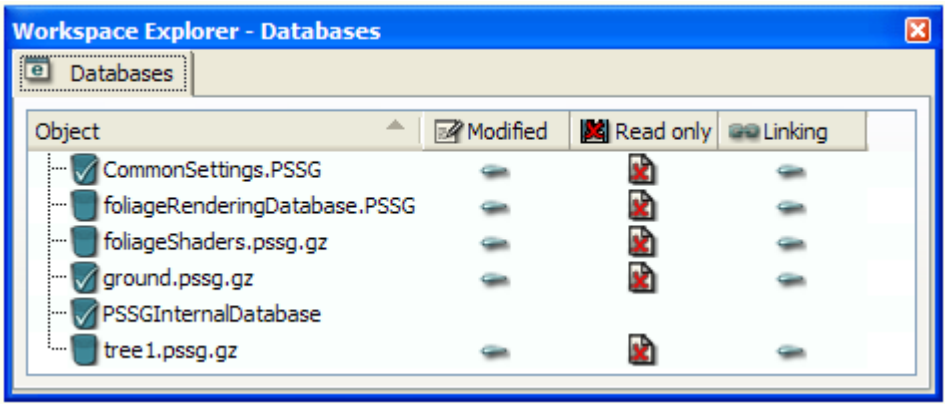
Clicking on the **Load Remote Database** button opens a dialog displaying a list of the databases that are currently residing on the host. Some of these databases can be copied to PhyreStation.

Figure 52 Available Remote Databases



Depending on your requirements, you can select all or some of the databases from the list (those that are not shown in red). The **Workspace Explorer – Databases** view shows the loaded databases in the workspace.

Figure 53 Loaded Databases in the Workspace Explorer

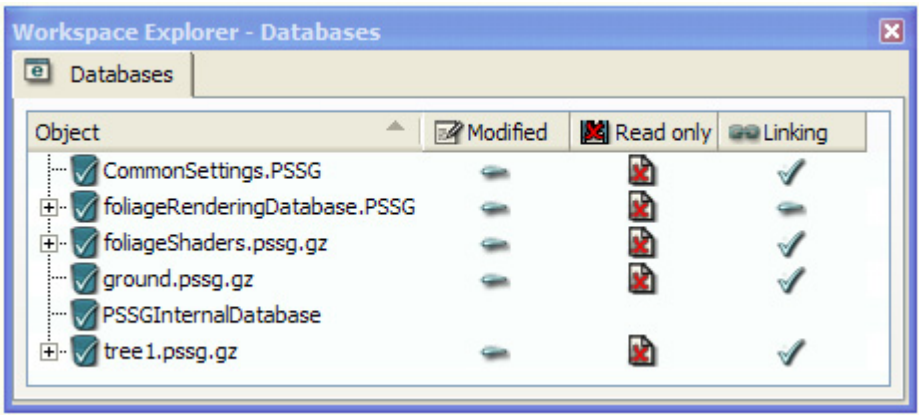


In *Figure 53*, note the following:

- The databases are marked with a **File does not exist** icon. This is because the equivalent files do not exist locally.
- The icon shows that some of the databases need to be resolved. Select the databases that need to be resolved and select **Resolve** from the context menu to populate the workspace with those PhyreEngine™ objects required by PhyreEngine™ to render any scenes.

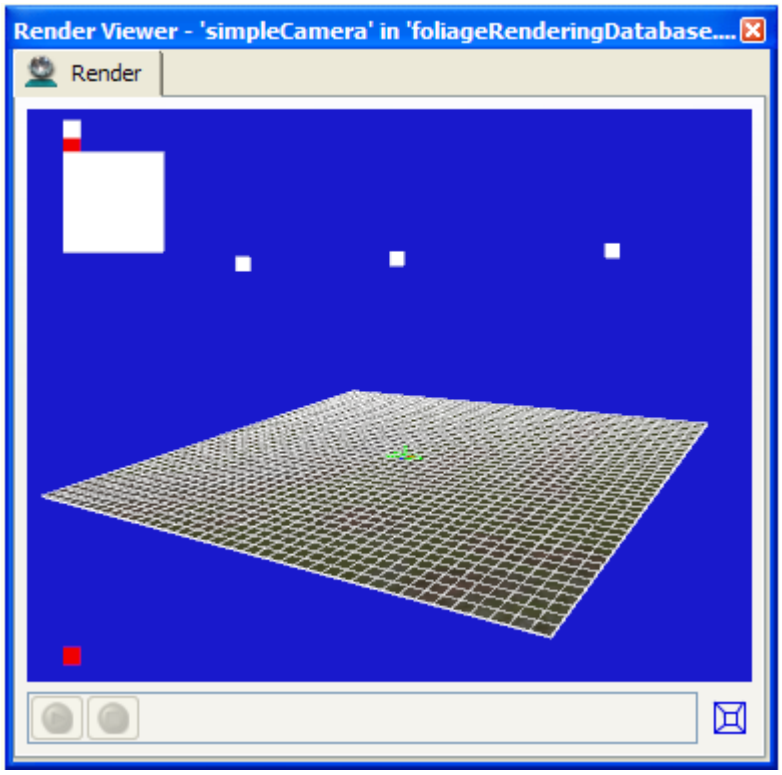
In this example, the user resolves all the databases in the workspace (see *Figure 54*).

Figure 54 Resolved Databases



The user selects the appropriate camera object or selects **View Scene...** from a database context menu to display a scene in the **Render Viewer,** as shown in *Figure 55*. This scene matches the scene displayed on the host (see *Figure 51*).

Figure 55 Scene Displayed in Render Viewer



Note, however, that the details in the rendered scene are very different to those in the scene displayed on the host. This is because the trees, the trees' shadows, and the text display are all dynamically created in real time on the host. The objects that represent these elements do not exist statically and so are not transferable to PhyreStation. This is not always true when copying databases to PhyreStation; it depends on your circumstances or requirements and how you have built your engine and data.

Apart from the visual aspect of inspection, you can examine other aspects of the databases copied across to PhyreStation; this can be valuable to you in determining the results of your work and your workflow.

11 Viewers

Overview

The following viewers are available in PhyreStation:

- Render Viewer – displays camera node objects
- Segment Set Viewer
- Shader Instances Viewer
- Texture Viewer
- Shader Program Viewer
- Shader Group Viewer

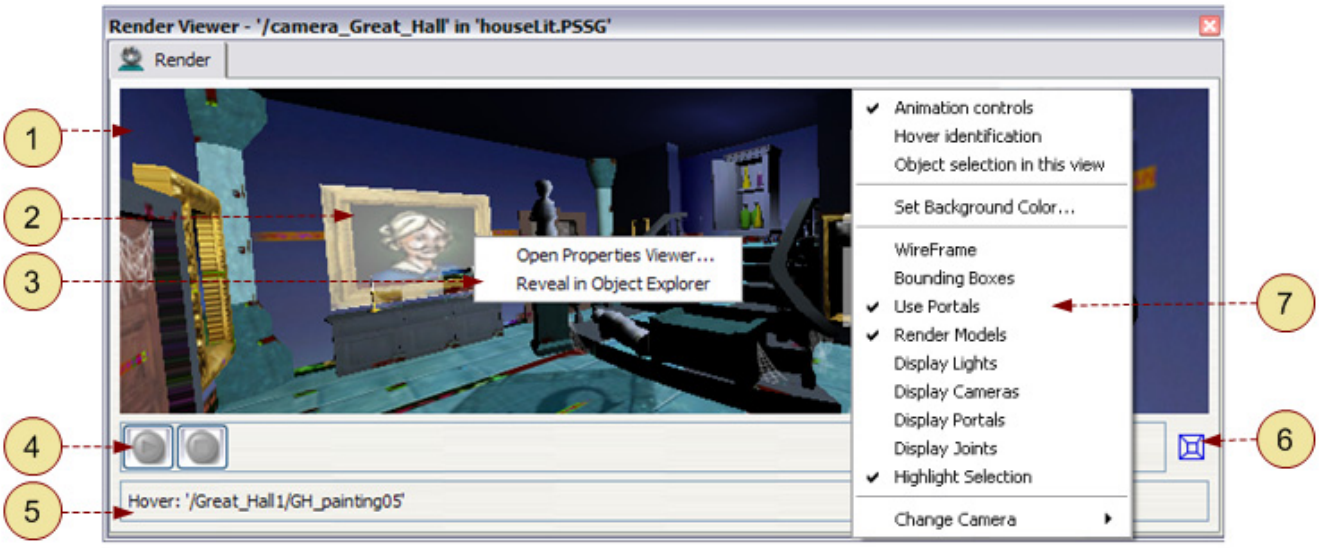
Render Viewer

The **Render Viewer** displays the current PhyreEngine™ scene from the point of view of a specific PhyreEngine™ camera node object. It is possible that you will not see anything if the camera is located in the wrong place or pointing in the wrong direction. You can manipulate the camera orientation or position by using keyboard controls or by changing attributes values of the camera object.

> **Hint:** If you wish to see the scene as a whole or look at a specific node in a scene, use either the **Database > View Scene** or **Node > View Scene Node** features. These options do not change the state of any objects in the database.

To display a camera node object in the **Render Viewer**, select **Open Render Viewer...** from the camera object's context menu.

Figure 56 Render Viewer



1. PhyreEngine™ rendered scene from the point of view of the chosen PhyreEngine™ camera object.
2. Selected PhyreEngine™ object(s) are drawn highlighted (highlight optional).
3. Context menu for the selected scene object.
4. Animation control buttons (display optional).
5. Mouse cursor hover over PhyreEngine™ scene object name (display optional).

6. Type of camera rendering the scene (display optional).
7. View context menu (right click mouse on view white space; use **Ctrl** key if needed).

Render Viewer Context Menu

The **Render Viewer** context menu (item 7) contains the following display options:

Context menu options (internal to PhyreStation)

- **Animation controls** – toggles the display of the animation controls.
- **Hover identification** – toggles the display of the Hover indicator. The Hover indicator displays the name of the PhyreEngine™ object that the mouse is currently hovering over.
- **Object selection in this view** – toggles on/off the ability to select a scene object using a left mouse button click. This allows you to navigate a scene without creating new PhyreEngine™ object selections, as shown in other windows.
- **Set Background Color...** – shows a color selection dialog.
- **Change Camera** – changes to another PhyreEngine™ camera view of the scene. (This option changes the subject for the view.)

Note: The operation of updating the Hover Indicator, when moving the mouse on a large scene, can be very slow. It is a very intensive task, so do not display the Hover indicator if it is not required.

Context menu options (originate from PhyreStationDLL)

These options control the rendering of the scene's layers.

- **WireFrame** – toggles between wire-frame and faceted views.
- **Bounding Boxes** – when enabled, draws red bounding boxes for all PhyreEngine™ node object types. A yellow bounding box indicates that the object is part of a selection.
- **Use Portals** – toggles the use of portals for visibility testing.
- **Render Models** – toggles the drawing of the scene on and off.
- **Display Lights** – toggles the display of lights in the scene.
- **Display Cameras** – toggles the display of other cameras in the scene.
- **Display Portals** – toggles the display of portals.
- **Display Joints** – toggles the display of joints.
- **Highlight Selection** – when enabled, the scene object is redrawn as if it is over-exposed, to show that the PhyreEngine™ object is part of a selection.

Note:

- If a PhyreStationDLL **Render Viewer** view context menu option is not appropriate for the current rendered scene, PhyreStation does not disable the option. You can still execute the command, but it may fail.
- PhyreStationDLL **Render Viewer** view context menu options are customizable and therefore may differ from those described in this guide.
- The available options that originate from the PhyreStationDLL may differ from those listed above.

Scene Object Context Menu

To be able to select scene objects in the **Render Viewer**, you must first enable the **Object selection in this view** option on the **Render Viewer** context menu. When you select objects in the **Render Viewer**, they are also highlighted in all other workspace windows, if appropriate.

You can right-click on a selected object to display the following context menu options:

- **Show Properties Viewer...** displays a static **Properties Viewer** window for the selected object. For information about static and dynamic **Properties Viewers,** see *Dynamic and Static Mode* in *Chapter 5, PhyreEngine™ Objects*.
- **Reveal in a Workspace Explorer** displays the selected scene object(s) in a suitable **Workspace Explorer** view.

Animation Controls

You can control animation data using the animation control buttons.


Play animation


Stop animation

These buttons are enabled only if the following PhyreEngine™ objects are available in the same PhyreEngine™ database as the camera: an animation set, an animation, an animation channel and at least two animation channel data blocks (time and value blocks).

Note:

- Changing the playback state of one **Render Viewer** window applies only to other **Render Viewer** windows that are displaying the same subject. For example, clicking the **Stop animation** button in one **Render Viewer** stops the animation in only those **Render Viewer** windows that are displaying a scene from the same database.
- The playing of an animation changes the state of database. Some new objects are added to the database, and other objects may be changed.

Camera Type Indicator

The selected PhyreEngine™ camera object can project the scene in either a perspective or orthogonal manner. This determines the navigation controls used to move the camera. The camera type indicator displayed in the lower right corner of the **Render Viewer** window shows the current camera type.


Orthogonal type camera


Perspective type camera

Navigation Controls

To change the camera view of the scene, hold down the **Alt** key and use the mouse buttons as described in *Table 7*.

Table 7 Render Viewer Controls

Mouse button held down	Camera	Effect when moving mouse
Left		Zoom viewport to magnify the image.
		Rotates the camera (that is, it looks around) on the spot. If one or more objects in the scene are selected, the camera rotates around the object or around the centre of the combined objects.
Middle		Moves (translates) the camera about the scene vertically or horizontally.
		Moves (translates) the camera about the scene vertically or horizontally.

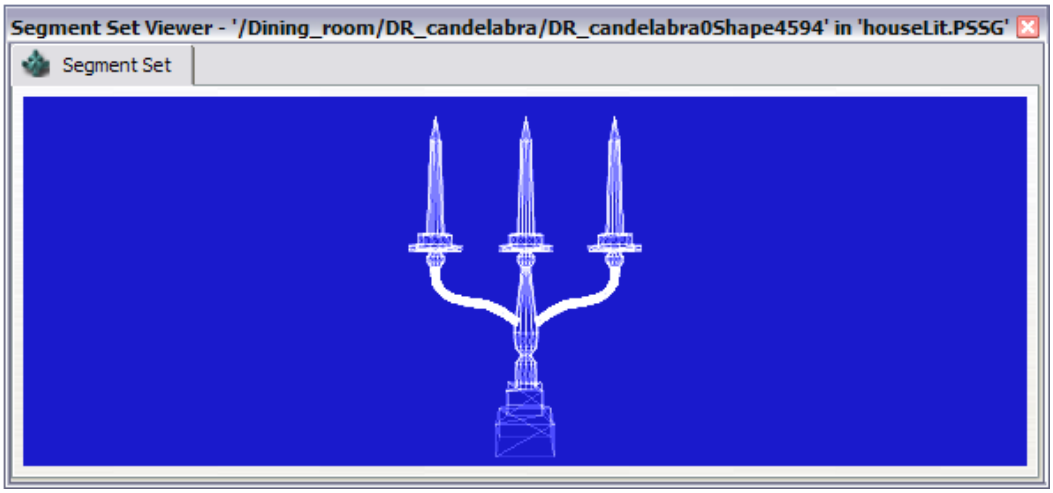
Mouse button held down	Camera	Effect when moving mouse
Right		Moves (translates) the camera forwards or backwards. This may appear to have no effect if viewing in orthographic projection, but the **Properties Viewer** will confirm the movement.
		Moves (translates) the camera forwards or backwards.

Note: Manipulating a camera by using the navigation controls will change the state of that camera object and so the state of the database. Temporary PhyreEngine™ workspace camera objects are an exception.

Segment Set Viewer

To display a segment set object in the **Segment Set Viewer**, select **Open Segment Set Viewer** from the object's context menu.

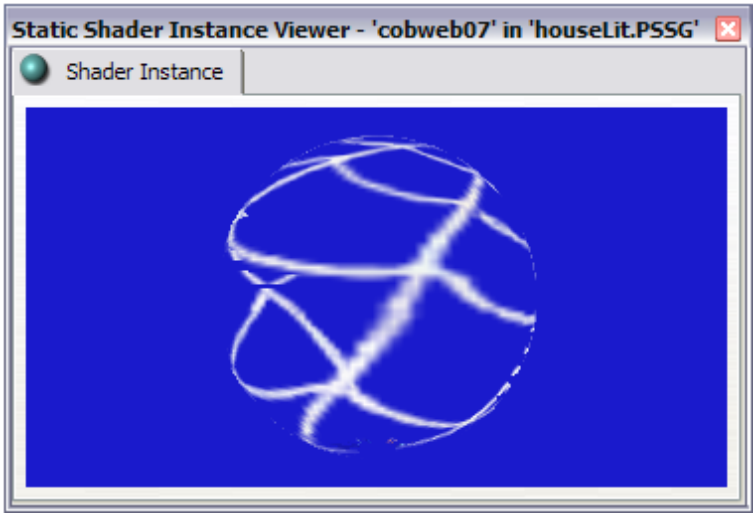
Figure 57 Segment Set Viewer



Shader Instance(s) Viewer

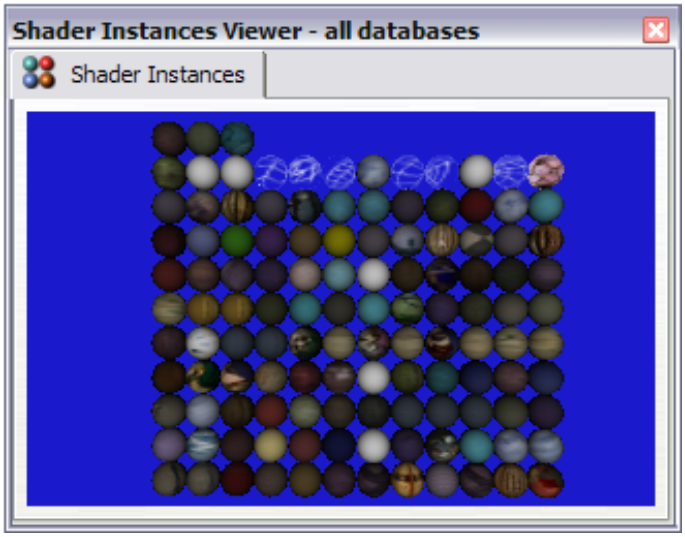
To display a single shader instance object in the **Shader Instance Viewer**, select **Open Shader Instance Viewer** from the object's context menu.

Figure 58 Shader Instance Viewer – Single Object



To display all shader instance objects from all the databases in the workspace, select **Open Shader Instances Viewer** from any shader instance object's context menu. This viewer is useful for comparing shader instances.

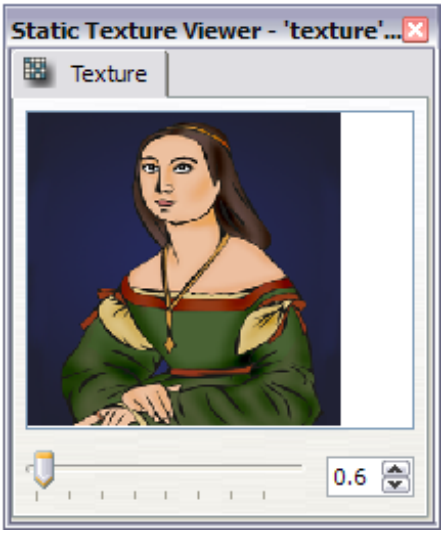
Figure 59 Shader Instances Viewer – All Objects



Texture Viewer

To display a texture object in the **Texture Viewer**, select **Open Texture Viewer** from the object's context menu.

Figure 60 Texture Viewer



Zoom Control

To zoom in and out of the **Texture Viewer** image, do one of the following:

- Use the zoom slider control.
- Edit the zoom factor in the spin control box. The initial zoom value is set to fit the image exactly into the viewer window.
- Press the **Ctrl** key and spin the mouse wheel.

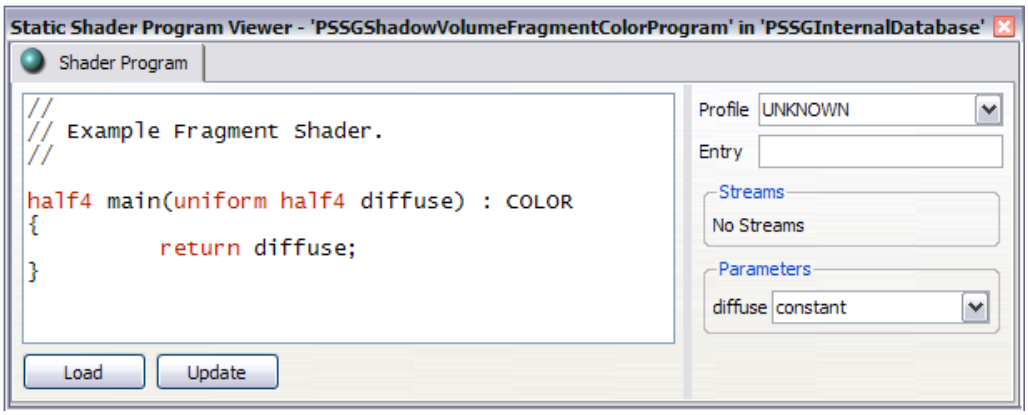
The magnification range for the **Texture Viewer** is from 0.1 to 16 times the image's original size. A magnification value of 1.0 shows the image at its true size.

Shader Program Viewer

To display a shader program object's program and parameter information, select **Open Shader Program Viewer...** from the object's context menu.

To choose another program for the object, click the **Load** button. To commit any changes to the shader program's parameters, click the **Update** button.

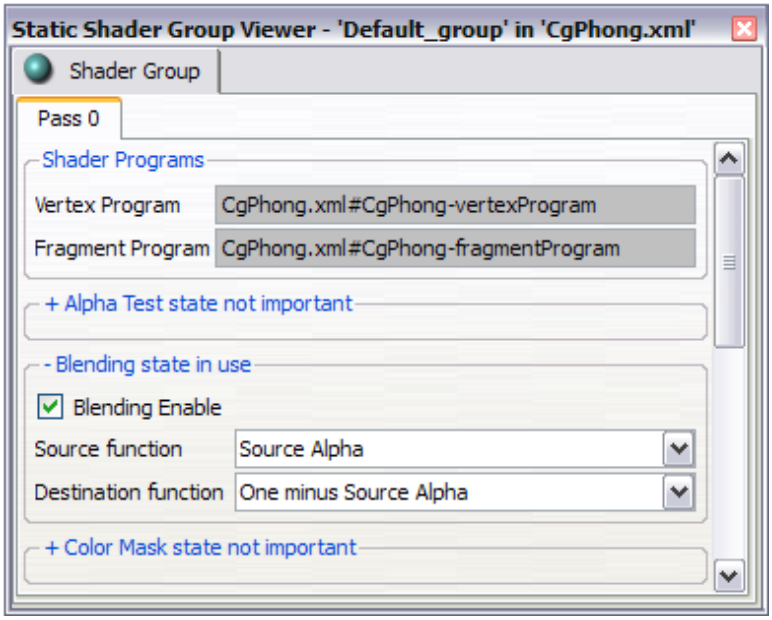
Figure 61 Shader Program Viewer



Shader Group Viewer

To display a shader group object's information in the **Shader Group Viewer**, select **Open Shader Group Viewer...** from the object's context menu.

Figure 62 Shader Group Viewer



Any changes that you make to the parameters in the **Shader Group Viewer** have an immediate effect; you can see the effect of your changes by displaying the subject in a **Render Viewer**.

When you collapse a group box, for example the **Alpha Test state** group box, the operation changes from 'in use' to 'not important' and PhyreEngine™ is instructed not to include this operation in the pass. When the group box is expanded, the operation is included in the pass. This setting is distinct from the enabled/disabled state indicated by the operation's **Enable** checkbox. For example, it is possible to have an operation included in the pass, but not enabled.

12 Graph Editors

Overview

In PhyreStation, there are three graph (or network) editors:

- Target blender editor – animation target blender objects
- Modifier network editor
- Modifier network instance editor

You use these graph editors to display and edit relationships between PhyreEngine™ objects and processes. All graph editors share some common behavior, but there are also some differences.

- Common graph behavior is provided by PhyreStation and cannot be modified.
- Behaviors that are specific to individual graph editors are driven by the class interfaces provided in PhyreStationDLL; you can modify these behaviors. Depending on the type of graph editor in use, specific behaviors may be indicated by the interface (for example, change of connection line color to show which connections are valid or invalid).

Common Graph Editor Behavior

This section uses the **Target Blender Editor** as an example to describe the graph editor interface and basic functionality. The interface and functionality described here are common to all graph editors.

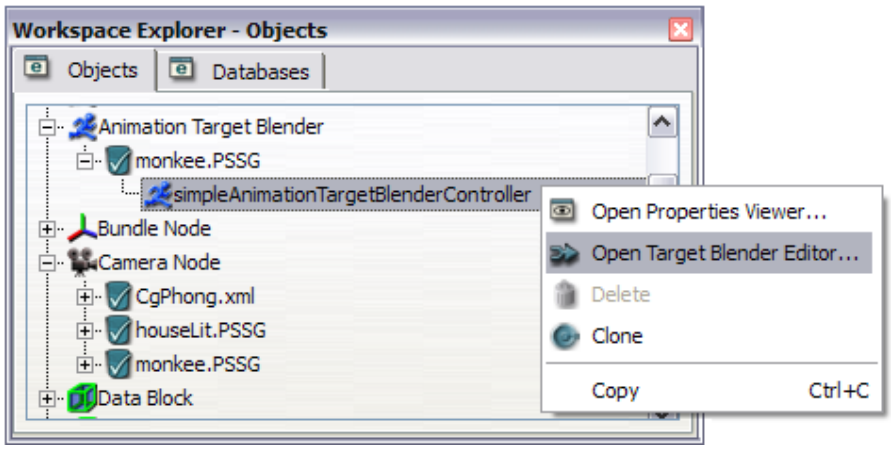
Opening a Graph Editor

To open the **Target Blender Editor**:

(1) Open the **Workspace Explorer Objects** view.

(2) Right-click an animation target blender object and select **Open Target Blender Editor...** from the context menu. The **Target Blender Editor** opens.

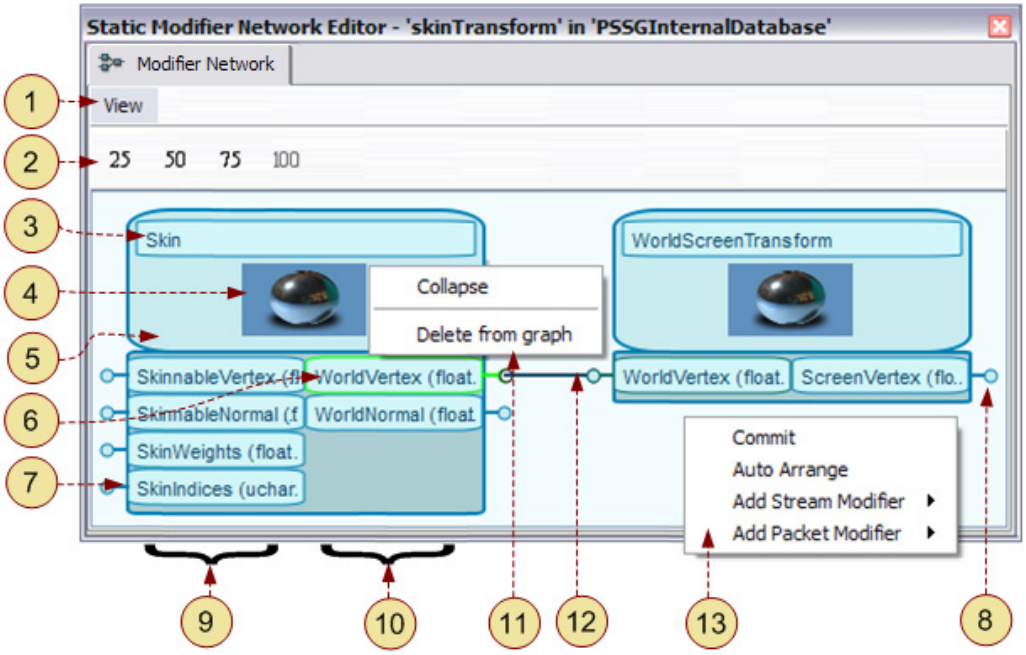
Figure 63 Opening a Target Blender Editor



(3) To apply changes to the relevant PhyreEngine™ database(s), select Commit from the editor's context menu.

Graph Editor Interface

Figure 64 Graph Editor Features

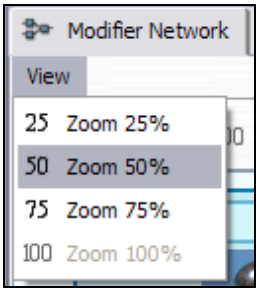


1. Menu bar
2. Toolbar
3. Graph node label: normally the name of the PhyreEngine™ object or process
4. Graph node icon: an image representing the PhyreEngine™ object or process
5. Graph node
6. Node connector (output), highlighted green to show that it is connected
7. Node connector (input) that is not connected
8. Handle to a node connector (output)
9. Set of input node connectors
10. Set of output node connectors
11. Graph node context menu
12. Line showing possible flow of data between two node connectors
13. Background context menu

Menu Bar

The Graph Editor has a single **View** menu. Use this menu to set the current zoom for the background area.

Figure 65 Graph Editor View Menu



Hint: Zoom can also be activated by holding down the **Alt** key and moving the mouse horizontally.

Context Menus

To display a context menu, right-click on the background area or on a graph node.

- The background context menu (item 13 in *Figure 64*) lists operations that affect the graph as a whole.
- Node context menus (item 11 in *Figure 64*) list options that are specific to graph nodes. When a context menu command is selected for multiple nodes, any nodes that are not valid targets of the command are ignored.

Context menu operations are registered either by PhyreStation or by PhyreStationDLL. PhyreStationDLL programmers may modify, extend, expose, or hide PhyreStationDLL information.

Graph Nodes

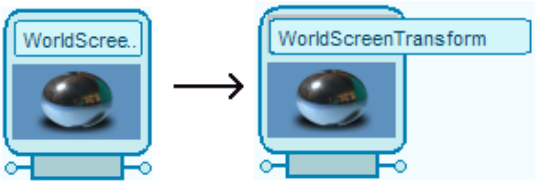
Overview

Objects and processes are represented in the editors by graph nodes. Nodes can be linked together via their node connectors. These connectors are classed as either input or output. Input implies that data flows into the node; output implies that data flows out of the node.

By convention, the graph editors show data flowing from left to right. Input node connectors are displayed on the left side of graph nodes, and output node connectors on the right.

Node and connector labels are shortened to the width of the node. To show them in full, hover the mouse over the node or connector label.

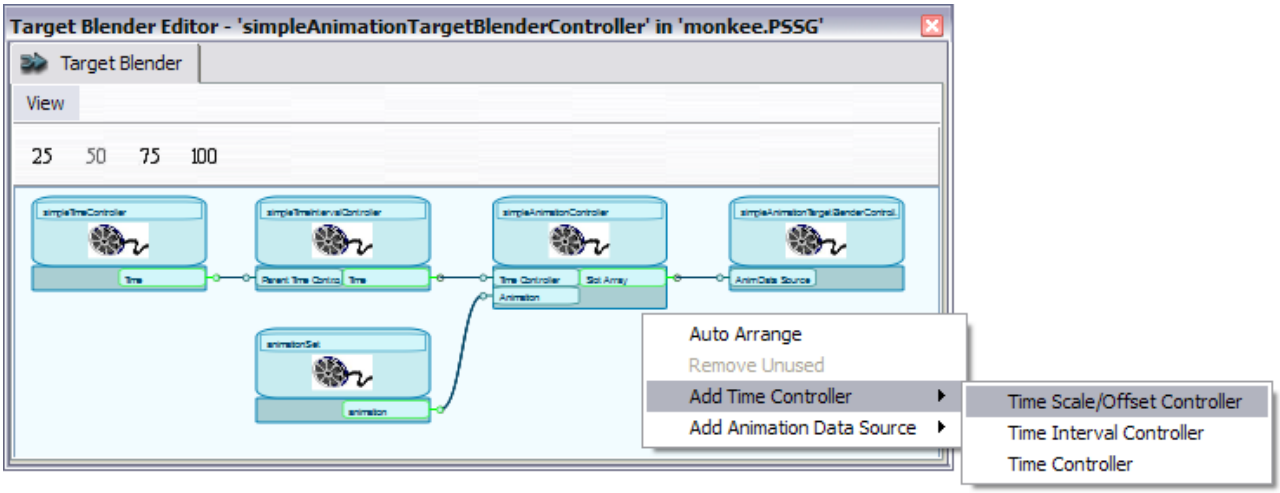
Figure 66 Viewing a Node Label



Adding Nodes

To add a graph node, right-click the background area and select the required type from the context menu.

Figure 67 Adding a New Graph Node



Moving Nodes

To move a graph node, drag the node to the required position. The node's connecting lines are highlighted during dragging.

To arrange the nodes in a grid-like manner so that data flows from left to right, select **Auto Arrange** from the editor's context menu. Nodes without connections are placed at the base of the background area.

Deleting Nodes

To delete a graph node, select **Delete** from the context menu. The **Delete** option is disabled if the node has a dependency.

To remove all non-essential unconnected nodes from the graph, select the **Remove Unused** option from the context menu.

Resizing Nodes

You can resize graph nodes to provide more space in the background area. The graph nodes are resized through four collapse states. To cycle on to the next collapse state, select **Collapse** from the node's context menu.

Figure 68 Cycling Through Graph Node Collapse States

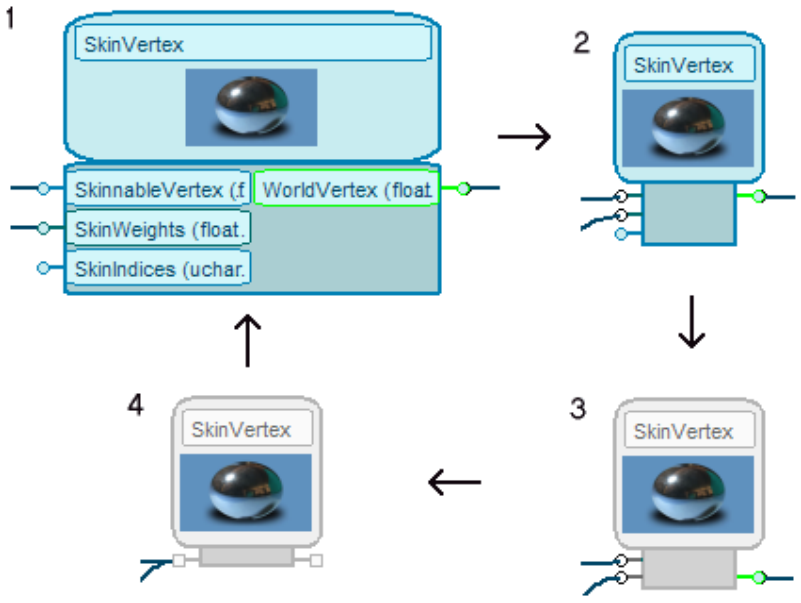


Table 8 describes these collapse states in detail.

Table 8 Graph Node Collapse States

State	Description
1	Display the graph node at full size with all its connectors, whether connected or not.
2	Display the node with all its connectors, but without connector labels.
3	Display the node showing attached connectors only. In this state, users cannot edit the node.
4	Display the node with all input connectors combined, and all output connectors combined. In this state, users cannot edit the node.

Finding Nodes

To locate a particular node in a **Workspace Explorer – Objects** view, select **Find in Workspace Explorer** from the node's context menu.

Connecting Nodes

A graph node can have any number of connectors, either on the input side or on the output side. Each input connector on a node may only accept connections from output connectors on other nodes. Likewise, each output connector on a node may only be attached to input connectors on other nodes.

To make a connection, perform the following steps (see *Figure 69*):

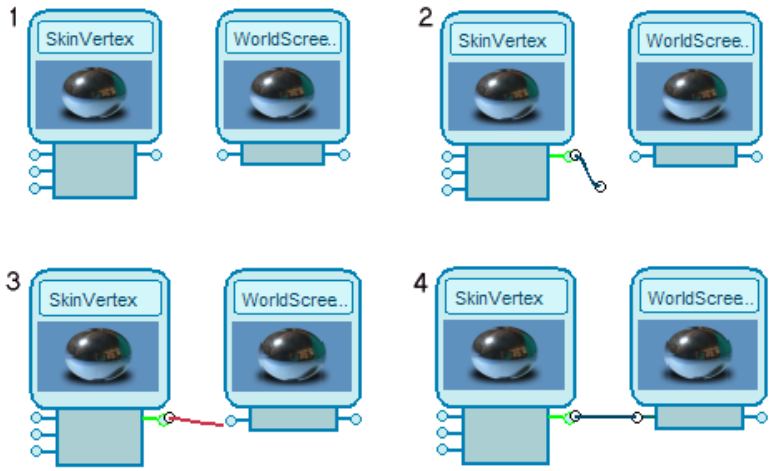
(1) With the left mouse-button, click a source connector on a graph node. (This may be an input or output connector.)

(2) Drag the mouse to the target connector on a different node. (This may be an input or output connector.) The PhyreStationDLL implementation is queried for connection validity. If the connection is valid, the connecting line is colored red, otherwise it remains black.

(3) Release the mouse button to make the connection.

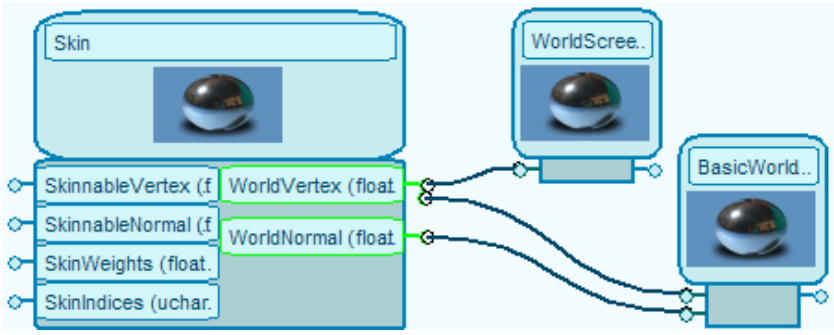
Note: Depending on the type of graph editor, a new connection may not be saved to the PhyreEngine™ database until a **Commit** command is issued.

Figure 69 Connecting Graph Editor Nodes



If supported by the PhyreStationDLL implementation, a node connector may be connected to multiple nodes simultaneously. An example of this is shown in *Figure 70*.

Figure 70 Multiple Node Connections



Removing Connections

To remove a connection, click on one of the two node connectors and drag the linking line away so it points to an empty space. After you hold down the left mouse-button, you must pause briefly before performing the drag; this helps you to avoid deleting the links accidentally when moving nodes around the work area.

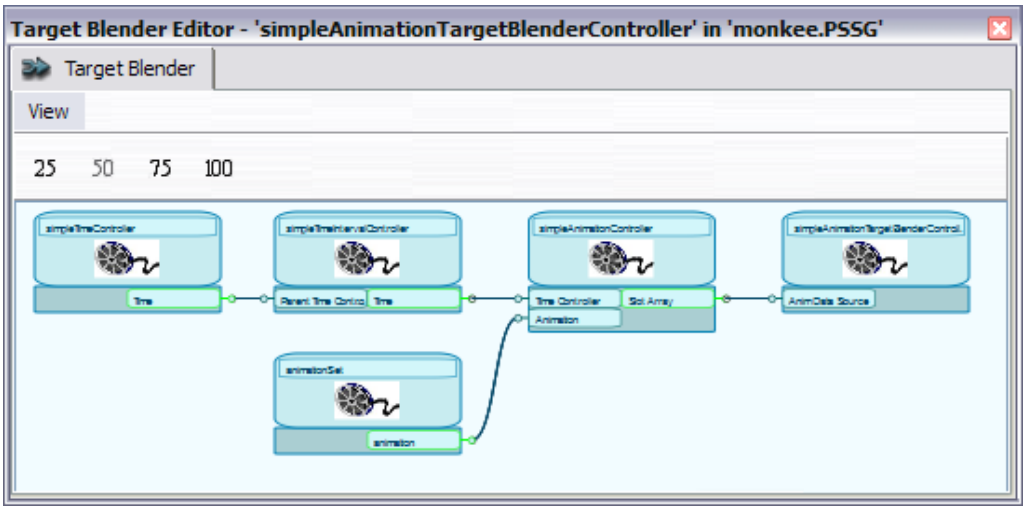
Specific Graph Editor Behavior

Target Blender Editor

The **Target Blender Editor** is used to view and edit animation target blender PhyreEngine™ objects.

> **Note:** Animation target blenders are not shown in the **Workspace Explorer – Objects** view until the associated animation is played (in the same database).
> To play the animation, open the **Render Viewer** window for the relevant camera object and click the **Play** button.

Figure 71 Target Blender Editor



Graph nodes in the **Target Blender Editor** represent the animation target blender and a range of time controllers and animation data sources.

To add new time controllers and animation data sources to the graph, select **Add Time Controller** and **Add Animation Data Source** from the background area context menu.

> **Note:** Any changes made using the **Target Blender Editor** are applied to the relevant PhyreEngine™ database immediately.

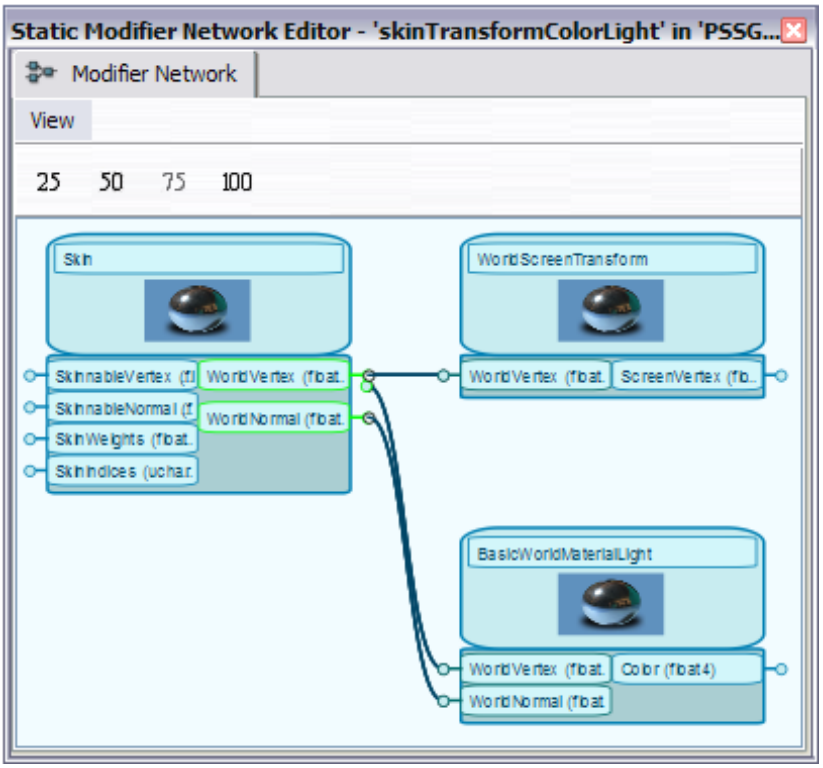
See the *PhyreEngine™ Programming Guide* for more details about animation time controllers and data sources.

Modifier Network Editor

The **Modifier Network Editor** is used to view and edit modifier network PhyreEngine™ objects.

> **Note:** Modifiers are processes rather than PhyreEngine™ object types, and therefore are not shown in the **Workspace Explorer – Objects filter** view. Modifiers can only be edited using the **Modifier Network Editor**.

Figure 72 Modifier Network Editor



Graph nodes in the **Modifier Network Editor** represent modifiers.

To add new stream modifiers and packet modifiers to the graph, select **Add Stream Modifier** and **Add Packet Modifier** from the editor's context menu.

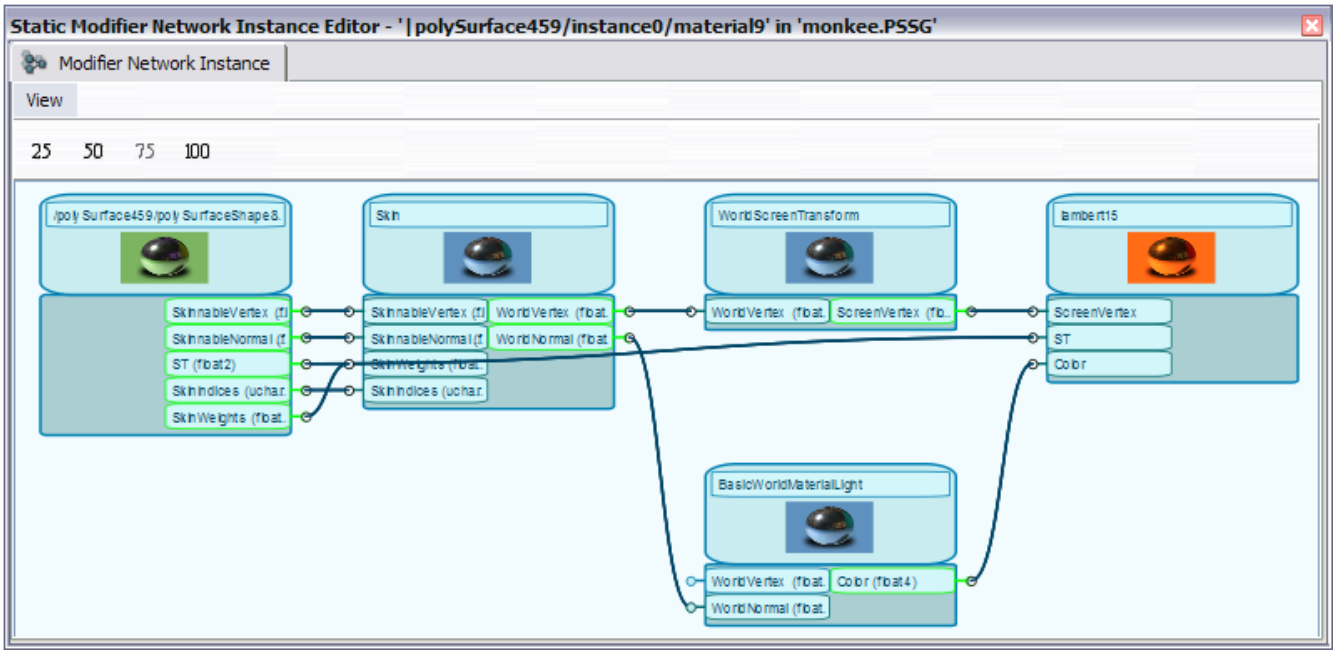
> **Note:** Any changes made using the **Modifier Network Editor** are applied only after the **Commit** context menu option is selected. Corresponding modifier network instance objects are not affected until then.

See the *PhyreEngine™ Programming Guide* for more details about modifiers.

Modifier Network Instance Editor

The **Modifier Network Instance Editor** is used to view and edit modifier network instance PhyreEngine™ objects.

Figure 73 Modifier Network Instance Editor



Graph nodes in the **Modifier Network Instance Editor** represent render data sources and have a green image. (Modifiers are blue, and shader instances are orange.)

To add new stream modifiers and packet modifiers to the graph, select **Add Stream Modifier** and **Add Packet Modifier** from the background area's context menu.

Any render data sources or shader instance objects that are represented in the graph, and that are changed externally to the editor, are automatically updated to reflect the properties of the object that was just changed. This does not apply to nodes that represent modifiers, as they do not have an equivalent PhyreEngine™ object.

Note: Any changes made within the **Modifier Network Instance Editor** are applied only after the **Commit** context menu option is selected.

See the *PhyreEngine™ Programming Guide* for more details about modifiers.

13 Animation Editor

This chapter describes the PhyreStation waveform editor – the **Animation Editor**.

Overview

The **Animation Editor** is a waveform editor associated with the PhyreEngine™ animation object type. It allows you to view and edit the animation data in the animation's animation channels.

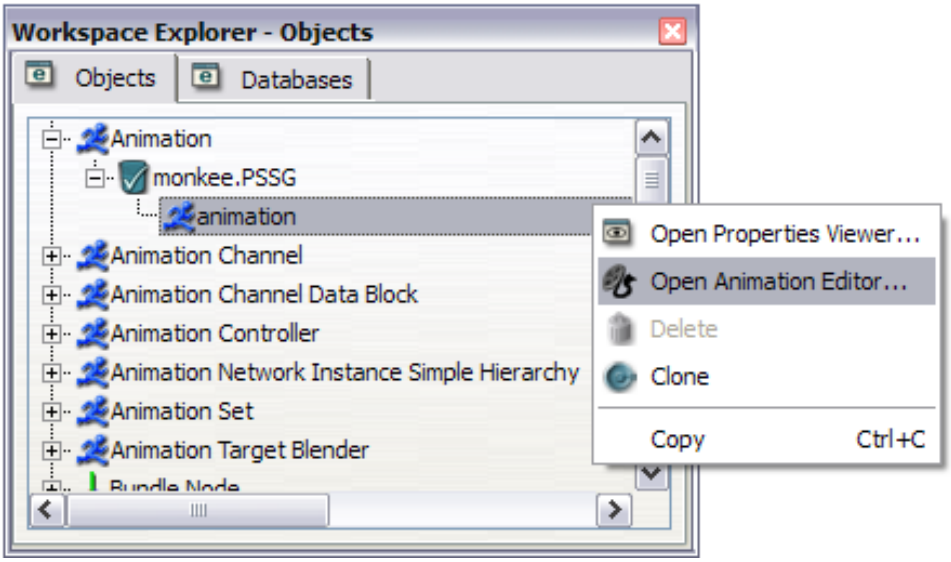
Opening the Animation Editor

To open the animation editor:

(1) Open the **Workspace Explorer – Objects** view.

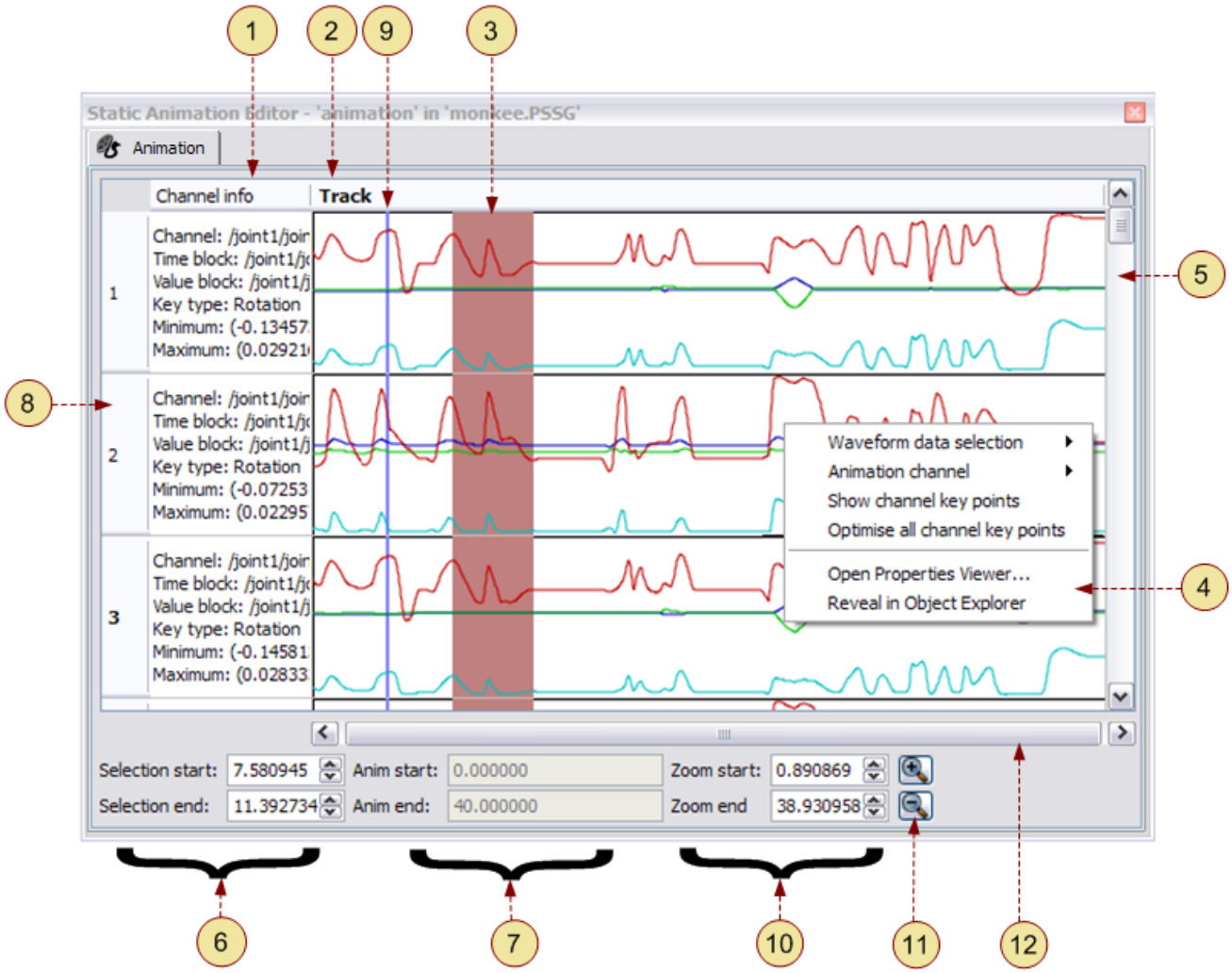
(2) Right-click on a PhyreEngine™ animation object and select **Open Animation Editor...** from the context menu.

Figure 74 Opening an Animation Editor



Animation Editor Interface

Figure 75 Animation Editor



1. Animation channel information (read only).
2. Animation channel waveform data. Time is represented on the horizontal axis.
3. Selection band (area in pink).
4. Animation editor context menu.
5. Vertical table scroll bar to scroll through the rows of animation channels.
6. Fine adjustment of the selection band's start and end position.
7. Animation's start and end information (read only).
8. Selected table row highlighted by a black border.
9. Time line.
10. Editor's zoom start and end information.
11. Editor's zoom in and out buttons.
12. Scroll bar to scroll all animation channel information.

Animation Channel Information

The **Channel info** column (item 1 in *Figure 75*) contains information about the animation waveform data (item 2). The column shows the PhyreEngine™ animation channel name, the names of the channel's value and time blocks, and the type of animation data held.

Animation Waveform Data

The **Track** column (item 2) graphically represents the animation waveform data. For rotations, each of the four components of the quaternion is shown. For translation, the three Cartesian coordinates are shown.

Time Line Position Information

Click on any part of a row in the **Track** column (item 2) to display a time line (item 9). The time position in the waveform data is shown in the selection spin boxes (item 6).

Waveform Data Selection

The selection band (item 3) allows you to work with a selected portion of the data. To select a band of data, click with the left mouse button and drag the mouse across the waveform data required.

Context Menu

To display the context menu, right-click on the background (item 4). The context menu lists operations for the currently selected table row, the animation channel, or a selection of waveform data.

Context menu operations are registered either by PhyreStation or by PhyreStationDLL. You can modify, extend, expose, or hide PhyreStationDLL menu options, as required. For more information, see *Chapter 6, PhyreStation Commands*.

Selection Spin Boxes

You can increase or decrease the currently selected region using the selection spin boxes (item 6). You can also edit the displayed number directly.

Waveform Start and End Boxes

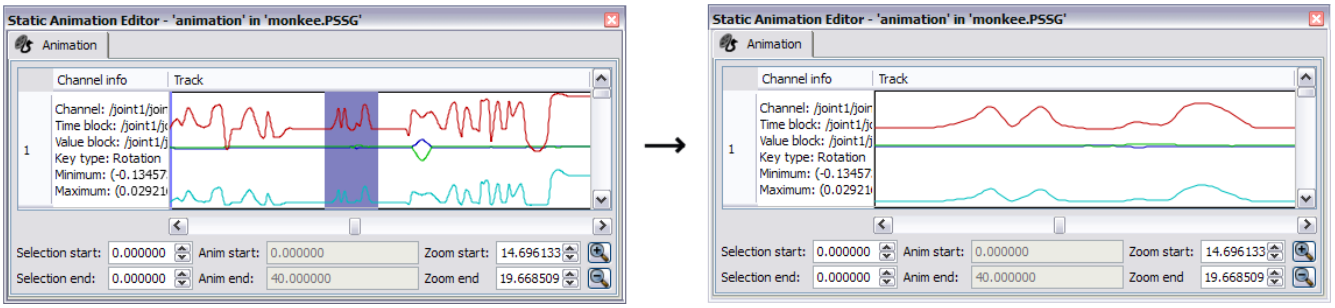
The **Anim Start** and **Anim End** boxes (item 7) display read-only information about the animation's total length in time.

Waveform Zoom Controls

You can magnify the waveform data to show the data in greater detail. The start and end positions of the currently displayed region of data are shown in the zoom spin boxes (item 10).

- To adjust the zoom start and end positions, edit the values directly or use the spin controls.
- To zoom in or out quickly, use the zoom buttons (item 11). After a zoom has been made, a scroll bar (item 12) is enabled to allow access to non-visible parts of the waveform data.
- To zoom into a portion of the waveform data, click a waveform data row using the middle mouse button and drag the mouse. A zoom selection band appears (see *Figure 76*). Release the middle mouse-button to complete the zoom procedure.
- To reset the view to show all the waveform data, click the middle mouse-button in the waveform data area.

Figure 76 Zoom Waveform Data



Editing PhyreEngine™ Animations

You can perform a number of operations on animation waveform data using the editor. You perform most of these operations by right-clicking on part of the waveform data and then selecting a command from the context menu. Other commands operate on the waveform data as a whole.

Note: Any changes applied to an animation using the editor are applied immediately to either the PhyreEngine™ Animation or PhyreEngine™ Animation Channels, or both. This, in turn, changes the relevant PhyreEngine™ database.

Optimise All Channel Key Points

The **Optimise all channel key points** context menu option works on all the animation channels' data for the animation being edited.

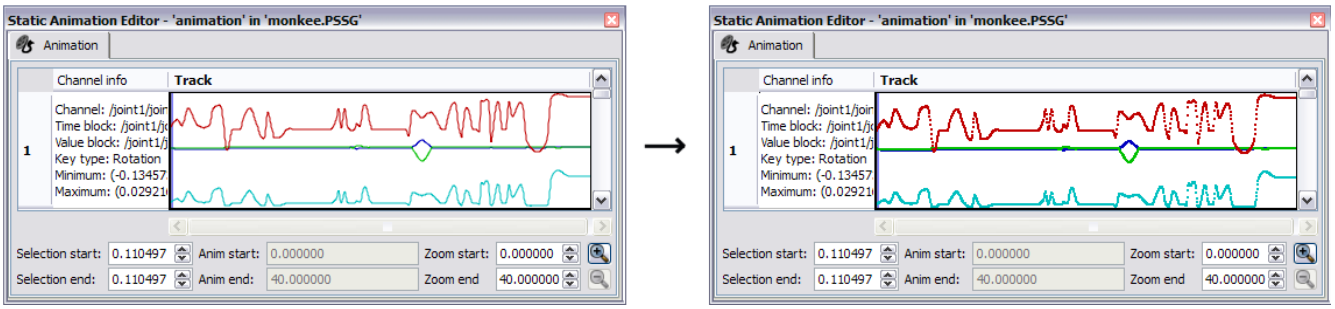
You can also reduce the number of animation key frames by dragging the mouse to select a portion of the waveform data (across multiple animation channels).

You can also perform this operation in a script by using either the `OptimizeAnimation` or `OptimizeAnimationChannel` script commands.

Show Channel Key Points

The **Show channel key points** option changes the waveform representation from a continuous line through the key frame data (the default) to discrete points that represent specific key frame data.

Figure 77 Waveform Data Representation



Crop Animation to Selection

The **Waveform data selection > Crop animation to selection** option reduces the animation waveform data by deleting all the waveform data from all of the animation's channels that lay outside of the selected region. Performing this operation reduces the length of the animation.

You can also perform this operation in a script by using the `CropAnimation` script command.

Delete Selection from Animation

The **Waveform data selection > Delete selection from animation** option allows you to delete a selected region of animation waveform data from all of the animation's channels; this operation reduces the length of the animation.

You can also execute this command in a script by using the `RemoveAnimationInterval` script command.

Create a New Animation from Selection

The **Waveform data selection > Create a new animation from selection** option allows you to create a new PhyreEngine™ animation object using regions of animation waveform data selected from each of the animation's channels.

You can also execute this command in a script by using the `CreateNewAnimationFromInterval` script command.

Delete a Channel from the Animation

The **Animation channel > Delete channel from the animation** option allows you to remove an animation channel table row (item 8 in *Figure 75*) from the animation.

You can also execute this command in a script by using the `RemoveAnimationChannel` script command.

Note: Deleting a PhyreEngine™ animation channel object that is still referenced by a PhyreEngine™ animation, or other PhyreEngine™ object(s), can cause PhyreEngine™ to become unstable. Do not attempt this unless you are sure of the outcome.

14 Particle Editor

This chapter describes the PhyreStation **Particle Editor**, particle system operations such as adding and removing elements and animations, and the process of exporting and compiling a particle system ready for PhyreEngine™.

Overview

The PhyreStation **Particle Editor** allows you to create and edit a PhyreEngine™ Particle System. After defining a satisfactory particle system, you export the particle system to a definition file. The definition file is parsed and source code is produced. The source code is compiled with the code project and used during the run time of PhyreEngine™ to produce a particle system

A PhyreEngine™ Particle System is represented in PhyreStation by the following:

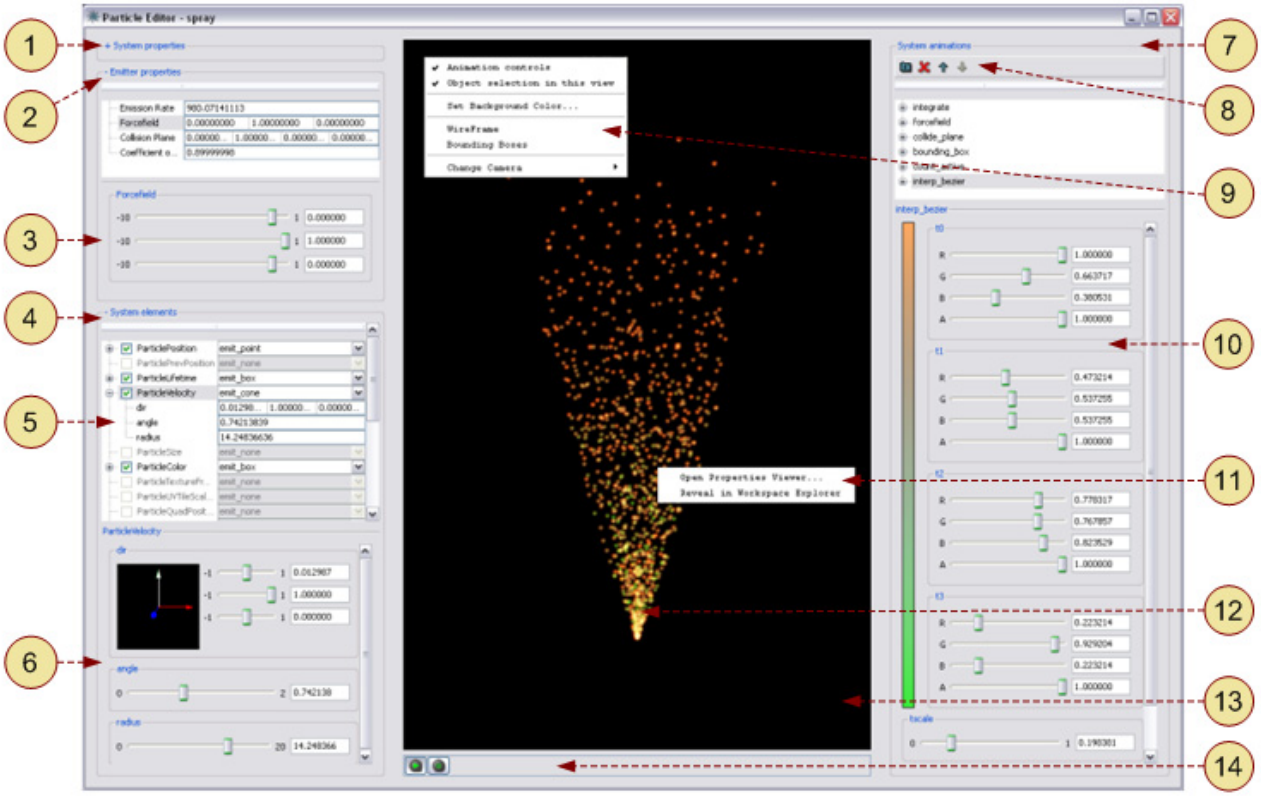
- Textures, PhyreEngine™ Render Streams, and PhyreEngine™ Shader Groups. See *Accessing Objects Using Viewers* in *Chapter 5, PhyreEngine™ Objects*.
- PhyreEngine™ Particle Modifier Network. See *Modifier Network Editor* in *Chapter 12, Graph Editors*.
- PhyreEngine™ Particle Modifier Network Instance. See *Modifier Network Instance Editor* in *Chapter 12, Graph Editors*.
- PhyreEngine™ Particle Modifier behavior – represented by the definition file.

For a high-level workflow that describes how to create a particle system, see *Workflow: Creating a Particle System* in *Chapter 2, Workflows*. For a detailed exercise that describes how to create an example particle system, see *Exercise 6: Set up a Particle System* in *Appendix A: Exercises*.

The Particle Editor

You create and edit particle systems using the **Particle Editor.**

Figure 78 Particle Editor



1. Particle system's properties (collapsed)
2. PhyreEngine™ Particle Modifier Behavior object's properties
3. Extended editing controls for a particle modifier behavior object's properties
4. PhyreEngine™ Particle System's behavior elements
5. Behavior element attributes
6. Extended editing controls for a behavior element's attributes
7. PhyreEngine™ Particle System's behavior animations
8. Particle system's behavior animations toolbar: add, delete, move up, move down
9. Render view's white space context menu
10. Extended editing controls for a Behavior animation's attributes
11. Context menu for the PhyreEngine™ Particle Emitter node object
12. The PhyreEngine™ Particle Emitter node object
13. Render view displaying the particle system being edited
14. Animation controls for the particle system

System Properties

The **System properties** area of the **Particle Editor** displays all the system properties.

- You can right-click fields containing PhyreEngine™ object links to access the context menu for that object. You can modify these fields by dragging existing PhyreEngine™ objects or files into the field.
- If your changes are valid, the **Update system** button is enabled. Click this button to apply your changes.

System Elements

The **System elements** area of the **Particle Editor** displays a tree view of all the currently supported behavior elements and their attributes. Depending on the particle modifier behavior object associated with the particle system, the tree view content may change.

- Non-ticked items are disabled, but are still editable. When enabled, the behavior element is added to the particle system and used. When disabled, the element is removed from the particle system.
- The top-level item for each element displays the `RenderDataType` for the element and the options available from the supported emission types (see the *PhyreEngine™ Core Library Reference*). You can access and edit each behavior element's attributes via the tree view's subitems. Any change made to an emission type of an element immediately updates the whole particle system.
- Some attributes have minimum and maximum allowable values. If, following a change to the emission type, a new attribute is determined to be out of range, the new value is limited to keep it valid.
- When you select an element's top-level item, extended editing controls for that element's attributes are displayed. Different attributes provide different editing capabilities. Attributes that represent direction and vector type values display a small orientation view to provide a three-dimensional representation of the slider values (item 6 in *Figure 78*).

The **System elements** area of the **Particle Editor** provides controls to perform element sorting.

- The **Sort key** drop-down list box displays a list of valid elements from which to select the sort key. To be valid, an element must be selected (or active) in the elements tree view.
- The **Sort type** drop-down list box displays a list of the available sort methods.
- The **Sort all** checkbox provides a mechanism to add or remove all active elements from the particle system's sort list. Elements can be added and removed from the sort list on an individual basis using the checkboxes in the **Sort** column of the element tree view.

System Animations

The **System animations** area of the **Particle Editor** displays the properties for the current behavior animations in the particle system.

- The animations are ordered in the tree view from top to bottom. Behavior animations nearer to the top affect how behavior animations nearer to the bottom update the particle system.
- The **System animation** toolbar allows you to change the order of the animations, and add or remove a behavior animation.
- The attributes displayed in the drop-down lists represent the current valid `RenderDataType` attributes for that behavior animation.
- Selecting an animation's top-level item also displays extended editing controls for that animation's non-`RenderDataType` attributes (item 10 in *Figure 78*).

Editing System Elements and Animations

You can add and remove behavior elements from the particle system's behavior by checking the element's checkbox on or off respectively. This, in turn, affects the `RenderDataType` items in the **System elements** area.

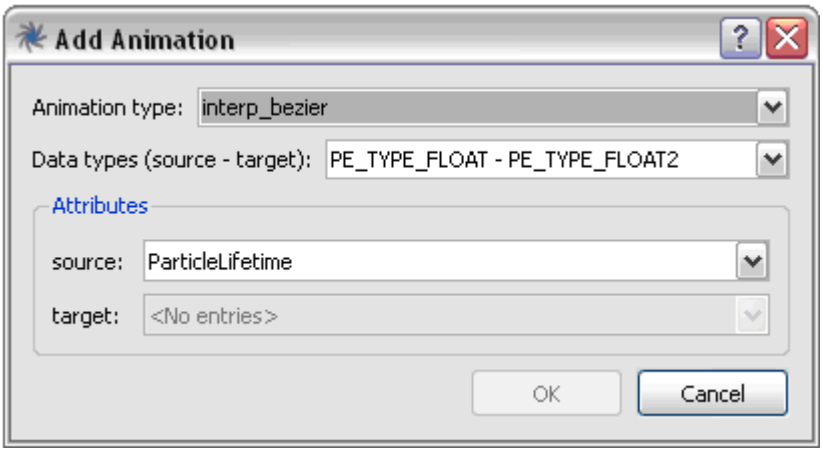
Adding and removing behavior elements can also affect any related behavior animations in the particle system. This is the case when the data type of the behavior element corresponds to the data type of any of the behavior animations' `RenderDataType` attributes.

- When you add a behavior element, the `RenderDataType` for that behavior element is added to the list of a compatible behavior animation's `RenderDataType` attributes.

- When you remove a behavior element, the `RenderDataType` for that element is removed from the list of any compatible behavior animation `RenderDataType` attributes. If removing the `RenderDataType` causes the attribute to have non-valid `RenderDataTypes`, you are informed of any behavior animations that are about to be invalidated, and prompted to either continue or cancel the operation. If you continue, the behavior element and invalidated behavior animations are removed from the particle system.

The relationship between the available behavior elements and behavior animation is also significant when adding new behavior animations. *Figure 79* shows a behavior animation that cannot be added because the particle system's behavior does not include any behavior elements with a `RenderDataType` to support the selected target data type.

Figure 79 Add Animation Dialog



In the dialog, an interp_bezier behavior animation is specified, which has the data types for the two `RenderDataType` attributes set at `PE_TYPE_FLOAT` and `PE_TYPE_FLOAT2` respectively. The particle system behavior element has the `ParticleLifeTime RenderDataType`, which has a data type of `PE_TYPE_FLOAT`. This is reflected by the item being available in the **source** drop-down list. There is currently no behavior element with a `RenderDataType` that supports `PE_TYPE_FLOAT2`, therefore the **target** drop-down list is empty and the user is unable to add a behavior animation with this configuration.

Creating a Particle System

Before you can use the **Particle Editor** to create a particle system, a PhyreEngine™ Particle Emitter Node object or PhyreEngine™ Visible Particle Emitter Node object must be present in the workspace (see the *PhyreEngine™ Core Library Reference*).

Note: For a detailed exercise that describes how to create an example particle system, see *Exercise 6: Set up a Particle System* in *Appendix A: Exercises*.

To create a particle system:

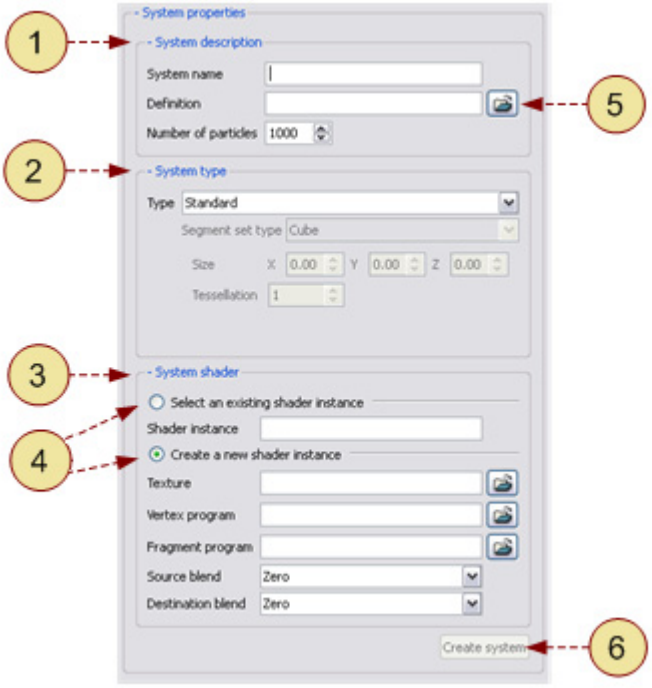
(1) Open a **Workspace Explorer Scene Graph** view and navigate to the database to which you wish to add the particle system.

(2) Right-click the root node object and do one of the following:

- To add a visible particle emitter node object, select **Add new child > Node > Visible Render Node > Visible Particle Emitter node** from the context menu.
- To add a particle emitter node object, select **Add new child > Node > Visible Render Node > Render Node > Particle Emitter node** from the context menu.

(3) Right-click the new node object and select **Create Particle System...** from the context menu. The **Particle Editor** opens.

Figure 80 Particle Editor – System Properties Group box



1. Particle system description properties
2. Particle system type properties
3. Particle system shader instance properties
4. Radio buttons to specify the shader instance mode
5. Particle system definition field
6. **Create system** button or **Update system** button (depending on mode)

(4) Define the properties for the new particle system in the **System properties** group box.

Fields with adjacent browse buttons can accept either files or existing PhyreEngine™ objects. To specify a file, drag the file from a file browser into the field. To specify a PhyreEngine™ object, select the object elsewhere in the application and drag it into the field.

All the fields are mandatory except the **Definition** field (item 5 in *Figure 80*). If you specify a definition file or object, PhyreStation reads in and displays the behavior elements and animation behaviors defined in that file or object. If a **Definition** is not specified, the **Particle Editor** displays default behavior and a new particle modifier behavior object is created.

(5) After you enter a valid set of particle system properties, the **Create system** button is enabled. Click the **Create system** button to create the system and populate the **Particle Editor** with the available system properties. If a system **Definition** was specified, the system elements and animations are shown. If no **Definition** was specified, these areas are empty.

When you create a new particle system, a PhyreEngine™ Particle Modifier Network Instance object is also created.

Editing a Particle System

To edit a particle system:

(1) Open a **Workspace Explorer – Objects** view and navigate to the required PhyreEngine™ Particle Modifier Network Instance object.

(2) Right-click the particle modifier network instance object and select **Edit Particle System...** from the context menu. The **Particle Editor** window is displayed.

(3) For details about editing the particle system properties, see *The Particle Editor* in *Chapter 15, Particle Editor*.

Exporting the Particle System

During the editing of a particle system, the behavior elements, behavior animations, and their attributes are continually committed to the workspace. At any point, you can export the PhyreEngine™ Particle System definition file by compiling the particle system. The compilation operation creates the definition file and compiles it to create the source code, ready to be inserted into your PhyreEngine™ project.

To compile the particle system, do one of the following:

- Open a **Workspace Explorer Database** view and select **Compile Particle Systems** from the PhyreEngine™ database object's context menu.
- Type in the `DBCompileParticleSystems` command, with parameters, into the **Command Window**.
- Execute a Lua script that uses the command `DBCompileParticleSystems`. See *Chapter 7, Scripts*.

15 The GUI

This chapter describes the important features of the PhyreStation user interface.

Overview

The PhyreStation main window contains three non-overlapping regions:

- Toolbar area
- Work area
- Status bar

Note: When using PhyreStation on a Microsoft platform, the GUI can vary depending on the style of the desktop chosen.

Toolbars

The toolbar area contains a set of toolbars.

Figure 81 PhyreStation Toolbars

File Toolbar

The **File** toolbar allows you to create, open, and save a workspace file.

The script button  allows you to locate and execute a script file. This button is disabled when a workspace is open.

Objects Toolbar

The **Objects** toolbar displays one button, the **Find** dialog button. The **Find** dialog allows you to search the workspace for PhyreEngine™ objects.

Workspace Toolbar

The workspace toolbar has two buttons. The  button is a toggle button to show or hide the dynamic **Properties Viewer.** The  button displays another **Workspace Explorer** window.

Trace Toolbar

The **Trace** toolbar has two toggle buttons. The  button displays the **Command** window. The  button displays the **Log** window.

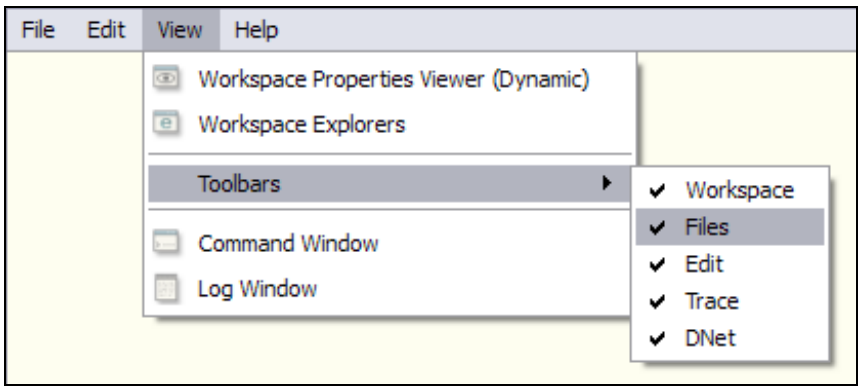
DNet Communication Toolbar

For more information about the **DNet Communication** toolbar, see *Chapter 10, DNet Communication*.

This toolbar is enabled only when a workspace is open.

To define which toolbars are displayed, select **View > Toolbars** from the main menu.

Figure 82 Toolbar Selection Menu



To reposition a toolbar or make it float in its own window, drag the toolbar handle.

Status Bar

The main window's status bar displays the following information:

- General application status messages.
- Command status messages (displayed during the execution of a command or a script). These messages are also sent to the **Log** window. See *The Log Window* in *Chapter 17, PhyreStation Log, User Preferences, and Error Handling* for details.

Work Area

The work area is the area within the main application window where other application (or frame) windows can appear.

You can move frame windows so that they are floating either inside the work area or anywhere on the desktop.

Docking Windows

You can dock a frame window at the top, bottom, left, or right of the work area if there is space for the window. A frame window can also be docked to another frame window to create a tab page. A floating frame window can be docked to the main window only if it is smaller than the available space.

To free a docked frame window, drag the window caption bar. To toggle a frame window between floating and docked, double-click the caption bar.

To place a frame window on, or very near, the work area border without it snapping to be docked, hold down the **Ctrl** key while dragging the window.

Moving Tab Pages

You can move a tab page out of a frame window by dragging the tab part of the page. You can then reposition the tab page as follows:

- Drop the tab page either inside or outside the work area. The tab page appears in its own frame window.
- Drop the tab page into any frame window in the same application instance. The position of the tab page in the frame window is indicated by a floating green arrow.

Depending on the current application and workspace preference settings, when a workspace is re-opened, the application can restore windows to their previous positions and sizes along with their content (this is not always possible – it depends on the PhyreEngine™ database's content).

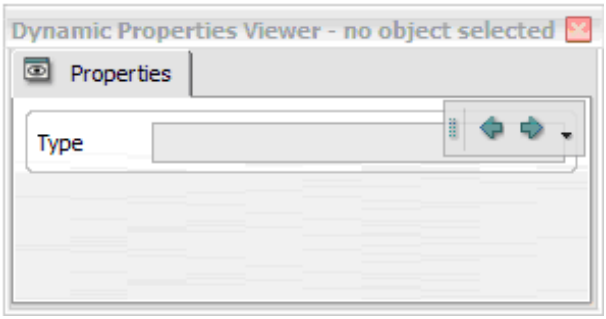
Window History

Most windows and views in PhyreStation operate an object or subject **History Navigation Toolbar**. This toolbar allows you to view, in chronological order, all the objects that were displayed in the window since it was opened.

In all windows and views except the object **Properties Viewer** (see *Editing Objects* in *Chapter 5, PhyreEngine™ Objects*), the history toolbar works as follows. When the history toolbar is first displayed, it is disabled. It remains disabled until you drag another object (of a suitable type) onto the view. When one or more other objects have been dropped onto the view, you can navigate to objects previously displayed in that view.

To display the **History Navigation Toolbar**, select **Show History** from the view's tab context menu.

Figure 83 History Navigation Toolbar



Use the **History Navigation Toolbar** as follows:

- To scroll through the objects that have been displayed in the window, click the left and right arrows. As you scroll through, each object and its properties are displayed in the viewer.
- Hold down the left mouse button on an arrow to display a list of all previous (left) or subsequent (right) objects. You can select an object from the list to display its properties again in the viewer.
- To show a list of all objects that have been displayed in the window since it was opened, click the drop-down arrow on the right of the toolbar. The objects are listed in the order in which they were displayed. You can select an object from the list to display its properties again in the viewer.
- To clear the history buffer, select **Clear list** from the history list.

Note:

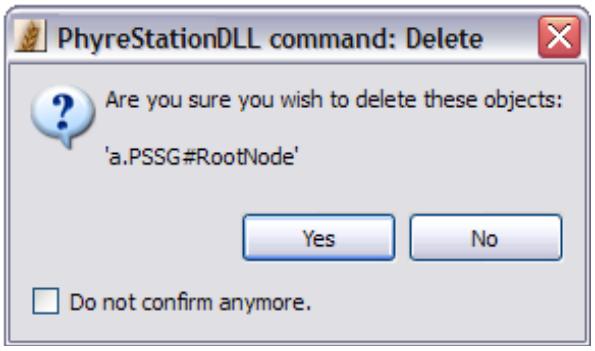
- When you delete an object from the workspace, the object is not deleted from the history buffer unless it is the current subject of the view. If you choose an object that no longer exists from the history buffer list, it does not update the view; it is just removed from the buffer.
- In some views, the **History Navigation Toolbar** may be 'drawn over' or disappear. This is especially true for views that draw a 3D scene. To resolve this, either move the toolbar to a part of the view that does cause the toolbar to be drawn over or click in the area where the toolbar was last seen; it will still operate.

Dialog Boxes

Most dialog boxes in PhyreStation are self-explanatory. However, the following dialogs demonstrate unique behaviour.

Confirm Deletion Dialog Box

Figure 84 Confirm Deletion Dialog Box



To prevent this dialog box from being displayed, check the **Do not confirm anymore** checkbox.

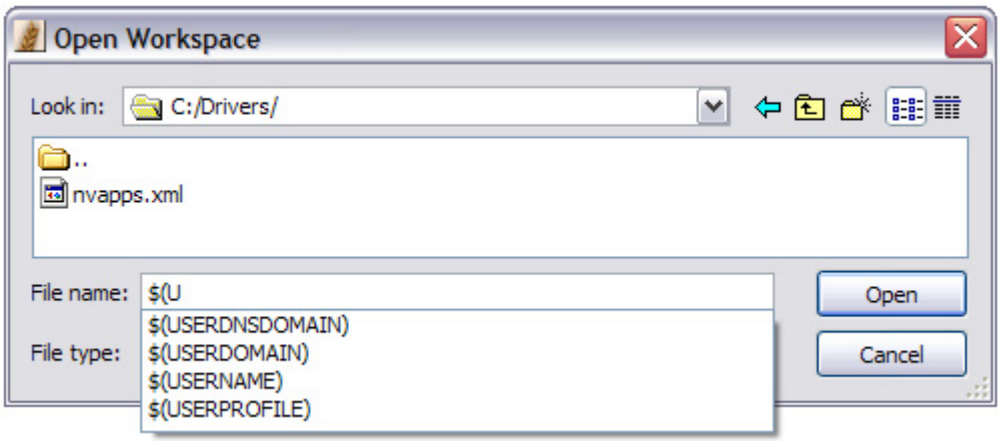
To re-enable the dialog box, select **Edit > User Preferences** and set the **Prompt for Deletion MessageBox** preference.

If you press the **shift** key and the **delete** key together, the dialog does not appear unless the deletion has other implications, for example if the object is referenced by other objects in the workspace.

File Dialog Box

PhyreStation uses file dialogs to locate a directory in order to load or save a file. These dialogs include a file auto-completer. You can enter the first characters of an environmental variable, for example $(U, or you can use wildcards, for example %USER%. The auto-completer lists all the matching available environmental variables.

Figure 85 File Completer



PhyreStation Resource Manager

Similarly to most GUI applications, PhyreStation needs to display icons and images. If the application cannot locate these resources, it replaces the missing resource with a default item. A grey square with a question mark is displayed where a GUI icon is missing. The resources for PhyreStation are located in the `[PhyreStation_root_directory]/Resources` directory.

16 Installation & Architecture

Installation

For all installation details, see the file `readme_e.txt` located in PhyreStation's installation root directory. This provides version information, details of any prerequisite software, lists of installed files, support information, and other installation information not included in this guide.

Licensing

You do not require a separate Nokia Qt software license to install and use the Qt run-time library that is supplied with PhyreStation.

For additional licensing information, see `[PhyreStation_root_directory]/Readme_e.txt`.

Architecture

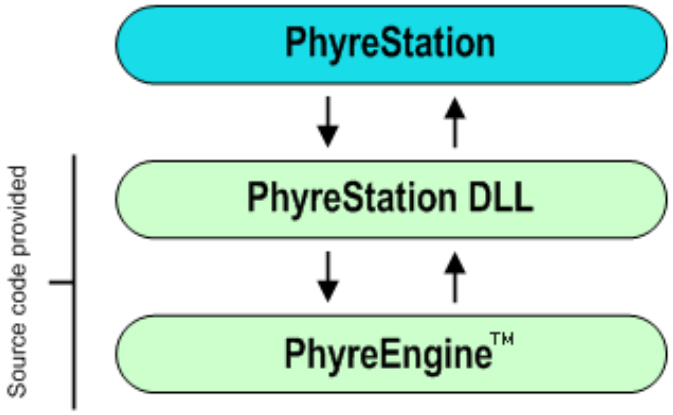
PhyreStation is supplied as a set of precompiled binary application files, including an executable and a number of run-time libraries. See `[PhyreStation_root_directory]/Readme_e.txt` for the full file list.

Note: PhyreStation source code is not available to third parties.

External Components

PhyreStation relies on a number of external components. The most important components are PhyreStationDLL and the PhyreEngine™ core library. The relationship between these components is shown in *Figure 86*.

Figure 86 Application Components



PhyreStationDLL

The PhyreEngine™ API and the PhyreStation GUI domains communicate via an intermediary component called 'PhyreStationDLL'. The PhyreStationDLL also provides the ability to extend PhyreStation's command set or handle your custom PhyreEngine™ objects.

Similar to the way in which third parties can modify and extend underlying PhyreEngine™ source code, you can also extend PhyreStation by modifying the PhyreStationDLL source code. See *Chapter 8, Customizing PhyreStation*.

PhyreEngine™

For details of the PhyreEngine™ graphics system, see the *PhyreEngine™ Programming Guide* and the *PhyreEngine™ Core Library Reference*.

Version Information

To obtain version information about PhyreStation and its components, select **Help > About PhyreStation...** from the main menu. See *About PhyreStation* in *Chapter 1, PhyreStation Overview*.

Application Instance ID

The application instance ID, as shown in the **About PhyreStation** dialog, identifies the particular instance of the application that is running. Normally, the instance ID is generated by the application when it starts. However, you can specify the ID when starting PhyreStation from the command line. The ID can be used by the command line parameters to target a specific instance of the application to carry out a new task.

17 PhyreStation Log, User Preferences, and Error Handling

This chapter describes PhyreStation's logging system, user preferences, error handling, and localization.

The Log Window

Overview

PhyreStation logs all user operations and feedback messages to the **Log** window. This information helps you to monitor the progress of command execution and perform general troubleshooting.

If PhyreStation terminates due to a serious system error, a copy of the **Log** window is saved to the file `[PhyreStation_root_directory]/PhyreStation.htm`. This file may help you to diagnose the root of the problem, and it may be requested by PhyreStation support if you refer the error to them.

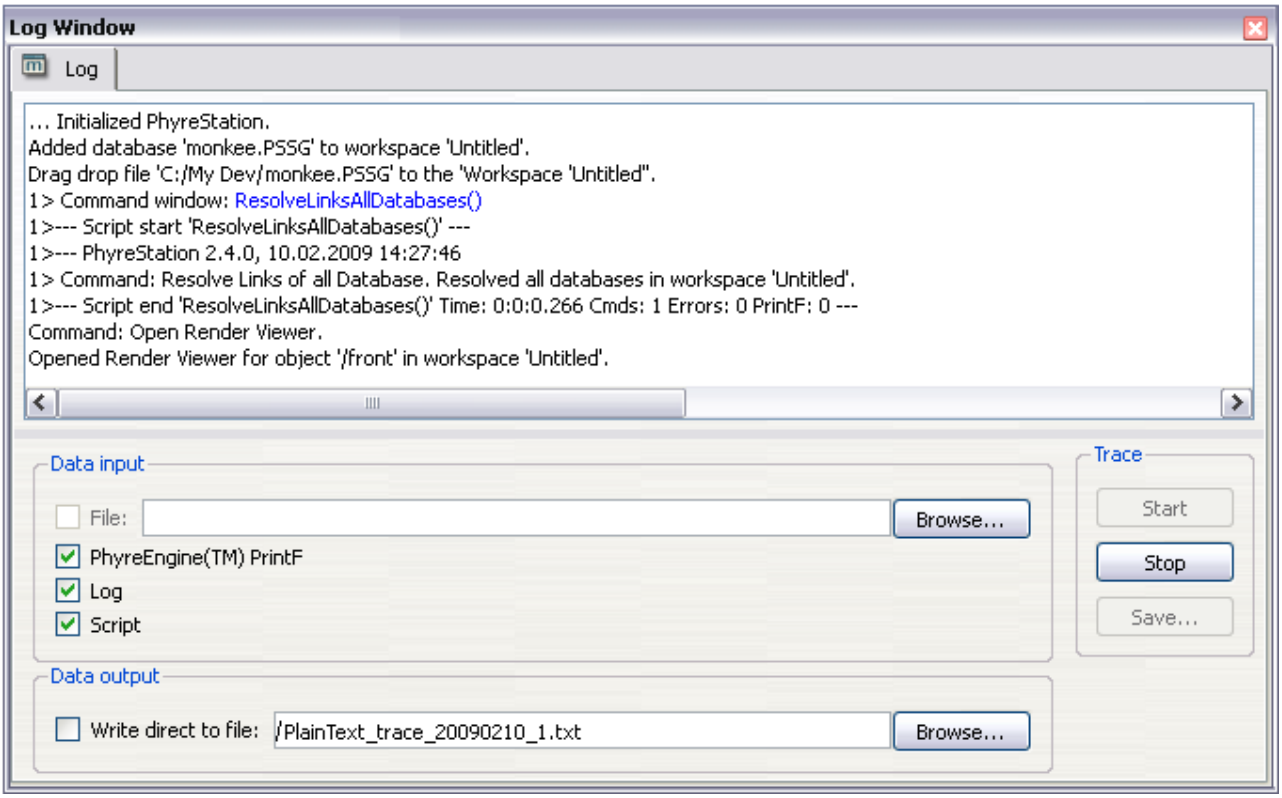
Displaying the Log Window

To open the **Log** window, select **View > Log Window...** from the main menu or click the **Log** window toolbar button.

Figure 87 Log Window Toolbar Button

The **Log** window contains a text area and an optional control group box. To hide or show the control group box, right-click on the **Log** tab and select the appropriate option from the context menu.

Figure 88 Log Window



Text in the **Log** window can be selected and copied.

Log Window Control Group Box

The **Log** window controls allow you to start and stop the log process, save log information, and filter the displayed text by selecting input sources.

Trace

- To start or resume the monitoring process and the display of information, click the **Start** button.
- To stop the monitoring process, click the **Stop** button.
- To save the currently displayed information to a file, click the **Save...** button.

> **Note:** It is only possible to save the information to the output stream file when the **Log** window has been stopped.

Data Input

To define the sources of input data to the log process, use the checkboxes in the **Data Input** group box.

- To display the contents of a specified file in real time, enter a filename in the **File** field and select the corresponding checkbox.
- To display PhyreEngine™ `PSSG_PRINTF` messages, select the **PSSG** checkbox.
- To display application and command messages, select the **Log** checkbox.
- To display script execution messages, select the **Script** checkbox.

By default, all checkboxes are selected except the **File** checkbox.

Data Output

To write the information shown in the **Log** window directly to a file during the monitoring process, select the **Write direct to file** checkbox. A data output file is created in the same folder as the application. The file name is generated automatically and includes the creation date and an increment index. By default, the **Write direct to file** checkbox is not selected.

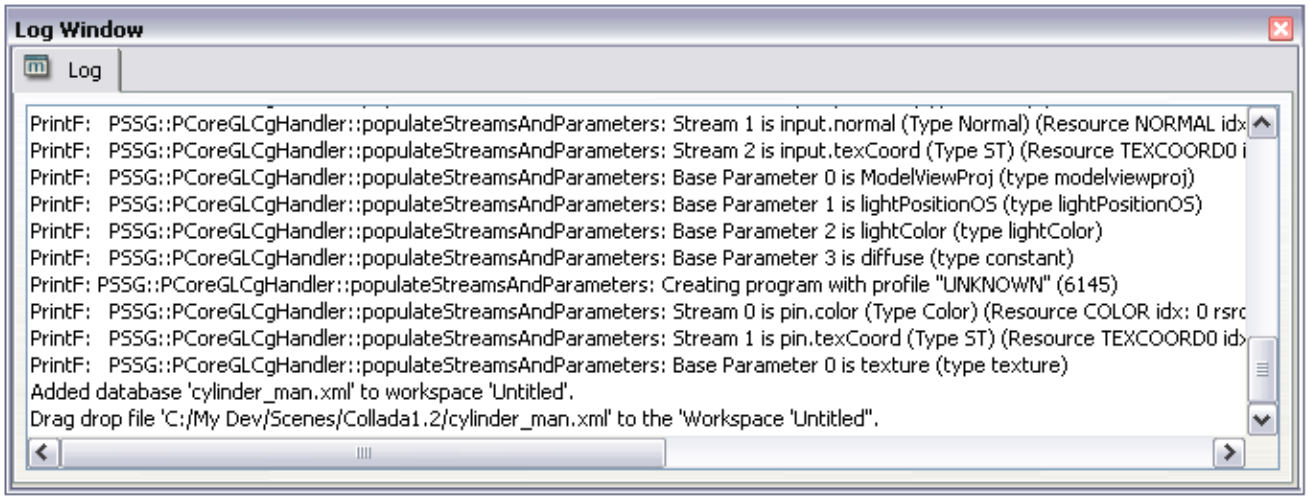
> **Note:** This data output file is not the same as the file specified by the **Save...** button.

The data output file is useful if you are experiencing crashes and are unable to obtain up-to-date trace information from the **Log** window. The data output operation is different from the `PhyreStation.htm` as it logs activity for the life of the application and only records Log information, not script or PhyreEngine™ information, even if the **PSSG** or **Script** checkboxes are selected

Feedback from PhyreEngine™ User Operations

The **Log** window logs all significant user operations and displays the feedback messages sent from any command that uses the PhyreStation command message feedback system. However, for reasons of speed and efficiency, this is not always practical. A developer using PhyreEngine™ may gain more detailed information on PhyreEngine™ operations by using PhyreEngine™ `PSSG_PRINTF`. The PhyreEngine™ `PSSG_PRINTF` messages can be seen by enabling the **PhyreEngine™ PrintF** checkbox. For example, this can be particularly useful when PhyreEngine™ is loading a COLLADA file; PhyreEngine™ reports any inconsistencies or incompatibilities as it parses the file. An example of the messages is shown in *Figure 89*.

Figure 89 Log Window PSSGPrintF



`PSSG_PRINTF` functionality is provided by PhyreEngine™ and is available for use anywhere in PhyreEngine™ or PhyreStationDLL code. You can redirect `PSSG_PRINTF` output to `stdout` by calling `PPhyreEngineDll::unregisterPhyreEnginePrintfCallback()` declared in `PhyreStationDLL.h` or `PhyreEngine::PhyreEngineResetPrintfCallback()` declared in `PhyreEngine.h`. This removes `PSSG_PRINTF` output from the **Log** window permanently.

Note: It is not possible for PhyreStation to act on messages sent to the user via the PhyreEngine™ `PSSG_PRINTF` input method.

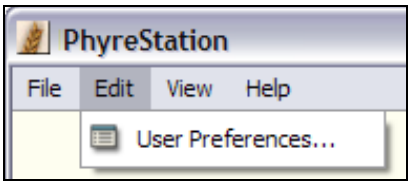
User Preferences

PhyreStation operates three categories of preferences:

- Preferences to control the way workspaces are saved and restored. See *Workspace Data* in *Chapter 4, Workspaces*.
- Application-wide preference settings, which are stored in the `prefs.xml` file.
- User settings, based on the currently loaded workspace session. When enabled, workspace preference settings override application settings.

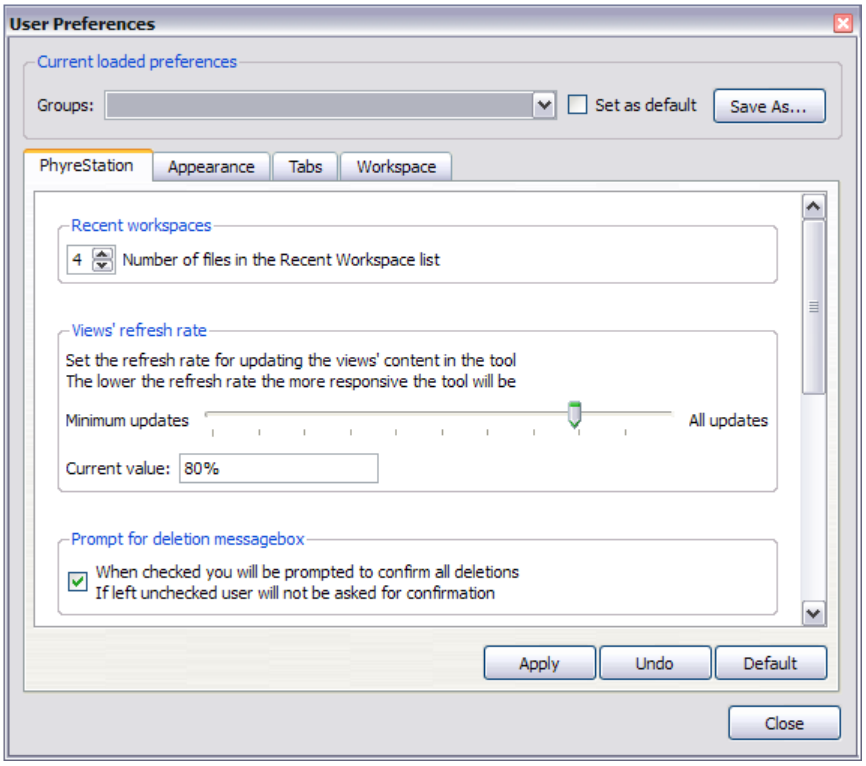
When you run PhyreStation for the first time, a preference file, `prefs.xml`, is created. You can edit, save, and load PhyreStation's user preferences in the **User Preferences** dialog box. To open this dialog, select **Edit > User Preferences...** from the main menu.

Figure 90 Edit Menu



The **User Preferences** dialog consists of the **Current loaded preferences** group box and a tabbed page area.

Figure 91 User Preferences Dialog Box



User Preferences Tabbed Pages

The tabbed pages display all user preferences for the PhyreStation application and its various components.

The **Default** button resets preferences in the current tab to their default values. Each preference has a default value that is 'safe' with all other PhyreStation user preferences.

If you attempt to apply an invalid value to one or more preferences, the **User Preferences** dialog displays a red feedback message in the relevant preferences' group box.

In some cases, your setting may be valid, but not recommended. In this case, a blue feedback message is displayed in the relevant preferences' group box.

Note:

- Resetting to the default preferences for one tab does not guarantee they will be compatible with other modified preferences. Feedback messages will warn you of any conflicts.
- Each tab has its own undo/redo history, and the history resets when switching between different preference files.

Current Loaded Preferences Group box

By default, a workspace can (if enabled) store the state of the user's preferences. Each workspace can specify and override the application settings.

You can use the **Current Loaded Preferences** group box to manage multiple sets of preference settings (called Groups) for a single workspace. This is useful if you require different PhyreStation configurations for different tasks in a workspace. At any time, you can specify a different preference group from the **Groups** drop-down list.

To enable this feature, set the **Remember and restore the workspace preference settings** checkbox to on in the **Workspace** tab page. The current preferences for the workspace or the Groups will be saved to the workspace file when it is closed.

To create a new group, change the preference settings as required and save them with a different name. The new group is added to the current set of groups in the workspace.

The **Set as default** checkbox allows you to specify which group is loaded by default when the workspace is next opened.

> **Note:** Any Groups specified in a workspace are cleared if Groups are not enabled; that is, if the **Remember and restore the workspace preference settings** checkbox is not set.

Using Application Settings from a Previous Version

When PhyreStation first starts, a dialog may appear asking whether you want to convert and use settings from a previous version. This dialog appears only when PhyreStation cannot find settings for the current version while finding other, older settings. Choose a version to use; PhyreStation will try to use those settings. It may not always be possible to convert and use the old settings exactly, so there may be some compromises.

Localization

Currently, PhyreStation is available only in an English language version. All displayed text is in English. This includes messages shown in the **Log** window, labeling in the main window's menu bar, text shown in dialog boxes, and so on.

However, it is still possible to use PhyreStation on systems using language and/or regional settings other than English. For example, you can load or save databases using Japanese file paths, and PhyreEngine™ objects may have Japanese names.

Localization Support for Commands

Localization support for commands is made possible by communicating UTF8 format strings through PhyreStationDLL. This is achieved by encoding all strings in PhyreStationDLL to UTF8 format (rather than ASCII, for example).

(UTF8 is a Unicode-type encoding format involving multibyte sequences with mark-up information. It is an efficient way of storing non-ASCII characters, for example, multilingual characters. UTF8 is a superset of ASCII, so that text saved to UTF8 is readable by legacy applications requiring ASCII, for instance.)

PhyreStation assumes that each string it receives from PhyreStationDLL (that is, const char arrays and std::string objects) is encoded in UTF8, before converting it to a format suitable for displaying. Although you are free to encode your own private or internal strings in whichever way you prefer, you must encode strings for viewing or editing in PhyreStation to UTF8 before passing them to PhyreStationDLL. To ensure that strings are always correctly displayed, it is recommended that you work exclusively with UTF8.

Because UTF8 is backwardly compatible with ASCII, Western European and U.S. users already developing under ASCII do not need to do anything; PhyreStation will display ASCII strings correctly.

Non-ASCII (for example, Japanese character) users must convert their strings to UTF8 before communicating through PhyreStationDLL. This means that:

- Source code files containing hard-coded strings must be saved with UTF8 encoding. (From MS .net, source files can be saved from the **Save File As...** dialog box: click the drop-down menu to the right of the **Save** button, choose **Save with Encoding...**, and then select the **Unicode (UTF-8 without signature) – Codepage 65001** option.)
- PhyreStation Workspace XML files must be saved with supported encoding. PhyreStation automatically saves these files correctly. The text must support saving with UTF8 encoding (for example, MS .net).
- PhyreStation database HIER files must be saved with supported encoding. PhyreStation automatically saves these files correctly. The text must support saving with UTF8 encoding (for example, MS .net).

Error Handling

Software errors can be categorized into two types:

- User errors are caused by the user entering invalid data or performing invalid operations, for example, attempting to write to a read-only file or entering an invalid script command. In PhyreStation, user errors are generally handled with appropriate messages in the **Log** window and/or warning message boxes. Occasionally, the **PhyreStation Error** dialog box is shown.
- System errors are usually more serious and indicate a problem with the underlying code or system, for example encountering a bug or running out of system resources. All system errors are displayed in the **PhyreStation Error** dialog box.

All errors are also directed to the `[PhyreStation_root_directory]/ PhyreStation.htm` file.

For PhyreStation Support

For PhyreStation support contact details, see `[PhyreStation_root_directory]/Readme_e.txt`.

Appendix A: Exercises

This appendix presents exercises to assist users in learning how to use various features of PhyreStation.

Overview

These exercises teach you the very basic operations you will probably use in any work with PhyreStation. It is recommended that you perform these exercises in sequence as some exercises are a continuation of the preceding exercises.

- Exercise 1: Basic Workspace and Database Exercise. Create a new workspace, create a database, add an existing database and resolve database links.
- Exercise 2: Database Links. Break a database link, unload and load a broken database, reestablish and resolve links.
- Exercise 3: View, Find, and Edit PhyreEngine™ Objects. Objects exercise. View, find, and edit objects in different ways.
- Exercise 4: Create a Script File and Associate with a Workspace. Scripts exercise. Create a script, associate it with a workspace, and execute the script in different ways.
- Exercise 5: Consolidate PhyreEngine™ Databases.
- Exercise 6: Set up a Particle System. Create a particle system.

Exercise 1: Basic Workspace and Database Exercise

This exercise introduces you to the PhyreStation workspace.

Step	Instructions	Additional Information
1. Run PhyreStation.	Locate the PhyreStation.exe file using the OS file explorer and double click on it. After a few moments, PhyreStation starts, displaying a default layout.	Close the current workspace if one is currently open. This will probably be the **Blank Workspace,** which is created if PhyreStation's preference for it is enabled. To find the preference, select **Edit > User Preferences...** from PhyreStation's main menu and click on the **PhyreStation** tab. See *The Current Workspace* in *Chapter 3, Workspaces*.
2. Create a new workspace.	Select **File > Workspace > New...** from the main menu, and then enter a name and a location for your new workspace. When the workspace is saved, a file is created in the specified location.	A workspace is similar to a project file. You can add or create objects in the workspace such as PhyreEngine™ databases and PhyreEngine™ objects. Scripts can also be associated with a workspace. Only one workspace can be open at any time. See *Chapter 3, Workspaces*.
3. Create a PhyreEngine™ database.	Choose **File > Database > New database...** from the main menu, and enter a name and a location for your new database. When the database is saved, a file is created in the specified location.	PhyreEngine™ has two types of database; `.pssg` and `.heir`. `.pssg` is a binary type file; `.heir` is an XML text file. See *Chapter 4, Databases*.

Step	Instructions	Additional Information
4. Open a Workspace Explorer – Databases view.	Click on the  icon on PhyreStation's toolbar to open a **Workspace Explorer** view. Right click on the view's white space to display the context menu, and select **Databases**. The new database you have just created is displayed in the **Workspace Explorer – Databases** view as a top-level database.	A **Workspace Explorer** allows you to navigate the current workspace and select objects. A workspace has several filters: 1. **Databases** 2. **Scene – Graph** 3. **Objects** 4. **Custom Groups** 5. **Scripts** Choosing a filter determines which objects in the workspace are displayed and how the objects are displayed. See *The Workspace Explorer* in *Chapter 3, Workspaces*.
5. Display Workspace Explorer filter properties.	To display a filter view's properties, select **Edit > User Preferences...** from PhyreStation's main menu, click on the **Workspace** tab, and select the **Enable workspace explorer filter properties** checkbox.	The **Workspace Explorer** filters' properties display extra information about an object such as whether one object is used by another (linked or referenced). See *Workspace Explorer Filter Properties* in *Chapter 3, Workspaces*.
6. Add a PhyreEngine™ database.	Find the PhyreEngine™ `monkee.pssg` database file and drag it on to the workspace. The `monkee.pssg` database is normally located in the PhyreEngine™ **Scenes** directory under the **Monkee** directory.	A database can be added to the current workspace by: • File menu • Database filter view white space context menu • Drag dropping a database file from outside of PhyreStation A COLLADA file (`.dae` or `.xml`) can also be added to the current workspace. PhyreEngine™ converts the file to a binary `.pssg` type file. See *Adding an Existing Database* in *Chapter 4, Databases*.
7. Resolve a PhyreEngine™ database.	In the **Workspace Explorer – Databases** view, select `monkee.pssg`, a top-level database. Right-click to show the context menu, and then select the command **Resolve Links**. The icon for `monkee.pssg` changes to a tick () to show that its links were successfully resolved. Underneath the top-level database are the immediate linked databases that PhyreEngine™ has determined are required to gather additional necessary resources.	A context menu for a selected item(s) displays a set of commands or actions that can be carried out on the item. Custom commands can be defined by you in the PhyreStationDLL. See *Adding PhyreEngine™ Object Commands to PhyreStation* in *Chapter 8, Customizing PhyreStation*. The **Resolve Links** command instructs PhyreEngine™ to find all the objects needed to render the scene. It also tells PhyreStation which PhyreEngine™ objects to add to the workspace. Note: work cannot be carried out on link databases. See *Link-Resolving a Database* in *Chapter 4, Databases*.

Step	Instructions	Additional Information
8. The Log window.	The **Log** window displays the status of the tool or recently executed commands. If the **Log** window is not already displayed, click on the  icon on the toolbar.	The text displayed in the **Log** window is collected from general application activity, commands executed, and special PhyreEngine™ PSSGPrintF messages. The text displayed in the **Log** window is also output to a PhyreStation.html file. See *The Log Window* in *Chapter 17, PhyreStation Log, User Preferences, and Error Handling*.
9. Exercise complete. Go to exercise 2.		

Exercise 2: Database Links

This exercise introduces you to handling databases.

As well as holding data or resources in databases, PhyreEngine™ has the important ability to share resources amongst a set of databases. This is automatically initiated when a PhyreEngine™ object makes a reference to a required object in another database. When PhyreEngine™ resolves a database, it checks whether these references are valid. If PhyreEngine™ finds one or more 'links' that cannot be matched, the resolution of the database has failed.

PhyreStation can display a top-level database's references to the first level of linked databases. Any failed resource links are displayed in the **Workspace Explorer – Databases** view as a database with a  icon.

Step	Instructions	Additional Information
1. Work through Exercise 1.		
2. Break a database's link.	Locate PhyreEngine™'s **Scenes\Monkee** directory. Select `monkeychest.pssg` and change its name temporarily.	Because you changed the name of the database file, PhyreEngine™ will not be able to locate a file called `monkeychest.pssg` when the database is next resolved.
3. Display a database with broken resource links.	In the **Workspace Explorer – Databases** view, select `monkee.pssg`. 1. Select **Unload** from its context menu. 2. Select **Load** from the context menu. 3. Select **Resolve Links** from the context menu. Notice that the icon for `monkeychest.pssg` has now changed to a  icon.	PhyreEngine™ has been instructed to release the resources required for `monkee.pssg` (also releasing the resources in the linked databases) and reacquire them. However, it cannot now locate everything because a database has been renamed. See *Unloading and Reloading a Database* and *Link-Resolving a Database* in *Chapter 4, Databases*.

Step	Instructions	Additional Information
4. Display a database with no broken resource links.	Change the renamed database back to its original name (`monkeychest.pssg`). Repeat step 3. The database icons should now revert back to a  icon.	
5. Exercise complete.		

Exercise 3: View, Find, and Edit PhyreEngine™ Objects

This exercise describes how to find PhyreEngine™ objects in the PhyreStation workspace and how to edit the attributes of a PhyreEngine™ object.

Step	Instructions	Additional Information
1. Work through exercise 1.		
2. Open a Workspace Explorer – Objects view.	Click on the  icon on PhyreStation's tool bar to open a **Workspace Explorer**. Right-click on the view's white space to bring up the context menu, and select **Objects**.	The **Workspace Explorer – Objects view** displays all PhyreEngine™ objects belonging to all databases that have been resolved successfully. Each category hides a list of objects and the database they belong too. See *Browsing Objects* in *Chapter 5, PhyreEngine™ Objects*.
3. Change how the objects are displayed.	Right-click on the view's white space to bring up the context menu, and select **View By Database**.	If no white space is available in the view, select any category. Select the same category again while holding down the **Ctrl** key, then right-click to bring up the view's context menu.
4. Open a Workspace Explorer Scene-Graph view.	Click on the  icon on PhyreStation's tool bar to open another **Workspace Explorer**. Right-click on the view's white space to bring up the context menu, and select **Scene-Graph**.	Only objects derived from a node object type are shown in the **Workspace Explorer – Scene-Graph** view. This view displays a subset of the objects shown in the **Objects** view. Any node that is a child of a root node is rendered in the PhyreEngine™ scene.
5. Open a Render view.	Some databases can contain camera nodes. Expand the database `monkee.PSSG` to display the scene root node **/root**. Expand the root node to display a node hierarchy. Select the camera node **/front**. This will be the subject or primary object for view. Select **Open Render Viewer...** from its context menu. From the whitespace context menu (blue background) choose the following; **Object selection in this view**, **Bounding Boxes** and **Highlight Selection**.	The **Render View** displays the current scene graph from the selected camera's point of view, in this case the **/front** camera position and its orientation in the scene. You may have to rotate and move the camera to see anything useful. Use the **Alt** key and the mouse buttons while moving the mouse. Do not click in the bounding rectangle area of the monkey. See *Render Viewer* in *Chapter 11, Viewers*.

Step	Instructions	Additional Information
6. Open a Properties Viewer.	Select the **/front** object and select **Open Properties window...** from the context menu.	The **Properties Viewer** displays the object's attributes. Some of the attributes' values can be changed directly in the GUI. Black fields are editable fields. Changing an attribute's value has an immediate effect. See *Editing Objects* in *Chapter 5, PhyreEngine™ Objects*.
7. Change an object's attribute value.	Click on the  icon on the tool bar to open a **Command Window**. Type the following into the command line edit field as follows: `SetObjectProperty( "monkee.pssg#/front", "FarPlane", 0, 101.0 )` Hit the return key. The **Render Viewer** will be updated to show the change.	Some commands are available from both the context menu and by typing in an equivalent script command. The **SetObjectProperty()** command is only available as a script command. See *Methods of Executing Commands* in *Chapter 6, PhyreStation Commands*. If you look at the **Log** window while changing an object's attribute, you will notice the same command being executed as the one you typed in the **Command** window. See *The Log Window* in *Chapter 17, PhyreStation Log, User Preferences, and Error Handling*. The tool's **Help Viewer** contains all the commands, their syntax, and descriptive information. See *The Help Viewer* in *Chapter 1, PhyreStation Overview*.
8. Locate a node object using the Find dialog.	Click the  icon on PhyreStation's tool bar to open the **Find** dialog. In the dialog, select **Workspace** view and **Selection.** Type ***poly*** into the **Find** edit box and hit the return key. The **`monkee.PSSG#/polySurface459`** skin node object should be found.	PhyreStation searches all objects contained in the workspace to find a match or partial match. See *Finding Objects in the Workspace* in *Chapter 5, PhyreEngine™ Objects*.
9. Select an object in a view.	Select **`monkee.PSSG#/polySurface459`** in the **Find** dialog. Note that the same object is also highlighted in other views. Select **/joint1** in the **Scene-Graph** view, and then select the **`polySurface459`** object in the **Objects** `monkeychest.pssg` view. The selection changes in all views.	You may have to scroll a view and expand the items in the view to reveal the selected object. A light blue selected item in a view indicates that a child of the item is hidden. Expand the item to reveal the grey selected item. See *Selection from Multiple Windows* in *Chapter 5, PhyreEngine™ Objects*.
10. Drag and drop objects.	Find the node whose name ends in **`/joint34`**. Select it in the **Scene-Graph** view. While holding down the left mouse button, drag the node to the root node **/root**, and then release the mouse button. Select **Reparent Node** from the pop up menu. You can drag and drop across views.	To see the effect of the drag drop operation, click the animation play button  in the **Render View**. Click the  button to stop the animation. Note that not all PhyreEngine™ databases have animation data so these buttons may be greyed out.

Step	Instructions	Additional Information
11. Create a new object.	Right-click `monkee.PSSG` in the **Workspace Explorer – Databases** view and select **Add new object > Node > Light Node** from the context menu. Open a **Properties View** for the existing light object **/pointLight1**. Copy the translation vector values to the new light object. Change the values a little so the light is placed in a different position in the scene. Reparent the new light object to the **/root** node. Select **Display Light** in the **Render View** to see the new light node.	In the **Scene-Graph** view, you will see the new object under the **`monkee.PSSG`** database.
12. Delete an object.	Select the camera object **/front** and choose **Delete** from the context menu. Confirm the deletion. The object is removed from the workspace and PhyreEngine™ removes it from the database. The workspace and the database are now marked as 'dirty' (indicated by the asterisk in the title bar and a tick in the view's properties under **Modified)**.	Linked objects cannot be deleted using the GUI (look at the views' properties). However, they can be deleted using the **`DeleteObject()`** script command. Care must be taken when doing this as it may cause PhyreEngine™ to become unstable.
13. Close the workspace.	Select **File > Workspaces > Close** from PhyreStation's menu bar.	Before the workspace is closed, PhyreStation displays a confirmation dialog listing any databases that still need to be saved. If the databases are not saved, changes will be lost.
14. Exercise complete.		

Exercise 4: Create a Script File and Associate with a Workspace

This exercise introduces you to handling PhyreStation scripts. A script is text file with the file extension `.lua`. The language used for scripting is Lua. Most of the commands registered with PhyreStation can be used in a script (that is, they are registered for operating in a script). Most of these commands can be found in the PhyreStationDLL.

This exercise demonstrates how to execute a script from PhyreStation's various modes of operation; execute a script silently (no GUI – use in batch file), execute a script and start PhyreStation, run PhyreStation and then execute a script.

Step	Instructions	Additional Information
1. Work through exercises 1 and 3.		
2. Create a script.	Open any text editor program such as Microsoft Notepad. Type in the following: **`print( PSSGVersion())`**. Save the text file with the name **MyScript**, with the file extension `.lua` replacing `.txt` if necessary.	The statement **`print()`** is a native Lua command. The command **`PSSGVersion()`** is a command defined in the PhyreStationDLL. It is also registered in the PhyreStationDLL so that the command can operate from within a script.

Step	Instructions	Additional Information
3. Run PhyreStation.	Run PhyreStation either by clicking on its icon or by executing from the OS command line (with no parameters specified). If no a current workspace is open, for example, the default 'blank' workspace, create a new workspace.	
4. Open a Workspace Explorer Scene-Graph view.	Click on the  icon on PhyreStation's tool bar to open a **Workspace Explorer**. Right-click on the view's white space to display a context menu, and then select **Scripts** (Scripts view).	The **Workspace Explorer – Scripts** view lists all the scripts associated with the current workspace. A script may be shared with many workspaces so it is recommended that you store your script files in a central location. A script can call another script.
5. Associate a script with a workspace.	Locate the script file you have just created and drag it on to the script view. The script is now associated with the current workspace and will remain so until removed from the workspace.	When the workspace is saved, the script has a path relative to the workspace stored in the workspace file. See *Associating Scripts with a Workspace* in *Chapter 7, Scripts*.
6. Execute a script from the Script view.	Open a **Log** window. Select the script file **MyScript** in the Scripts view and select **Run** from its context menu. In the script's report to the **Log** window, you should see the version of PhyreEngine™ that PhyreStation is using printed out.	If you also open the **Command** window you will see the results of the executed script. Most commands that are executed in a script report give either a success or error message. The command **`PSSGVersion()`** does not give a message as it is unnecessary. See *Executing Scripts* in *Chapter 7, Scripts*.
7. Save the workspace script association.	From the menu bar choose **File > Workspaces > Save As...**. Type **`MyWorkspace.xml`** for the file name and click **Save**.	
8. Execute a script from the Command window.	Locate the script file you just created and drag it on to the **Command Window** edit box. The script file is automatically executed once the drop is complete.	You can use the method described in this step to execute scripts that are not associated with a workspace.
9. Execute PhyreStation along with a script from an OS command line window.	Open an OS command line. Type the following: **`PhyreStation.exe -script="<path>\MyScript.lua" -scriptLog="<path>\MyScriptLog.txt" -scriptQuit="true"`** and hit return. PhyreStation executes the script and quits immediately. You can find the report of the execution in the script log file.	When PhyreStation runs in batch mode, no GUI appears. PhyreStation must be in your system's path or you must explicitly type the path to the executable. Batch mode is useful when you need to automate a set of PhyreStation tasks amongst other workflow batch processing jobs.

Step	Instructions	Additional Information
10. Execute PhyreStation along with a script from the OS command line window and allow it to continue to run after the script has finished – script GUI mode.	On the OS command line, type the text from step 9 but change the last parameter to **`-scriptQuit="false"`** and hit Return. PhyreStation executes as before but will now not quit and startup as if you started it from its icon.	Executing scripts from the command line is useful when you need to carry out tasks such as loading a database when PhyreStation starts up. Note a script log file is not produced in this mode. See *Executing Scripts from the Command Line* in *Chapter 7, Scripts*.
11. Execute a script that is contained within a workspace file from the OS command line.	On the OS command line type the following: **`PhyreStation.exe -workspace="<path>\MyWorkspace.xml" -script="MyScript.lua" -scriptQuit="false"`** and hit return. The same script as in steps 9 and 10 is executed.	Some script information is reported to PhyreStation's log file **PhyreStation.html**.
12. Exercise complete.		

Exercise 5: Consolidate PhyreEngine™ Databases

This exercise demonstrates a basic workflow example most developers will typically come across when working with PhyreEngine™. The typical situation is there are a set of 'raw' PhyreEngine™ databases which need to be combined into one database suitable for the target platform. The resultant database contains only those assets that are relevant with any duplicate objects removed. The final database is also compressed.

It is assumed you have completed all the exercises prior before working through this exercise.

Step	Instructions	Additional Information
1. Run PhyreStation.		
2. Create a new workspace with the 'raw' databases to process.	Locate PhyreEngine™ scenes example Monkee. Add database `monkee.pssg` to the current workspace. Select the database and choose **Resolve Links** from its context menu.	
3. Create a new script and add it to the workspace.	Create a new PhyreStation script file in the same directory as the new workspace file and call it **`process.lua`**. Drag the new script file on to a **Workspace Explorer – Scripts** view to add it to the workspace.	See the script code in *Example Script – process.lua* at the end of this exercise. Note that some of the commands in the script are included only to update PhyreStation so that you can see the results of this exercise.
4. Save the workspace.	Save the current workspace as **`MyWorkspace.xml`**.	
5. Exit PhyreStation.		

Step	Instructions	Additional Information
6. Execute a script associated with a workspace file from the OS command line.	On the OS command line type the following: **`PhyreStation.exe -workspace= "<path>\MyWorkspace.xml" script= "process.lua" -scriptQuit="true" -scriptLog="scriptLog.txt"`**	
7. Embed the script process into a automation batch file.	It is often necessary to embed the execution of a script into larger job or workflow. The **PhyreEngine™ Game Templates** are typical examples. For instance, the **`UnstoppableSpeedSmash`** example contains a batch file **`updatePSSGFiles.bat`** in its **ArtWork** directory. That batch file executes a further three batch files, each of which execute PhyreStation with different scripts to operate the workflow necessary to process the assets required for the example.	**PhyreEngine™ Game Templates** are examples of how to achieve the best from PhyreEngine™ for a particular genre of game. They also demonstrate typical workflows for developing software using PhyreEngine™.
8. Exercise complete.		

Example Script – process.lua

```
-- Create a new monkee.PSSG database from an existing monkee.PSSG but with the
-- minimal assets to render it in a scene.

-- It will contain enough to render the monkee and be visible, lit, but will -
- not contain any animation data.

--

srcDb = "monkee.PSSG"
destDb = "monkeeDst.PSSG"
PSSG = os.getenv("SCE_PSSG")
outDir = "C:\\<Your path>"

--
-- Resolve database monkee.PSSG in the current workspace
--
ResolveLinks( srcDb )

--
-- Acquire all the databases PhyreEngine(TM) has loaded
-- remove the PhyreEngine(TM) internal database and source database
--
print( "Acquire databases..." )
local dbList = { GetDatabases() }
local dbListTextures = dbList
for i,db in dbList
do
 if db == "PSSGInternalDatabase" or db == srcDb
 then
       dbListTextures[ i ] = nil
 end
end

---
--- Create a new database
---
print( "Create new database..." )
```

```
NewDatabase( outDir, destDb )

---
--- Copy over the objects we need then delete objects
--- we still do not need
---
print( "Coping and deleting objects..." )
CloneObjectDeep( srcDb .. "#/root", destDb )
DeleteObject( destDb .. "#/top" )
DeleteObject( destDb .. "#/side" )
DeleteObject( destDb .. "#/front" )
local roots = { GetObjectsOfType( destDb, "ROOTNODE" ) }
for i,r in roots
do
 RemoveUnusedLeafNodes( r )
end
ResolveLinks( destDb )  -- For PhyreStation, update with new objects
---
--- Copy to the new database all the external textures
---
local newTextureList = {}
for i,a in dbListTextures
do
 local textureList = { GetObjectsOfType( a, "TEXTURE" ) }
 for j,b in textureList
 do
       local result,newTexture = CloneObject( b, destDb )
       print( "Copying " .. b .. ". New name is " .. newTexture )
       table.insert( newTextureList, j, newTexture )
 end
end

---
--- Update Shader Instance to use the internal textures
--- (they will be wrongly assigned, but this is an example case only)
---
local shaderList = { GetObjectsOfType( destDb, "SHADERINSTANCE" ) }
for i,a in shaderList
do
 local newTexture = newTextureList[ i ]
 print( "Update attribute for " .. a .. " texture " .. newTexture )
 SetObjectProperty( a, "Parameters", 0, newTexture )
end

---
--- Save the new database
---
SaveDatabaseCompressed( destDb )
```

Exercise 6: Set up a Particle System

This exercise demonstrates how to set up a PhyreEngine™ Particle System ready to be inserted and run in an instance of PhyreEngine™.

It is assumed that you have completed all the previous exercises before beginning this exercise.

Step	Instructions	Additional Information
1. Run PhyreStation.		
2. Create a particles database.	Using your current workspace or a new workspace, create a new database and call it `particles.pssg`.	
3. Create a Visible Particle Emitter node.	Select the `particles.pssg` database and create a root node. In the **Scene-Graph** view, select the root node and add a new **Visible Particle Emitter Node** object.	The **Visible Particle Emitter Node** object can be found under **Visible Render Node**. A **Particle Emitter Node** can also be used in its place.
4. Create a Particle System.	Select the **Visible Particle Emitter Node** object and select **Create Particle System...** from the context menu. This will open the **Particle Editor** window.	
5. Edit required properties.	Edit the properties necessary to enable the particle system. In the **System properties > System description** group box, edit the following fields: **System name**: type **colorWheelSystem** **Definition**: locate file **`colorWheel.xml`** **System Type**: select **Packetized** In the **System properties > System shader** group box edit the following fields: • **Texture:** locate file **`colorWheel.bmp`** • **Vertex program**: locate file **`quadParticleRenderVert.cg`** • **Fragment program**: locate file **`particleRenderFrag.cg`** • **Source blend** : select **Source Alpha** • **Destination blend** : select **One** When the above is entered correctly, the **Create System** button is enabled. Click this button. The Particle Editor displays the new particle system.	The properties not edited here are default values which can be left as they are. The files needed for this exercise can be found in PhyreEngine™ in the advanced sample **particleUberModifier**. When this step is complete, you can proceed to edit the other enabled properties in the editor window. Any edits have an immediate effect.
6. Compile the Particle System.	Right-click the `particles.pssg` database and choose **Compile Particle Systems** from its context menu. This command generates a separate definition file and the necessary source files containing an optimized particle modifier for the particle system.	This step prepares the new particle system. Note that it may be necessary to run one of the **Compile Cg Programs** commands on the database, depending on the particle system's target platform, before compiling the particle system.

Step	Instructions	Additional Information
7. Save the particle system.	Select the `particles.pssg` database and choose **SaveDatabase** from the context menu.	
8. Use the particles database with PhyreEngine™.	The new particle systems database is ready to be used. Locate the particle systems database and copy it to where it may be used by PhyreEngine™.	The separate source files generated from the **Compile Particle Systems** command also need to be included in your project and compiled. The generated optimized particle modifier then needs to be registered with PhyreEngine™.
9. Exercise complete.		

Glossary

This glossary presents technical acronyms and other terminology that is specific to PhyreStation, or that may be unfamiliar to some readers.

Term	Definition
Action	A command carried out by the user, usually from a context menu. See *Command*.
Animation Editor	A specialized form of a *Waveform Editor*.
API	Application Programming Interface.
Application instance ID	A number that uniquely identifies every running instance of the PhyreStation on the same platform. It can be used to target a specific instance to carry out work.
Application preferences	*User preferences* for the PhyreStation's application wide settings. See *workspace preferences*.
Art asset	An item or collection of items, of art data.
Auto-completer	A pop-up menu displaying possible, complete alternatives based on what has been typed into a line edit box. When you select an item in the list, the item is entered as though you had typed it in.
Batch command	A *command* that internally spawns one or more secondary commands or a command executed with multiple objects selected in the GUI.
Batch mode	The mode in which PhyreStation runs when an instance of PhyreStation is executed from the *command line interface* in batch mode with a script specified. A main application window is not shown. This mode of operation is specified by a PhyreStation command line parameter.
Behavior animation	A Particle System's animation behavior.
Broken database	A *PhyreEngine™ database* that has not been successfully resolved.
COLLADA	File format for art assets.
Command	Every non-trivial PhyreStation operation involves the execution of a command. Commands may originate either from the *PhyreStationDLL* component, or from within PhyreStation itself. PhyreStation invokes commands by various means.
Command line interface	The command line interface (cli) or command line window is an OS window in which you execute system commands.
Command line interpreter	The **Command** window's line edit box where script or Lua commands can be entered to be executed.
Command line parameter	An extra parameter that can be specified when using PhyreStation from the *command line interface*.
Command window	A PhyreStation window used to execution Lua or script enabled commands.
Custom Groups	A user defined object set that is assigned a label and a subset of objects in the current workspace.
Custom Groups filter	A **Workspace Explorer filter** view that displays the current workspace's user defined *Custom Groups*.
Connector	See *node connector*.
Current workspace	The workspace currently on display in PhyreStation.
Database	See *PhyreEngine™ database*.
Database filter	A **Workspace Explorer filter** view that displays the *PhyreEngine™ databases* contained within the current workspace.
DNet	PhyreEngine™ Debug utility library and communication protocol.
DNet Communication toolbar	A PhyreStation application toolbar that is used to communicate with a host over a network.
Filter view	The **Workspace Explorer** *window* can display different views of the objects in the workspace.
Find Dialog	A PhyreStation dialog that allows the user to search for objects in the workspace.

Term	Definition
Frame window	A window that contains one or more *Tab* windows. It may *float* or be docked to the *work area*.
Ghost	See *Group Ghost object*.
Graph	Refers to one of PhyreStation's three *work area* graph editors: the **Target Blender Editor**, the **Modifier Network Editor**, or the **Modifier Network Instance Editor**. Also known as a *network*.
Graph node	A pictorial node in a *graph*. Compatible graph nodes can be connected together through their *node connectors*.
Group	A single user-defined label that may, or may not, have object representations assigned to it. See *Custom Groups*.
Group Ghost object	A representative object of an object that does not currently exist in the workspace.
Group object	A representative object of a real object in the workspace. Group objects are shown in the *Custom Groups filter* view.
GUI	Graphical User Interface.
GUI mode	The mode in which PhyreStation runs when an instance of PhyreStation is executed from the *command line interface* with no parameters specified, or when the user has clicked on the PhyreStation icon. The main application window is shown.
Help window	PhyreStation online help window.
Highlighting	Refers to an object in the workspace that is 'selected' and visible in one or more windows. It also refers to an object that is shown to be part of a selection of objects in the workspace.
Internal database	See *PhyreEngine™ internal database*.
Link	A connection or reference between two *PhyreEngine™ objects* from different *PhyreEngine™ databases*.
Linked database	A *PhyreEngine™ database* that is referenced by another *PhyreEngine™ database* and is shown in the *Database filter view*.
Top-level database	A *PhyreEngine™ database* that is added to the current *workspace* and can be seen in the *Database filter view*.
Link-resolving	A *PhyreEngine™ database* is said to have been *link-resolved* when all its referenced elements – including all its dependencies – have been successfully verified. A database is *link-unresolved* if its links failed for some reason.
Log window	A PhyreStation window that logs important user operations and *command* feedback messages.
Lua	High-level programming language, embedded into PhyreStation to provide automation of *script commands*.
Main window	PhyreStation's main application window.
Menu bar	The main window's menu bar is the horizontal region underneath the *title bar*, containing menus for performing tasks.
Modifier Network Editor	An editor for graphs/networks associated with *PhyreEngine™ modifier network* objects.
Modifier Network Instance Editor	An editor for graphs/networks associated with *PhyreEngine™ modifier network instance* objects.
Network	See *graph*.
Node	See either *graph node* or *PhyreEngine™ node object*.
Node connector	An interface on a *graph node* used for connecting to other graph nodes.
Object	An object in the workspace; normally a representation of a *PhyreEngine™ object*.
Object filter	A **Workspace Explorer** filter view that displays all the *PhyreEngine™ objects* of any resolved databases in the currently loaded workspace.
OS	Operating System.

Term	Definition
Particle Editor	A PhyreStation window used to create or edit PhyreEngine™ Particle Systems.
Particle System	A PhyreEngine™ Particle System; a collection of processes that can either update or calculate new data from a given particle stream fed into other particle streams.
Preference	See *user preference*.
Primary object	A *tab* window's or editor's (depending on its function) *subject* object. Changing properties of the subject may cause *subjects* of *secondary object windows* to change too.
Properties Viewer window	A PhyreStation window that displays the attributes of an object in the workspace. Some of the attributes may be editable.
PhyreEngine™	High-level graphics rendering system, including a scene-graph and other components.
PhyreEngine™ database	Stored collection of *PhyreEngine™ objects*.
PhyreEngine™ internal database	A *PhyreEngine™ database* that is always present in every workspace. Contains mandatory *PhyreEngine™ objects* and default objects typically required by all development projects.
PhyreEngine™ node object	Type of *PhyreEngine™ object*. Node objects can be connected together in a hierarchical fashion to represent the physical or logical structure of a graphical scene. Specialized node types can be derived to perform specific functions when rendering the scene, such as adding lights or geometry to the scene. See *Scene-Graph*.
PhyreEngine™ object	Representation of an *art asset*, and the basic building block of PhyreEngine™. Multiple objects can be stored together in a *PhyreEngine™ database*.
PhyreEngine™ object link ID	PhyreEngine™ assigns a unique name for every PhyreEngine™ object in a *PhyreEngine™ database*. The PhyreEngine™ link ID is concatenated to the *PhyreEngine™ database* name as follows: *databaseName*#*objectName*.
PhyreEngine™ object type	A category of a *PhyreEngine™ object*.
PhyreStation	A GUI tool for managing *PhyreEngine™ objects*, *PhyreEngine™ databases*, using automation.
PhyreStationDLL	A PhyreStation component through which programmers can add or modify *PhyreEngine™ object* commands or custom *PhyreEngine™ objects*.
PhyreStation error	An error that occurs in the application code while PhyreStation is running. PhyreStation has two categories of errors: non-fatal (PhyreStation can recover) and fatal (PhyreStation tries to exit gracefully).
Qt	*Trolltech Qt* is a cross-platform GUI toolkit library; it is embedded into PhyreStation.
RenderDataType	PhyreEngine™ name given to the type of result requested when specifying a *Particle System's behavior element*.
Render Viewer	A **Render Viewer** is a window that displays the rendered scene from the point of view of the camera node object subject for that window.
Resolved	See *link-resolved*.
Scene-graph	A hierarchical assembly of *PhyreEngine™ object node types* and geometry representing the physical or logical structure of a graphical scene. See ***Scene-Graph** filter*.
Scene-Graph filter	A **Workspace Explorer** filter view that displays all the *PhyreEngine™ scene-graphs* in the current workspace. See *Scene-Graph*.
Script	See *script file*.
Script command	A command registered for PhyreStation script operation via the *PhyreStationDLL*, providing PhyreStation automation. Script commands can either be executed directly from the **Command** window, or incorporated in to a *script file*.
Script file	A text file containing a sequence of either Lua or PhyreStation *script commands*.

Term	Definition
Script filter	A **Workspace Explorer** filter view that displays all the scripts associated with the current *Workspace* session
Secondary object window/secondary view	A *tab* window (depending on its function) whose *subject* has a dependency on another (*primary)* object elsewhere in the workspace. If an attribute of the *primary* object changes, the *secondary* object may also reflect a change due to its relationship with the *primary* object's change.
Segment Set Viewer	A PhyreStation window that renders a *PhyreEngine™ segment set* object.
Shader Group Viewer	A PhyreStation window that displays information about a single *PhyreEngine™ Shader* Group object and allows some limited ability to change the parameters and attributes of the shader process.
Shader Instance Viewer	A PhyreStation window for viewing a single *PhyreEngine™ shader instance* object.
Shader Instances Viewer	A PhyreStation window for viewing all the *PhyreEngine™ shader instance* objects in all of the loaded and successfully resolved *PhyreEngine™ databases* in the current workspace.
Shader Program Viewer	A PhyreStation window for viewing the *PhyreEngine™ Shader Program* object and its program. Provides some ability to change the parameters and attributes of the shader process shader program.
Script GUI mode	The mode in which PhyreStation runs when an instance of PhyreStation is executed from the *command line interface* with a script specified. A main application window is shown after the script has finished . This mode of operation is specified by a PhyreStation command line parameter.
Status bar	The horizontal region at the bottom of the *main window*, providing state information and other feedback.
Static mode	A PhyreStation *Tab* window whose *subject* remains unchanged when the user selects further workspace objects.
Subject	Some windows such as the **Render Viewer** window have a workspace object as a subject, normally a *PhyreEngine™ object*. The window displays information about that subject or related objects. The subject is normally chosen by the user from a *filter view*.
System animation	A Particle System's animation behavior item.
System element	A Particle System's behavior element.
Target Blender Editor	A PhyreStation editor for graphs/networks associated with *PhyreEngine™ animation target blender* objects.
Texture Viewer	A PhyreStation window that displays an image of a *PhyreEngine™ texture* object.
Title bar	The main window's title bar is that horizontal region at the top of the *main window*, usually displaying the name of the current workspace.
Toolbar	The main window's toolbar is that horizontal region – found underneath the *menu bar* – comprising toolbar buttons for performing common tasks.
User preference	A configuration setting for some features of the PhyreStation. User preferences can be modified by users and persist between PhyreStation sessions.
Untitled workspace	An empty workspace, which is opened automatically when PhyreStation starts up (if the *user preference* is enabled).
View filter	Selected in a *Workspace Explorer* from a list of available filters. It determines how certain information is displayed.
Waveform Editor	A PhyreStation editor for displaying and editing data that has waveform characteristics, such as *PhyreEngine™ animation channel* objects.
White-space	The area of a window that is void of any content.
Work area	PhyreStation's main window region within which *Frame windows* or *Toolbars* can be docked.

Term	Definition
Workspace	A PhyreStation working environment that can contain a set of *PhyreEngine™ databases,* associated *script* files, PhyreStation Groups or other or other workspace 'objects'.
Workspace Explorer window	A window that allows a *view filter* to be chosen from a list of available filters. It allows the user to manage the type of information they wish to see, to select information or objects, and then carry out tasks on the recent selection.
Workspace file	XML file that stores *workspace* information and user preferences.
Workspace preferences	*User preferences* for the PhyreStation's *workspace* settings. See *Application preferences*.