

# RenderWare Studio

## User Guide

---

Version 2.0.1

Published 07 September 2004  
© 2002-2004 Criterion Software Limited. All rights reserved.

# Contents

---

<b>What's new</b>	8
Version 2.0.1	9
Version 2.0	11
Upgrading from version 1.2	20
Database conversion errors	25
Database format differences	26
Retaining Workspace customizations	27
Known issues	30
Version 1.2.2	31
Version 1.2.1	32
Version 1.2	34
Version 1.12	46
Version 1.11	50
Version 1.1	51
Version 1.01	58
Version 1.0	60
Version 1.0 RC1	62
<b>Introduction</b>	
What is RenderWare Studio?	66
What does RenderWare Studio do?	69
Basic concepts	71
Game database	73
Game database XML files	76
Customization	80
System requirements	82
<b>Getting started</b>	85
First steps	86
Requesting a permanent license	95
Getting started with VBScript	96
<b>Designing games</b>	
Creating and editing a project	97
Creating a new project	98
Importing assets	101
Asset properties	102
RF3 files	104
Controlling the camera	108
Camera flight controls	111
Moving the camera in orbit mode	113
Setting camera sensitivity and frame rate	115
Using keys to control flight	117
Creating and working with entities	119
Selecting entities	

Moving, rotating, and scaling entities	123
Aligning with existing entities	125
Creating an entity from a template	127
Creating a spline curve	128
Creating box and trigger volumes	130
Light mapping	132
Tutorial 1: Light mapping user interface	133
Tutorial 2: Preparing assets for light mapping	141
Tutorial 3: Creating and using entity lights	145
Tutorial 4: Area lights	149
Tutorial 5: Light Map Preview Settings window	151
Tutorial 6: Display options window	154
Setting Design View and Workspace options	158
Display options	160
Snapping at intervals when moving or rotating	162
Organizing your game into folders	166
Global folder	168
Changing levels in a stand-alone game	169
Sharing attributes	170
Sharing entities and folders	174
Importing from another project	175
Deleting objects vs. removing references	177
Adding sound to your game	178
Adding screens to your game	179
Sequencing entity actions	180
Saving target data to files	183
Developing for multiple platforms	184
Creating a custom platform	185
Managing files with NXN alienbrain	187
alienbrain Features in Workspace	188
NXN Database Explorer window	192
Validating the game database	193
Viewing your game on a target	197
Creating or editing a target connection	199
How targets get asset data	204
Creating a custom action for a target	208
Starting the GameCube Comms Server	210
Starting the Game Framework	212
Connecting to a target	213
Connecting to a GameCube SN-TDEV	214
Connecting to a GameCube DDH or GDEV via a broadband adapter	215
Running the Game Framework under AtWinMon (PlayStation 2)	216
Debugging your game	219
Graphing variables over time	220
Viewing data streams with NetTest	224
Packaging and distributing your game	

Distributing a PlayStation 2 game on CD	235
Packaging a game that contains custom resource handlers	237
<b>Workspace user interface</b>	238
Main menu	244
Toolbars	251
Keyboard shortcuts	257
Windows	262
Assets	264
Asset Light Map	267
Attribute Shares	269
Attributes	270
Behaviors	273
Design View	276
Design View Toolbar	278
Event Map	284
Events	286
Game Explorer	288
Help (for behaviors)	291
Light Attributes	292
Light Map Preview	294
Log windows	297
Object Information	299
Preview	300
Profiling Tools	305
RF3 Asset Templates	309
RF3 Project Templates	311
Scene Light Map	313
Search Results	315
Sequencer	317
Stream Viewer	325
Target Files	329
Targets	330
Templates	334
Saving layouts	335
<b>Developing behaviors</b>	336
Tutorial	337
Lesson 1. Creating a behavior	341
Lesson 2. Adding an action	344
Lesson 3. Editing behavior attributes	346
Lesson 4. Creating an event	348
Lesson 5. Receiving an event	350
Lesson 6. Sending an event	352
Lesson 7. Debug macros	359
Lesson 8. Inheritance	364
Lesson 9. Creating a behavior without using CAtomicPtr	366
<b>Customizing the attribute editor</b>	

Attribute editor architecture	370
Attribute editor script integration	371
Classes, commands, entities and attributes	372
Attribute editor controls	373
Control interfaces	374
Attribute editor control wizard	376
Attribute editor data	378
Saving attribute data	381
Creating new attribute editor controls	382
Lesson 1. Creating an ActiveX attribute control - basic	385
Lesson 2. Creating a List box attribute control - advanced	396
HTML attribute editor controls	405
Adapting the game framework	407
Building	408
Core architecture	413
Tips	421
<b>Customizing Workspace</b>	<b>423</b>
Understanding Workspace	424
Application source files	426
The RenderWareStudio directory	428
Understanding .Layout files	430
What Enterprise Host does	432
How Workspace works	434
Startup processing	435
Normal operation	437
Mapping Workspace windows to Author objects	440
Application settings file	443
Programming Workspace objects	445
Additional initialization	446
Selecting game entities	447
Broadcasting method calls	450
Global methods and constants	453
Typical customizations	454
Tutorial: Creating a Workspace layout	455
Creating the "Simple" layout	456
Tutorial: Making a new menu bar and toolbar	462
Creating a new menu bar object	463
Adding items to a menu bar object	464
Installing a new menu bar in Workspace	467
Complementing a menu bar with a toolbar	469
Merging menus at run time	472
Customizing the Preview window	474
Creating ActiveX controls for Workspace	477
Tutorial: Creating a custom ActiveX control	481
Using the RenderWare Studio ActiveX Control Wizard	

Building and debugging the control	483
Adding a command button	486
Handling the button click event	488
Tutorial: Adding an ActiveX control to Workspace	490
Installing a control in the Workspace user interface	491
Debugging controls in Workspace	493
Firing events from an ActiveX control	494
Handling ActiveX control events in Workspace	497
Tutorial: Closer interaction with an ActiveX control	500
Creating and coding the edit control	501
Deeper integration with RenderWare Studio	504
Opening arbitrary asset types for editing	508
Adding to the RenderWare Studio settings file	512
<b>Example projects</b>	<b>514</b>
Alpha sort	515
Animation	516
Area triggers	517
Audio	519
Camera look-at-point	520
Dynamic lights	522
Environment map	523
First-person player	524
FX motion blur	525
FX particle spray	527
FX pixel shader	533
FX vertex shader	534
G3 pipelines	535
Maestro	537
Multiple levels	538
Replicator FX	539
RF3 files	542
Shared attributes	543
Sky dome	544
Spline camera	545
Split screen	546
Split screen to view PVS	547
<b>Glossary</b>	<b>548</b>
<b>Trademarks</b>	<b>552</b>
<b>Contact us</b>	

# Release notes

---

The release notes summarize the major changes in each release:

Release name	Date
<a href="#">Version 2.0.1</a> (p.9) ← <b>this release</b>	July 2004
<a href="#">Version 2.0</a> (p.11)	April 2004
<a href="#">Version 1.2.2</a> (p.31)	November 2003
<a href="#">Version 1.2.1</a> (p.32)	October 2003
<a href="#">Version 1.2</a> (p.34)	August 2003
<a href="#">Version 1.12</a> (p.46)	April 2003
<a href="#">Version 1.11</a> (p.50)	February 2003
<a href="#">Version 1.1</a> (p.51)	November 2002
<a href="#">Version 1.01</a> (p.58)	6 September 2002
<a href="#">Version 1.0</a> (p.60)	23 August 2002
<a href="#">Version 1.0 Release Candidate 1 (RC1)</a> (p.62)	1 August 2002

For more detailed information, view the change log:

- Click **Start ▶ Programs ▶ RenderWare Studio ▶ Documentation ▶ Changelogs**

# Release notes for Version 2.0.1

---

This release of RenderWare Studio resolves a number of issues in areas including memory usage, using RF3 files, and the RenderWare Studio command line. It also incorporates several requested features, such as Maya-style flight controls in the Design View, and an enhancement to integration with NXN alienbrain.

## Installation

The installation program will update an existing installation of RenderWare Studio 2.0. Alternatively, you can install a complete version of RenderWare Studio 2.0.1 from scratch.

## New features

### **Maya-like flight controls in Workspace**

Workspace's Design View now supports flight controls like those in Alias's Maya software, as well as those in Discreet's 3Ds Max. For example:

- Alt+left mouse button** selects orbit mode
- Alt+middle mouse button** pans the camera
- Alt+right mouse button** dollies the camera

### **Workspace announces current alienbrain change set at startup**

At startup, RenderWare Studio Workspace now displays the name of the current alienbrain change set in its Version Control Log. If you're working with different change sets for different applications, this serves as a reminder that you may need to switch.

Optionally, you can configure Workspace to display a dialog allowing you to switch change sets whenever you open an alienbrain-managed project.

### **New Game Production Manager documentation**

The documentation that ships with RenderWare Studio 2.0.1 includes a new tutorial on the Game Production Manager and the writing of new build rules.

## Bugfixes

### **Excessive Workspace memory requirements**

Under some circumstances, RenderWare Studio 2.0 loaded the same texture many times, resulting in high memory usage. This has been fixed in 2.0.1.

### **Failure to import a level from another project**

If the entity/asset GUIDs of a level being imported matched those of the open project in Workspace 2.0, importing that level would fail. This has been fixed in 2.0.1.

### **Workspace hangs if /close is specified on the command line**

Using /close to terminate Workspace automatically after running from the command line caused RenderWare Studio 2.0 to hang. This has been fixed in 2.0.1.

**RF3 files cause the console to crash**

In RenderWare Studio 2.0, using RF3 files in your projects could cause the console to crash under some circumstances. RF3 files work correctly in 2.0.1.

**Using direct set functions causes the console to crash**

In some situations, setting attribute values directly causes the console to crash in RenderWare Studio 2.0. This has been fixed in 2.0.1.

**Sequencer corrupts character-type attributes**

In Workspace 2.0, if an entity taking part in a sequence has a behavior with an attribute of type `RwsChar`, the value of that attribute could become corrupted when the project containing it is saved to and loaded from disk. This has been fixed in 2.0.1.

**Sequenced entities not properly displayed in Workspace attribute controls**

When using the sequencer's keyframe editor to configure attributes, incomplete information is passed to some Workspace attribute controls in RenderWare Studio 2.0, resulting in display problems. This has been fixed in 2.0.1.

# Release notes for Version 2.0

---

## Overview

RenderWare Studio 2.0 incorporates some significant changes over previous versions to improve usability, multi-user database modifications, customization, and performance. Several new Workspace tools have been added, such as the Sequencer for animation and cut-scene authoring. These release notes describe the new features, changes, and information for users upgrading from previous versions.

## Genre Pack 1 - First Person Shooter

Genre Pack 1 provides a pre-built game engine for FPS-type games, so a game designer can immediately start populating a FPS game with game objects, behaviors, and logic.

Game authoring within Workspace exploits RenderWare Studio features such as events, and also includes a trigger editor for creating and modifying game trigger volumes. The game engine provides behaviors for main player, non-player characters, weapons, pickups, health, armour, ammo etc. Effects behaviors include coronas, lights, and animations.

Source code is provided for the game engine, incorporating base classes for customers to build their own behaviors into the system.

The game engine supplied in this Genre Pack also uses RenderWare Audio, RenderWare AI, and RenderWare Physics.

The GP1 installer for this is separate to the actual RenderWare Studio install, and is optional.

The Genre Pack 1 has not been tested on the Playstation Debug machine.

## Workspace

### New tools

#### **Sequencer**

The new [sequencer tool](#) (p.317) is a general-purpose editor for animating sets of entity attributes over time. You can use this to create cut scenes, for example, or as a way of creating easily playable coordinated sequences within the main game play.

#### **Trigger editor**

A new general-purpose [volume editor](#) (p.130) lets you interactively create arbitrary box volumes in the Design View. While Genre Pack 1 interprets these as trigger volumes, their usage is completely customizable by the game. Different textures can be applied to the volumes, and build rules modified to extract the relevant information for the game.

#### **Light mapper**

A new tool that ties together several elements of RenderWare Studio allowing you to set up and preview light maps in the Workspace Design View. This includes the ability to create and position lights, or to tag

materials as area light sources and parameterise as appropriate.

### **Network test**

The [NetTest tool](#) (p.224) displays a text description of the contents of a RenderWare stream. NetTest can either:

- Listen to an IP port, and describe a stream as it is being sent to a target. This is especially useful for debugging target connections, so that you can see exactly what data is being sent to the target across the network.
- or
- Open an existing stream file (.rws or .stream).

### **Plugin data editor and stream viewer**

The stream viewer allows you to look inside RenderWare streams, and attach custom editors to handle the different types of plugin data. Edits made to the stream data are preserved in the asset's XML file, and so can easily be reapplied should the asset be re-exported from its original source.

## **New window management user interface**

A new [window management mechanism](#) (p.238) gives users greater control and flexibility over the placement and grouping of windows in Workspace. The main difference that existing users will notice is a new look and arrangement to Workspace.

Windows can be grouped into arbitrary sets of “stackers”, and can also be toggled between tabbed views and floating groups. Any tab can be moved to another window by simply dragging and dropping. Stackers can be collapsed to save screen space.

Several task-oriented layouts are provided; each Workspace user can modify these, create their own layouts, and distribute layouts to other users.

## **Preview control source code**

The source code for the Workspace [Preview control](#) (p.300) is supplied as an optional component (you need to select this when installing RenderWare Studio). As well as allowing on-site customization, this code provides examples of integrating with Workspace and using such features as drag and drop within custom controls.

## **Diagnostic tool enhancements**

The Diagnostics tool (available from the Windows Start menu) has been enhanced to also show version information for all the various DLLs required by the software. The aim of this is to enable easier confirmation that upgrades and patches to individual components have actually been applied.

## **Shared attribute handling**

Two features have been added to improve the usage of shared attributes:

- You can move and copy attributes between shares using drag and drop or copy and paste keys.
- The attribute editor displays an icon next to attributes that belong to shares. This makes it much more obvious that changing a shared value will affect

several entities rather than just the selected set.

In addition, a related feature that applies to attributes in general is that the attribute editor displays an icon over the attributes that have been set for that entity.

## Active target toolbar

A new feature of Workspace takes advantage of the observation that most users only use a single target in day to day use. This can now be tagged as the “active target”. Toolbar shortcuts to frequent operations such as launch, build, and connect operate on the active target.

## Customizable HTML-based help for behaviors

Allows programmers to provide game designers with documentation for behaviors, or for game designers to share their own hints and tips with other designers.

For example, if your game has a behavior named `CMyBehavior`, and you save an HTML file named `CMyBehavior.htm` in the `help` folder under your project's source root folder, then you can display that HTML file in the new Workspace [Behavior Help window](#) (p.291) by right-clicking the behavior and selecting [View Help](#).

Genre Pack 1 supplies an example of this behavior help.

## Create template by dragging entity

A popular feature request from existing users: you can now create templates (or template folders) by dragging entities (or folders) from the Game Explorer window to the Templates window.

## Design View enhancements

New features in the Design View window:

### Filter assets by platform

To show only the assets for selected platforms, select **Options > Display**, and then set the **Platform Filtering** options.

### Vary lighting

To select which objects are lit in the Design View, and the brightness of the lighting, select **Options > Display**, and then set the **Lighting** options.

### Material pick mode

To edit the material properties of an entity, click the Pick Materials toolbar button , and then click the entity in the Design View window. The

material properties dialog appears in a floating window.

### Adjustable “locate” size

Previously, when you pressed **F3** (or selected **Selection > Locate**) to move the camera to an entity, the camera was sometimes too close, or too far, from the entity. You can now set this distance: select **Options > Flight**, and then set the **Zoom Factor**.

### Adjustable mouse zoom

Previously, if you selected an entity and then used the mouse wheel to

zoom in and out, the magnification steps could be too great, and you could zoom past the entity too quickly. To fix this, the magnification steps now get progressively smaller as the camera zooms in. You can also adjust the sensitivity of the mouse wheel: select **Options ▶ Flight**.

### **Freeze entities to avoid unwanted selections**

To make it easier to select only the entities you want, you can now “freeze” entities, so that they can no longer be picked: right-click entities in the Design View or Game View window, and then select **Freeze**.

### **Cut, copy, and paste entities**

To create a copy of one or more selected entities, right-click the selection in the Design View, select **Copy**, point where you want the new copy to appear, and then right-click again and select **Paste**. The new entities have the same name as the originals, with “Copy of ” prepended. (Instead of right-clicking, you can press **Ctrl+C** to copy and **Ctrl+V** to paste.)

Similarly, you can cut an entity, and then paste it elsewhere (this has the same effect as simply moving the entity).

### **Spline editor: adjustable node size and offset**

New Spline Options dialog (**Options ▶ Spline**) allows you to set the offset position of new nodes, and the display size of nodes, when creating a [spline curve](#) (p.128).

## **Preview window displays worlds, sectors, normals**

The [Preview window](#) (p.300) has been extended to show world objects (.sp files) in addition to the atomic, clump, and animation objects. The world view also enables visualization of sectors and surface normals.

As described elsewhere, the source code for this control is also available as part of this release.

## **Game database**

### **Load on demand brings speed improvements and scalability**

Users with large [databases](#) (p.76) will gain performance benefits from the new load-on-demand mechanism that loads only the parts of a database that a user is interested in. The mechanism has been designed to provide good scaling behavior as games move towards next-generation consoles with very large data sets.

Existing users who have their own custom controls that access the game database should refer to the section on upgrading existing customizations.

## **RF3 enhancements**

When you open a project, the Message Log window now tells you if you have not set the RF3 project template path. The Build Log window shows more information about the progress of RF3 exports.

The RenderWare Exporter RF3 project and asset template editors are now available as windows in Workspace.

## **Persistent properties**

This new feature allows you to attach user-defined property data to any database

node. This allows you to add arbitrary data to assets, for example. The data can optionally be persisted. The data is accessed through an API, also available at the automation (script) level.

## PurgeDeleted tool

The PurgeDeleted tool, supplied in `Studio\Programs\Tools\PurgeDeleted`, deletes database XML files left over from previously [deleted objects](#) (p.177).

## NXN alienbrain

RenderWare Studio 2.0 requires [NXN alienbrain](#) (p.187) version 7.

### **Change set functionality included**

You can now use change sets for atomic multi-change submissions

### **Multiple checkouts**

You can now check out more than one object at a time or select single objects to check in or revert.

### **alienbrain properties and version history**

Context menu options allow you to view the alienbrain properties and version history directly from Workspace.

### **Custom columns**

You can customize the alienbrain Manager client window to show the Name, Description, and Type tags of a [game database](#) (p.76) XML file.

### **Script-based integration**

The interface between RenderWare Studio and alienbrain is now managed by script within the REN file.

### **Removal of “local edit” feature**

The provision of multiple checkout support means that the **Local Edit** feature, introduced in RenderWare Studio 1.2, is redundant and has been removed.

## alienbrain Database Explorer window

This [window](#) (p.192) is a scaled-down version of the NXN manager client window that is hosted by RenderWare Studio. It allows you to perform all the version control operations on your game database that you would in the full alienbrain client manager window from inside Workspace.

## Game Production Manager (GPM)

### **Command line**

You can run the GPM from a command line.

### **Improved feedback and error reporting**

During a build, a missing behavior will now appear in the Build Log as an error and stop the build, preventing the incorrect stream from getting to a console. You can also use the [database validator](#) (p.193) to find missing behaviors for an entity.

### **Build activity logs**

Running the GPM now produces build activity logs; these are XML files that record which rules were run, which target files were updated, and why

(which dependent files were out-of-date).

When building the game data stream, these logs are saved as `BuildActivity.xml` in the `Build Output` folder under your project.

When exporting [RF3 files](#) (p.104), these logs are saved as `BuildActivity_asset_name.xml`, in the same folder as the exported stream file.

### Rule format changes

Minor additions to the build rule XML syntax, including the attributes `alwaysbuild` and `strictvalidation`.

## Windows Explorer plugin

Adds a [RenderWare Studio Name column](#) (p.76) to the Windows Explorer [Details](#) view, allowing you to more easily identify game database `guid.xml` files.

## Customization

### Split REN files

The powerful customization mechanism that enables users to add their own controls and logic to Workspace has been extended. Before RenderWare Studio 2.0, the REN file was a monolithic entity containing the definition of Workspace's user interface and its corresponding VBScript code. In version 2.0, this information is split into a set of [application source files](#) (p.426) that are managed together as a project.

### REN projects

In version 2.0, you still open a REN file in Enterprise Author when you want to customize Workspace. However, that file now has the same role as the `.vcproj` file in Visual C++, or the `.rwstudio` file in RenderWare Studio. Each control in Workspace's user interface has its own, dedicated configuration files. As well as providing greater modularization and easy browsing, this scheme simplifies deployment of customizations and handling of updates.

### Scripts can be JScript

As supplied, the script code for the Workspace user interface is contained in a set of VBScript (`.vbs`) files. In RenderWare Studio 2.0, however, you can choose to use JScript (`.js`) files for your user interface code. The files that handle alienbrain integration use JScript, for example.

## API changes

### Extension object: add .dff files into Design View

The new `LoadStream` method allows you to add `.dff` files into the Workspace Design View window.

### RWSPersist API toolkit removed

The RWSPersist API toolkit has been removed, since loading and saving of the RenderWare Studio project is now handled by the core API.

```
RWS_API RWSID RWSProjectLoad(const RWSChar* const szLocation);
RWS_API void RWSProjectSave(RWSID GameID, const RWSChar* const szLocation,
```

```
RWSSaveFlags Flags);
RWS_API RWSID RWSProjectImport(const RWSChar* const szProjectFilename,
const RWSChar* const szFolderGUID);
RWS_API RWSID RWSLoad(const RWSChar* const UID, RWSIDType Type);
RWS_API void RWSSave(RWSID ID);
```

**Note:** The last two items allow the loading and saving of individual objects. They also have script equivalents in RWSScript.

The version control functions are now in VersionControl.js.

```
IsProjectManaged()
GetLatestVersion()
IsPathManaged(strPath)
GetPersistStatusInformation(ID)
GetFileStatus(strFilename)
GetFileVersionString(strFilename)
GetFileUsersString(strFilename)
CheckOutFile(strFilename, bOnlyIfLatest, bShowDialog)
UndoPendingChangesForFile(strFilename)
SubmitPendingChangesForFile(strFilename)
ImportFile(strFilename)
CheckOutAPIObject(oAPIObject, bOnlyIfLatest, bShowDialog)
UndoPendingChangesForAPIObject(oAPIObject)
SubmitPendingChangesForAPIObject(oAPIObject)
GetLatestVersionOfAPIObject(oAPIObject)
SubmitDefaultChangeSet()
UndoDefaultChangeSet()
OnPermit(ID, RefID, Action)
```

(This is not a complete list.)

## Broadcast events

The new [broadcast mechanism](#) (p.450) in RenderWare Studio 2.0 simplifies adding custom controls to the Workspace user interface. It allows code in any control to call methods with the same name in every other control in the application.

Workspace uses this mechanism to create a set of “standard” method calls that are guaranteed to be broadcast when certain events occur (such as the application starting up, or a project being saved). If a control should take action when such an event takes place, it just needs to implement a method with the appropriate name. Failure to implement a “standard” method does not result in an error.

## Develop custom attribute editor controls in HTML

In addition to using your own custom ActiveX controls as attribute editor controls, you can also use HTML to define a custom attribute editor control.

## New documentation

As well as extensive updates that reflect feature changes, the RenderWare Studio 2.0 documentation includes new topics and tutorials on [customizing Workspace](#) (p.423), and on the relationship between Workspace and the Game Production Manager.

## AppWizards supplied as Visual C++ 7.1 projects

The custom AppWizards that help you to create [ActiveX controls for Workspace](#) (p.477), and [attribute editor controls for Workspace's Attributes window](#) (p.382), have

been updated. They now install into and generate code for Visual C++ 7.1, which ships with Visual Studio .NET 2003.

## Enterprise Author changes

Enterprise Author shares the user interface improvements demonstrated by Workspace, and manages the split REN file project architecture. It also includes support for application splash windows, and offers some new features to the applications it creates.

### Scriptable extensions to objects

In Enterprise Author, you can associate a script file with any object in the interface of the application you're creating. Code in this file can handle events fired by the object, but it can also add functionality *to* the object. The script methods you write here can be called from other objects as though they're members of the associated object itself. (This is sometimes called "script joining".)

### Stand-in control

It can happen that a control required by an Enterprise Author-generated application is unlicensed, or unregistered, or wrongly located. The new stand-in control means that Workspace, for example, can continue to operate—in a restricted mode—in this situation. The stand-in control takes the place of the missing control in the user interface, reports its name, handles calls to methods on the missing control without raising errors, and causes warnings to appear in the Message Log window.

## Game Framework

### .NET projects supplied, VC6 dropped

The Game Framework source is now supplied with Visual Studio .NET project files (.vcproj), not Visual Studio 6 project files (.dsw and .dsp).

### Sequencer behaviors

Game Framework includes behaviors for use with the [sequencer](#) (p.180).

### Fast attribute system

Provides a table of offsets to behavior data, allowing you to set values directly, rather than using the slower method of sending an entire array of values.

## Bugs fixed

A number of bugs from previous versions have been fixed as part of the 2.0 development. The following list simply highlights those that have been raised as particularly important customer issues, rather than an exhaustive list of what has been done.

- Wrong context menus (right-clicking sometimes displayed the incorrect context menu)
- Fixed destruction order (game database objects are now destroyed in reverse order of their construction.)
- **File ▶ Save Project As...** copies build rules (*project\Build Rules* folder)

- Game Explorer closing folder leaves selection (closing a folder with selected sub-items now clears the selection)
- Texture dictionary generation now works with non-embedded assets
- Attribute editor scrolling fixed
- Undo stack no longer cleared by a save
- Attribute window display speed improved

## Upgrading

To upgrade from RenderWare Studio 1.2, see the detailed, step-by-step [upgrade procedure](#) (p.20).

# Upgrading from RenderWare Studio 1.2 to 2.0

---

This topic describes how to upgrade a game development team from RenderWare Studio 1.2 to 2.0 with minimal downtime. Specifically, this topic describes how to upgrade a RenderWare Studio 1.2 game database, managed by NXN alienbrain 6, to the RenderWare Studio [2.0 game database format](#) (p.26), managed by NXN alienbrain 7.

## Summary

The recommended approach is to upgrade the alienbrain server first, and then let your team continue working with RenderWare Studio 1.2 and the alienbrain 6 client. Meanwhile, a system administrator tests the upgrade to RenderWare Studio 2.0 and the alienbrain 7 client on one PC, using a snapshot of the database. When the system administrator has completed testing, they instruct the team to check in all files and stop working. Then the system administrator gets the latest database files from the server, converts them to 2.0 format, and then replaces the 1.2 database files on the server. Finally, each team member upgrades their PCs to RenderWare Studio 2 and the alienbrain 7 client, and then resumes work.

Use the table below to familiarize yourself with the overall procedure. The table presents an abridged version of the procedure; to perform the upgrade, follow the detailed steps presented after the table.

**Note:** In this topic, `project.rwstudio` refers to the RenderWare Studio project that you want to upgrade.

System administrator	Game development team	Time
<b>Stage 1. Upgrade the alienbrain server</b> to version 7. Leave all PCs with alienbrain 6 client.	No access to database during server upgrade.	
<b>Stage 2. Test upgrade on a single PC</b> , using a snapshot of the game database:	Continue working with RenderWare Studio 1.2 as before.	
<ol style="list-style-type: none"> <li>1. Install the alienbrain 7 client.</li> <li>2. Install RenderWare Studio 2.0.</li> <li>3. Open <i>project.rwstudio</i> in Workspace and convert it to 2.0 format.</li> <li>4. Resolve any database conversion errors.</li> <li>5. Build the game data stream and connect to a target. Check that the game still runs.</li> </ol>		
<b>Stage 3. Upgrade Workspace customizations</b> (if any) on test PC.		
<b>Stage 4. Final database upgrade:</b>	Check in all files.	
<ol style="list-style-type: none"> <li>1. Ensure that all database files are checked in, and then get latest.</li> <li>2. Check out <i>project.rwstudio</i> and its <i>project</i> folder.</li> <li>3. Delete your local copies of any game database XML files left over from deleted objects.</li> <li>4. Open <i>project.rwstudio</i> in Workspace and convert it to 2.0 format. Save the converted project to a new location.</li> <li>5. Resolve any database conversion errors.</li> <li>6. Build the game data stream and connect to a target. Check that the game still runs.</li> <li>7. Delete your local copy of the original <i>project.rwstudio</i> and its <i>project</i> folder.</li> <li>8. Save the converted project to the original <i>project.rwstudio</i> location.</li> <li>9. Exit Workspace.</li> <li>10. Delete files on the server that exist in the 1.2 version of the database, but not in the converted 2.0 database on your PC.</li> <li>11. Check in the converted <i>project.rwstudio</i> and <i>project</i> folder.</li> <li>12. Import to the server the new-for-2.0 database files.</li> </ol>	Stop work.	
<b>Stage 5. Upgrade each team member's PC</b> to the alienbrain 7 client, RenderWare Studio 2.0 (with Workspace customizations carried over from 1.2), and the converted game database.	Each team member can resume work as soon as their PC is upgraded.	

If you want only a single period of downtime, rather than the two described above, then you can upgrade the alienbrain server (stage 1) during stage 4. However, there are benefits to upgrading the alienbrain server in a separate, initial stage:

- It minimizes the number of changes that you are making to your system at the same time, making it easier to diagnose problems.
- The idea of stage 2 is to test the upgrade on a single PC before rolling it out to the game development team. If you leave the alienbrain server upgrade until later, then stage 2 will not be a true test of the upgraded environment; you will not be able to test the alienbrain functionality, because RenderWare Studio 2.0 does not work with alienbrain 6.

## Stage 1. Upgrade the alienbrain server

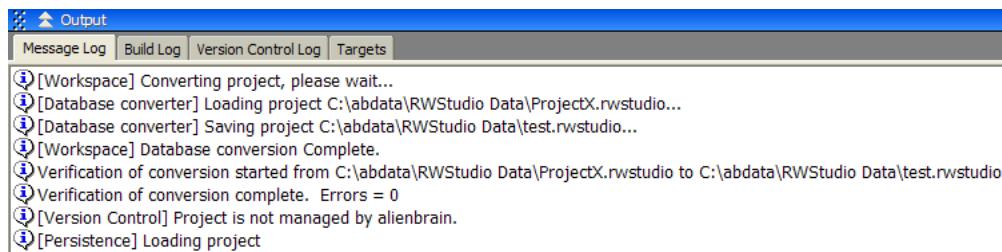
1. Follow the instructions provided with alienbrain to upgrade the server from version 6 to 7.

After upgrading the alienbrain server, your team can continue working with RenderWare Studio 1.2 as before.

## Stage 2. Test upgrade on a single PC

On a test PC:

2. Follow the instructions provided with alienbrain to upgrade the client software from version 6 to 7.
3. Uninstall RenderWare Studio 1.2 (**Start ▶ Control Panel ▶ Add or Remove Programs**).
4. Delete the RenderWare Studio 1.2 installation folder.  
The default installation folder for RenderWare Studio 1.2 was C:\RWStudio. For 2.0, the default is C:\RW\Studio (alongside the other RenderWare components) so it's a good idea to clean up the old installation folder.
5. Use the alienbrain client to get the latest version of *project.rwstudio* and its *project* folder.
6. Ensure that your PC meets the [system requirements](#) (p.82) for RenderWare Studio 2.0. In particular, ensure that you have installed DirectX 9.
7. Install RenderWare Studio 2.0.
8. Open *project.rwstudio* in Workspace.  
Workspace recognizes that the game database is in 1.2 format, and displays a "Proceed with database conversion?" dialog.
9. Click **Yes** to convert the database.
10. Save the converted 2.0 database as *test.rwstudio* in the same folder as *project.rwstudio*.  
Workspace converts the database, verifies the conversion, and then loads the converted database:



11. Resolve any database [conversion errors](#) (p.25) reported in the Message Log window.
12. Build the game data stream and connect to a target. Check that the game still runs.

## Stage 3. Upgrade Workspace customizations

13. Read the advice on [retaining](#) (p.27) any customizations you've made to Workspace, and installing your custom controls into the user interface.
14. On your test PC, implement the most important of these changes, and prepare a set of files for distribution to your users after they've performed their upgrades.

The fragmentation of Workspace source files in RenderWare Studio 2.0 makes it easier for you to roll out less critical changes after the initial upgrade.

## Stage 4. Final database upgrade on test PC

15. Instruct the team to check in all files and stop work on the database.
  16. Ensure that all database files have been checked in.
- On the test PC:
17. Use the alienbrain client to get the latest version of, and then check out, *project.rwstudio* and its *project* folder.
  18. On your local copy of the database, delete any *guid.xml* files left over from previously [deleted objects](#) (p.177).

To do this, run the *PurgeDeleted.wsf* Windows script file supplied in the *Studio\Programs\Tools\PurgeDeleted* folder. (Do not yet delete the copies of these deleted files from the alienbrain server. We'll do that in a later step.)

19. Open *project.rwstudio* in Workspace, and save the converted project as *new.rwstudio* in the same folder.
20. Build the game data stream and connect to a target. Check that the game still runs.
21. Use Windows Explorer to delete your local copies of *project.rwstudio* and its *project* folder.
22. Switch back to Workspace, and save *new.rwstudio* as the original *project.rwstudio*.
23. Exit Workspace.

Next, we will replace the original 1.2 database files on the server with the

- converted 2.0 database on your PC.
24. Use the alienbrain client to delete the server copies of all files in the *project* folder that exist only on the server. These are the files that exist in the 1.2 version of the database on the server, but not in the converted 2.0 database on your PC:
    - The `Folders\guid.xml` file for the global folder (this is now `Folders\RWS_GlobalFolder.xml`).
    - Any `guid.xml` files for deleted objects (that you have deleted from your local copy of the database, using the `PurgeDeleted` tool).

**Tip:** In the alienbrain client, files that exist only on the server have dimmed icons with dashed outlines.
  25. Use the alienbrain client to check in `project.rwstudio` and its *project* folder.  
This replaces the old 1.2 database files on the server with the new 2.0 files on your PC, while retaining their revision history.
  26. Use the alienbrain client to import the [new-for-2.0](#) (p.26) database files:  
`project\Folders\RWS_AssetHierarchyFolder.xml`  
`project\Folders\RWS_AttributeShareHierarchyFolder.xml`  
`project\Folders\RWS_GlobalFolder.xml`  
`project\Folders\RWS_TemplateHierarchyFolder.xml`  
  
 As per previous versions of RenderWare Studio, **do not** import the `project\project.settings` file to the alienbrain server.  
  
 You have now replaced the old 1.2 database on the server with the 2.0 database.
  27. Delete `new.rwstudio`, `test.rwstudio`, and their corresponding folders (`new` and `test`) from the test PC.
  28. As a final test, start Workspace, and open the converted, alienbrain 7-managed project. Build the game data stream and connect to a target. Check that the game still runs.

## Stage 5. Upgrade each team member's PC

- On each team member's PC:
29. Install the alienbrain 7 client.
  30. Uninstall RenderWare Studio 1.2.
  31. Delete the RenderWare Studio 1.2 installation folder (default is `c:\RWStudio`).
  32. Delete the local copy of the `project.rwstudio` and the contents of the *project* folder, **except** for the `project\project.settings` file.
- Note:** The format of the `project.settings` file is the same in RenderWare Studio 1.2 and 2.0.
33. Use the alienbrain client to get the latest version of `project.rwstudio` and its *project* folder.
  34. Ensure that the PC meets the system requirements, and then install RenderWare Studio 2.0.
- Each team member can resume work as soon as their PC is upgraded.

# RenderWare Studio 1.2 to 2.0 database conversion errors

---

The database converter can report the following errors:

## **File missing: *file***

The file exists in the original 1.2 version of the game database, but not in the converted 2.0 version.

This error is not reported for deleted objects (objects whose files contain a `<Deleted/>` element), or for the global folder XML file (which, in 2.0, has been renamed to `RWS_GlobalFolder.xml`).

## **No internal XML element for file: *file***

Either:

- The file is not valid XML  
or
- The value of the `id` attribute of the first child of the root element does not match the `guid` in the `guid.xml` file name. For example:

```
<?xml ... ?>
<RenderWareStudio version="2.0">
    <Asset id="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX">
        (Must match base file name):
    </Asset>
</RenderWareStudio>
```

## **Child link missing. File = *file* Child = *child***

The file refers to a child object, but the file for the child object does not exist in the converted database.

In RenderWare Studio 2.0, if an object has children, then its corresponding database XML file contains `<ChildReference>` (p.26) elements that refer to the children.

## **Parent link missing. File = *file* Parent = *parent***

This file was referred to by its parent object (see error described above), but this file does not contain a `<ParentReference>` element that refers to the parent object.

## **Unspecified problem at: *info***

Contact RenderWare Studio support.

For more information, see the script that reports these errors:

`Studio\Programs\RenderWareStudio\Scripts\VerifyConversion.vbs`

# Differences between RenderWare Studio 1.2 and 2.0 game databases

---

You do not need to know these details to follow the upgrade procedure, but it's useful to note that there are only a few differences between RenderWare Studio 1.2 and 2.0 game databases:

- In RenderWare Studio 1.2, if a database object had children, then the object's `guid.xml` file contained `<ObjectReference>` elements that referred to those children. In RenderWare Studio 2.0, these are now `<ChildReference>` elements:  
`<ChildReference id="guid" type="object type">`  
 where *object type* identifies the type of child object referred to by the *guid* (for example, an entity typically has one or more `RWSAssetID` children).
- If a database object has parents, then the object's `guid.xml` file contains new `<ParentReference>` elements that refer to its parents:  
`<ParentReference id="guid" type="object type">`  
 where *object type* identifies the type of parent object referred to by the *guid* (for example, `RWSFolderID`).
- The XML file for the global folder has been renamed from:  
`project\Folders\guid.xml`  
 to:  
`project\Folders\RWS_GlobalFolder.xml`
- The following new XML files store the hierarchies shown in the Assets, Attribute Shares, and Templates windows, respectively:  
`project\Folders\RWS_AssetHierarchyFolder.xml`  
`project\Folders\RWS_AttributeShareHierarchyFolder.xml`  
`project\Folders\RWS_TemplateHierarchyFolder.xml`
- The value of the `version` attribute, in the root `<RenderWareStudio>` element of each XML file, has been changed from "1.2" to "2.0":  
`<RenderWareStudio version="2.0">`

# Retaining Workspace customizations

---

The change in Workspace's definition from a monolithic REN file to a collection of application source files makes *new* customization easier to do, and easier to integrate into future revisions of the application. It does mean, however, that you need to do a little work to integrate any *existing* customizations into the new architecture.

To reproduce your changes to Workspace in the new version of the software, you need to know exactly what those changes are, and where they're likely to take place in the new "split" REN file. To help you discover this information, both Workspace and Enterprise Author allow you to convert old-style REN files to the new format by loading and saving them. This means that you can:

1. Load and save an original RenderWare Studio 1.2 REN file, creating a "split" version of that document.
2. Load and save your customized version 1.2 REN file, creating a "split" version of that document.
3. Run `WinDiff` (or a similar file comparison utility) against the two sets of files generated above.

Following this procedure will give you a good indication of where to start customizing a new Workspace installation. Because of the size and quantity of the changes between versions, however, it is unlikely to give you the complete picture. Ideally, you need to know the *intent* as well as the location of the changes you made to the old version.

In some cases, you'll be able to repeat your old changes in the new Workspace source files. In others, the recommended techniques for making Workspace customizations will have changed. In others still, the original technique will not work at all. The rest of this topic suggests strategies for dealing with different customization types in the new version of the application.

**Note:** The section of the documentation called [Customizing Workspace](#) (p.423) contains detailed information and tutorials on the new customization techniques.

## Customization strategies

In general, customizations to Workspace fall into four categories:

- Changes to the positions of windows in the application at startup, including the creation of new layouts
- Modifications to Workspace's supplied menus and toolbars
- Creation of new objects (controls, menus, toolbars, etc.) for Workspace
- Alteration of the supplied default script code to incorporate new objects

## Layout changes

The new version of Workspace introduces new windows, gets rid of some old ones, and comes with a range of built-in layouts. It also stores information about window positions and layouts in a different format. For these reasons, it is not possible to take old layout specifications and turn them into new ones.

If Workspace's built-in layouts do not cater for your needs, you can recreate your custom ones (or equivalents of them) by hand in Workspace itself, or in Enterprise Author. When you've created a new layout for Workspace, distributing it to users just involves [copying .Layout files from your computer to theirs](#) (p.335).

## Menu bar and toolbar modifications

In pre-2.0 versions of Workspace, adding items to menus (or buttons to toolbars) meant inserting those items and their handlers into the supplied REN file.

However, changing a supplied file can make future upgrading difficult, because you have to make the changes again if that file is updated. The approach is still *possible* under version 2.0, but it's recommended that you choose one of the alternatives:

- Don't add new buttons to existing toolbars. Instead, [create new toolbars](#) (p.469) and insert them at suitable locations in the user interface. This insulates you from any future changes to the default toolbar set.
- Don't add new items to existing menus by modifying the menu bar objects. Instead, [create new menus](#) (p.462) that contain only the items you want to add, and use [menu merging](#) (p.472) to customize Workspace's menus at run time.

Using events, or the new [broadcast mechanism](#) (p.450), you can merge menus when Workspace starts up, or at various points during execution—when a context menu is displayed, for example. This technique allows you to edit the built-in menus without actually changing their implementation.

## Custom objects

If you've created custom objects (ActiveX controls, pages, or menu bars) for previous versions of Workspace, the process of loading and saving the old REN file will generate the [.RENObj and .vbs files](#) (p.426) that you need in order to install them into the new version. This is a big step towards reproducing your old customization, and depending on how the object works, it may even be all you need to do. In general though, there will be further considerations (usually involving script code—see the next section), and there will certainly be steps that you can take to make customization easier in future.

**Note:** In most cases, ActiveX controls that you've written for earlier versions of Workspace will work in the new version without recompilation. Should you want to update these custom controls, you should consider recreating them using the new version of the [RenderWare Studio ActiveX control wizard](#) (p.477).

## Script code customization

Any new object that you install into Workspace requires script code so that it can interact with the rest of the application. In the new version, you should try to write this code only in the script module of the object in question, rather than adding to the supplied .vbs files. To help, there is a range of global events and broadcast method calls that you can use to make your objects to do the right thing at the right time.

### ActiveX controls

Custom ActiveX controls can perform many different tasks, but their interactions with Workspace tend to be similar. It's a common requirement for them to initialize themselves or change their appearance in response to things happening

in the running application, such as:

- Application startup
- Changes in the selected entity
- Projects being loaded or saved
- Game code being built

In previous versions of Workspace, it was typical to add code to existing methods to make the necessary calls to new objects. For the new version, that code should instead be placed in broadcast method implementations in the script modules belonging to the new objects.

Workspace broadcasts method calls for all four events mentioned here. It also does it when context menus are displayed (as described above), and on a number of other occasions. The separation of custom code from supplied code that this enables makes it easier to roll customizations out to your users, and protects you from future changes to Workspace. As long as those broadcast method calls are made, you don't need to worry about the supplied code.

## Pages

In terms of their interactions with Workspace, pages need to be notified of the same events as ActiveX controls, and there is no difference in the way this should be dealt with. If you've previously added code to the supplied scripts, you should try to transplant it to the .vbs files of your custom pages.

## Menu bars

The new technique for customizing default menus was described earlier, but there are also changes to the way that menu item selections are handled. If you're creating a new menu, you should not use the IDs of items to determine which one has been selected. The new version of Enterprise Author allows you to [specify the name of an event to be fired by each menu item](#) (p.467), and you should deal with selections by handling these events in your code.

# Known issues

---

This section contains known issues and workarounds that have been identified in RenderWare Studio 2.0. These issues are current as of May 2004.

## **Previous releases must be uninstalled before installing RenderWare Studio 2.0**

RenderWare Studio's new installer will support future updates without requiring version 2.0 to be uninstalled. Before you install 2.0, however, you must uninstall any earlier RenderWare Studio version, including any beta or preview versions of RenderWare Studio 2.0.

## **Installing Genre Pack 1 after installing RenderWare Studio 2.0 may require a reboot**

The Genre Pack 1 installer can fail to detect an installation of RenderWare Studio 2.0, if you attempt to attempt to install both packages in sequence.

If you experience this problem, quit the Genre Pack 1 installation and reboot your PC before attempting to install Genre Pack 1 again.

## **Creating a stand-alone version of Workspace in Enterprise Author does not work**

Do not attempt to use Enterprise Author's **File ▶ Make Executable...** menu item to create an executable version of RenderWare Studio Workspace. Files created in this fashion lack many key features.

## **Genre Pack 1 artwork looks very dark on some systems**

Under some PC and graphics card combinations, the DirectX version of the console can make the Genre Pack 1 artwork look very dark. You may need to adjust the gamma correction on your graphics card to get the best results.

## **Missing AI lesson in the Genre Pack 1 tutorial**

Due to a late change in the integration of RenderWare AI with RenderWare Studio 2.0, the lesson on using artificial intelligence within Game Engine 1 has been temporarily removed from the documentation. A future documentation update will contain a lesson on this subject.

## **Testing merged changes before submitting them to alienbrain**

After resolving conflicts in a changelist submitted to alienbrain through Workspace, it is not possible to test the modified files in Workspace before finalizing the submission.

If your changelist causes conflicts, cancel it in Workspace, switch to the NXN alienbrain Manager Client, and attempt to submit it again from there. When you've resolved the conflicts, but before you commit the changes, switch back to Workspace. You can then reload your project and test the merged changes before committing them.

# Release notes for Version 1.2.2

---

This release fixes some installation and NXN alienbrain issues, and one minor user interface bug. The installation program will update existing RWStudio 1.2 and 1.2.1 installations, or alternatively can install a complete version. To update versions prior to 1.2, please see the release notes for that version.

## Changes

### **Installation fix**

The system search path was being modified such that in certain instances it was found to interfere with the licensing of RenderWare Graphics. This new install supersedes the manual fix suggested to customers who had this issue.

### **NXN alienbrain speed issues**

A number of alienbrain-related performance issues have been resolved. Any customers that used the emergency patch DLLs from CSL support should update to RWStudio 1.2.2, which contains more complete changes.

### **Attribute share drag and drop**

The ability to create attribute shares by drag and drop has been restored to this version.

# Release notes for Version 1.2.1

---

## Overview

This release of RenderWare Studio is primarily to address database load times, bugs and documentation issues found subsequent to the release of version 1.2. A small number of minor feature requests have also been included.

## Installation

The setup process will upgrade existing RenderWare Studio 1.2 installations to version 1.2.1.

If RenderWare Studio has not been previously installed on a machine, then the setup process will install a full version 1.2.1., therefore it is not necessary to install 1.2 before 1.2.1.

**Note:** If a version of RenderWare Studio earlier than 1.2 is installed, then users should follow the upgrade instructions in the [1.2 release notes](#) (p.34).

## New features

### Faster load

Database load time has been significantly improved. The improvement factor varies considerably with the database, but tests have shown typical gains of between 2x and 5x over previous versions.

### Fewer steps required to create an asset

A quick way to bring a new asset into the game database is available from a context menu in the asset view. Right click in the window and select New ▶ Asset.

### Build All and Clean All options

The Build operation in the 1.2 Workspace simply built the currently active level. The entire game database can now be rebuilt from the Workspace using new “Build All”, and “Rebuild All” options. A corresponding “Clean All” operation is also provided.

### Command line options

The clean and build options in the Workspace can now be run from the command line for batch build processes, to clean and/or build all folders in the game. For example:

```
C:\RW\Studio\Programs\RWS stu dio.exe /game
"C:\RW\Studio\Examples\Example Alpha Sort.rwstudio" /clean
"Connection2"
```

### Connect without build

In version 1.2, selecting “Connect” to a console caused the Game Production Manager (GPM) to check if any streams needed rebuilding, and then invoke a build if necessary. Even if nothing had changed, the check processing itself caused unwarranted delays. To alleviate the delay, the “Connect” operation has been changed so that it no longer does this initial check, and a new menu has been added called “Build and Connect” that works like the old mechanism.

**Note:** Since the decision to check on data validity has now been left to the user, the success of the “Connect” operation depends on the database not having changed significantly. If problems occur, simply execute “Build and Connect”.

### **Game Production Manager documentation**

The introduction to the GPM has been extended, and a simple tutorial added. (More GPM documentation will be included in the next release, including details on the supplied build rules.)

## **Game Framework**

Changes to the game framework have been minimal for this release. These include:

### **Warnings generated on Maximum level with ProDG now removed**

Some small changes have been made to the Game Framework files to remove warnings that were generated at maximum warning level setting when compiling with SNSystems™ ProDG for the PS2 target.

# Release notes for Version 1.2

---

## Overview

RenderWare Studio 1.2 contains many changes with major enhancements in the following areas:

- The introduction of a fully customizable and flexible game build process
- Structured game databases
- Improved 3D interaction
- Memory and performance tools

The new features are explained in more detail below. Existing users should read the notes on upgrading existing installations.

The Game Framework is largely untouched since release 1.12, except to show and exploit some of the new features such as the memory and performance diagnostic tools, linear memory manager, and global entities.

## New features

### **Game Production Manager** (automated asset management and image production pipeline)

The major feature of this release is a customizable and flexible mechanism for transforming original game data (assets, entities etc.) into the final deliverable format. The Game Production Manager (GPM) handles dependencies, custom processing and stream creation. It forms the core of all the game data processing and sits at the centre of the whole game development process.

The GPM offers the following capabilities:

- Ensures consistency of data across all team members.
- Rule-driven asset dependencies and transformations. This enables, for example, automated creation of texture dictionaries.
- Incremental in-game asset handling managed by an integral “make”-like component.
- Completely customizable.
- Support for asset validation (the facility to abort the build based on custom rules).
- Batch-driven capability to enable automated, repeatable full builds of all or parts of the game.
- Easy custom network packets/command definition.
- Rapid development of the final build process.

The Game Production Manager is a set of COM interfaces for creating production-quality game data. The core of the GPM consists of the RWSMake COM object. This object is responsible for defining a process whose output is the final game data image on disk. The process is similar to

the processing of a makefile during a source-code build, with a few advantages:

- You write the build rules in a script language (VBScript), so the process is much more dynamic.
- You can set up more complex dependency/target relationships through the use of cross-dependencies and independent sub-build processes.
- Dependency/target relationship can optionally be entirely self-contained through the AddSubtask mechanism.
- Rule files that define separate tasks are logically separated due to the nature of the architecture.
- Data of any form (from built-in script types to COM objects) can be shared across the whole process, or within each task.
- The process is often simplified because rule arguments can be any type of object.
- Less tools are often required, as code can optionally be written in script.
- Development of the final build process can be very rapid due to the nature of script development.
- Error reporting during development of a process is more accurate (rule name and line/character number).
- The process can be deliberately halted at any point (with meaningful messages) if certain requirements are not met (for example, number of triangles too high).
- Any tools can be better integrated, as there is no longer a restriction on tools being command-line driven.
- Complex processes are simpler to debug because the entire dependency hierarchy is automatically available via the IRWSSubtask interface.
- A centralized location for reporting information, warnings, and errors is immediately available at any point in the process.

### **.rf3 support**

RF3 files contain platform independent intermediary export data in XML format. Unlike .rws files, which are RenderWare Graphics optimized binary files, .rf3 files contain no rendering optimizations, and are simply a snapshot of the raw 3D data. RenderWare Platform developers can compile these .rf3 files into optimized platform-specific RenderWare Graphics binary files, (.rws, .rp2, .rgl, .rx1 etc.) using a custom tool.

With this release of RenderWare Studio, the compilation process is handled by the Game Production Manager transparently. A benefit of this is that original art assets can be stored as .rf3 files, and automatically converted to platform-optimized files without extra processing steps required by artists and designers.

### **3D interaction interface**

The user interface for interacting with the 3D Design View has been overhauled to make it much easier to mix selection and movement operations. Using a set of modifier keys, artists and designers can quickly

move around objects of interest without having to use menus and toolbars to change modes.

Various related enhancements have also been made to the dragging mechanism; for example, there is a new selection lock feature, and a mode to drag objects along a surface.

### **RenderWare Studio Manager COM interface** (for script programming)

In addition to the existing C++ API, you can now program the RenderWare Studio Manager via a COM interface. This allows you to use the RenderWare Studio Manager in script applications, including:

- Enterprise applications (.ren), such as the RenderWare Studio Workspace
- Visual Basic applications
- HTML applications (.hta)

This feature enables easier and more powerful customization, and has been used as part of the implementation of some of the other features such as the database search tools. It is also extensively used by the Game Production Manager.

### **XML format for .ren files**

The customizable user interface is described by a .ren file, which defines the ActiveX controls, menus, scripts, and dialogs used by the Workspace. Prior to version 1.2, this was a custom format binary file, but is now XML. This provides easier tracking of changes, particularly for the user interface scripts that tie everything together, in addition to making it easier for more than one person to edit the user interface at the same time. See notes below on updating your existing .ren files to the new format.

### **Graphics and animation preview tool**

The new Preview tool for the Workspace lets you inspect graphics objects and animations. You can either drag objects from the current game database, or drag asset files from Windows Explorer (without having to import them into the game database). The Preview tool also provides relevant statistical information such as number of triangles and bones.

### **NXN alienbrain integration changes**

The alienbrain integration has been enhanced to provide several new features:

- Option to automatically import assets into alienbrain when added to the game database.
- When getting the latest version of the database from alienbrain from within Workspace, users are offered the option of getting all the latest assets.
- Identify who has what checked out from alienbrain by displaying a "Checked Out" column in the Game Explorer window.

### **Structured database**

Earlier versions of RenderWare Studio provided a limited form of structuring the game database that has been completely overhauled and replaced with a structured database hierarchy. This removes the idea of fixed, level-oriented containers; now you can organize entities, assets, and templates into nested folders. Key elements of this new database structure

are:

- “Folders” replace “Collections”. The folder is similar in concept to the earlier collection idea, except that it can now contain other folders. The folder also replaces the previous “Level” object, as it offers the same functionality. For certain situations, typed folders can be created that restrict the contents of the folder.
- “Attribute Shares” replace “Attribute Collections”, which is effectively a dedicated Manager API object type.
- Multiple references (shortcuts) to objects and folders can be created.

Existing users should note a slight change to the Asset window. In previous releases, double-clicking an asset thumbnail showed an in-place 3D preview, allowing the object to be zoomed and rotated. Since assets can now be arranged into hierarchies, double-clicking has been chosen as a natural way to open the asset folder. To preview thumbnails, you now right-click the thumbnail and then click **Activate Preview**.

### **Global entities**

Within the new structured database hierarchy, there is always one folder that can be used to store entities and assets that are “global”. This *global folder* is not removed from the console when the active folder is changed. Typical uses of this are for game engine-type entities that exist for the entire lifespan of a game, rather than being level-dependent.

The global folder is automatically added to new projects.

### **Database search**

A facility has been added to run a number of common search operations over the database. For example, to find all entities that reference a particular asset, or all objects that match a particular name. As these searches are written using the scripting interface to the RenderWare Studio Manager API, you can easily extend the searches for your own needs. For your convenience, all the searches are in a script module called “Searches” that you can find by loading the `CustomInterface.ren` file into Author. Most searches are initiated by context menus defined in the `.ren` file.

### **Entity and folder game reset**

A new feature that allows a subset of entities to be reset in the design build of the game has been added for this release. This is useful when tweaking game play, and the designer just wants to reset something before trying a different value. This feature can also operate on folders, so that entire parts of the game can be reset just by sending the entity attribute packets, rather than resending all of the assets.

### **Linear Memory Allocation**

This new game framework optimization exploits the Game Production Manager to create contiguous memory buffers for looking after multiple entities of the same type. This can be enabled on a per-behavior basis, and helps prevent memory fragmentation.

### **Event editing and viewing enhancements**

The event handling user interface has been improved. In addition to substantial changes to the Event Visualizer window (now called Event Map) to enhance clarity and navigation, a new Events window has been created to allow browsing of all event messages in the system. Event messages can be dragged from this view and dropped in the relevant attribute editor slot.

## **Memory Statistics tool**

This release introduces a new tool to display memory usage over time of a running game. This helps identify areas of memory fragmentation, for example. It can also graphically differentiate which subsystem (game, framework, RenderWare Platform etc) is responsible for different allocations. The display is interactive, and can link directly to the line of code responsible for any given allocation.

## **Framework performance monitor**

Another new tool provides a per-function performance breakdown of a running game. Two statistics are provided:

1. Time in a function (and all that it calls)
2. Cumulative call counts

You can drill down the call stack to identify performance hot spots.

This tool is not intended as a substitute for a full performance analyzer. It is aimed at being able to quickly identify areas of performance concern while in game design mode, without having to break away from the RenderWare Studio environment.

## **Spline Editor**

The Workspace now has a new tool for creating and editing spline assets. This exploits two other new features of this release:

1. Pipeline for editing custom asset types within Workspace. The spline editor can be seen as an example of an integrated tool that can create and modify assets within Workspace, while taking full advantage of alienbrain.
2. Generic path editing module. This is a programmable API that encapsulates handling of generalized graph objects (nodes linked by edges in arbitrary ways) while integrating into the pick and edit facilities of Workspace.

## **New Attribute Editor tutorial**

This release introduces a new attribute editor tutorial that concentrates more on the game data than the first tutorial which described mainly the mechanics of adding a custom attribute window into the Workspace.

# **Miscellaneous frequently requested features**

This section lists some of the frequently requested suggestions that have been implemented as part of this release.

## **Delete key in Design View and Game Explorer windows**

You can now delete selected objects in these by pressing the **Delete** key.

## **Context menu in Design View**

With an object selected, right-clicking in the 3D Design View displays a context menu for various common operations. Note that, when in Camera Flight mode, this option is disabled as the right mouse button is used for panning.

**Note:** All context menus in the Game Explorer and Design View windows can be customized by editing the `.ren` file that defines the Workspace user interface.

## **Expand game database tree to selected object**

With an object selected in the 3D Design View, you can find it in the main Game Explorer window by right-clicking to display the context menu and selecting the relevant option.

#### **Automatic texture dictionary update**

This facility is part of the new Game Production Manager.

#### **Raytest API in 3D User Extensions**

The 3D user extensions API now provides a method to manage raytests within the 3D view.

#### **Folders in folders**

This is part of the new structured database feature.

#### **Template collections / set piece dragging**

Part of the new structured database, folders can be turned into templates.

Dragging a template folder into the Design View instantiates a copy of all the contained items into the view.

#### **Removed large flat entity and asset lists**

As part of the structured database development, the game database view (now called "Game Explorer") has been simplified to remove the large flat lists of entities and assets.

#### **Flattened attribute editor**

The attribute editor now defaults to showing the exposed attributes for all subclasses, while retaining the facility to filter this as before.

#### **.NET project files for Game Framework on Xbox**

The Xbox projects for the Game Framework are all provided as .NET projects.

#### **Unique attribute packet identifiers**

Some customers have been using levels as ways of creating different streams for use at various parts of the game, and had to work around the issue that attribute packet IDs were unique only within a stream. This is now fixed. See below for changes needed to attribute handler code.

#### **Default timeout value**

The default network send timeout for connections has been set to 10000ms (10 seconds). If this timeout period expires whilst the workspace is sending data to a connected target, the connection will be dropped. If you wish to debug your connected target code during a data send, you should check the "infinite" timeout checkbox in the connection properties dialog. This stops the connection being dropped whilst you are debugging for more than the timeout period.

#### **Choice of IDEs for viewing source**

You can now set which editor should be used when viewing source code from within Workspace. The choice is between Visual Studio 6, Visual Studio .NET, and CodeWarrior. Alternatively, if you just use the system default association, Workspace will attempt to locate the editor that is associated with C++ header files. Users should note, however, that in some circumstances this means that the file can be loaded but the correct line will not be selected. It is recommended that you explicitly set your editor of choice from the **Options ▶ Workspace** dialog to ensure proper navigation within source files.

## **DirectX9**

RenderWare Studio Workspace requires a DirectX9 runtime and drivers to be installed first. The installation will fail if it does not detect the correct versions of DirectX. You can download the DirectX9 runtime from [Microsoft](http://www.microsoft.com/windows/directx/default.aspx) ([www.microsoft.com/windows/directx/default.aspx](http://www.microsoft.com/windows/directx/default.aspx)).

## Upgrading from earlier versions

This section describes steps you may have to take to upgrade existing RenderWare Studio game databases so that they will work with and take advantage of the new features. In particular, the entire database to console pipeline can now be automated using the Game Production Manager components.

The update process will be easier if you have already upgraded to RenderWare Studio 1.12. It is recommended that users who have not yet upgraded to 1.12 should do so first.

## Uninstall previous releases

Any previous releases of RenderWare Studio should be uninstalled prior to installing 1.2. Safe copies should be made of any custom .ren files, as these may be needed for reference later.

After uninstalling earlier versions it is recommended that you delete or rename the existing RWStudio top-level installation folder.

## Upgrading your database

Updating your existing database is not trivial, but Criterion support is available to help on any questions regarding the upgrade process. The complications lie in managing the downtime of the database when nobody should be making modifications. At the end of this section is a suggested set of steps that should help you through the process.

### Databases created pre-RenderWare Studio 1.1

If your database was created with a version of RWStudio before 1.1, you must ensure that the GUID enforcer tool has been passed over the database.

**Note:** It is important that you run GUID enforcer and check your database validity before installing 1.2.

If, however, your database was first created with RWStudio 1.1 or later, then you should not need to run the GUID enforcer. If in doubt, run the GUID enforcer and validate the integrity of your database before installing 1.2.

To run the GUID enforcer, follow these steps before uninstalling earlier releases:

1. Ensure RenderWare Studio Workspace is not running.
2. Run GuidEnforcer.exe (in the Programs/Tools folder).
3. Browse to your game database.
4. Ensure that the **Generate GUIDs to replace all IDs** check box is not selected.
5. Click **Enforce GUIDs**.
6. When processing has finished, click **Cancel** to exit the enforcer.
7. Repeat for all game databases you wish to update to 1.2

## Upgrading the database

RWStudio 1.2 will load older versions of the database but then disable certain operations that could cause problems with the new database structure. Users are informed of this in the Message Log window. The recommended upgrade mechanism is for the game database administrator to load the existing database, and then do a **File ▶ Save As** immediately to a different project name or location.

When loading old databases, a global folder will automatically be created.

After saving the database to its new location, you should reload the database to ensure all operations that were disabled due to an old version being loaded are re-enabled.

It is strongly advised that all users submit all changes to the original database first, so the new database is clean and has no checked out files. Due to the coordination effort involved, the database administrator should do a dry run first to ensure any custom tools and modifications have been changed to work with the new release. After conversion the new database can be added to alienbrain as normal, and then all users should switch to this new database. See later for an example update sequence.

It is recommended that when you add the new database to alienbrain that you do so using the **File ▶ alienbrain** menu in the Workspace, rather than attempting to do it directly using the NXN alienbrain client software. This will ensure that only the relevant files are placed under revision control.

## Texture dictionaries

Since the GPM now creates texture dictionaries automatically, there is no longer any need to explicitly add these to the game database. You should remove any texture dictionaries as part of the database update process.

## Upgrading customized REN files

The supplied REN file has changed considerably since RWStudio 1.12. The simple but potentially more error prone way of upgrading is to simply reapply the changes you made, assuming these are recorded somewhere. This is just as you would have done for earlier releases.

Alternatively, you could attempt to exploit the new XML feature to find the differences for you:

1. Load in an original unmodified `CustomInterface.ren` file into 1.2 Author.
2. Save this to `Original112.ren`.
3. Load your modified `CustomInterface.ren` into 1.2 Author.
4. Save this to `Modified112.rencode>`
5. Using a text compare program (such as WinDiff) compare `Original112.ren` and `CustomInterface.ren`.
6. Take note of what changes you made to the original `.ren` file. Depending on what controls you had in your `.ren`, you will see a great many changes early on in the file in what looks like random characters. This is binary data encoded as ASCII, and should be ignored. Instead, scroll to the bottom to look at the differences in script.

The differences should show you what modifications you made to the previous

release. If you load the 1.2 CustomInterface.ren into Author it should be a easy to reapply these edits to the 1.2 user interface. You may have to search for equivalent functions, but the overall structure of the .ren file remains the same.

There will be little to gain by comparing the 1.12 and 1.2 .ren files as they differ so much it is unlikely that a text comparison tool will show any useful differences.

## Custom controls and attribute editors

Any custom controls and attribute editors will need recompiling against the new 1.2 APIs and headers. Depending on what kind of control you have, some source code changes may be required. If you get any errors in compiling, linking, or execution that are not obvious to fix, please contact CSL Support.

## Exploit Game Production Manager and structured database

As part of the update process there will be opportunities to exploit the many new features of the build process. The following list are some things to consider:

- Texture dictionary creation is now automatic and are created dynamically, so you should remove texture dictionaries as explicit assets.
- Investigate custom rules for handling any special transformation of data you may have. For example, if you have a tool already that does some processing on an original asset (say, a script), and transforms it into a console runtime file, then there is scope for automating this with a custom rule.
- If some of your data has an ordering dependency that you hope artists and designers preserve when using the Workspace, consider explicitly specifying dependencies with a custom build rule. This removes the burden on the artists and designers.
- Create folders and move objects to take advantage of the new structured database facility.
- Move game level entities into the global level. This will remove possible duplication of shared entities that you may have in your existing database.

## Example sequence for upgrading to RWStudio 1.2

These are the recommended steps on which to base your update process. For more details see above.

1. Run GUID enforcer if needed, *using your current version of RenderWare Studio.*
2. Install RWStudio 1.2 on a single machine.
3. Get latest from alienbrain.
4. Load the project into Workspace.
5. Do File->Save As to make a copy of the database in 1.2 format to a temporary location. If your project is managed by alienbrain, you should then do a Save To alienbrain, writing over the top of the newly created project.
6. Reload your project to force all 1.2 options to be enabled.
7. Recompile any custom ActiveX and attribute controls you may have had in earlier versions.

8. Make any UI customizations to REN file.
9. Test it all works.
10. Get all users to checkin all outstanding changes to the game database.
11. Lock the game database against any checkouts and changes. Do this from the alienbrain client.
12. Remove the game database from the single machine, and get latest to ensure a fresh copy.
13. Load into RenderWare Studio Workspace.
14. Save database to new location.
15. Add new game database to alienbrain. It may be easier for the rest of your users if you move the old database to a new location, and replace with the new one. If you do this, be sure to lock the database against unintentional changes while doing the update.
16. All users remove their local copies of the game database. This removes the risk of getting a mixture of 1.12 and 1.2 databases.
17. Update the rest of the team to 1.2, including tools and customization, and get latest on the new alienbrain database.

**Note:** It is strongly recommended that you take backups at each stage of the conversion, as this will help trace any problems that may occur

## Changes required in game code to upgrade from 1.12 to 1.2

This section describes what changes may have to be made in your game code to support the new features of this release.

Upgrading from 1.12 to 1.2 is much simpler than for 1.11 to 1.12. We suggest checking out your own copies of the files under `gfcore`, `startup`, and `mainloop` and copying the 1.2 files over them. You can then use your version control tool to go through each file and check the differences to bring across any edits you may have made.

### Attribute handlers

Attribute handlers are now identified by the unique persistent ID rather than the ID, so in the attribute handler code you will see code like:

```
void Add(unsigned int instanceId);
static void Remove(unsigned int instanceId);
static CAttributeHandler* Find(unsigned int instanceId);
```

changed to:

```
void Add(RWSGUID & instanceId);
static void Remove(RWSGUID & instanceId);
static CAttributeHandler* Find(RWSGUID & instanceId);
```

This means that the attribute handlers are globally unique eliminating the problems with loading multiple streams. To make use of this you need to ensure you have run GUID enforcer on any databases created before RWStudio release 1.1 (see above).

### Resources

RWSGUIDs are now used to identify resources in a similar way to the attribute

handlers. The UID of an asset in the game framework will be the same as the ID used for the resource in the workspace.

## Stream changes

The format of the data in the `RwStream` objects for assets and entities has changed due to the unique IDs mentioned previously, and also for the flagging of objects as global. The functions that have been updated accordingly are:

```
CreateEntity() in attribhandlerstrfuncs.cpp  
LoadAsset() and LoadEmbeddedAsset() in resourcemanagerstrfunc.cpp
```

Details of the data format changes can be found in the `target_change.log.txt` file which is in the root RenderWare Studio install directory (typically `C:\RW\Studio`).

## New files to add to your projects

There are a few new files in `gfcore` that you will need to add to your project:

### Function profiling

To take advantage of the new performance profiling tool in the metrics layout in the workspace, add the following files

```
core\function profiler\Profile.h  
core\function profiler\Profile.cpp
```

### Memory profiling

To take advantage of the new memory profiling tool in the metrics layout in the workspace, add the following files

```
core\memoryhandler\memoryprofile.cpp  
core\memoryhandler\memoryprofile.h
```

### GUIDs for entities

This feature solves the problem of using multiple streams by providing a unique if for every instance of a behavior, and means adding the following file to your project.

```
core\misc\rwsguid.h
```

### Allocation policies

You will need to add the following files to handle the new linear memory manager feature

```
core\attributehandler\Allocation  
Policies\dynamicallocationpolicy.h  
core\attributehandler\Allocation  
Policies\linearallocationpolicy.h  
core\attributehandler\linearallocationpolicy.h  
core\attributehandler\linearallocationpolicy.cpp
```

## Common problems upgrading Game Framework code

Some common problems to look out for when upgrading the Game Framework code:

### Ensure that `$(RWGSDK)\include\xxx` is before `$(RWASDK)\include\xxx` in the project settings

If this is not done the wrong version of `rwplcore.h` will be included causing compile errors.

**'UseLinearMemory' : is not a member of 'MyBehavior' compile errors**

The following errors may come from your code:

```
...\\console\\game_framework\\source\\modules\\FPS\\FPSTriggers\\MyBehavior.cpp(54)
:
error C2039: 'UseLinearMemory' : is not a member of 'MyBehavior'

...\\console\\game_framework\\source\\modules\\fps\\fpstriggers\\MyBehavior.h(63) :
see declaration of 'MyBehavior'

...\\console\\game_framework\\source\\modules\\FPS\\FPSTriggers\\MyBehavior.cpp(54)
:
error C2065: 'UseLinearMemory' : undeclared identifier
```

In 1.2 the class factory has been modified to optimize memory allocations for entities. For each behavior you will need to choose an allocation policy. We have provided two allocation policies: LinearAllocationPolicy and DynamicAllocationPolicy. We suggest the LinearAllocationPolicy as this will minimize fragmentation and the overhead of an allocation per class instance. To fix the compiler error you need to derive the behavior from one of the allocation policies. That is:

```
class MyBehavior : public CSystemCommands, public
CEventHandler, public CAttributeHandler
{}
```

change to:

```
#include "framework/core/attributehandler/allocation
policies/LinearAllocationPolicy.h"

class MyBehavior: public CSystemCommands, public
CEventHandler, public CAttributeHandler, public
LinearAllocationPolicy
{}
```

**CAttributeHandler::m\_Debug variable has been replaced with  
CAttributeHandler::m\_Flags**

If you have code such as

```
If (CAttributeHandler::m_Debug)
{}
```

You need to change it to:

```
if(CAttributeHandler:: m_Flags & uATTRIBUTEHANDLER_FLAG_DEBUG)
{}
```

**Classes derived from CEventHandler**

If you have a class which is derived from CEventHandler yet is not a behavior (in that it is not derived from CAttributeHandler), and you want the event system to delete it when the event system is purged you will need to call RegisterForAutoDelete. The auto deletion is now an “opt in” scheme rather than “opt out”. See also related name changes in the change log file.

# Release notes for Version 1.12

---

This release provides a new refactored game framework and a number of bug fixes to the Workspace.

## **This is an update, not a complete installation**

To install this version, you must have Version 1.11 installed.

**New users:** install Version 1.11 before installing this version.

**Existing users:** do not uninstall Version 1.11 before installing this version.

## Installation

1. If you are installing from a CD-ROM:
  - a. Insert the CD-ROM.
  - b. The setup program should start automatically. If it does not, then run `Setup.exe` in the CD-ROM root folder.

If you are installing from a download:

  - a. Download the 1.12 update `.zip` file.
  - b. Extract the zipped files to a temporary folder of your choice (for example, `C:\RWS_Update`).
  - c. Run `Setup.exe`.
2. Click **Next**.

A dialog box informs you that a backup of the existing game framework console folder will be created.
3. Click **OK**.

A dialog box informs you that a backup of the custom interface `.ren` file will be created.
4. Click **OK**.

The setup program completes the installation.
5. Click **Finish**.

## Game framework

The most significant change to this release is the refactoring for the game framework to make it easier to include just those services that your game requires. For the purpose of providing the simplest RenderWare Studio framework application, a new empty framework is also provided. These are described below.

As part of the refactoring, a number of framework and documentation bugs have been fixed.

### Refactored and empty frameworks

RenderWare Studio consists of two parts:

- The Workspace that runs on the PC  
and
- The game framework that is the source code that developers are expected to use to create their game

The game framework projects contain many toolkits and behaviors that can be confusing to developers when trying to extract the code they actually want to use. Dependencies within the core have also caused problems. Based on the feedback from existing developers, for version 1.12 we have refactored the game framework and created an empty framework.

By refactoring the game framework, we have solidified the core components, reduced the dependencies within the core, and moved the less generic components into toolkits. The refactored framework makes the distinction between the core, toolkits, and behaviors much clearer and allows developers to select just the components of the framework that they need much more easily.

In addition to this, rather than providing a large code base and expecting developers to strip this down to what they need, we have also provided the empty framework. This is a much smaller code base providing just enough functionality to boot up the consoles and connect to the Workspace. The game framework can be considered a specialization of the empty framework for the purpose of demonstrating a wide range of game behaviors; it is also a good starting point for getting a game up and running very quickly. For developers already using the game framework, upgrading to the refactored framework should be fairly painless, depending on how much the core of the framework has been modified. Details of the changes can be found in the change list installed in the framework directory, but an overview is provided here.

For behaviors, the majority of the changes required will be to the include paths (for example, in `FPSRender.cpp`) as well as a few class and function name changes.

```
#include "../../../../../framework/core/Macros/debugmacros.h"
#include "../../../../../framework/core/MainLoop/render.h"
#include "../../../../../framework/core/DebugTools/DebugTools.h"
#include "../../../../../framework/core/World/CRwCamera.h"
#include "../../../../../framework/core/World/CRpWorld.h"
```

are now:

```
#include "framework/core/macros/debugmacros.h"
#include "framework/mainloop/render.h"
#include "framework/toolkits/debugtools/debugtools.h"
#include "framework/toolkits/world/clevel.h"
#include "framework/toolkits/world/helpers/worldhelper.h"
```

As you can see from the paths some of the components that were part of the core are now toolkits; also, the projects have been set-up with a path to the source folder, eliminating the need for complex paths.

### New framework installation issues

During installation you will be asked to change the location of the existing installed game framework source. This is to prevent possible overwrites of changes developers have made to their local copies, and also provide a point reference to difference against the new framework. New users who are using RenderWare Studio for the first time can ignore this, as they will not have any modifications since earlier installations. It is recommended that the installed game framework is copied to a folder that is not under the installation folder.

### SN-TDEV for Nintendo GameCube users

This release of RenderWare Studio supports the SN System's implementation of the T-DEV device. See the procedure for [Connecting](#) (p.214) RenderWare Studio to an SN-TDEV.

### Broadband adapter support for GameCube

Support is provided for the GameCube broadband adapter. Apart from a good performance boost in connection times and design-time interactions, using the broadband connector means that the GameCube Commserver application is no longer necessary.

### Area Trigger behaviors

The new framework provides some simple area triggers and an [example](#) (p.517) showing how they are used.

## Workspace changes

While there are no significant new features in Workspace, a few usability changes and a number of bugs have been fixed.

### Entity visibility persisted between sessions

The visible/hidden state of entities in the 3D edit view is now saved in the user's local game settings. Note that the information is not saved to the game database.

### Alienbrain changes

Extra checks and validations provide better feedback with alienbrain. The speed of check-ins has also been considerably improved.

### Improved diagnostics

The [Message Log window](#) (p.297) provides better feedback on what is happening in the system, making it easier to identify problem assets and entities. This also helps to identify instabilities related to viewing asset thumbnails in the [Assets window](#) (p.264). For any persistent asset-related problems, contact Developer Relations for assistance and the latest information on solutions available in the RenderWare Graphics active build.

# Release notes for Version 1.11

---

This release is a minor release that fixes some bugs and works with version 6.0 SR1 of NXN alienbrain.

## Installation

Users should uninstall previous versions of RenderWare Studio before installing 1.11.

## Changes

### **Bugfix - Windows XP Install**

The software now installs under Windows XP correctly.

### **Bugfix - Failed load causing all subsequent loads to fail**

If the last file being parsed contained a <Deleted> tag, then subsequent loads would fail. Fixed.

### **Bugfix - Workspace graphics crash with textured wire frames**

In some circumstances a crash could occur in the workspace when one of the ortho view display modes was set to textured wire frame. Fixed.

### **Bugfix - Intermittent Workspace crash**

The fix to the crash caused by the vector attribute control that was previously available with a post 1.1 patch has been included in 1.11.

### **Bugfix - Files missing for Maestro exporter**

The installer has been updated to include texdict.c and texdict.h, which were missing from the 1.1 install.

### **alienbrain 6.0 SR1**

alienbrain integration is now linked to 6.0 SR1 (124) NxN libraries.

Users are advised to upgrade alienbrain servers and clients to 6.0 SR1.

### **Updating to RenderWare Graphics 3.5**

RenderWare Studio has been tested against RenderWare Graphics 3.4. However, to enable users who upgrade to later versions, RenderWare Graphics 3.5 will now install compatible versions of the RenderWare Studio components required to handle files exported from the latest builds of RenderWare Graphics. This means that customers will no longer have to wait for an update to RenderWare Studio to handle the Active Graphics releases.

The game framework has been modified to compile against beta versions of RenderWare Graphics 3.5 (in addition to 3.4), and so will require a minimum of modification (if any) for when the final release of 3.5 is made.

# Release notes for Version 1.1

---

Version 1.1 contains many changes since Version 1.0. This topic outlines installation issues, new features, and information for users upgrading from earlier versions of RenderWare Studio.

Please refer to the rest of the documentation for more detailed descriptions of the new features.

## Installation

### **Upgrading from earlier versions of RenderWare Studio**

Existing users should uninstall previous versions of RenderWare Studio before installing Version 1.1. Information on game database changes and framework code changes can be found below.

### **NXN alienbrain 6**

This version of RenderWare Studio will work with version 6 of NXN alienbrain. Existing users of NXN alienbrain 5 must upgrade to version 6 before installing RenderWare Studio Version 1.1.

### **Fixed period trial license**

The licensing mechanism has been changed to install a trial license for a fixed period after installation, rather than timing out on a fixed date. This allows new users to get started with the software immediately. Please note, however, that you should still apply for a full license in the normal way in order to use the software after the trial period, which is set at 10 days.

When the license is a few days away from expiring, a message will appear every time you run Workspace. You should use this as a reminder to apply to CSL for a full license using the License Manager application supplied.

Please note that attempting to change the clock on your PC will not work, and in fact may prevent RenderWare Studio Workspace from starting at all until a full valid license is installed.

When you do receive your license, simply copy it into the Programs folder.

## Workspace

### **Shared attributes**

Shared attributes provide an elegant and natural solution to many games design problems, such as changing universal parameters in a single operation. RenderWare Studio uses attribute collections to provide a powerful mechanism of sharing sets of attributes among many entities.

### **Custom platform flags for build and console link**

With 1.0, data streams sent to the console could only be customized with some hard coded flags which differentiated the platforms (e.g. PS2, Xbox etc). Release 1.1 extends this by allowing the flags to be customizable. When sending data to the console, objects are only sent if a target's flags are a subset of the entity or asset.

### **Validation of attributes and behaviors**

Class definitions change as a game evolves. For example, attributes may be added, removed, or have their types change. To prevent the data in the game database getting out of step with the game source code, a new

database integrity check feature has been added.

### **Multiple source directories for source code parsing**

Multiple directories can be specified for parsing behaviors from by using a semi colon to separate each path. In addition the parser has been enhanced to cope with comments and split lines.

### **Improved texture dictionary creation**

Texture dictionary creation has several enhancements including bump and environment mapping support, and ensuring that only textures referenced by the target platform are added. The texture dictionary generated is per level now rather than per game, so it only includes textures referenced by the selected level.

### **Import levels**

A level can be imported into a game database from another game database. Users who have databases from earlier versions of RenderWare Studio should read the section below on upgrading their game databases if they wish to use this feature.

### **Enhanced 3D interactions**

When moving entities around within the 3D view in Workspace, actual values and changes are displayed in a numerical output toolbar. You can also click on these numbers and type new values for more accuracy. In addition, the 3D picking has been split into three distinct modes - translate, rotate, and scale, rather than the two mixed modes that were in release 1.0. New toolbar icons have been created to select the mode, and users can cycle through the modes using F10 or by double clicking on the pick handles. A new option allows object dragging in either world or local coordinate systems.

### **Customisable flight keys**

Users can change the keys for flying a camera around a scene. See the Options menu.

### **Capture and display of console metrics in Workspace**

The framework can link to a dynamic display in the Workspace of metrics information generated by the game running on the console. Users can send their own data up the link and it will be displayed in the same graphing tool.

### **Hide objects in Workspace**

Users can hide objects in the Workspace view to help remove clutter from complex scenes. There is also a new display mode for the edit view that can show an untextured view of the world.

### **Enhanced layouts**

Based on user feedback the layouts have been subtly changed and extended. Extra toolbars have been added for common operations.

### **Easy creation of multiple entities**

A new feature has been added to the drag and drop method of creating new entities to make it easy to create many entities. While dragging a template or asset in the Design View, pressing the space bar will create a new entity. This provides a way of "stamping" down many new objects. Users may want to take advantage of the active collection facility to group together the new entities.

### **Large databases**

Several changes have been made to improve the handling of large

databases (of the order of thousands of entities and multiple levels). In addition, new user interface features such as the ability to hide objects from the edit view, new display modes, and clip plane control can also be exploited to help performance.

## Framework

The following list is a fairly high level view of the changes made to the game framework. For more detail see the target\_changelog.txt file installed with the software.

### **Improved particle behaviors**

The particle behaviors have been enhanced to provide texturing support and better performance. In addition, the classes have been refactored to make it easier to customize particle systems using derived classes. The CFX\_PartSpray behavior has been updated with new controls to allow use of the texturing and to allow rendering mode and blend modes to be specified.

### **Improved audio behaviors**

The exploitation of RenderWare Audio has been enhanced to ease the integration of audio content in to a game. It is now possible to use events to fade the output device and individual voices. Notification events are sent when the fade has completed. The reverb has been factored out of the AudioGlobalMixer behavior allowing multiple reverbs around the world which can be triggered by events. The built in default reverb types can be derived from to create custom types. The AudioSound3D behavior has been enhanced to utilize features from RenderWare Audio 2, like virtual voice priority, virtual voice bias and Min/Max distance attenuation. A new AudioGroup behavior can put Wave Dictionaries into separate sound groups. Using the AudioGroup behavior can fade sounds in these groups. Debug flags have been added to AudioSound3D, AudioGlobalMixer and AudioReverb in order to view all parameters and channel strip details on the PS2.

### **Support for RenderWare G3x pipelines**

A set of behaviors have been written that encapsulate the high speed G3x pipelines in RenderWare 3.4 and later. Use of these can make significant performance improvements on the Playstation 2 platform.

### **Support for MatFX plugins and vertex shaders**

New behaviors exploit the RenderWare MatFX plugins for effects such as environment and bump mapping. This mechanism also applies to vertex and pixel shaders where relevant for each target platform. See the environment map [example](#) (p.514).

### **Skydome behavior**

FPSSkyDome has been supplied to provide skydome rendering support to games. See the sky dome [example](#) (p.514).

### **Translucency sorting**

FPSRender can now handle sorting of atomics for displaying objects with translucent surfaces.

### **Support for ATWINMON on PS/2 Test Station**

Users of the Playstation 2 Test Station can use Sony's "ATWINMON" utility to load executables from the host PC, rather than having to cut new CDs every time a change to the software occurs. It is also possible with this utility

to load files existing on the host PC's file system.

### **Support for SN Systems profiling module**

For ProDG users the game framework has been modified so that it can initialise the profiling module. The information can be viewed in the ProDG debugger's profiling window.

### **Generalized mechanism for sending data from console to Workspace**

New framework services provide the ability to send data from the game into the Workspace.

### **Ability to create files on PC from console**

A new panel has been added to the workspace that allows streams sent up from the game console to be saved to any location on the disk. This feature exploits the feature described previously for sending data from the console to the Workspace.

## **Documentation and Samples**

The documentation has undergone a number of changes and additions. Key changes are detailed below.

### **Attribute editor customisation**

The mechanism for users to create and integrate their own attribute edit controls into the Workspace is now documented.

### **Custom drawing in edit viewer**

New documentation has been added describing how to exploit the 3D View Extensions interface, which provides the ability to add your own graphics into the edit view.

### **Manager API documentation**

The RenderWare Studio Manager API documentation has been extended to include more description and background. There are also code samples for common operations such as accessing a database, creating entities, sending data to and receiving data from the game.

### **New tutorial**

A new tutorial has been created that shows how to use the framework without making use of the smart pointers. The idea of this is to help users less familiar with C++ and those who require more explicit visibility of object lifetimes.

### **Samples**

A number of new samples are provided to show some of the new framework features such as alpha sorting, audio, motion blur, pixel and vertex shaders, G3 pipelines, shared attributes, and sky domes.

## **Updating from Release 1.01**

This section describes any special steps to take if you are updating your version of RenderWare Studio from version 1.01 to 1.1.

## **Game Database**

Users of RenderWare Studio with databases created before version 1.0 may find that they have objects with IDs such as "Asset1", "Entity6", etc. Such IDs were also used in example data supplied with 1.0 and 1.01. To ensure future compatibility with RWStudio tools, and the newly introduced "Import level"

feature, it is recommended that you run the GUIDEnforcer tool over your data. This will convert all objects with non-GUID IDs and update all references, renaming files where appropriate. For further information on the use of GUIDEnforcer please see the accompanying help.

## Game Framework

Below is a list of the changes that will have the greatest impact on developers upgrading from 1.01 to 1.1 i.e. those that may require changes to their own code or projects.

### Handling missing behaviors

Modified ClassFactory to issues warnings if a behavior is requested that is unavailable on the target platform. Should you get such a warning then select properties on the entity in the workspace and select the target platform, or add the missing behavior to the project. The reason for the warning is that it helps with debugging when working with multiple platforms or projects.

### logical.h

Removed logical.h that means that RWS\_implies and RWS\_iff are no longer defined.

### macros.h

The file:

core/macros/macros.h

has been moved and renamed to:

core/CAttributeHandler/attributemacros.h

CAttributeHandler.h includes attributemacros.h so there should be no need for a behavior to include it specifically. CAttributeHandler/attributemacros.h contains the definitions of the macros used for attribute tweaking i.e. RWS\_BEGIN\_COMMNDS, RWS\_ATTRIBUTE etc.

### Renamed function

The function:

```
void ClumpHelper::ClumpRenderVisible(RpClump * clump);
```

has been renamed to:

```
void ClumpHelper::RenderAllVisibleAtoms(RpClump * pClump);
```

### Karma Initialization

CKWorldObject::SetWorld() method added to enable you to set up the collision world for Karma once the world is loaded. This used to be done in the constructor for CKWorldObject but this is called before the world is loaded. SetWorld is now called from CKarmaParams.

Use the CKarmaParams behavior with a world asset to specify the collision world to use with Karma.

### New particle system

As described above, there is a new textured particle system. Basically, rewritten the particle system to allow more code reuse on derived systems (i.e. the textured system). Supports various rendering modes (to remove interparticle and or intersystem problems) and now allows blend modes to be modified. Texturing can be animated using sub-textures.

## BugFix/Enhancement in CResourceHandler

The following code:

```
virtual void *Load(const RwChar *psName, const RwChar *psType,
const RwChar *psResourcePath, RwStream* pStream, RwUInt32
uiStreamSize) = 0;
```

has been changed to:

```
virtual void *Load(const RwChar *psName, const RwChar *psType,
const RwChar *psResourcePath, RwStream* pStream, RwUInt32
uiStreamSize, RwUInt32 &uiResourceSize) = 0;
```

This enables a ResourceHandler to "return" the size of the resource that has been loaded which is stored by the resource manager within the CResource class for each resource.

## Handling transparency

FPSRender now supports depth sorting of atomics, this is useful for sorting translucent atomics. CRpLevel has been extended to contain by default two RpWorlds - an opaque world and a translucent world. CRpLevel also provides functionality for attaching a hint to an atomic or clump which indicates which world they should be added to. A new behavior CSetCRpLevelHint is provided which can set this hint value.

In order for CRpLevel to add atomics/clumps to the correct world CRpLevel provides the following RpWorld equivalent functions AddAtomic, AddClump, AddLight. For a detailed description of CRpLevel please refer to the help files.

A new alpha sort [example](#) (p.514) has been provided which demonstrates three ways to setup a world and make use of the depth sorting functionality of FPSRender, and RpWorld management of CRpLevel.

The code in CEntity to move objects from one world into the new world has been removed as a result of the changes to CRpLevel. Worlds must now be created before objects are added to them.

**Important note:** entities that need to be added to the world must be created after the world is created (CRpLevel::SetOpaqueWorld() being called), i.e. if using FPSRender this should typically be the first entity in the hierarchy list shown in the workspace.

## CAnimSet change

Extended CAnimSet to store the name of the animation.

CAnimSet::AnimData structure now has an extra member:

```
const RwChar *pName_;
```

The constructor for AnimData now takes an extra param which is the name of the animation:

```
AnimData( const RpHAnimAnimation* pRawData, const RwChar*
pName );
```

CAnimSet::AddAnimation gets this name from the animation resource.

To use the animation name with CAnimSet call:

```
CAnimSet::GetDstAnimationByName:
RwBool GetDstAnimationByName(const RwChar *animName, RwInt32
*animIndex);
```

This method searches CAnimSet::animData\_ and if the name is found returns TRUE, otherwise FALSE. animIndex stores the index of the animation if found. The index can then be used to set the desired animation in the usual way via CAnimSet::SetDstAnimation. The animation asset in RenderWare Studio needs to have the 'Send Name to Target' check box ticked in order to send the name with the Resource which is picked up in CAnimSet::AddAnimation.

### **DebugTools**

Refactored DebugTools.h and DebugTools.cpp, extracted code from DisplayDebugTools::Poll to create:

```
void DrawLines(void);
void DrawEllipses(void);
void DrawPolys(void);
void Draw3DText(RwCamera *pCamera);
```

The following methods:

```
extern RpLight *Show_Light (RpLight * light, void *data);
extern RpWorldSector *Show_Sector(RpWorldSector *worldSector,
void *data);
```

Have been renamed as:

```
extern RpLight ShowLightCB (RpLight * light, void *data);
extern RpWorldSector *ShowSectorCB(RpWorldSector *worldSector,
void *data);
```

### **Platform-specific startup code**

The startup folder in core now has platform-specific code with sub-folders:

```
startup/win32
startup/gcn
startup/xbox
startup/sky
```

### **HandleSystemCommands**

HandleSystemCommands for CatomicPtr, CclumpPtr, and ClightPtr are now implemented in AtomicHelper, ClumpHelper and LightHelper so that they can be used without using CatomicPtr etc. I.e.

CAtomicPtr::HandleSystemCommands calls  
AtomicHelper::HandleSystemCommands(RpAtomic \*pAtomic).

### **Restructured startup folder**

Startup folder within core has been restructured to now contain sub folders for each platform.

### **Filename case convention**

All source files for the framework are now lower case, and all #include references to them have also been changed to ensure lower case. This also applies to the folder names.

# Release notes for Version 1.01

---

The main feature of Velerse 1.01 is support for RenderWare Graphics 3.4. The only other changes are further documentation and bug fixes, described below.

## Support for RenderWare Graphics Version 3.4

RenderWare Studio no longer supports versions of RenderWare earlier than 3.4.

### CodeWarrior users

You might have to change the values of the environment variables RWGSDK and RWASDK defined by the RenderWare 3.4 setup program. The setup program uses forward slashes ('/'), but CodeWarrior expects backslashes ('\'). To change these values:

- Click **Start ▶ Settings ▶ Control Panel ▶ System ▶ Advanced ▶ System Environment Variables**

### SN ProDG for PS2 users

Before building the supplied project, select the Group libraries option. This enables ProDG to correctly link the project.

For example, if you are using the Visual C++ integration:

- Select the **VSI Control Panel** (typically shown on the toolbar next to **Run PS2 Elf**), and then select **Group libraries**.

## Licensing

This install comes with a license that will expire on 1 October 2002. To continue to use the product after then you need to use the supplied License Manager application

## Revised lightmapping sample

The lightmapping sample has been changed slightly so that the lights used to generated the lightmap are in a similar position to those modelled in the scene. The sample also uses colored lights.

## Bug fixes

- RWSMemory.cpp and Karma classes fixed to prevent potential crashes when using the memory diagnostic tools.
- The ProDG project files for Gamecube are now included in the install.
- Save now removes deleted objects when RWStudio used without NXN alienbrain.
- Project saves without NXN alienbrain installed no longer generates spurious error message.
- Save of write-protected project without NXN alienbrain now behaves correctly.
- View mode is now saved in the settings file.

- Resource root is now saved correctly for a new project.
- alienbrain status now shown for objects that are in the database but not yet in the game.
- Object selection now cleared properly when another object selected.

# Release notes for Version 1.0

---

This page details the major changes since Release Candidate 1 (RC1).

## Installation

You should uninstall any previous versions of RenderWare Studio before installing. Users wishing to be able to customize the Workspace should select the Custom Install option when prompted. Alternatively after a standard install, you can add optional components using Add/Remove Programs from Control Panel.

Any existing game databases created with versions of the software earlier than RC1 will need to be upgraded before they can be used. See below for details. RC1 databases will not require any modification to work with the final release.

This version of RenderWare Studio will only work with RenderWare 3.3.

If NXN alienbrain is installed prior to RenderWare Studio then Studio's integration will be enabled automatically. If you install alienbrain after Studio then you will need to rerun the install using **Control Panel ▶ Add/Remove Programs**, select RenderWare Studio Change/Remove, choose the Modify option, and then select the Alienbrain component.

## Changes since RC1

The majority of changes since RC1 have been bug fixes and cosmetic changes, with only a few functionality enhancements.

### **Support for left-handed mouse**

Workspace now correctly uses the global Windows setting for left handed mouse settings.

### **Direct save to NXN alienbrain**

You can now save a project directly into alienbrain, rather than having to use the alienbrain client.

### **Visual indication of NXN alienbrain status**

Items in the game database and asset windows show the status of that asset in alienbrain such as check out status, and whether the file is managed or not.

### **Workspace settings persisted between sessions**

Workspace settings such as current window positions, layout, and auto load of last file are now persisted between sessions. You can modify what gets saved from the **Options ▶ Workspace** menu.

### **Orbit center set to current object automatically**

In RC1 the orbit center was only set to the selected object when you used the Fit option (**F3**). This has been changed so that it follows the last selected object around.

### **Drag object in plane**

Double clicking the drag handles on a picked object will change the handles so you can drag the object in any of the three major planes (XY, YZ, XZ).

Double click again to change back to rotation handles.

### **PS2 Broadband support**

Support for the Broadband Ethernet adapter (DTL-H20400) via ifc001.cnf and dev001.cnf files. To switch to this configuration, edit net000.cnf, and uncomment the relevant section.

### ProDG support for GameCube

Project files for compiling the samples using ProDG for Gamecube GDEV are now provided. These projects require Dolphin SDK 1.0, 12 December 2001 patch level 4 (or later); they will not link with earlier versions. The framework for the Gamecube DDH has not been tested.

## Framework changes

Programmers should look at the framework change log for technical details on all the changes to the framework code.

### Change to FX lights

Between RC1.0 and RC2.0 the hierarchy of the FX lights classes was changed. This means that any old databases that use the FX light classes will need to be updated. There is no quick and easy way to update a database to fix the problem - you have to go through each light and set up the parameters again.

## Documentation

The documentation has been further enhanced. The main help file available from the Help menu in Workspace provides a single point of access to the four separate RenderWare Studio help files:

- Workspace (also supplied as printer-friendly Acrobat PDF)
- Game Framework API reference (online Help only)
- Manager API reference (online Help only)
- Enterprise Author (also supplied as printer-friendly Acrobat PDF)

We value all feedback on the documentation; please use the envelope icons at the bottom of each page to provide direct feedback to our technical authors.

# Release notes for Version 1.0 Release Candidate 1 (RC1)

---

Release Candidate 1 contains nearly all the functionality of the final 1.0 release. This page details the major changes since the last development release (RWSDevRel 1.08a). Users should also read the change log files that get installed with this release for a more detailed view of the changes.

## Installation

You should uninstall any previous versions of RenderWare Studio before installing RC1. Users wishing to be able to customize the Workspace should select the Custom Install option when prompted.

Any existing game databases created with previous versions of the software will need to be upgraded before they can be used. See below for details.

## Game database

The game database has been substantially enhanced since the last release. Refer to the documentation for details of the structure, which includes an entity relationship diagram that shows how different parts of the database relate to each other. It also describes how the file structure is persisted.

### .rwstudio file extension

The extension for the top level game file has been changed from .xml to .rwstudio.

### Collections

There are two types of collections - one for assets and another for entities. This allows you to arbitrarily group your data to suit your own uses.

### Templates

With entity templates you can set up a single entity's behavior, assets, and parameters, and save that as a template object. This template can then be used to easily create multiple instances of that entity.

### Database upgrade tool

Any existing games databases from earlier versions of RenderWare Studio will need to be upgraded. An automatic tool called RWSDatabaseConverter.exe is provided in the bin folder of your RWStudio installation. On running this application you will be asked to provide the location of the database to convert. You have the option to create a new 1.0 version in a different location. Upgrading is a safe operation as none of the original data will get overwritten by the conversion process.

### Integration with alienbrain

Support for alienbrain now includes the capability to check in changes from the RWStudio Workspace, and to synchronize the latest version of a game with the central alienbrain server. Support for reverting edits is also provided. Users are informed if they attempt to modify parts of the game locked open by other users.

### Tray dog icon no longer required

The “dog” icon that appeared in the system tray when a database was loaded has been removed. The database server is no longer a separate application, but has been moved to be in-process to provide better performance - particularly on large datasets.

## 3D navigation

The mechanism for users to navigate around their scenes within Workspace has been radically enhanced. In addition to the existing Camera Flight mechanism, an alternate user interface has been provided that will be more familiar for artists to use. The documentation describes how to use this new feature. To support the new modes for easy navigation and inspection, a number of hot keys have been added. For example, F3 will fit the camera to the currently selected object, and will also reset the orbit centre. If the 'O' key is held down while picking in the view, then the orbit centre will change appropriately.

## Workspace user interface

A large number of changes have been made to the user interface. Some of the key changes are described below.

### Keyboard accelerators

Windows like Alt-key menu accelerators are now fully supported. Note that if you customize RWStudio Workspace using the Author tool, you can also gain access to the accelerators and modify or extend as needed. In addition, there are a number of hotkey accelerators for common operations. See documentation for details.

### Menus

The menus have been restructured based on feedback from customers. Some features (such as debug tools) that used to exist as panelled dialogs have now moved to menu options for ease of access.

### Lister windows

The underlying implementation of the lister windows (that show entities, the game tree, assets etc) has been radically enhanced. You can now change the order of items in the list (using ALT+arrow keys), group and sort objects. All listers are customizable to show different views on your data - use the right button on your mouse both within the view and also on the column headers.

### Entities without assets

It is now possible to add entities which do not require assets directly into the game level. This will reduce the need to have so many dummy objects in your level that simply act as placeholders for code.

### Full Design Views

From any layout a user can expand the 3D views to fill all of the window, and then toggle back again. This provides an easy way of hiding all the user interface windows when inspecting a scene. Use the F11 key or the **Views** menu.

## Framework changes

Programmers should look at the framework change log for technical details on all the changes to the framework code.

### Category and description macros

Programmers can add categorization and descriptions of their behaviors that is picked up and displayed in the Workspace. The former provides a nice grouping mechanism when used in conjunction with the customizable column headers in the new behavior lister.

#### **Added iMsgDoRenderDirectorsCamera**

Instead of camera/rendering behaviors checking whether the directors camera is enabled, using  
`if(DebugSwitches::GetSwitch(DebugSwitches::m_enable_DirectorsCamera) == true)` this is done once within render.cpp. The decision as to when to render is made by either linking to iMsgDoRender or iMsgDoRenderDirectorsCamera.

#### **Refactored CLight**

Updated sample XML files containing references to CLight, to use CFXColorLight instead (same parameters, same names, same order). Needed as CLight has been deleted from the example behaviors. NOTE: If you are already using the CLight behavior in your game your .xml files will need to be updated to use a different light e.g. CFXColorLight. Otherwise there will be no light in the game.

#### **Access to asset name**

Code added to resource handler and network layer to receive name of assets in network packets. Added FindByName to resource handler as a helper.

#### **Enhanced communications layer memory handling**

Dynamic memory allocations removed from network code by reimplementing as custom RwStreams to read or write network data.

#### **Memory handling functionality**

Added memory handling functionality. System now tracks allocations and frees. Reports leaks, boundary violations and total/peak usage. The code is included via the preprocessor headers and is therefore in all files. Only requirement for use is that the 'RWS\_NEW' macro is used instead of 'new' and 'RWS\_OP\_NEW' is used instead of 'operator new' or 'RwMalloc'. Data is written to the debug console log.

#### **RenderWare Audio 2.0 support**

The audio toolkit and sample behaviour has been updated to work with RenderWare Audio 2.0, which should be released around the same time as RenderWare Studio 1.0.

#### **Added iMsgRequestCurrentCameraMatrix event, and SendTransformToWorkspace method**

The workspace can request a camera matrix from the console, this is used to locate the editor when the camera on the workspace is being controlled by the game. In order to extend this so that multiple views can be used on the console the iMsgRequestCurrentCameraMatrix event has been created so that any behaviour can choose to send its transform back to the workspace using the SendTransformToWorkspace. FPSPlayer and CCameraLookAtPoint have been modified to take advantage of this facility.

#### **New CAttributeHandler functionality**

There are new methods and a macro in CAttributeHandler:-

- CAttributePacket \*Clone(void) const
- CAttributePacket \*Update(const CAttributePacket &rAttr)

- macro RWS\_IMPORT\_COMMANDS

This allows you to create a class factory which easily instance a specific behaviour with a specific asset and attributes on the console at run time, using the CAttributePacket data as if the workspace had created it.

## Documentation

The documentation has been substantially enhanced, and is now provided as compiled HTML files. The main help file available from the Help menu in Workspace provides a single point of access to the 4 separate books - Workspace, Framework API reference, RWStudio Manager API reference, and Enterprise Author user guide. Separate printer-ready PDF versions are also installed on your disk.

We value all feedback on the documentation - please use the envelope icons at the bottom of each page to provide direct feedback to our technical authors.

## Features missing from release 1.0

The following features will be included in the full release:-

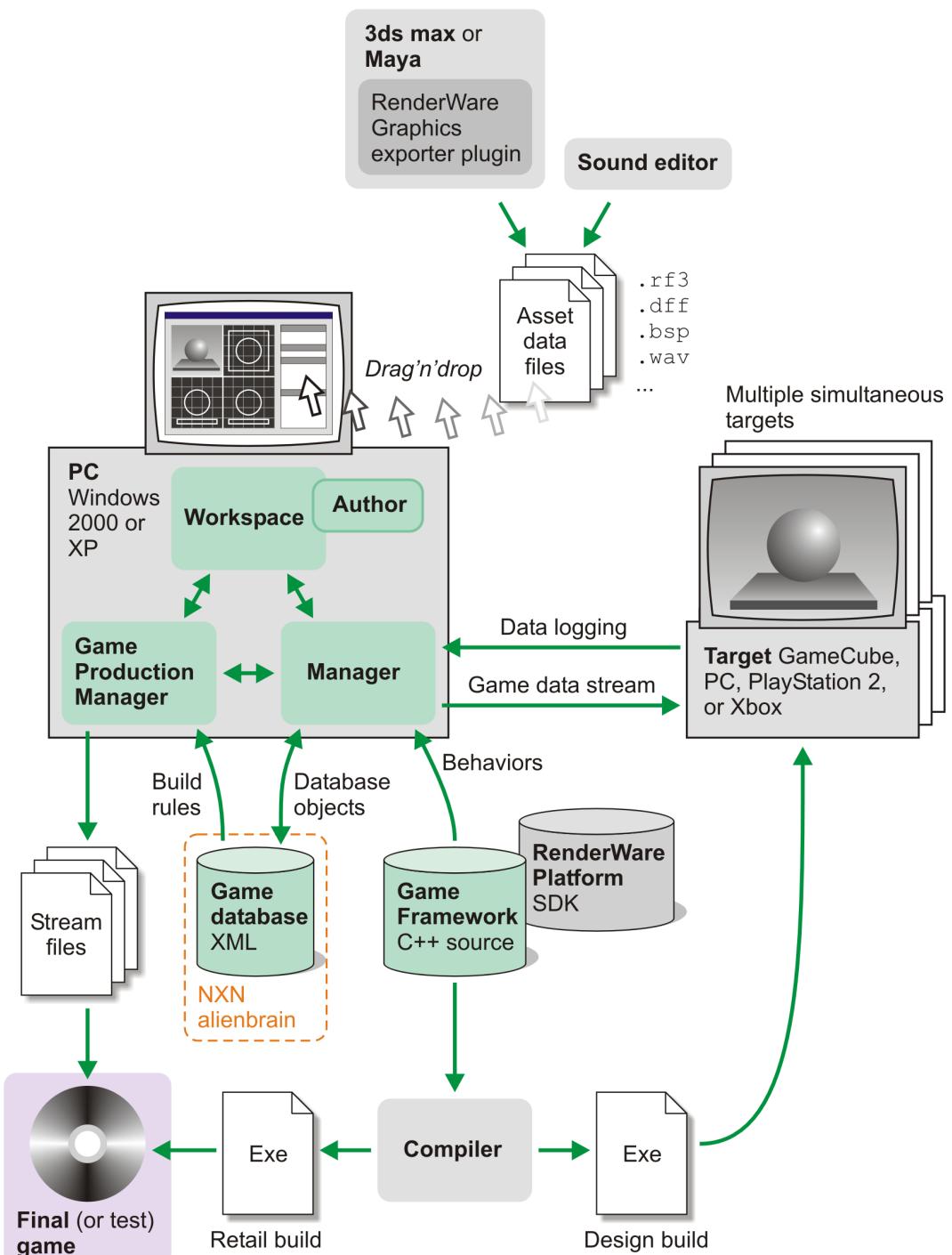
- Miscellaneous cosmetic changes to the user interface, including new icons, window titles, usability enhancements.
- Enable direct save to NxN alienbrain (currently available only via the NxN alienbrain client).
- Visual indication in the Workspace of check out status of items.
- Directors camera reset on undo.
- Ensuring all desktop settings are persisted between sessions.
- Copy/paste mechanism extended to Design Views for an easy way of dropping lots of objects into the scene.
- Further enhancements to the 3D edit interface (e.g. drag objects in a plane).
- More complete Manager API documentation.
- Windows file association or .rwstudio files with the Workspace executable.

The main focus of the period between RC1 and full release is the fixing of bugs, and more documentation. Customer feedback is welcome.

# What is RenderWare Studio?

RenderWare Studio is an integrated, open and extensible suite of tools that allows the entire game development team to work together in a collaborative, real-time environment. It's built upon a framework that empowers you to develop gameplay as soon as your game assets become available.

Driving the RenderWare suite of technologies, RenderWare Studio provides a real-time communications link to all target platforms (PlayStation 2, Xbox, Nintendo GameCube, and PC), automatically generating and managing the native file formats.



RenderWare Studio consists of:

### **Workspace** (`RWStudio.exe`)

The Windows-based graphical user interface of RenderWare Studio. You can use Workspace to design your game, and immediately see your changes reflected in the running game on one or more targets (connected to the Workspace PC via an Internet Protocol network).

The Workspace user interface is a set of ActiveX controls displayed as dockable child [windows](#) (p.238) inside the Workspace application window. (The Workspace application window itself contains a single ActiveX control—a “docking frame”—that is responsible for displaying the child windows.)

Workspace was developed in a Visual Basic-like application development tool called **Enterprise Author** (or just “Author”). RenderWare Studio supplies both Author and the Workspace source files, allowing you to [customize](#) (p.80) Workspace; for example, by adding your own ActiveX controls.

### **Game Framework**

A set of C++ classes that you use to develop your game with RenderWare Studio. The Game Framework is supplied with RenderWare Studio as source code. It is designed to be incorporated into your own game code from the earliest “proof of concept” through to the final commercial game.

The Game Framework includes C++ classes known as *behaviors*. These define what each of the different kinds of [entity](#) (p.71) in your game can do. To identify a C++ class as a behavior, you annotate its header file with `RWS_` macros. Workspace parses the Game Framework source for these macros, and uses them to build a list of behaviors, and also to display the user interface for editing behavior [attributes](#) (p.270).

RenderWare Studio supports various [C++ development environments](#) (p.82), depending on your target platform.

### **Manager**

A set of functions for maintaining the game database and communicating (via a network connection) with targets. The Manager offers two programming interfaces:

#### **C API**

Used by the ActiveX controls in the Workspace.

#### **COM interface**

Used by the Workspace source files and by the Game Production Manager build rules. This COM interface is a “wrapper” for a subset of the C API functions.

Workspace is one example of an application that uses the Manager; you can use these interfaces to incorporate the Manager in your own applications.

### **Game Production Manager (GPM)**

A set of COM interfaces for creating production-quality, game-ready data. To use the GPM, you write build rules that define the relationships between your “source” files (such as asset data files) and the file formats required for the game (such as “stream” files).

RenderWare Studio is supplied with build rules for common asset types, but you can easily customize these or write your own build rules to handle new asset types.

### Game database

All information about the game (except the Game Framework C++ source) is stored in a database of [XML files](#) (p.76).

RenderWare Studio supports version control and team development of game databases with the integrated NXN [alienbrain](#) (p.187) asset management tool (supplied with RenderWare Studio).

## Importing artwork or audio assets

To import your 3ds max or Maya art assets into RenderWare Studio, use the RenderWare Graphics exporter plugin to create .rf3 files (or the legacy .bsp or .dff file formats). For other assets, export the appropriate file types (for example, .wav for audio). Then drag one or more of these asset data files from Windows Explorer to the Assets window in Workspace.

When you drag an asset data file to the Assets window, Workspace (via the Manager) creates a new asset in the game database. This asset database object stores the relative path of the asset data file (relative to an absolute path known as the *resource root*).

To create a new entity in your game, you drag an asset from the Assets window to the Design View window (that displays a 3D view of your game). You can then attach a behavior to the new entity by dragging the name of a behavior to the entity.

# What does RenderWare Studio do?

---

- RenderWare Studio takes the **designer ▶ programmer** loop out of the edit cycle by allowing real-time viewing on the console of changes made in the editor. The programmer can test new behaviors before passing them to the designer.
- RenderWare Studio also cuts out the need for the **games designer ▶ programmer ▶ tools developer** cycle, as RenderWare Studio dynamically generates the user interface for new behaviors.
- RenderWare Studio generates the user interface that the designer needs to set parameters and manipulate new artwork and behavior in the editor, eliminating the need for code to pass via the tools programmer to the designer. Behavior code becomes an asset like any other that can be viewed and used in the RenderWare Studio Workspace. Pre-made modules that have been tested to work within a standardized framework mean that a lot of the back and forth between the programmer and the designer is taken out of the equation. As long as the standard for interaction between objects is adhered to modules can be guaranteed to work together.
- The PC-hosted RenderWare Studio Manager ensures all new art, sound and code assets are made available to the whole team from the moment they are checked in.
- Real-time dynamics and collision give instant feedback as to how collisions and forces will react with the game world.
- RenderWare Studio deploys an intuitive “drag and drop”, object-based user interface, which facilitates the rapid prototyping and sequencing of all game events and behaviors, from camera views through to in-game characters.
- RenderWare Studio allows you to “decorate” your game code with attribute tags, from which the RenderWare Studio Manager automatically creates menus, sliders and buttons, to represent adjustable parameters within the Workspace. Example: With a particle effect you could adjust and save its color, gravity and spread. These tags are then automatically removed for a final release.
- Artists and designers can create real-time atmospherics by placing, then adjusting, lighting setting and special effects within the game worlds for each platform. Example: They can tune and save “fog” settings to appear exactly how they want it.
- With the fully integrated alienbrain™ asset management software by NXN, the entire team will be working on actual game parameters in real time, on the target console, with optimal results.
- Developers can observe when in-game parameters change via the GUI, for powerful, efficient play testing and debugging by any team member. All four run-time applications can then be generated “on the fly”, so there is always a latest build with the latest settings, for all the team to use.
- RenderWare Studio can handle source code from any version control system. Asset importer plugins link RenderWare Studio with your preferred art package. Developers can extend the coverage of any import path, as all

are open and well documented; to whatever standard or customised asset creation formats and functionality are required.

- Customizable tools can be built very quickly with RenderWare Studio, leveraging all the underlying services it provides, resulting in a phenomenally powerful game development environment from the outset.

# Basic concepts

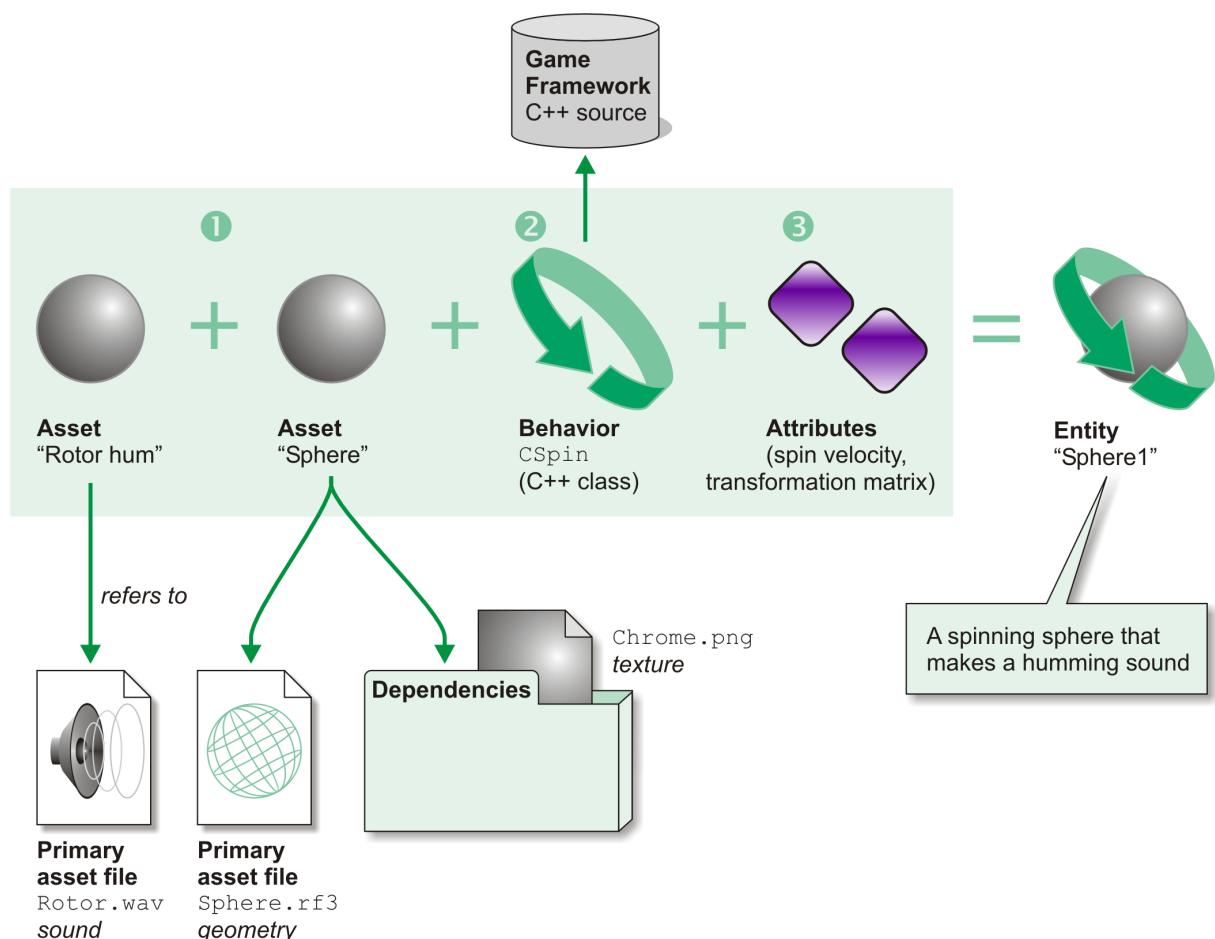
---

## A game is made up of entities

In RenderWare Studio, the objects in a game are called *entities*. Here are some examples:

- A human figure that walks or runs around the game world in response to movements on a control pad
- A button that can be pressed
- A door that slides open when a button is pressed
- A heads-up display
- A wall
- A spinning sphere that makes a humming sound (as shown in the example diagram, below).

## What does an entity consist of?



An entity consists of:

**1** References to zero or more assets.

Assets identify data such as geometry, sound, or text. Each asset refers to a *primary asset file* and, optionally, a *dependencies folder*. Typically, the primary asset file is an .rf3 file (or a legacy .dff file) that contains geometry data, and the dependencies folder contains the textures named in the primary asset file.

**2** A reference to a *behavior*.

A behavior is a C++ class that defines:

- How and where an entity appears in the scene.
- The actions that an entity can perform.
- The events that an entity can trigger.
- The events to which an entity can respond.

Behaviors are defined in the C++ source of the *Game Framework*. The Game Framework contains the game logic.

**3** Values for the entity *attributes*.

An attribute is a parameter that controls the behavior.

Many behaviors have a “transformation matrix” attribute that defines the position and orientation of the entity in the scene.

By default, attribute values apply only to a particular entity, and are stored with that entity. However, you can also [share](#) (p.170) attribute values between entities.

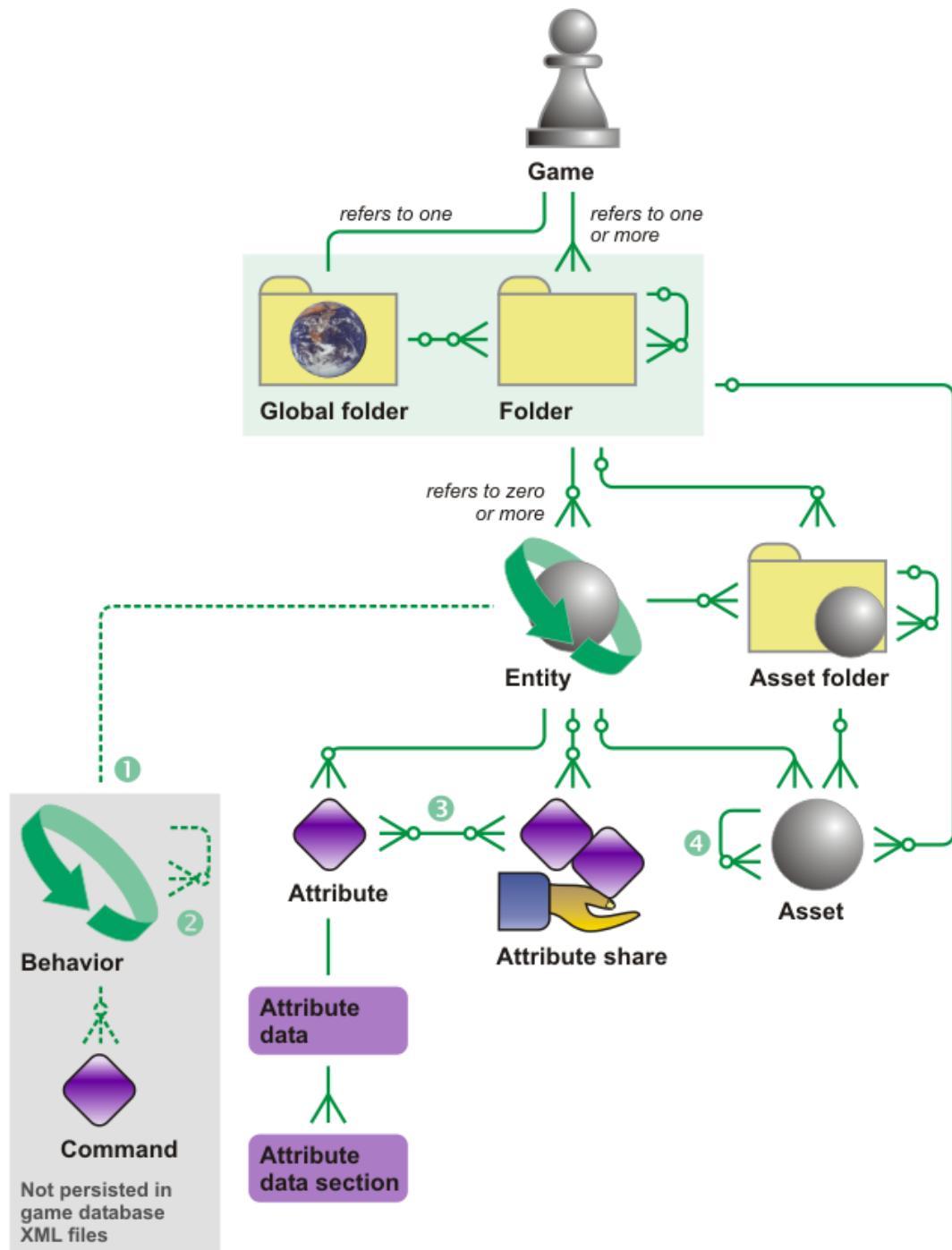
## How are entities stored?

Each entity is stored in the [game database](#) (p.73) as a separate XML file. (Assets are stored in the same way.)

The XML file for each entity contains the values of its attributes, the name of its behavior, and references to its assets. (The same behavior and assets can be referred to by many entities.)

# Game database structure

Database designers refer to the diagram below as an *entity relationship* diagram: it shows the types of object in a database, and the relationships between the objects. (In that context, *entity* is the generic term for an object in a database. However, in RenderWare Studio, *entity* is also a specific type of object.)



- 1** Except for behaviors and commands, each object in the game database is stored in its own individual [XML file](#) (p.76). The solid lines in the diagram above indicate the persistent links between these objects. An entity refers to the name of the C++ class that defines its behavior. The Workspace adds behaviors and their commands to the database when it parses the Game Framework source.

(The Workspace parses the source each time you open a project, create a new project, or click the  toolbar button.)

- 2** A child behavior refers to the base class of its parent. For example, in the game database, CSysytemCommands is a child behavior of CEntity; in C++ terms, CSysytemCommands is a base class of CEntity. This parent/child relationship reflects the hierarchy shown in the Game Explorer window and the Behaviors windows of the Workspace.
- 3** An attribute can belong to zero or more attribute shares.
- 4** Only assets created from .rf3 files can have child assets.

A game database consists of the following types of object:

### Game

The root object in the game database hierarchy. A game database can contain only one game object. Currently, RenderWare Studio supports only one game per project (.rwstudio file).

### Folder

A container for other objects in the game database. You can use folders to organize a game database into a structured hierarchy. There are various types of folder.

Only the following types of folder are significant in the game database hierarchy:

**Folder** (without any additional qualifying term; if you like, a “normal” folder)

Can contain entities, assets, asset folders, or other folders. Displayed only in the Game Explorer window.

Depending on the genre of your game, the highest-level folders might correspond to the levels in your game.

### Asset folder

Can contain assets or other asset folders. Displayed in the Assets and Game Explorer windows.

### Global folder

The global folder is a special type of folder containing objects that are persistent throughout a game (not just one level of a game). A game has one, and only one, global folder. The contents of the global folder are sent to the target console before any other folders. A global folder can contain the same types of object as a normal folder (assets, entities, other folders etc.), but the global folder cannot be nested inside another folder.

The other folder types—attribute share folders, behavior folders, and template folders—exist only to organize the display of objects in their associated Workspace windows.

## Entity

Typically, each individual object in your game (such as human figures, vehicles, tools, weapons, doors, buttons, levers) is represented by an entity in the game database. An entity consists of:

- References to zero or more assets
- A reference to one (and only one) behavior
- Values for the attributes of the behavior

For example, a car entity might consist of:

- A reference to an asset that defines the geometry of the car and its texture (such as paint color and exterior detailing)
- A reference to a behavior that defines how the car moves
- Attribute values relating to the behavior (such as the amount of rubber the car's tires leave behind on the road as it accelerates)

## Asset

Identifies data that can be used by an entity. Among other properties, an asset contains the relative paths of:

### A primary asset file

Contains the “primary” data for an entity (for example, a .dff or .rf3 file, defining the geometry of an entity).

### A dependencies folder

The folder containing data files required by the primary asset file (for example, texture bitmaps).

Several entities can use the same asset. You can attach assets to an entity either one by one, or you can group assets into asset folders, and then attach a mix of individual assets and asset folders to an entity.

**Note:** In general usage, the term *asset* is sometimes used to refer to a primary resource file. However, in RenderWare Studio, the term *asset* refers to an object in the game database that points to a primary asset file, not to the primary resource file itself. Like any other object in a game database, an asset is stored in an XML file.

## Attribute

Stores the value of a parameter that controls the behavior of an entity. For example, a vehicle entity might have an “exhaust color” attribute that controls the color of the exhaust smoke that its behavior causes the entity to emit. Each entity can either have its own individual attribute values, or use *attribute shares* to share the same attribute values as other entities.

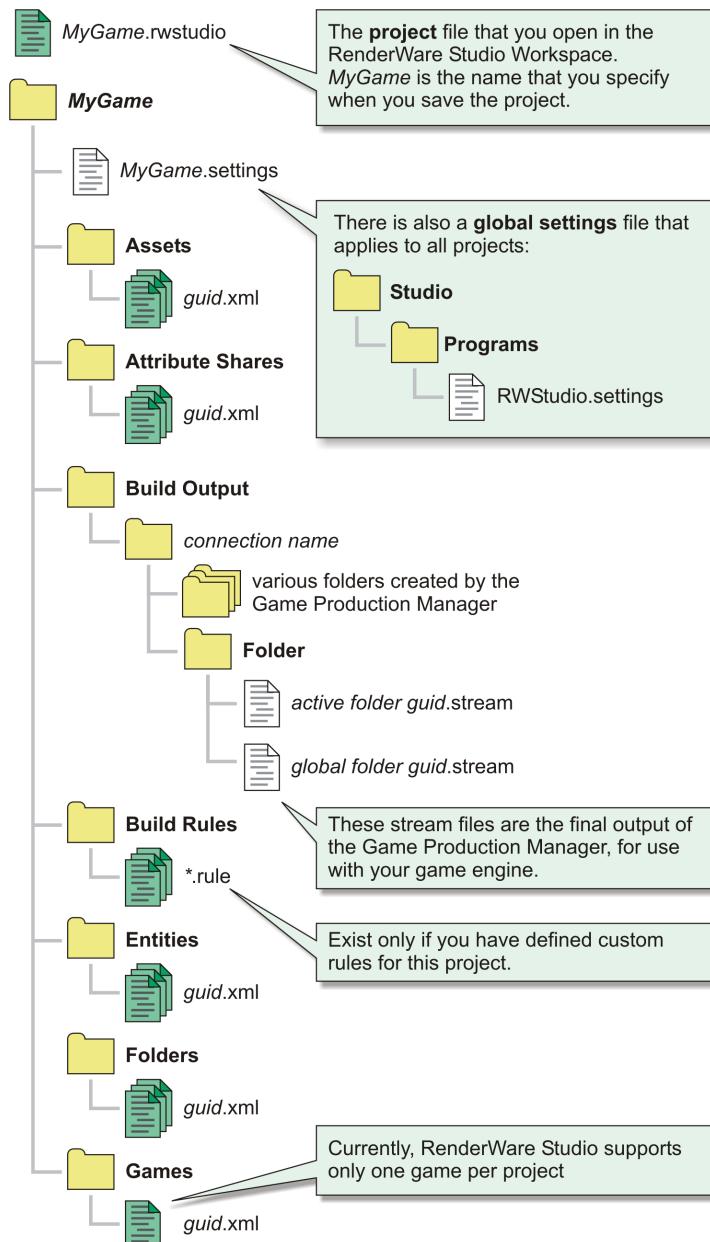
To change attribute values, use the Attributes window.

Behavior commands and entity attributes are closely related. Commands define the attributes that an entity can have. Attributes store the values of the command parameter values for an entity.

# Game database XML files

---

RenderWare Studio stores the database for each game in a set of XML files:



## Note:

- In the diagram above, only the files shown as are considered to be part of the game database. If you use NXN alienbrain (p.187), then these are the only files that Workspace checks in or out. Other files are shown here for completeness.
- For general information about XML, go to [www.w3c.org/xml/](http://www.w3c.org/xml/) ([www.w3c.org/xml/](http://www.w3c.org/xml/)).

## *MyGame.rwstudio*

The RenderWare Studio project file. This is the file that you specify when you want to open a game, or save a game. When you save a new game for the first time, or you save an existing game to a new location, then, in addition to creating a new .rwstudio project file, RenderWare Studio creates a new folder structure, as shown in the diagram above.

The .rwstudio project file points to the location of the folders that contain the *guid.xml* files (described below).

**File ▶ Save As** copies the XML files from their current location to the new location (relative to the new .rwstudio project file). On the **File** menu, click **Save As**, and enter the new file name (*NewGame*). RenderWare Studio creates the new project file and folder structure, and copies the related XML files.

**Deleting an object in Workspace does not delete its corresponding XML file.** When you save a project, rather than deleting the XML file for a deleted object, Workspace replaces the file contents with a <Deleted/> tag. This allows you to use RenderWare Studio with version control systems that do not delete client files (such as NXN alienbrain and Microsoft SourceSafe).

To delete these files, use the PurgeDeleted tool, in  
Studio\Programs\Tools\PurgeDeleted.

## MyGame.settings

Contains the [game properties](#) (p.98).

## guid.xml

Each *guid.xml* file defines an object in the [game database](#) (p.73).

## RenderWare Studio Windows Explorer extension

A typical game project can contain hundreds of files called *guid.xml*. Naming asset and entity files this way is convenient for RenderWare Studio, but not so convenient for you when you're browsing your project's directories. For this reason, RenderWare Studio includes the option to install a Windows Explorer extension that adds a human-readable **RenderWare Studio Name** column to directory listings.

To benefit from this feature, you need to select the **Extras ▶ Explorer Extensions** item in the RenderWare Studio Setup wizard. (You can do this at installation time, or later through Control Panel's Add/Remove Programs tool.) Once it's installed, you must enable it for each folder you want to use it in:

1. In Windows Explorer, browse to a folder containing some RenderWare Studio assets, and switch to the **Details** view.
2. Right-click any one of the column headings, and select **More....**
3. In the dialog that appears, select **RenderWare Studio Name**, and click **OK**.

When you close the dialog, the new column appears in Windows Explorer:

Name	Size	Type	RenderWare Studio Name
{1AD4FFBA-0919-4C93-B098-0F69D4535473}.xml	1 KB	XML Document	weapon_flare_s.png
{1EEEF1B9-DDF2-419B-B590-DEF69C384291}.xml	1 KB	XML Document	Ground Floor Patrol Route
{28C189D9-553E-47F7-9321-34CA7E730759}.xml	1 KB	XML Document	Corona.png
{2EEBA455-22D7-4B5A-B31E-C73A1F28FB60}.xml	1 KB	XML Document	third guard
{328B12B9-C2B6-4876-AA3D-1D469B158C71}.xml	1 KB	XML Document	concrete_bullet_hole.png
{3853D661-5BDF-435B-A8D5-2A38F059B1A4}.xml	1 KB	XML Document	Shadow
{3F0B2409-1AA8-4213-B574-7259E89932CA}.xml	1 KB	XML Document	Second Floor Patrol Route

**Note:** The **RenderWare Studio Name** column takes its data from the <name> elements of the game database XML files.

## Load game on demand

The new load-on-demand mechanism loads only the parts of a database that a user is interested in:

- The XML parser is modified to load data as clients of the database request

it.

- The XML parse of individual objects is quick enough so as not to produce a perceived delay when data is requested.
- Minimal changes should be required in workspace/ tool code.

Certain Manager API calls cause one or more XML files to be loaded:

**RWSGetFirst**

May be called to enquire on the first object of the specified type.

**Caution:** Calling this function can slow down loading the game as it loads and allocates memory to every object of the given type.

**RWSChildAdd**

Results in a load of both parent and child.

**RWSChildRemove**

Results in a load of both parent and child.

**RWSChildSwap**

Causes the parent to be loaded.

**RWSChildGetFirst**

Causes the parent to be loaded.

**RWSParentGetFirst**

Causes the child to be loaded.

**RWSObjectGet**

Causes the object to be loaded.

**RWSTagGetData**

Causes a load for any tags that are serialised—global folder, template entity and folder type.

**RWSPropertySet**

Causes a load of the object.

**RWSPropertyGet**

Causes a load of the object.

**RWSPropertyGetFirst**

Causes a load of the object.

# Customization

---

Building upon well-known technologies such as ActiveX (COM) and Active Script, the component-based nature of RenderWare Studio makes the software highly customizable. There are a wide variety of modification options to suit the workflow, practices, and custom tools of a games development team.

This topic describes some of the different kinds of customization possible, and provides pointers to sources of further information.

## Changing the user interface

Using Visual C++ 7.1 and/or Enterprise Author, you can create new windows and dialogs for RenderWare Studio Workspace. This could be to provide a new way of presenting standard information to the user, or to display and manage new asset or entity types.

By the same token, you can create menus, toolbars, and context menus that either perform new operations on standard Workspace features, or allow users to interact with any new windows or dialogs you've created. Alternatively, you can add new items to the existing set of menus and toolbars.

## Integrating new controls

Imagine that you've created an ActiveX control that deals with a new kind of asset, and that entities created from this asset ought to be represented alongside other game entities in RenderWare Studio's main window (Design View). You can add your own graphics to that window by using the 3D View Extensions interface.

**Tip:** If the entities dealt with by a new ActiveX control can be represented as collections of nodes and their interconnections ("graphs"), you can call methods of the Graph Tool API from the ActiveX control to create and manage those collections.

Workspace's Attributes window displays controls for editing the behavior attributes of any entity in a game. If a new entity type involves a new attribute type, there's a RenderWare Studio wizard to you to create a control for editing its value.

## Dealing with new types

Custom assets could require custom file types, and you can customize Workspace to manage these seamlessly:

- To make Workspace recognize a file type, you use Enterprise Author to modify `GetAssetFileType()` in the *GlobalScript* script module.
- To make custom assets appear correctly in Workspace's list windows, you modify the `RWStudio.settings` file to associate their type with an icon (or even a separate viewer control).
- To link a new asset type to Workspace's search mechanisms, you modify the *Searches* script module, and use the Manager COM interface in the implementation of the search logic.

- To process the assets that make up a game into game-ready data, you need build rules. For new asset types, you can modify existing build rules, or create them from scratch.

Among the built-in rules is `TextureDictionary.rule`, which produces a list of all the textures used by every asset in a game, and generates a stream file containing it. To do this, the rule uses a tool called `texdicgen`. You can find the modifiable source code for this tool in RenderWare Studio's `Programs\Tools` directory.

## Testing and profiling

Communication between Workspace and a running game is vital for testing and profiling purposes, and this too can be customized. On the PC side, networking headers are defined in `Programs\Workspace\Build Rules\BufferTools.vbs`, which can be modified. The code in this file provides a good reference to what needs to be implemented. The corresponding definitions for the Game Framework are in `strfunc.h`.

For traffic in the other direction, the `RWSComms_OnConnectionBufferReceived()` method, which can be found in the `BuildScriptManager` script module, can be modified to handle custom network traffic from the game to Workspace. As above, custom commands can be defined using `strfunc.h` if needed.

# System requirements

---

## Hardware

Minimum hardware requirements:

### Display adapter

3D hardware acceleration, such as NVidia GeForce 2 (or better) or ATI Radeon.

Must support the following display drivers:

**DirectX 9** ([www.microsoft.com/windows/directx/](http://www.microsoft.com/windows/directx/))

To run the Workspace.

**OpenGL** or **DirectX 9** ([www.microsoft.com/windows/directx/](http://www.microsoft.com/windows/directx/))

To run the Game Framework on PC targets.

### Display resolution

1024 x 768 (1280 x 1024 recommended).

### Processor

1 GHz.

### Memory

1 GB RAM minimum.

### Disk space

375 MB free disk space, of which approximately 275 MB is used by the program files and 100 MB is used by the examples supplied with RenderWare Studio.

## Operating system

Either:

- Microsoft Windows 2000, Service Pack 2 or later
- or
- Microsoft Windows XP

To install RenderWare Studio 2.0, you must have administrative privileges on the operating system.

## Consoles

### Nintendo GameCube

The [SN-TDEV](#) (p.214) development system.

### Sony PlayStation 2

You will need a Broadband Ethernet adapter or a USB Ethernet connection (both supported via the `net000.cnf` file).

**Tip:** We recommend the SCPH-10350 network adapter for the broadband ethernet connection to the PS2 Test deck or, for the USB ethernet connection, the list of compatible USB adapters found by navigating to the release notes for the AN986 Ethernet Driver in the USB Ethernet Drivers section of the [PlayStation 2 Developer Network](#) ([www.ps2-pro.com](http://www.ps2-pro.com)).

### Microsoft Xbox

Xbox Development Kit (XDK) December 2003 (or later) for both the Xbox console and the development PC.

## C++ compilers

RenderWare Studio has been tested with the following C++ compilers:

Compiler	Target platform			
	PC	GameCube	PlayStation 2	Xbox
MetroWerks ( <a href="http://www.metrowerks.com/">www.metrowerks.com/</a> ) CodeWarrior	○	○	●	○
SN Systems ( <a href="http://www.snsys.com/">www.snsys.com/</a> ) ProDG	○	● 1	●	○
Microsoft ( <a href="http://www.msdn.microsoft.com/">www.msdn.microsoft.com/</a> )	●	○	○	●

1 Supplied projects require Dolphin SDK 1.0, 8 May 2003 patch level 4 (or later); they will not link with earlier versions.

## Other software

### **Microsoft Forms 2.0 ActiveX controls**

These controls are supplied with Microsoft Office; they are also available for free from Microsoft, as part of the Microsoft ActiveX Control Pad.

If either of these two packages is installed on your system, then you already have these controls.

If you do not have these controls, then you need to download and install the Microsoft ActiveX Control Pad:

1. Go to the [Microsoft Developer Network \(MSDN\) website](http://msdn.microsoft.com/) ([msdn.microsoft.com/](http://msdn.microsoft.com/)).
2. Search for Knowledge Base article 224305, which contains a link to the ActiveX Control Pad page.
3. Install the ActiveX Control Pad on your computer.

### **Microsoft Internet Explorer**

Version 6 (supplied on the RenderWare Studio setup CD) or later.

### **Adobe Reader**

Supplied on the RenderWare Studio setup CD.

RenderWare Studio user documentation is supplied in Acrobat PDF and Microsoft HTML Help (.chm) formats. To view the Acrobat PDF files, you need to install Adobe Reader.

### **RenderWare AI**

Version 3.71 or later

### **RenderWare Audio**

Version 3.7 or later.

### **RenderWare Graphics**

Version 3.7 or later.

### **RenderWare Physics**

Version 3.7 or later.

### **NXN alienbrain**

If you use [alienbrain](#) (p.187) to manage your RenderWare Studio projects, then you must use alienbrain version 7.

## Getting started

---

If you have not used RenderWare Studio before, then we recommend that you:

1. Read [What is RenderWare Studio?](#) (p.66).
2. Complete the [First steps tutorial](#) (p.86).

**Note:** If you have just installed RenderWare Studio, then you will be using a temporary license. To continue using RenderWare Studio when the temporary license expires, you need to [request a permanent license](#) (p.95).

# First steps (duration: 15 minutes)

## In this tutorial

You step through the following basic tasks:

1. Opening an existing project.
2. Flying around the game world.
3. Adding an entity to the game.
4. Moving and rotating an entity.
5. Viewing the game on a target.

In RenderWare Studio, the term *target* refers to a console or PC platform for which you are developing a game. In this tutorial, you view the game in a separate window on the PC running RenderWare Studio.

6. Attaching a behavior to an entity.
7. Setting the behavior attributes.

The behavior that you attach in this tutorial causes the entity to rotate; you set attributes that control the rotation speed.

8. Saving the project.

## Opening an existing project

1. Start RenderWare Studio.
2. In the **File** menu, select **Open Project....**
3. Browse to the following folder:

C:\RW\Studio\Examples

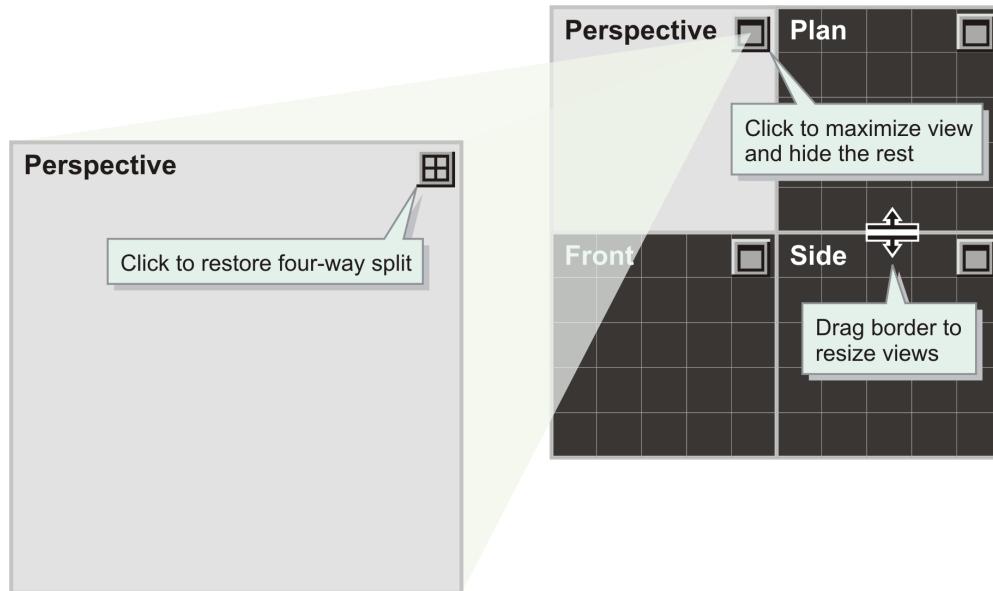
This is where the installer places example projects by default. If you changed this setting during installation, the path you need to use here will be different.

4. Open Tutorial.rwstudio.

The tutorial project loads in the Workspace.

## Flying around the game world

5. In the Design View window, click the  icon in the top-right corner of the top-left pane (the “perspective view”).

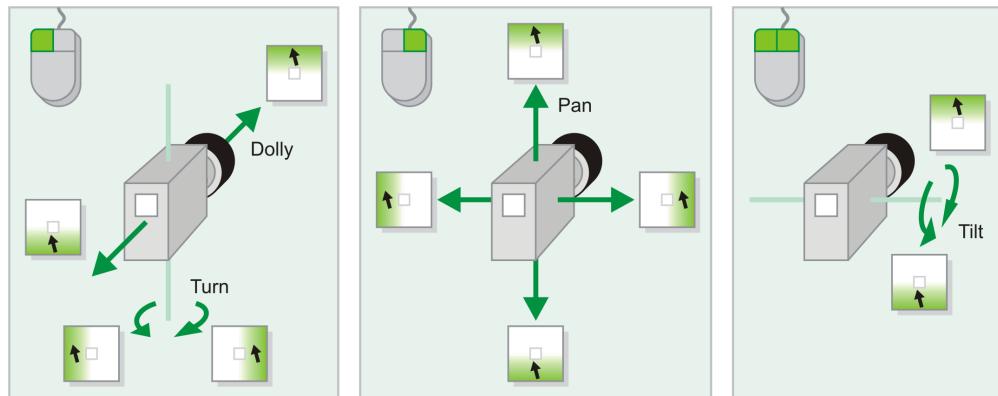


The 3D or perspective view expands, hiding the three orthographic views.

To fly around the game world, you control the movements of a virtual camera; the perspective view shows the camera's current viewpoint. To control the camera, you can use either the mouse or the keyboard.

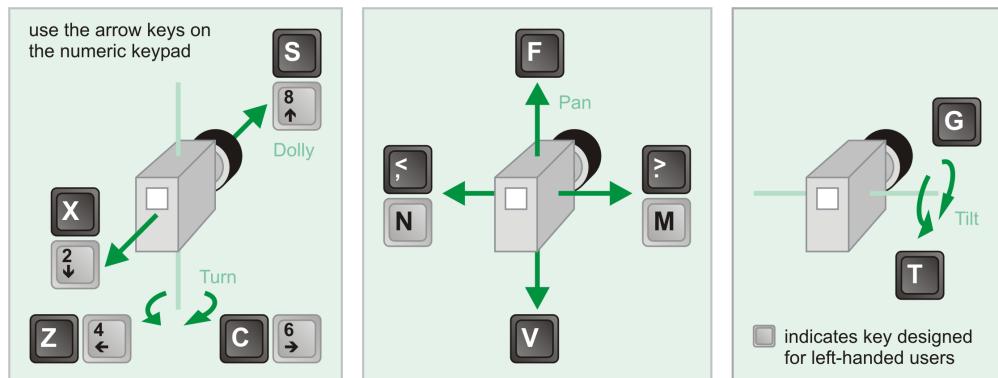
6. Click the  (Flight Camera) toolbar button.
7. Fly around in the perspective view.

## Mouse flight controls



To dolly *into* the game world, for example, point to the top half of the perspective view, and then hold down the left mouse button. The closer to the edge of the perspective view you point, the faster you go.

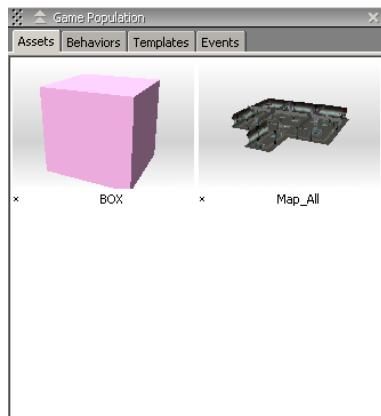
## Keyboard flight controls



8. Try the other **camera control buttons** (p.10)  (**Orbit Camera**),  (**Pan Camera**), and  (**Zoom Camera**).

## Adding an entity to the game

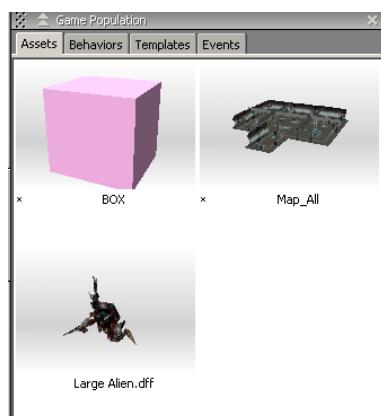
9. Display the [Assets window](#) (p.264) (which might appear as a tab in the Workspace layout you are using; see the explanation below).



### Use of the term “window”

In Workspace, you can switch between user interface layouts that present different arrangements of (mostly) the same elements. What appears in one layout as a tab might appear in another layout as a separate window. This documentation refers to all such user interface elements as “windows”.

10. In Windows Explorer, navigate to the C:\RW\Studio\Examples\Models folder.
11. Drag the file Large\_Alien.dff from Windows Explorer to the Assets window.



12. Drag the “Large Alien” asset from the Assets window to the Design View window. Drop it in a clear space in the game world.

A new “Large Alien” appears in the window.



Dragging an asset into Design View creates a new [entity](#) (p.71). At this point, the behavior of the entity will be the default CEntity behavior, and its name will be something similar to *Entity<nnnn>*.

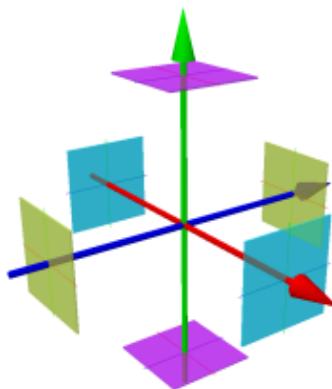
**Tip:** To delete an entity, expand the game folder in the [Game Explorer](#) (p.288) window, right-click the entity name, and then select [Delete](#) (p.177). This does not affect the asset.

## Moving, rotating, and scaling an entity

13. Click the  (Pick and Move) toolbar button.

14. Click the entity in the perspective view.

This selects the entity, displaying it as a wireframe with *drag axes* (p.123):



15. Drag a rod to move the entity along an axis; drag a square panel to move the entity in any of the x, y, or z-planes.

16. Try the  (Pick only),  (Pick and Rotate) and  (Pick and Scale) toolbar buttons.

**Tip:** To center the camera viewpoint on the selected entity, press **F3**. Press **Shift+F3** to center the view on the selected entity without moving the camera to that entity.

17. When you have finished moving, rotating, and scaling the entity, press **Esc** to unselect it.

## Viewing the game on a target

18. Click the Targets tab in the Output window.

Targets				
Name	Status	Address	Platform	Target
GameCube	Disconnected	Enter IP Address here	GameCube	GameCube
Genre Pack 1: Xbox	Disconnected	Enter IP Address here	Xbox	Xbox
Local PC - DirectX	Disconnected	localhost	DirectX PC	Local PC
Local PC - OpenGL	Disconnected	localhost	OpenGL PC	Local PC
PS2 Debug Station	Disconnected	Enter IP Address here	Playstation 2	Test Station
PS2 Development Station	Disconnected	Enter IP Address here	Playstation 2	Development Tool
Xbox	Disconnected	Enter IP Address here	Xbox	Xbox
Genre Pack 1: DirectX	Disconnected	localhost	DirectX PC	Local PC
Genre Pack 1: PS2 Debug Station	Disconnected	Enter IP Address here	Playstation 2	Test Station

The **Targets** (p.330) window shows the targets on which you can view your game. Specifically, it shows target *connections*, defining how RenderWare Studio connects to each target.

19. In the Targets window, right-click either the **Local PC - DirectX** or **Local PC - OpenGL** connection, depending on which display drivers your computer

supports (see note below). The context menu for the connection appears.

### DirectX versus OpenGL display drivers

Unless you have an old display adapter, it is likely that your computer supports both DirectX and OpenGL. However, while the OpenGL display driver is typically supplied and installed with the display adapter, you might have to [download](http://www.microsoft.com/windows/directx/) ([www.microsoft.com/windows/directx/](http://www.microsoft.com/windows/directx/)) and install the DirectX display driver separately.

20. Click **Launch**. A new window, named RWS Console, appears on the desktop. This window shows the loading screen of the RenderWare Studio Game Framework.



21. Press **Alt+Tab** to return the focus to Workspace.
  22. In the Targets window, right-click the same connection again, and then select **Build and Connect**.
- The Workspace sends a stream of data to the Game Framework. This stream consists of information about all of the entities and assets in the current game level. When this operation is complete, the game replaces the loading screen in the RWS Console window.
23. Ensure that the  (**Director's Camera**) toolbar button is selected.

Selecting the Director's Camera overrides any camera control behaviors in the Game Framework. Instead, the console view follows the movements of the camera in Workspace. For example, in a “first person” game, the camera typically follows the movements of the player. If you select the Director's Camera, then, although you can still use your console's control pad to move the player, the console view moves with the Workspace camera, not the player.

24. Fly around the game world.

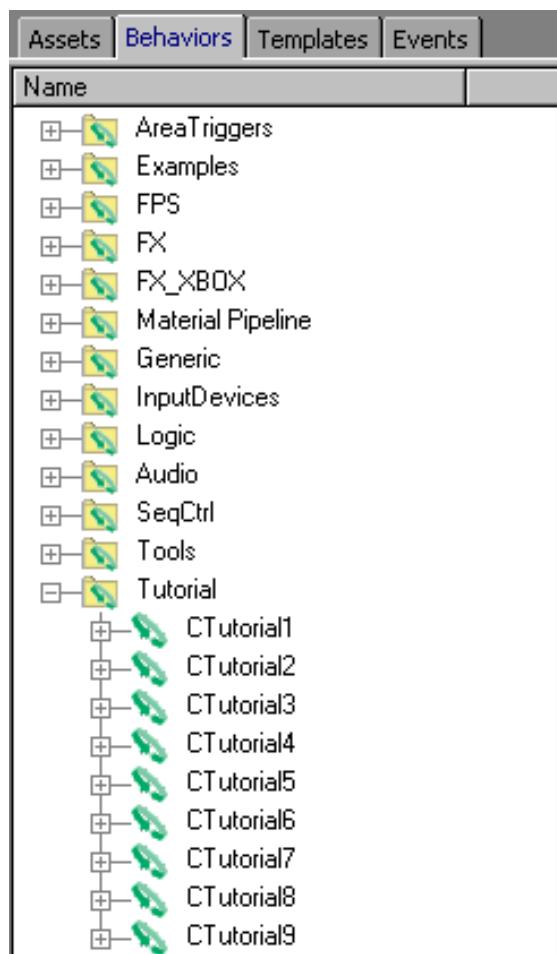
As you fly around, notice that the RWS Console window shows the same view as the Workspace.

25. Fly back to the entity you added, so that you can see it close up.

## Adding a behavior to an entity

26. Click the Behaviors window.

The Behaviors window shows the behaviors that you can attach to an entity.



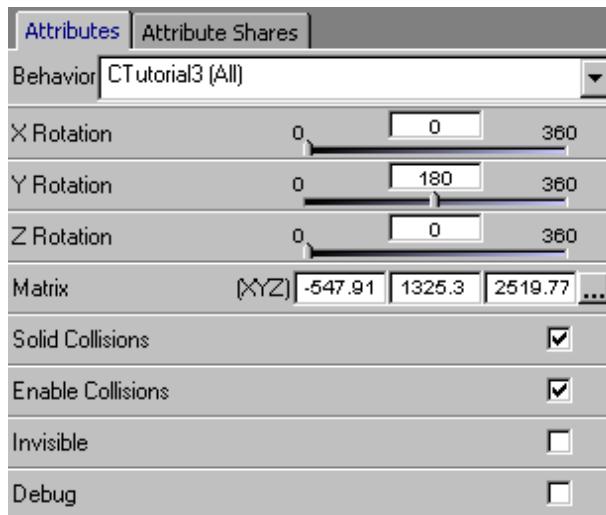
27. Drag the `CTutorial3` behavior to the entity in the perspective view. In the console view, the entity starts rotating.

An entity can have only one behavior; the `CTutorial3` behavior causes its entity to rotate.

**Tip:** Another way to attach a behavior is to right-click the entity in the Game Explorer window, and then select **Properties**. Now select the required behavior from the drop-down list of behaviors.

## Changing the attributes of a behavior

28. Select the Attributes window.



The Attributes window shows the controls for editing the behavior attributes of the selected entity. Attaching a behavior to an entity also selects the entity, so this window shows the attributes of the CTutorial3 behavior for the entity that you have just added.

The CTutorial3 behavior has three attributes that define the number of degrees its entity rotates per frame in each axis: *X Rotation*, *Y Rotation* and *Z Rotation*.

When you [create a behavior](#) (p.337) in the Game Framework, you specify an RWS\_ATTRIBUTE macro for each of its attributes. This macro defines the attribute's parameters, including its name, the type of editing control, and the range of allowed values. To see the code for this attribute, right-click the **X Rotation** attribute, and then select **View Source**. The appropriate header file appears in your code editor, containing the line:

```
RWS_ATTRIBUTE(CMD_rot_x, "X Rotation", "Specify the x axis rotation", SLIDER, RwReal, RANGE(0, 0, 360))
```

29. Adjust the slider control settings.

As you adjust the sliders, look at the console view, and notice how the speed of rotation changes.

## Saving the project

30. On the **File** menu, click **Save Project As....**
31. Type **First steps** and then press **Enter**.

Saving a project to a new file does more than just creating a new high-level .rwstudio file. It also creates a new set of [folders](#) (p.76), and copies the project's other XML files into these folders.

This is the end of the tutorial.

## Requesting a permanent license file

---

Each time you start RenderWare Studio, it checks its `Programs` folder for a file named `1servrc` (with no file extension). This file contains information about your license to use RenderWare Studio. When you install RenderWare Studio, you will be using the supplied temporary license file. To continue using RenderWare Studio after the temporary license expires, you need to replace this file with a permanent license file.

To request a permanent license file, visit the [Fully Managed Support Services](https://support.renderware.com/licensing/) (<https://support.renderware.com/licensing/>) website. To login to the FMSS website, you will need your customer username and password.

**Tip:** If you installed RenderWare Studio in the default location, then the path of the `Programs` folder is `C:\RW\Studio\Programs`.

# Getting started with VBScript

---

If you want to customize either the RenderWare Studio Workspace user interface or the Game Production Manager build rules, then you will need to know how to program in the Microsoft Visual Basic Scripting Edition (VBScript) language.

## VBScript documentation

The best place to begin learning VBScript is the [Microsoft Scripting website](http://msdn.microsoft.com/scripting/) ([msdn.microsoft.com/scripting/](http://msdn.microsoft.com/scripting/)). At the time of writing, you could download VBScript documentation in an HTML Help file called *Microsoft Windows Script 5.6 Documentation*. If you have the Microsoft Developer Network (MSDN) Library, then you already have this documentation: look in **Web Development ▶ Scripting ▶ SDK Documentation ▶ Windows Script Technologies**.

There are also many good books on VBScript, such as *VBScript in a Nutshell* (ISBN 1565927206, published by [O'Reilly & Associates](http://www.ora.com/) ([www.ora.com/](http://www.ora.com/))).

## Microsoft Script Debugger

You can use Microsoft Visual Studio .NET 2003 to debug VBScript code, but if that isn't installed on your computer, we recommend that you download and install the Microsoft Script Debugger. This tool, which is freely available by following the links from the [Microsoft Scripting website](http://msdn.microsoft.com/scripting/) ([msdn.microsoft.com/scripting/](http://msdn.microsoft.com/scripting/)), enables you to set breakpoints, step through code, examine the call stack, and query or change values.

**Tip:** Make sure that you download the correct version of the debugger. There's one for Windows 98 and Me, and another for Windows NT 4.0, 2000, and XP.

Before using the debugger, you need to edit the Windows registry, as described in Microsoft Knowledge Base article [252895](http://support.microsoft.com/support/kb/articles/q252/8/95.asp) ([support.microsoft.com/support/kb/articles/q252/8/95.asp](http://support.microsoft.com/support/kb/articles/q252/8/95.asp)):

1. Start the Registry Editor:
  - a. On the **Start** menu, click **Run...**
  - b. Type `regedit` and then click **OK**.
2. Find the following registry key:

HKEY\_CURRENT\_USER\Software\Microsoft\Windows  
Script\Settings\JITDebug

3. Set its value to 1.

## Starting the debugger

To invoke the debugger, insert a VBScript `Stop` statement in your script code. When you run the script and it reaches the `Stop` statement, the script halts execution and transfers control to the debugger.

For more information, see the documentation supplied with the debugger.

# Creating and editing the game database

---

This section provides basic information on how to:

- [Create and save a new project](#) (p.98)
- [Import assets into your project](#) (p.101)
- [Create an entity](#) (p.119)
- [Create and switch between folders](#) (p.166)
- [Import folders from another project](#) (p.175)

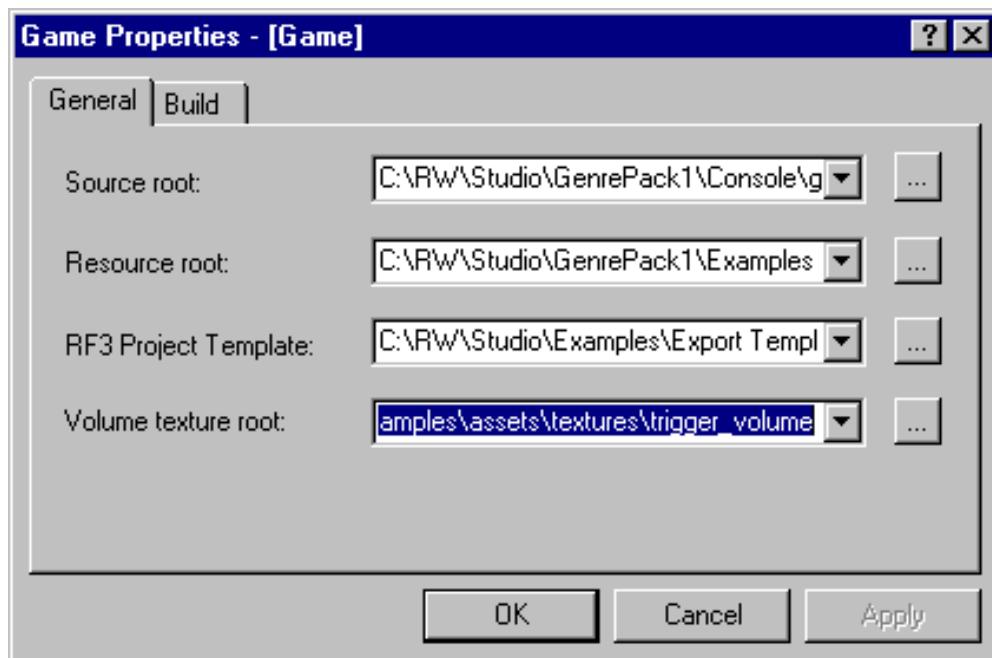
# Creating a new project

---

To create a new RenderWare Studio project:

1. On the **File** menu, click **New Project**.
2. In the Game Explorer window, right click the game icon  , and then click **Properties**

The Game Properties dialog appears.



3. Set the game properties:

### Source root

The absolute path of the root folder where the Game Framework C++ source files are stored. The Workspace uses this path to locate the source header files that define behaviors.

The Workspace parses the header files to determine:

- The list of behaviors that you can attach to entities.
- The attributes for each behavior, and the controls that the Attributes window displays for setting these attributes.

If you installed RenderWare Studio in the default folder (C:\RW\Studio), then the supplied Game Framework C++ source files are in:

C:\RW\Studio\console\game\_framework\source

### Resource root

The absolute path of the root folder where your asset data is stored.

Each asset in the game database specifies the relative paths of:

- A primary asset file (such as a .dff or .rf3)

and, optionally,

- A dependencies folder (for example, where textures for the primary asset file are stored)

The Workspace uses the resource root folder to resolve these relative paths into absolute paths.

### RF3 Project Template

The absolute path of the template used to convert RF3 files into streams that Workspace can display.

If you installed RenderWare Studio in the default folder (C:\RW\Studio), then the supplied template files are in:

C:\RW\Studio\Examples\Export Templates\Project

### Volume texture root

The absolute path of a folder containing textures that may be assigned to any box and trigger volumes you create in your game. The files in this folder appear in a drop-down menu when you click the **Texture selection** button.

4. Click **OK**.

A new project opens in Workspace.

5. Begin developing your game. The first step is to [import assets](#) (p.101).

## Saving your project

To save your project:

- On the **File** menu, click **Save**.

In addition to a `.rwstudio` project file, the Workspace also creates a [folder structure](#) (p.76) for your project.

## Editing game properties

To change the location of the source and resource folders after creating the project, right-click the game in the [Game Explorer](#) (p.288) window, and then click **Properties**.

# Importing assets into your project

---

To import assets into your project, either:

- Drag one or more [primary asset files](#) (p.71) (for example, .d<sup>r</sup>f files) from **Windows Explorer** to the Workspace [Assets window](#) (p.264).  
Workspace creates an asset for each file; except for [RF3 files](#) (p.104), which can contain more than one asset.

**Tip:** You can also drag a whole folder containing primary asset files into the Assets window.

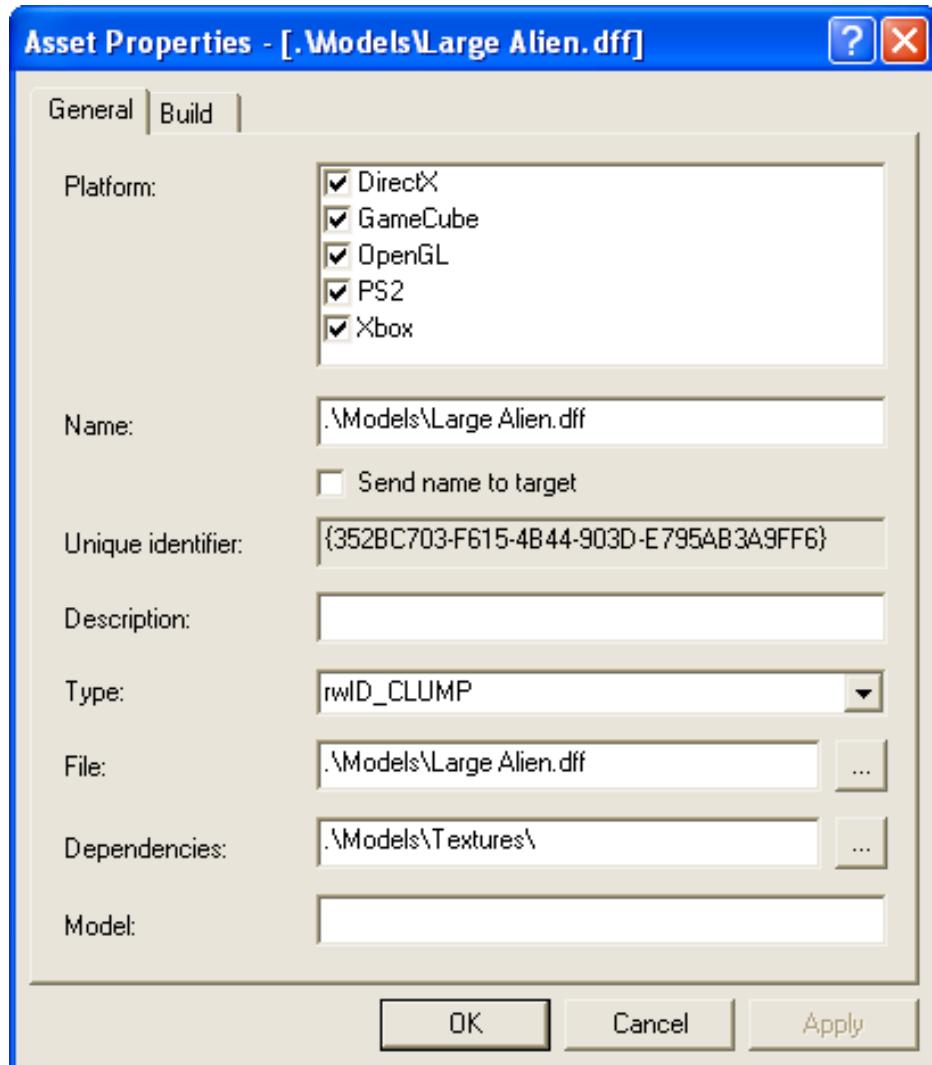
or

- Right-click inside the Assets window, and then click **New ▶ Asset**. A [property dialog](#) (p.102) for the new asset appears.

After importing an asset, you can use it to [create an entity](#) (p.119).

# Setting asset properties

To set asset properties, right-click the asset, and then click **Properties**.



## Platform

The platforms to which this asset applies. For each target, the game data stream includes—or refers to—only those assets that apply to the target's platform.

If this object applies only to some platforms, then deselect the other platforms.

## Name

The name of this asset. Must be unique for assets in the game. For example, you can have an asset and an entity both named “Alien”, but you cannot have two assets named “Alien”. This name can include numbers, spaces and letters in any combination of uppercase and lowercase. If you import an asset by dragging its primary asset file into the Assets window, then Workspace gives the asset the same name as the file (excluding the file extension; for example, `Large Alien.dff` becomes an asset named “Large Alien”).

**Send name to target**

Includes the name of the asset in the game data stream.

**Programmers:** If you want to use the `FindByName` method to refer to the asset by this name (rather than having to know its GUID), then select this check box.

**Unique Identifier**

The ID that the Game Framework uses to refer to this object. Also the name of the XML file for this item. This is a system-generated 128-bit value that uniquely identifies this item across all projects. This value is known as a globally unique identifier (GUID); also known as a universally unique identifier (UUID).

**Description** (optional)

Your description of this asset.

To sort assets according to their descriptions, right-click in the Assets window, click **View ▶ Detail**, right-click a column heading, select **Description**, and then click the Description column heading. (An arrow appears in the column indicating the sort order; to sort in reverse order, click the column heading again.)

**Type**

The RenderWare data type of this asset. For details, see the *RenderWare Graphics API Reference*.

**File**

The path of the primary asset file, relative to the game resource root folder.

**Dependencies**

The path of the folder containing the files pointed to by the primary asset file (for example, textures). Similar to the primary asset file, this path is also relative to the game resource root folder.

**Texture file formats**

Workspace supports only BMP and PNG texture file formats. If your assets use other texture file formats (for example, TIFF or RAS), then the Workspace displays those assets with all-white textures.

However, if the texture file format is supported by RenderWare Graphics, then the texture will appear correctly when you view the game on the target.

**Model** (optional)

Not currently used by RenderWare Studio; you can use this to store your own custom information about the asset.

**Tip:** To set the dependencies folder for several assets at once, select the assets, right-click one of the selected assets, and then click **Properties**. Any changes you make to the properties are applied to all of the selected assets.

# Working with RF3 asset files

---

When you import an RF3 file into RenderWare Studio (say, by dragging it from Windows Explorer to the Assets window), Workspace uses the RenderWare Exporters to convert the RF3 file into a RenderWare stream (.rws file) that Workspace can display. To create this stream, the Exporters use the RF3 project template that you specify in the [game properties](#) (p.98). The path of this template is stored in your [project .settings file](#) (p.76), in your project folder.

Similarly, when you build the game data stream for a target, Workspace exports the RF3 file to a platform-specific RenderWare stream that the target can display. This stream is created using the RF3 project template that you specify in the [connection properties](#) (p.199) for the target. The path of this template is stored in the `RWStudio.settings` file, in the RenderWare Studio `Programs` folder.

## RF3 asset hierarchies

RF3 files can contain multiple assets. When you import an RF3 file, Workspace creates an asset hierarchy in the game database: one asset representing the RF3 file itself, with a child asset for each asset defined in the RF3 file.

Name	Type	Description
└军事实验室	RF3 rwID_WORLD rwID_CLUMP rwID_CLUMP	
└军事实验室_vertexlit	rwID_WORLD	Auto-generated from: Models\military_lab.rf3
└牢房_cave_jail	rwID_CLUMP	Auto-generated from: Models\military_lab.rf3
└牢房	rwID_HANIMANIMATION	Auto-generated from: Models\military_lab.rf3
└crypt_tree_monster	rwID_CLUMP	Auto-generated from: Models\military_lab.rf3

**Tip:** The image above shows the tree view of the Assets window; this is useful for viewing RF3 assets. To see this view, right-click in the Assets window, and then select **View ▶ Tree**.

Even if the RF3 file contains only one asset, Workspace creates an asset hierarchy with a single child asset.

The [Build Log](#) (p.297) window displays the progress of the creation of each asset.

## Workspace RF3 asset streams use generic platform options

Workspace uses DirectX to display streams (for example, in the Design View window or as thumbnails in the Assets window). To the RenderWare Exporters, DirectX is a “generic” target platform, so the RF3 project template that you specify for Workspace must have generic platform options enabled:

```
<Platform>
  <Generic>
    <param name="Enable" type="bool" value="true" />
  :
  </Generic>
  <PS2/>
  <Xbox/>
  <GameCube/>
</Platform>
```

If the generic platform options are not enabled, then the Exporters will not create a stream that Workspace can use, and Workspace will not be able to display RF3-based assets.

If you are developing your game for a target that also uses generic platform options (for example, you are developing a DirectX PC game), then Workspace and your target reuse the same stream file.

## Keep platform-specific options in separate templates

The Exporters create streams for all platforms enabled by the export template options. When organizing RF3 project templates for use with RenderWare Studio, it is strongly recommended that you keep each set of platform options in a separate project template; or at least, that you enable only one set of platform options per RF3 project template. Here's why:

- It enables you to create the streams for each target platform separately, as required, rather than all at once.
- It avoids any clashes in the path names of the exported streams (described below).

## Where are RF3 asset streams stored?

RF3 asset streams are stored under the [resource root](#) (p.98) folder (the path is shown here split over several lines, for clarity):

```
resource root\
    RWStudio RF3 Exports\
        base name of RF3 project template\
            File property of asset (without .rf3 extension)\\
                name attribute of <asset> element in RF3 file.rws
```

For example, with a resource root of:

C:\models

and an RF3 file at:

C:\models\characters\alien.rf3

then the File property of the “alien” asset in the game database would be:

characters\alien.rf3

If the RF3 project template is named:

Generic.rwt

and the asset name in the RF3 file is:

<asset name="predator">

then the exported stream would be stored at:

C:\models\RWStudio RF3  
Exports\Generic\characters\alien\predator.rws

**Note:** In this example, the asset name (“predator”) and the RF3 file name (“alien”) are different. Typically, for the sake of simplicity, they are identical.

## Studio needs “clump-only” asset streams

Workspace and the game data stream require asset streams that contain clump data only, without any of the additional data that the Exporters can embed, such as group chunks, texture dictionaries, or effect dictionaries. If your RF3 export

template options instruct the Exporters to embed any of this additional data, then Workspace processes the exported stream, and creates another stream that contains just the clump data. This stream is stored in the same folder as the original exported stream, with the clump index as a suffix. For example

`predator1.rws`

This is the stream that is actually used by Workspace, or included in the game data stream.

If your RF3 export template options do not embed any additional data, so exported streams contain only clump data, then Workspace detects this and uses the exported streams, and does not create another stream.

## RF3 assets might appear differently on Workspace and consoles

In an RF3 export template, platform-specific options affect only the streams for that platform. The stream displayed by Workspace is exported using the generic platform options; the stream displayed on a PlayStation 2 is exported using the PS2 platform options. To keep the appearance of RF3-based assets as similar as possible between Workspace and your target platform (aside from any inherent display differences between the platforms), if you edit the platform-specific options for your target platform, then you should also edit the generic platform options for Workspace. Otherwise, the stream displayed on your target platform might appear markedly different to the stream displayed in Workspace.

## Location of RF3 asset templates

Each asset in an RF3 file can refer to an asset template, overriding the project template options.

Workspace looks for asset templates in a folder named `Asset`, alongside the folder that contains the project template:

`project template folder\..\Asset`

For example, if the path of the project template is:

`C:\RW\Studio\Examples\Export Templates\Project\Generic.rwt`

and an RF3 file refers to the asset template (notice the `Characters` relative folder path):

`Characters\Monster.rwt`

then Workspace looks for `Monster.rwt` at:

`C:\RW\Studio\Examples\Export Templates\Asset\Monster.rwt`

(that is, it ignores any folder path, and looks in `Asset`, *not* `Asset\Characters`)

**Tip:** We recommend that you store project and asset RF3 templates in sibling `Project` and `Asset` folders, as shown in the example above.

## All users should refer to the same RF3 project template file names

In a multi-user environment—for example, if you are using [NXN alienbrain](#) (p.187) to manage files—all users should refer to the same RF3 project template file names. Otherwise, if different users re-import updated versions of the same RF3 files, then the exported stream files will be created under different paths, resulting

in confusion between users.

## How does Workspace keep RF3 asset streams up-to-date?

When you import an RF3 file, open a project that contains RF3 assets, or build the game data stream for a target, Workspace uses a Game Production Manager (GPM) build rule to decide whether the RF3 asset streams are up-to-date. This build rule identifies the asset streams as *target files*; the RF3 file and project template are *dependent files*. If any of the target files do not exist, or any of the dependent files are newer than any of the target files, then the build rule exports new streams from the RF3 files.

For example, if you update an RF3 project template that is specified in a RenderWare Studio project settings files, and then you open that project in Workspace, the build rule recognizes that the RF3 project template is newer than the streams, and exports new streams.

**Note:**

- The RF3 asset streams for display in Workspace are updated only for the current active root folder, and for any assets that are currently displayed as thumbnails in the Assets window. If you change to another active folder, or display other assets as thumbnails, then those streams are exported as required.
- In RenderWare Studio Version 2.0, **asset templates are not treated as dependent files**. If you update an asset template, then, to ensure that your RF3 asset streams are updated, you should re-import the asset (for display in Workspace) and rebuild the game data stream for the target: right-click the asset, and then select **Rebuild and Reload** (p.264).

**Build engineers:** For details on how RF3 files are exported to streams, see the RF3Asset.rule build rule, stored in the RenderWare Studio Programs\Workspace\Build Rules folder.

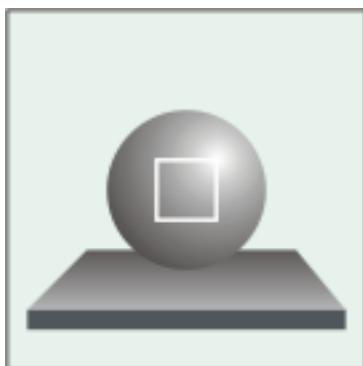
# Controlling the camera

---

To control the camera in the Design View window, you can use a mouse or a keyboard.

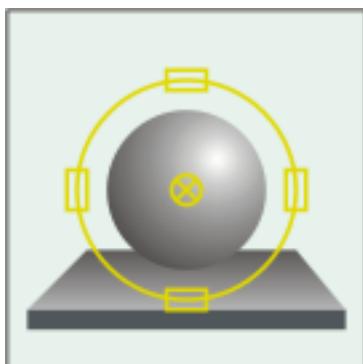
## Mouse controls

To control the camera with a mouse, you can choose between the following camera modes:



### Fly

 Fly the camera by pointing in the view, and then pressing and holding down a mouse button. This camera mode has separate controls for the perspective view and the orthographic views.

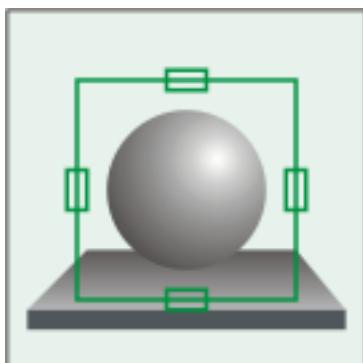


### Orbit

Orbit the camera around a point by dragging.

By default, the orbit center is the entity that you most recently [selected](#) (p.121). You [can move](#) (p.113) the orbit center to a different entity, or to an arbitrary point.

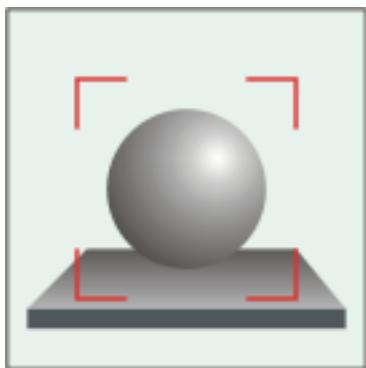
To constrain the orbit to a particular axis, begin dragging inside a  or  rectangle.



### Pan

Pan the camera by dragging.

To constrain the pan to a particular axis, begin dragging inside a  or  rectangle.



### Zoom

Zoom (dolly) the camera in and out by dragging.

#### Special tips for Max users:

- To zoom while in one of the other camera modes (pan, fly or orbit), turn the middle mouse wheel, or use **Ctrl+Alt+ drag** with middle mouse button
- To zoom while in pick mode, use **Ctrl+Alt+ drag** with middle mouse button.

To select the camera mode:

- Press **F10** to cycle between the modes  
or
- Click a toolbar button (shown above)  
or
- Use the **Selection** menu  
or
- While in **pick** (p.123) mode, press the middle mouse button and drag to switch instantly into pan mode.

Holding down the middle mouse button followed by one of the following modifier keys and then dragging will instantly switch into one of the other following **quick entry** modes—these modes are suitable for Max users:

- **Alt:** Orbit mode
- **Ctrl:** High-speed pan mode
- **Ctrl+Alt:** Zoom mode.

The following quick entry modes are suitable for Maya users. In all cases it is necessary to press the **Alt** key first before moving the mouse:

- **Alt+ Left mouse button:** Orbit mode
- **Alt+ Middle mouse button:** High-speed pan mode
- **Alt+ Right mouse button:** Zoom mode.

**Tip:** You can use all the quick access modes **except** orbit mode in all orthographic views as well as in the Design View window.

**Caution:** When using the middle mouse button in this way, you cannot use the boxes described above to constrain the orbit or pan to a particular axis.

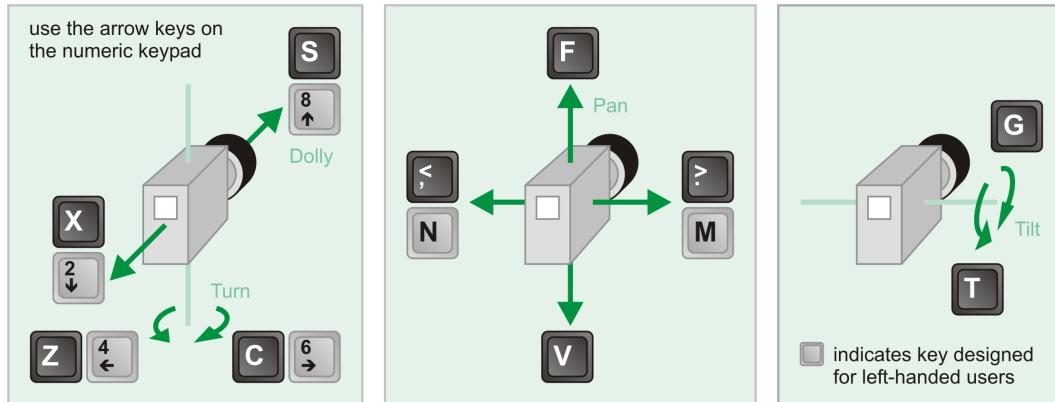
#### Special tips:

- To reset the view to its original state in any of the quick entry modes, click and release the right mouse button as you are dragging with the middle mouse button.

- You can also toggle between camera mode and pick mode by pressing **F9**.

## Keyboard controls

In any camera mode (and even in pick mode), you can use the keyboard to fly the camera:



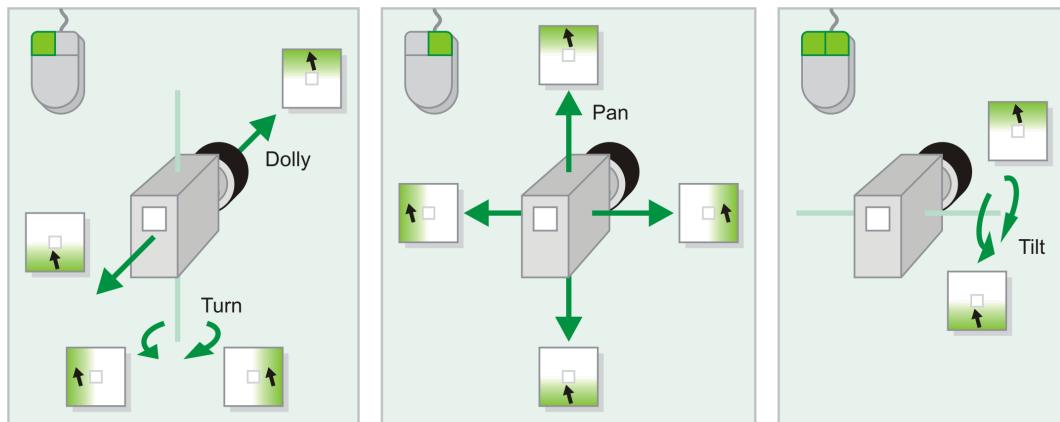
**Tip:** If the keyboard controls are not working, it means that one of the other windows has the keyboard “focus”. To correct this, click in the Design View window.

# Camera flight controls

**Fly**  camera mode (p.108) has separate controls for the perspective view and the orthographic views, as described below.

## Perspective view

In the perspective view, you can use the mouse to fly the camera around the game world:



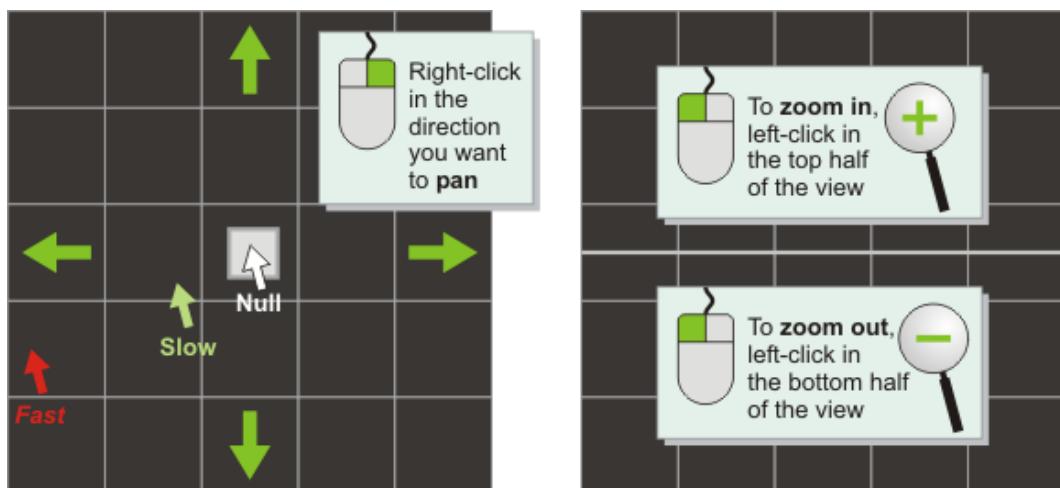
For example, to dolly (zoom) into the game world, point to the top half of the perspective view, and then hold down the left mouse button.

**Note:**

- The closer to the edge of the view you point, the faster you go.
- The center of the perspective view is a “null zone”; clicking here has no effect.

## Orthographic views

In the orthographic (plan, front and side) views, you can use the mouse to pan and zoom:



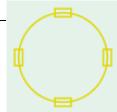
**Note:**

- The closer to the edge of the view you click, the faster you pan.
- The center of each orthographic view is a “null zone”; clicking here has no effect.

## Moving the camera in orbit mode

---

There are three orbit modes in RenderWare Studio. To change mode, first ensure that you are in orbit camera mode, then press the relevant button on the Selection toolbar. The camera behavior differs in each mode depending on whether or not an object is selected:

Button	Mode	Behavior without object selected	Behavior with object selected
	Orbit only	Camera orbits the world origin. 	Camera orbits the selected object or head object in a range of selected objects. 
	Orbit objects	The camera tilts and turns on its own axis. 	
	Camera arc		

The yellow circle changes appearance depending on the orbit mode and object selected.

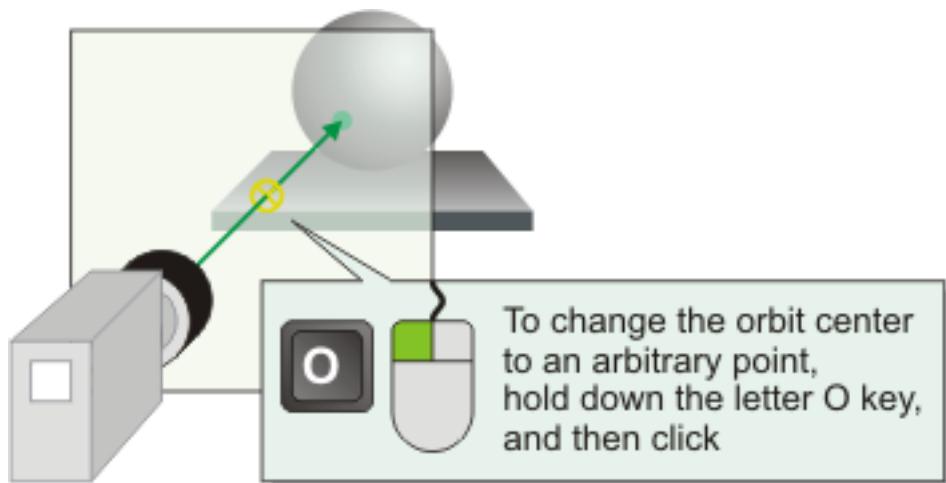
## Orbiting the camera around an alternate orbit center

You can move the orbit center to a different entity, or to an arbitrary point.

**Note:** While the camera is orbiting an alternative point, the other camera orbit modes are disabled.

1. Switch to [camera orbit](#) (p.108) mode. The yellow circle changes appearance:  


2. Press and hold the letter **O** key, and then click in the perspective view:



The Workspace determines the new orbit center by drawing a ray from the camera viewpoint through the point marked by the you clicked on the view plane. The new orbit center is the first geometry that the ray strikes.

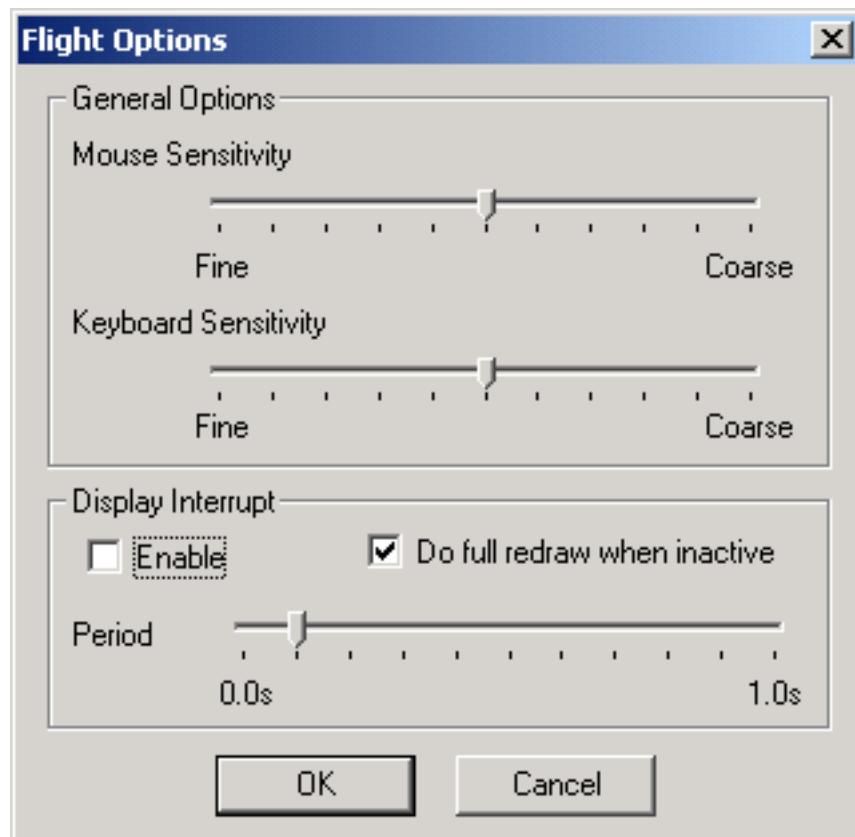
**Note:** The last orbit center  that you set is always remembered until you set a new one. Pressing the **O** key without clicking the mouse button toggles the last remembered orbit center on and off.

# Setting camera sensitivity and frame rate

To set how finely the camera responds to mouse movement or key presses, or to set a minimum frame rate for camera motion:

- On the **Options** menu, click **Flight...**

The Flight Options dialog appears:



## General options

### Mouse Sensitivity and Keyboard Sensitivity

Controls how finely the camera responds to the mouse and the keyboard. The finer the sensitivity, the greater the camera's response to the same mouse movement (or duration of key press).

## Display interrupt

If your game contains many entities, or you have a relatively slow computer, then camera motion can become jerky. This is because, by default, your computer renders each frame completely. If frames take a long time to render, then the camera position can change significantly between each one, resulting in jerky camera motion.

For smoother camera motion, you can set a maximum time limit for rendering each frame; if a frame is not completely rendered in that time, then your computer begins rendering the next frame. This sets a minimum frame rate for camera motion, at the expense of not always completely rendering every frame.

### Enable

Interrupts the rendering of any frame that takes longer than the time specified by **Period**, and begins rendering the next frame.

If you leave this check box deselected, then your computer renders each frame completely, regardless of how long it takes.

### Do full redraw when inactive

Renders the entire frame after the camera stops moving, regardless of how long it takes.

If you leave this check box deselected, and display interrupt is enabled, then the frame might not be completely rendered after the camera stops moving.

### Period

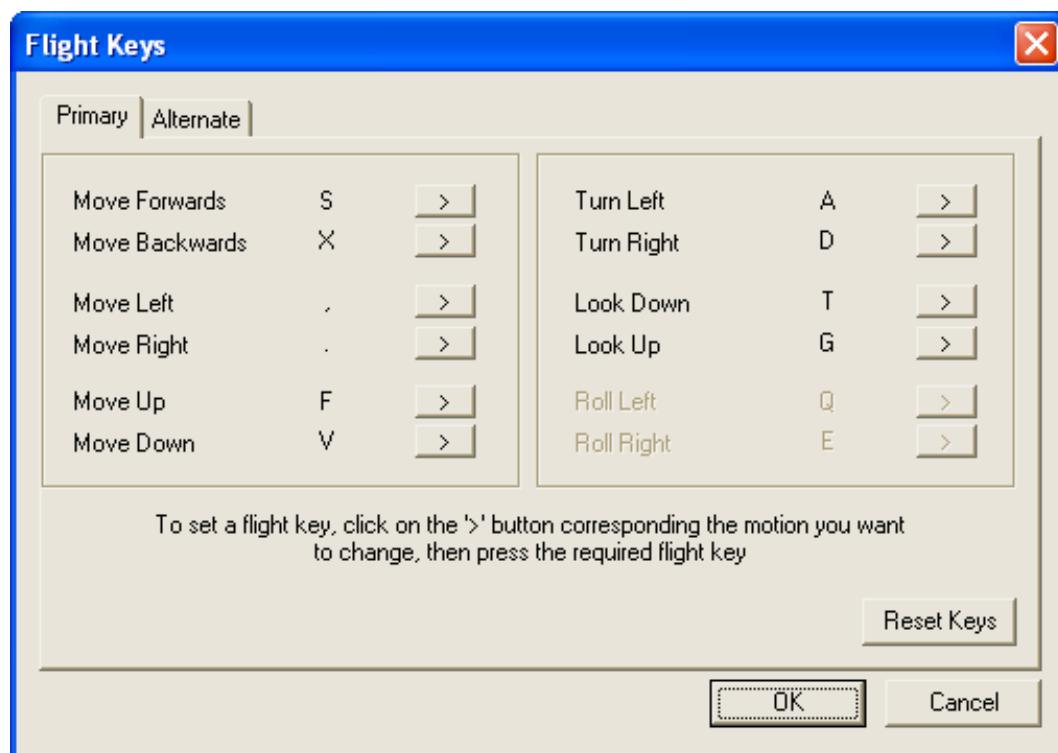
Sets the maximum time for rendering a frame (in seconds). For example, a value of 0.0416 sets a minimum frame rate of 24 frames per second.

# Using keys to control flight

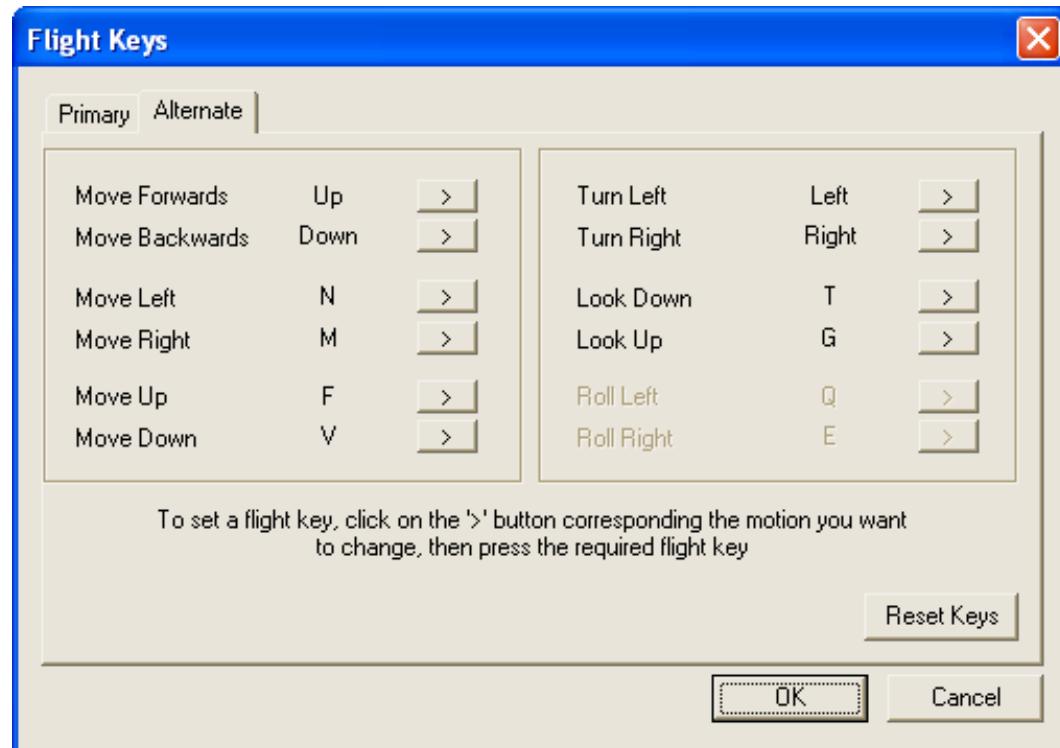
To view which keys control flight, or to change the assigned keys:

- On the **Options** menu, click **Flight Keys**

The Flight Keys dialog appears listing the primary keys assigned to flight movement.



To list alternate keys assigned to the same movement, click the Alternate tab.



To change the key for a particular action click the > button adjacent to the appropriate movement, such as *Move Forwards* or *Turn Left*. RenderWare Studio prompts you to press the key you want to use:



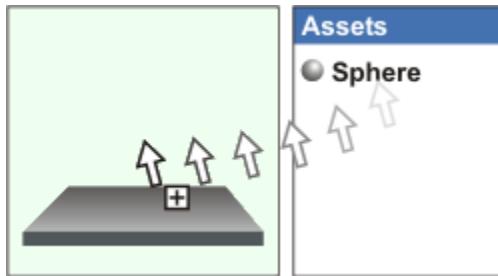
Press the key you want to use. The key you choose displays in the dialog next to movement.

# Creating an entity

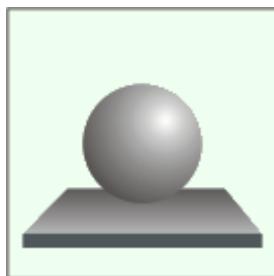
---

To create an entity in the current folder of your game:

1. Drag an asset from the Assets window to the Design View window. While dragging, you can adjust how the new entity aligns with existing entities.



2. The new entity appears in the Design View window, and is listed in the Game Explorer window.



Special tips:

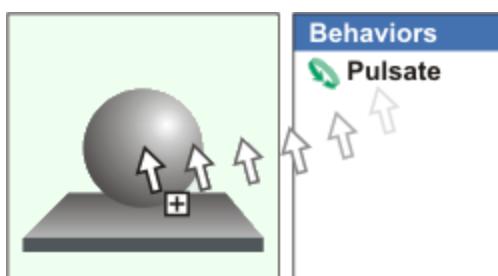
- Another way to add an entity is to drag a behavior directly from the Behaviors window to the appropriate folder in the Game Database window.
- Holding down the **Ctrl** key while dragging the asset constrains it to moving up and down the Y axis.

## Attaching a behavior

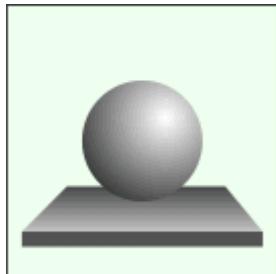
By default, Workspace attaches the CEntity behavior to new entities.

To attach a different behavior to an entity:

1. Drag the behavior from the Behaviors window to the entity in the Design View window.



2. If you connect to a target, then you can dynamically view the behavior in the game.



After attaching a behavior, you can set its attributes (for example, the speed and color of the pulsation) in the Attributes window.

For descriptions of the behaviors supplied with RenderWare Studio, see the Game Framework Help.

# Selecting entities

---

Before you can move, rotate, or scale an entity, or edit its properties or attributes, you must first select it.

## Selecting a single entity

To select a single entity, either:

- Click one of the “pick”[toolbar](#) (p.251) buttons, such , and then click the entity in the [Design View](#) (p.276) window.  
or
- Click the entity name in the [Game Explorer](#) (p.288) window.

In the Design View window, the selected entity changes to a wireframe representation, with “drag axes” around it.

In the Game Explorer window, the selected entity name is highlighted.

**Tip:** Instead of clicking the Pick toolbar button, press **F9** to toggle between pick mode and the current [camera mode](#) (p.108).

## Selecting multiple entities

In the Design View window, to select multiple entities:

1. Click the Pick toolbar button.
2. Press and hold down **Ctrl** while clicking each entity.

In the Game Explorer window, to select multiple entities...

- that are listed consecutively:
  1. Click the first entity.
  2. Press and hold down **Shift**.
  3. Click the last entity.
- that are not consecutive:
  1. Press and hold down **Ctrl**.
  2. Click each entity.

**Note:** You cannot select multiple entities that are in different folders. If you do select multiple entities in the Design View window that are in different folders, only *one* of the selected entities will be highlighted in the Game Explorer window.

## Deselecting entities

To deselect all entities, either:

- Click an empty area of the Game Explorer window.  
or
- Press **Esc**. (This works only if the Design View window has the keyboard

“focus”. To set the keyboard focus to the Design View window, click in the Design View window.)

To deselect one of several selected entities:

- Press and hold down **Ctrl**, and then click the entity.

## Disabling further (accidental) selection

If you have selected entities that you want to manipulate (move, rotate, or scale), and then you accidentally click another entity instead of the handle of a selected entity, you will need to start over, and reselect the entities.

To avoid this, turn on the *selection lock* immediately after selecting the entities. This disables further selection of entities in the Design View window (you can still select entities in the Game Explorer window).

To toggle the selection lock, either:

- Press the space bar.  
or
- Click the  toolbar button.

**Tip:** When this toolbar button is indented, the selection lock is on.

# Moving, rotating, and scaling entities

---

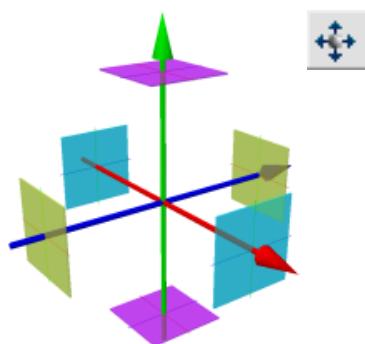
To move, rotate, or scale an entity:

1. Click one of the toolbar buttons shown below.
2. **Select** (p.121) the entity (for example, by clicking the entity in the **Design View** (p.276) window).

In the Design View window, the selected entity changes to a wireframe representation, with drag axes.

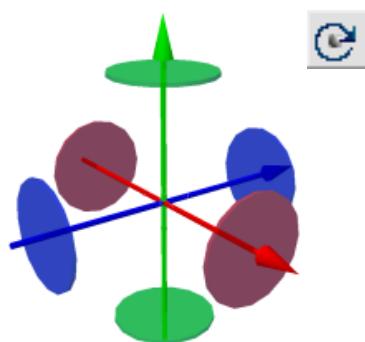
**Note:** If you click the World entity, you will see it change to a wireframe representation, but without drag axes. This is because you cannot move, rotate, or scale an entity whose primary asset file is a .bsp file.

3. Drag a part of the axes, as described below.



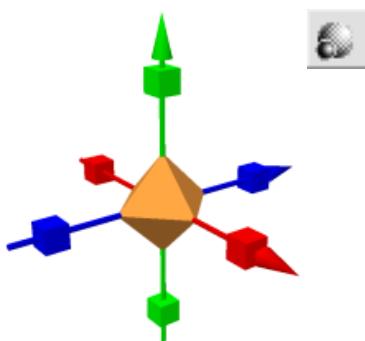
## Pick and Move

To move the entity along an axis, drag a rod. To move the entity in any of the XYZ planes, drag a square panel.



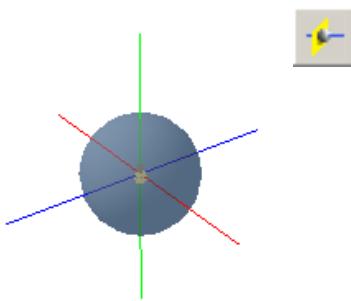
## Pick and Rotate

To rotate the entity around an axis, drag a rod or disc.



## Pick and Scale

To uniformly scale the entity, drag the octahedron at the center of the drag axes. To nonuniformly scale the entity along a plane, drag a rod or box.



### Pick and Drag

To move the entity along the floor (snaps the origin of the entity to the floor).

**Tip:** To reset any of the move, rotate, or scale operations as you perform them, click the right mouse button while you are dragging. Be aware that if you use this method, you are **resetting** the operation and your changes will not be added to the [Undo Stack](#) (p.251).

## Resizing the drag axes

To increase or decrease the size of the axes, press + or - (on the main keyboard, not the numeric keypad).

This keyboard shortcut works only if:

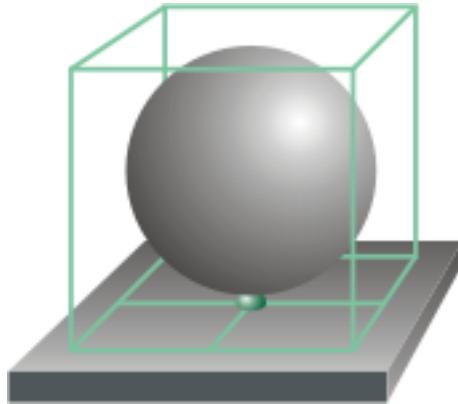
- The Design View window has the keyboard “focus”. To set the keyboard focus to the Design View window, click in the Design View window.  
and
- The [Automatically rescale drag axes...](#) (p.162) option is selected.

## Aligning with existing entities

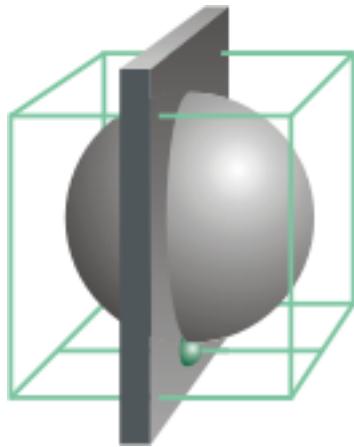
---

While you are dragging a new entity into the Design View window, the Workspace uses a point on the bounding box of the new entity as a collision test point. You cannot drag the new entity into existing entities past that point.

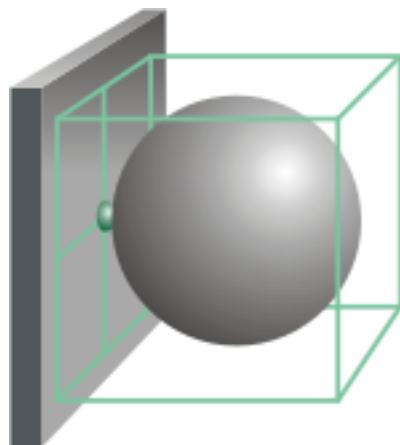
By default, the collision test point is at the bottom center of the bounding box. This allows you to easily align the new entity with a floor surface:



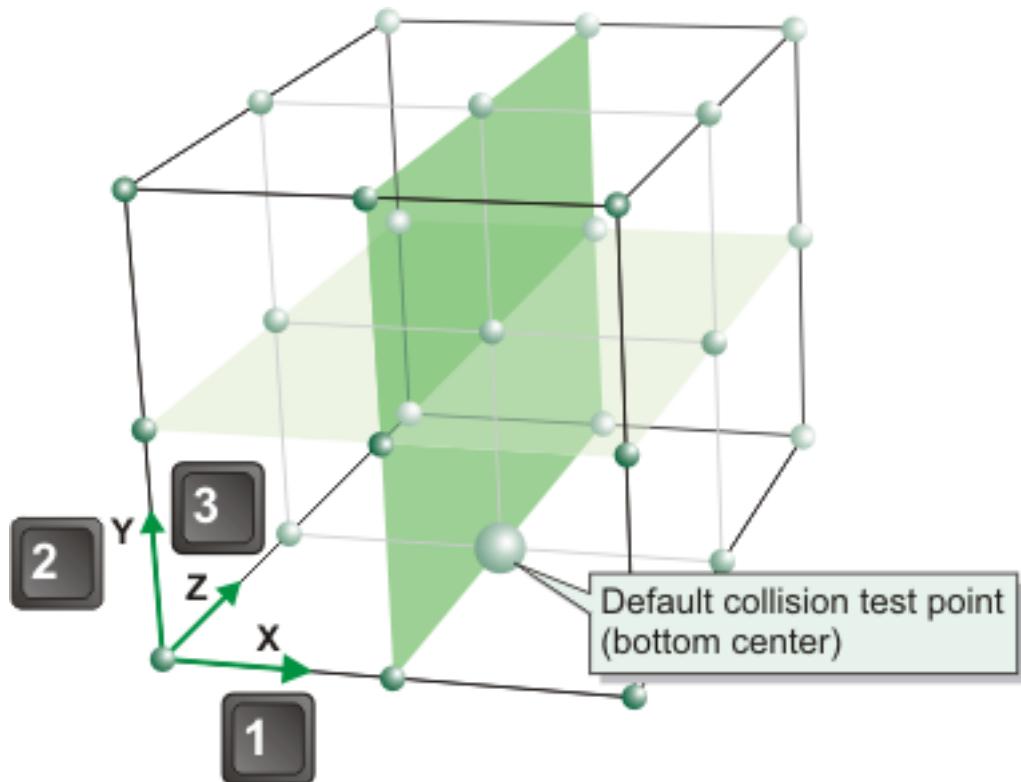
However, this is not always the most suitable location for the collision test point. For example, it allows you to move the entity into walls:



To change how the new entity aligns with existing entities, you can move the collision test point. For example:



You can move the collision test point to the following locations on the bounding box:



To move the collision test point while dragging a new entity into the Design View window:

- To cycle around locations in the **X** plane, press **1**.
- To cycle around locations in the **Y** plane, press **2**.
- To cycle around locations in the **Z** plane, press **3**.

This applies only while you are dragging a new entity into the Design View window. The next time you [move](#) (p.123) the entity, no collision testing occurs; you can move the entity freely to any location.

# Creating an entity from a template

---

Rather than creating an entity from an asset—and then attaching a behavior and editing its attributes—you can start with a special type of existing entity, known as a template.

**Tip:** If you want identical entities in different folders, then, rather than creating a new entity each time, you can [refer](#) (p.174) to the same entity.

## To create a template:

1. [Create an entity](#) (p.119) that you want to use as the template.
2. In the Game Explorer window, right-click the entity, and then click **Create Template**.

This creates a template (effectively, a copy) of the entity in the Templates window. It does not affect the original entity.

## To create an entity from a template

Either:

- Drag the template from the Templates window to the Design View window.  
or
- Drag the template from the Templates window to a folder in the Game Explorer window. (Using this method, you need to set the transformation matrix attribute of the entity.)

The new entity is a copy of the template: it begins with the same assets, behavior, and attributes as the template. Unless the template uses attribute shares (see below), each entity you create from a template is completely independent.

## Creating entities that share attribute values

If you want entities created from a template to share attribute values (that is, if you change an attribute value in one entity, it affects all entities created from the same template), then create the template from an entity that uses [attribute shares](#) (p.170).

# Creating a spline path

---

Splines can be used for moving game objects, such as a camera as in the [Spline camera example](#) (p.545), or non-playing cars controlled by AI in a racing game, along predetermined paths.

## Creating a spline

To create a new spline asset, right-click in the Assets window and select **New ▶ Spline**.

You can also right-click in the Assets window and select **New ▶ Asset** and then:

- Right-click the newly created asset
- Select **Properties**
- Browse to the spline asset file in the **File** field of the [Properties](#) (p.102) dialog that appears and then click **OK**.

**Note:** New spline assets can be checked in and out of [NXN alienbrain](#) (p.187) in the same way as any other asset.

After the spline asset is created, drag the new asset from the Game Explorer window to the Design View window. This automatically creates a new entity in the active folder.

In the Game Explorer window, right-click the new spline asset, and then select **Edit**; the [Spline toolbar](#) (p.251) becomes active.

## Using a spline exported from a design package

Splines exported from artwork packages such as 3ds max or Maya can also be edited using the spline editor. To use such files:

1. Use the RenderWare Graphics exporter plugin template to export files in the correct .spl file format.
2. Drag the exported spline asset into the Design View as you would any other asset. The exported spline file can be edited using the spline editor in exactly the same way as if it had been created in Renderware Studio from scratch.

## Editing a newly created spline

1. Click the **New nodes**  button, then click in the Design View to begin drawing the spline. Each time that you click in the Design View, a new node (or spline control point) is inserted and the segment between both nodes is joined together by a line.
2. After you have created as many nodes as required, to link the node segments that you have created into one loop as in the Spline Camera example, click on the **Close node**  button. (To re-open the loop for inserting further nodes, click the **Open node**  button.)

3. To alter the position of the nodes in the Spline Editor, click on one of the pick buttons such as the **Pick and Drag**  button, then click the relevant node in the Design View window. After the [drag axes](#) (p.123) have appeared, drag the node as you would any other object until that segment of the spline curve is the desired shape.

## Editing an existing spline

To edit a new spline asset, the spline must be in the [active folder](#) (p.166) in the [Game Explorer](#) (p.288) window.

To insert nodes into an existing spline path, either:

- Hold down the **Ctrl** key, click on two adjacent nodes, and then click the **Insert node**  button. A new node is inserted equidistant from the two selected points.  
or
- Follow these steps:
  1. To edit a closed spline asset, click the **Open node**  button. This allows you to add further control points.
  2. Click in the Design View window to create the next node on the spline path as though you were creating a new spline path from scratch.
  3. To close the spline asset again once you have added the final node, click the **Close node** 

# Creating box and trigger volumes

---

Box volumes are simple cuboid entities that you can place into your game without the need to import an asset. They can be used as placeholders for entities that will be created in the future, or (more commonly) as the basis for trigger volumes.

## Creating a box volume

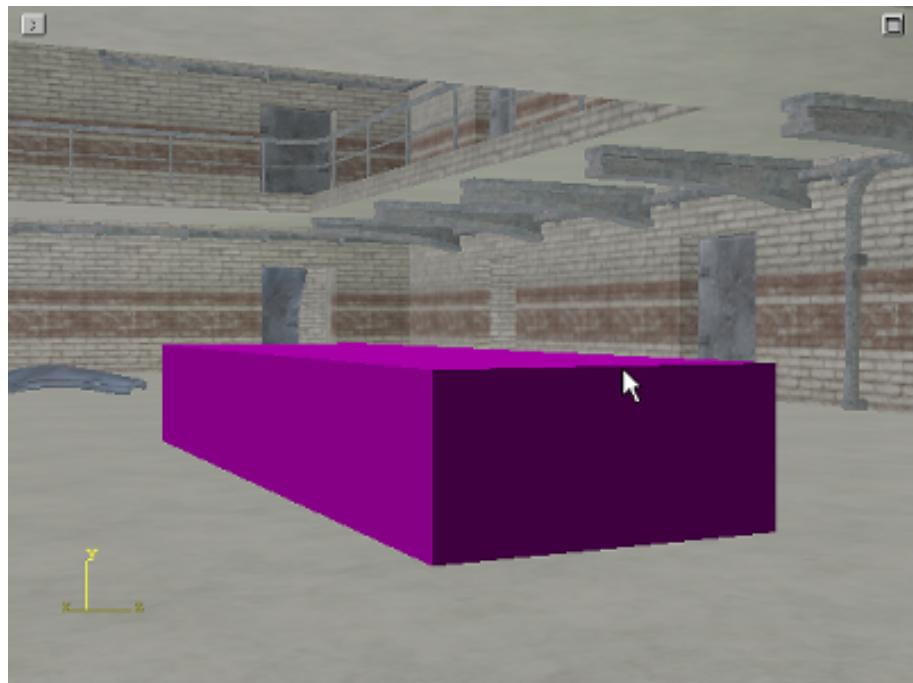
1. Ensure that the **Volume toolbar** (p.251) is displayed in Workspace. Then:
  - To create a box whose axes are aligned with the current view in Design View, click the **Create box volume** button .

or

  - To create a box whose axes are aligned with those of the game world, click the **Create axis-aligned box volume** button .
2. Click the **Texture selection** button , and choose the appearance of the box by selecting a texture from the list that appears.
3. In the Design View window, drag to draw the bottom or top face of the box. To fix the size of the face, release the mouse button.



4. Without clicking, move the cursor up or down to change the height of the box. To fix the height, click the button again.



A new box is added as an entity to the current working folder, and associated with the CEntity behavior.

**Note:** When you've finished creating box volumes, remember to select a different tool from the toolbar.

## Converting a box volume into a trigger volume

In Workspace, you can convert a box volume into a trigger volume by replacing the default CEntity behavior with one from the **Behaviors ▶ Triggers** category. The revised entity will send the event specified in its *Send Event/Trigger* attribute (and perform any other actions defined by its behavior) when the player enters the defined volume.

# Light mapping

---

This section of the documentation demonstrates how light mapping entities can be added to the game world using RenderWare Studio.

RenderWare Studio has a number of special facilities to aid in the effective light mapping of a level, including a custom layout designed specifically for implementing light mapping entities within the game world.

The effects of added light entities can be previewed in the **Design View** window of RenderWare Studio. Once the various parameters have been set to your satisfaction, the light map files required to light the level can be generated.

The following sequence of tutorials introduces the components of the RenderWare Studio user-interface related to light mapping and also demonstrates how to create and control light map entities. In addition, features such as the ability to control the interaction of materials with light are also introduced, as well as the subject of creating area lights.

# Light mapping user interface

## In this tutorial

We introduce you to the basic light mapping facilities of RenderWare Studio, showing how to set up for light mapping and indicating some of the most useful menus and windows.

In this tutorial, the Genre Pack 1 project is used throughout. The tutorial is split into a number of lessons:

- Selecting the light mapping layout.
- The light mapping toolbar.
- Viewing the **Light Map Preview Settings**.
- The **Scene Light Map** window.
- The **Asset Light Map** window.

## Selecting the light mapping layout

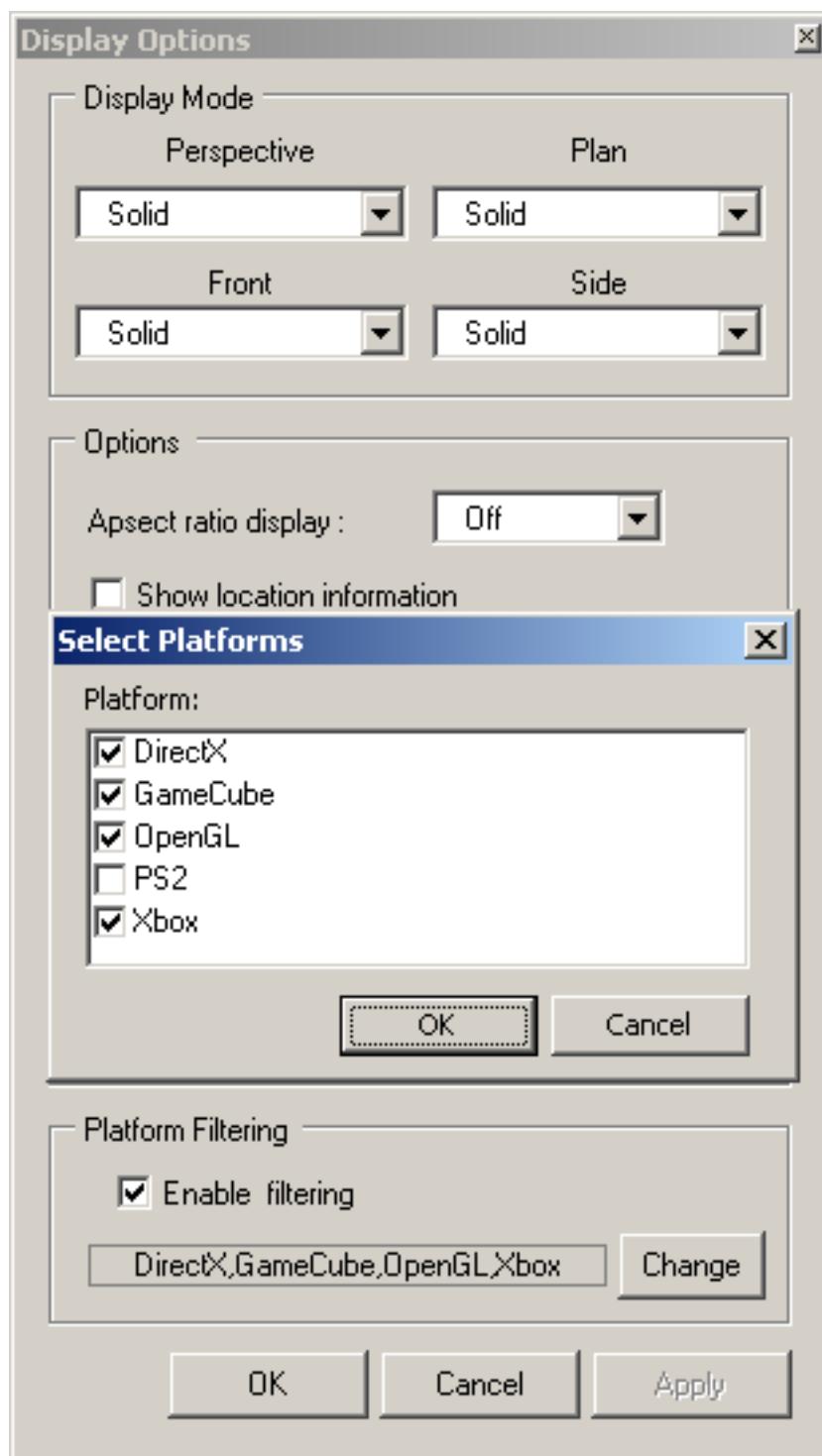
To tailor the RenderWare Studio environment to assist with carrying out general light mapping tasks:

1. Load the following RenderWare Studio project into Workspace:

```
C:\RW\Studio\Examples\GP1 Tutorial.rwstudio
```

**Note:** This assumes that Genre Pack 1 has been installed.

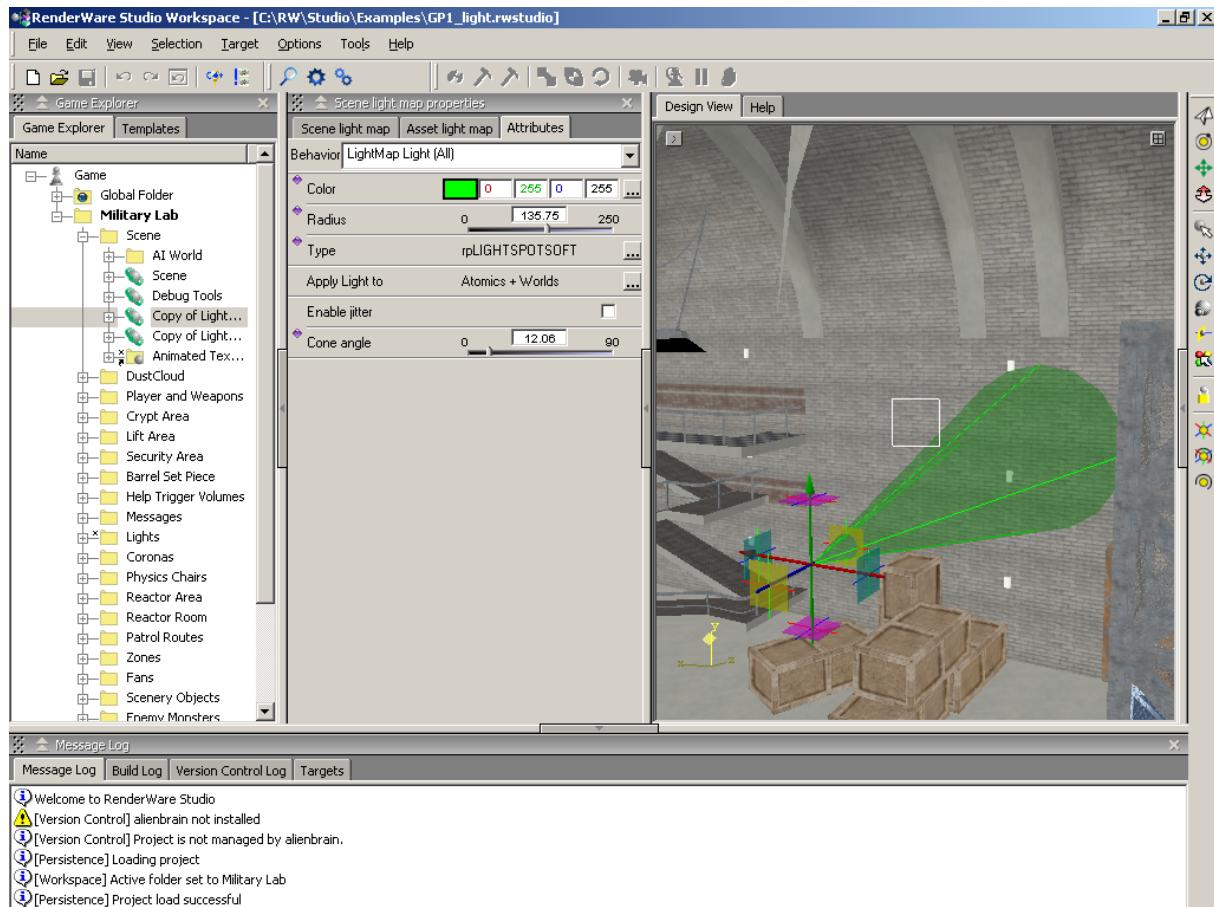
2. On the main menu select **Options ▶ Display options** to display the **Display Options** window.
3. On this window, in the **Platform Filtering** section, ensure that the **Enable Filtering** checkbox is selected.
4. Click the **Change** button. This will display a **Select Platforms** window. Deselect the PS2 checkbox and click **OK**. This ensures that PS2 specific data is not used, thus simplifying this tutorial. This is illustrated by the following screenshot:



5. Use **File ▶ Save Project As** to save the project to a new project called:  
`GP1_light.rwstudio` in the same directory.
6. On the main menu select **View ▶ Layouts ▶ Light mapping**. This layout tailors the user interface to the needs of light mapping by:
  - Eliminating unnecessary toolbars.
  - Adding a light map toolbar.
  - Adding a panel for scene light map.
  - Adding a panel for asset properties.

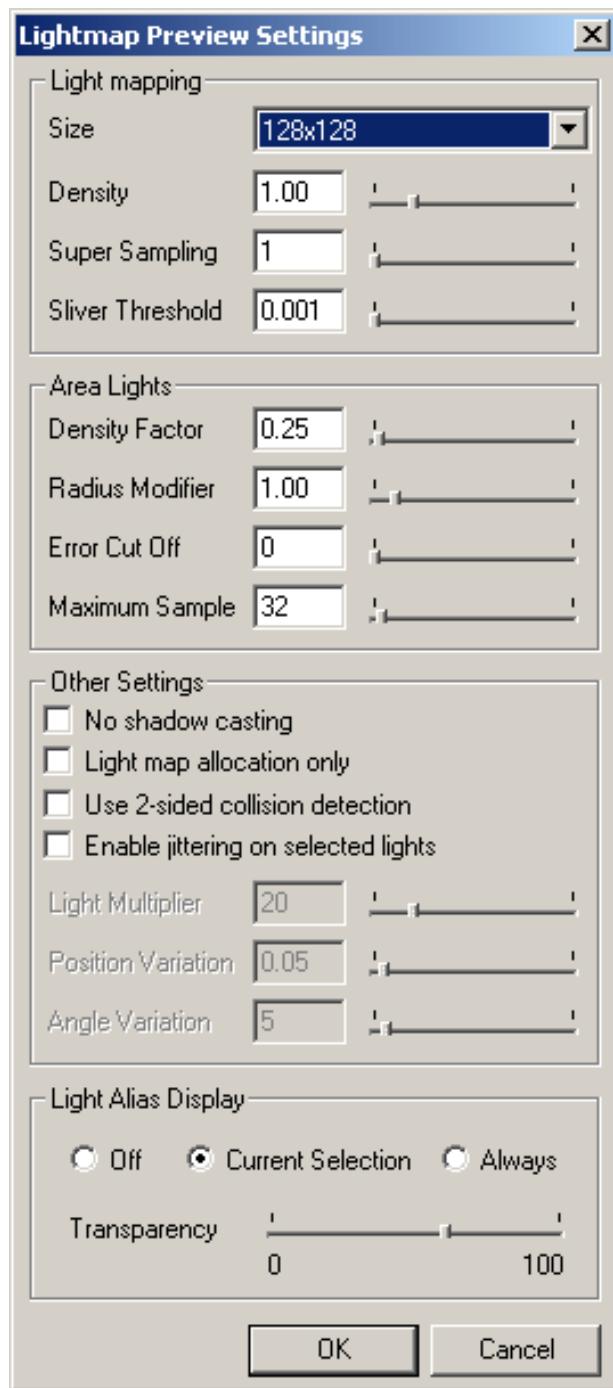
- Adding a template window for light creation.

A typical light mapping layout is illustrated by the following figure:



## Viewing the light map preview settings

The Studio environment allows you to preview the affects of lighting in the **Design View** window. This lesson shows how to select this window.



1. On the main menu **Options > Light map** can be used to view and modify light map preview settings. These affect the previewing of a level as seen in the **Design View** window.

Alternatively, select the settings button from the light mapping toolbar.

Further details on the **Light Map Preview Settings** window can be found in the [Light Map Preview Settings window](#) (p.294) topic.

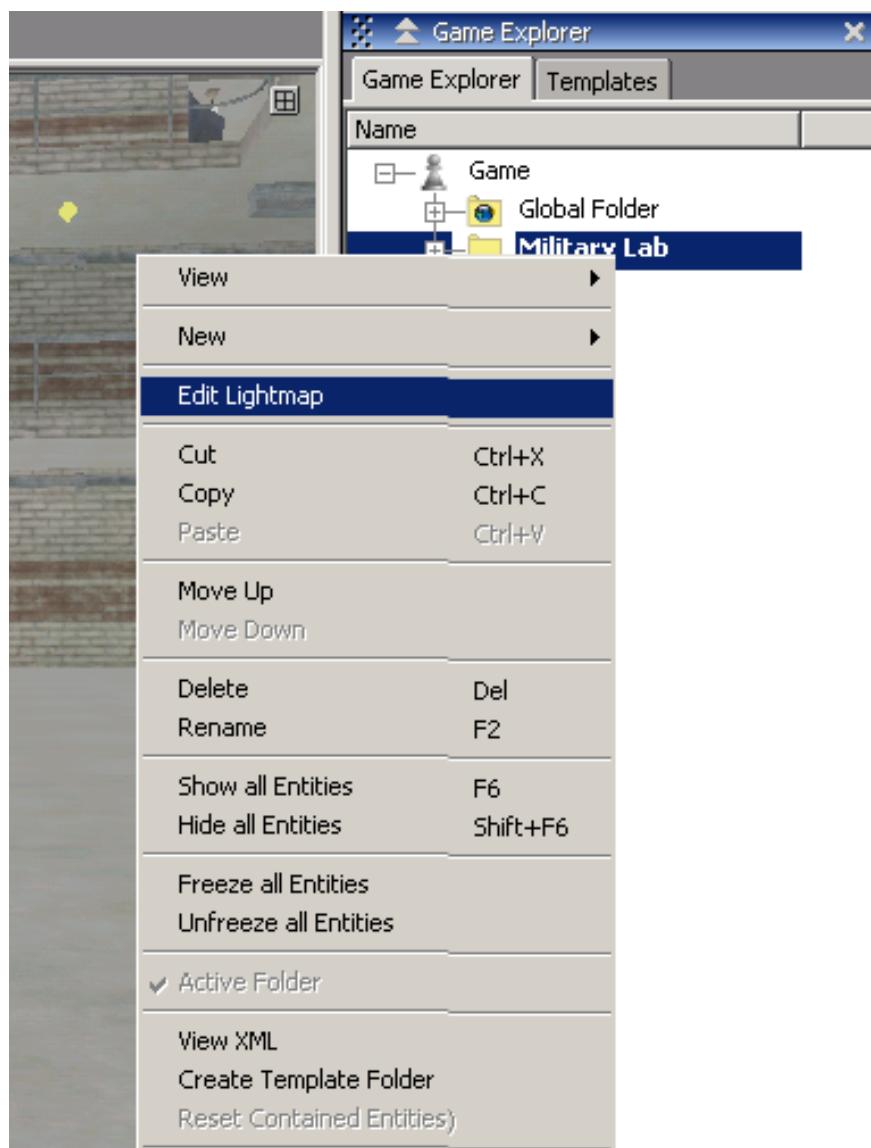
## The Scene Light Map window

The **Scene Light Map** window allows you select whether a scene is to be light mapped or not. It also allows various parameters associated with light mapping to be modified. Additionally, various characteristics of area lights can be set globally

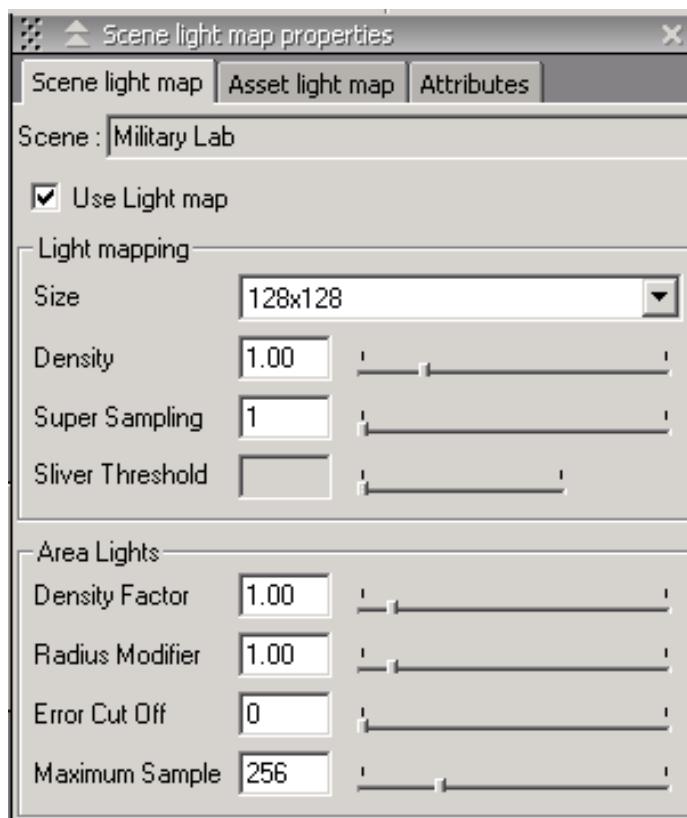
for the scene. The following procedure shows how to bring up the **Scene Light Map** window for a specific scene.

1. Right-clicking on a “scene folder” in the **Game Explorer** window will show an **Edit Lightmap** option.

To demonstrate this, in the **Game Explorer** window, right-click on the folder **Military Lab** and then select the **Edit Lightmap** menu option as shown in the following figure:



Clicking the **Edit Lightmap** option will activate the **Scene Light Map** window. This will appear as a tabbed window in the **Scene Light Map properties** window. This is also shown in the following figure:



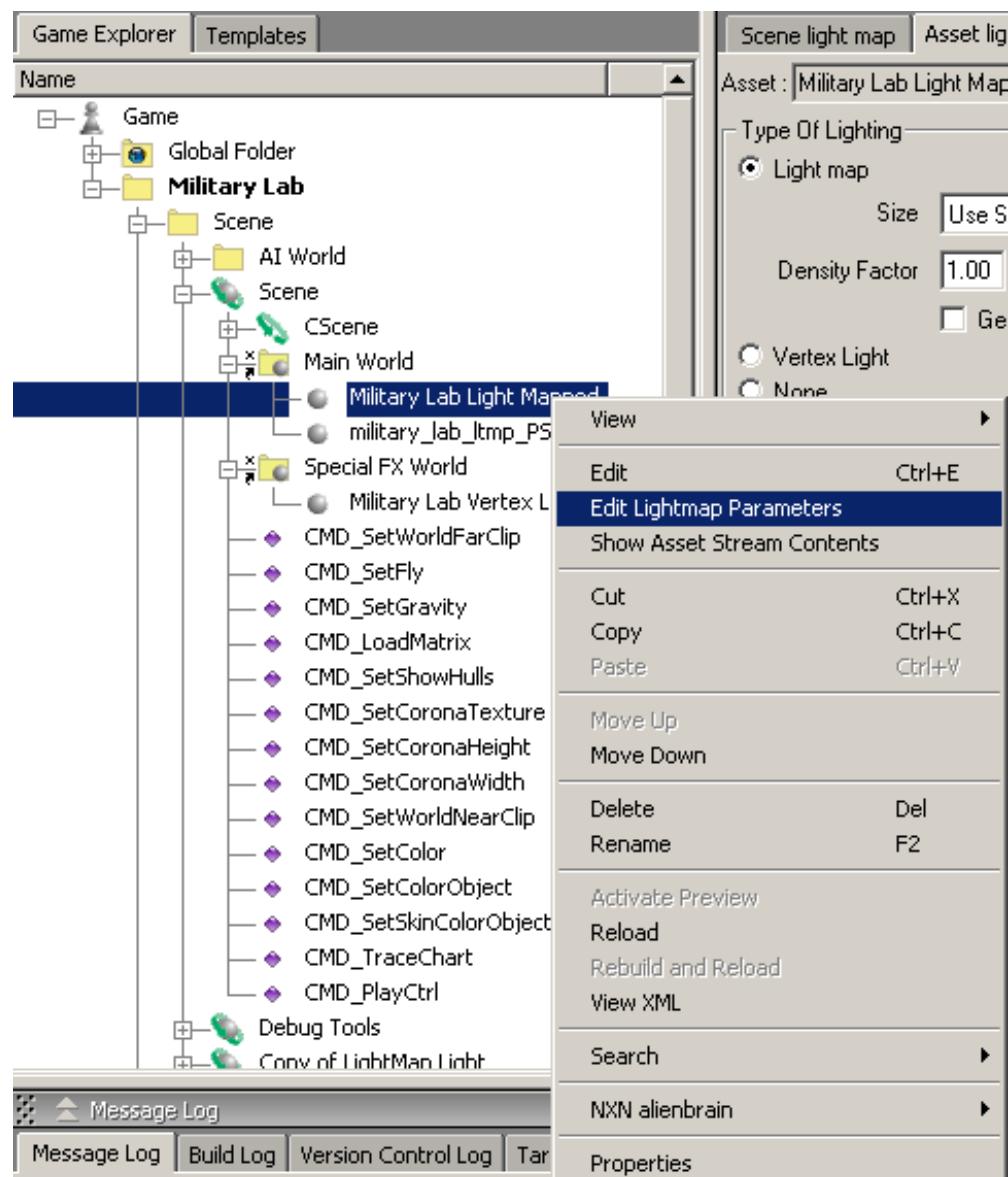
**Note:** If light mapping is enabled for a scene folder, light mapping will be applied *only* to objects within the folder itself or a sub-folder of it.

Further details of the **Scene Light Map** window can be found in the [Scene Light Map window](#) (p.313) topic.

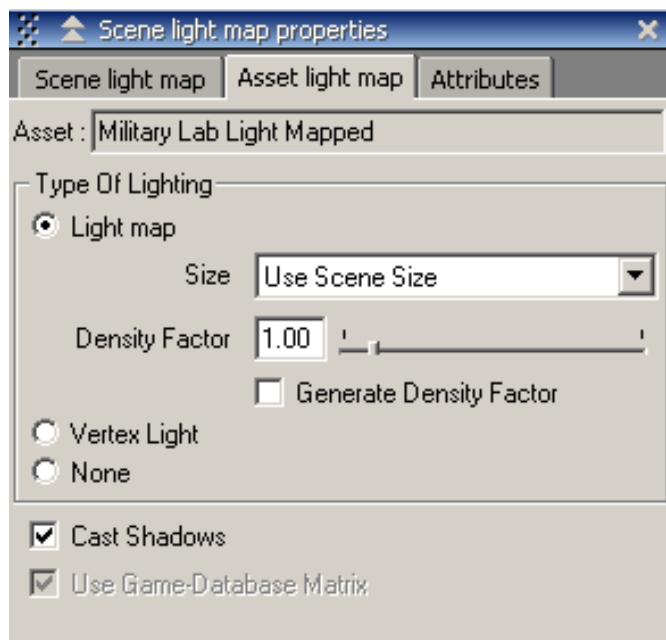
## The Asset Light Map window

The **Asset Light Map** window allows you to select the type of lighting to be applied to an asset, and whether the asset should be lit based on the game world and whether it will cast a shadow or not.

1. Right-clicking on an asset in the **Game Explorer** window will show an **Edit Light Map Parameters** option. To demonstrate this in the **Game Explorer** window, navigate to: **Military Lab ▶ Scene ▶ Scene ▶ Main World ▶ Military Lab Light Mapped**
2. Right-click on this asset. From the menu select **Edit Light Map Parameters** as shown in the following figure:



This will activate the **Asset Light Map** tabbed window in the **Scene Light Map properties** window. This is also shown in the following figure:



Further details of the **Asset Light Map** window can be found in the [Asset Light Map window](#) (p.267) topic.

# Preparing assets for light mapping

## In this tutorial

We show you how to:

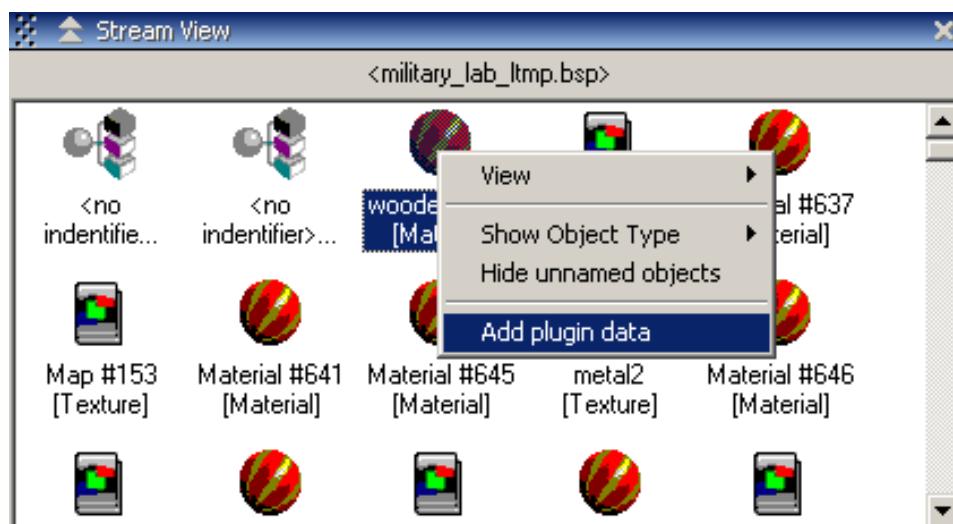
- Prepare objects for light mapping.
- Modify parameters associated with the light map plugin window.

## Preparing assets for light mapping

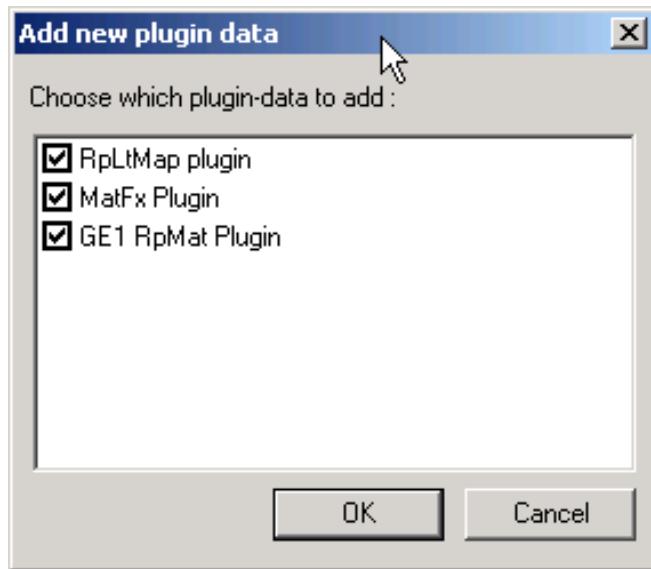
**Note:** The following steps only need to be carried out if assets have been exported from the content creation tool, such as 3ds max or Maya, *without* the light map data plugin having being enabled.

The following procedure shows how light map data can be added to an asset on a per-material basis.

1. Navigate to the asset to be light mapped. For example, in the **Game Explorer** window navigate again to the Military Lab Light Mapped asset.
2. Right-click the asset and select **Show Asset Stream Contents**.
3. For each material, right-click and select **Add plugin data**:



4. Select the check boxes to add plugin data:

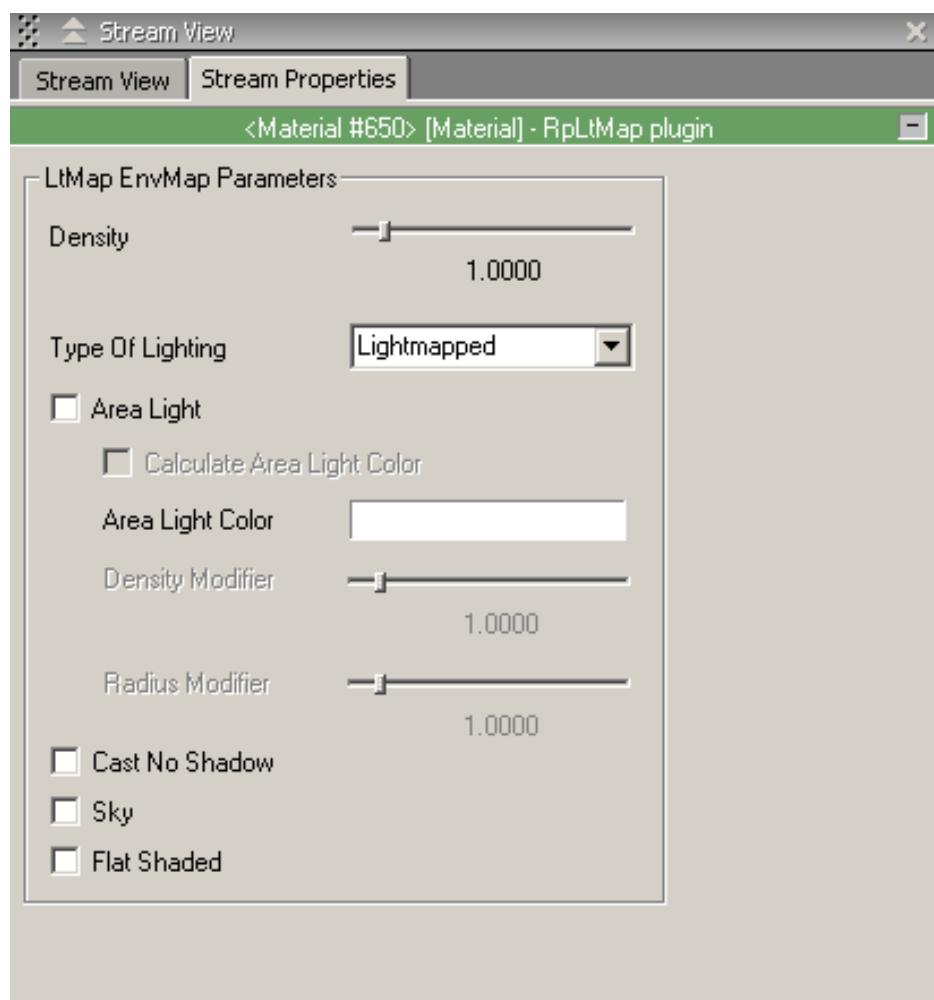


RpLtMap represents the RenderWare light map plugin. This needs to be selected in order to add light map data for the asset.

At this point the asset is now prepared for light mapping.

## Light map plugin window

When a material has had light map plugin data added to it, various parameters can be specified and adjusted. Surfaces using this material can be lit in several different ways, as well as being identified as area lights. The following window can be accessed by clicking on the material:



## LtMap EnvMap Parameters

### Density

This represents the number of point lights within an area light. This factor can be modified by both global and per-material modifiers.

### Type of Lighting

This drop down list box allows the lighting characteristics of surfaces using this material to be selected from one of:

#### None

No light will be emitted from surfaces using this material.

#### Light mapped

Surfaces using this material will be light mapped.

#### Vertex Lit

Surfaces using this material will be vertex lit.

### Area Light

Selecting the Area Light check box makes this material act as an area light. The following options can then be selected and modified.

#### Calculate Area Light Color

If selected the color will be calculated based on the color of the texture and the material used for the area light.

**Area Light Color**

Click in the color box to select the color of the area light.

**Density Modifier**

This per-material modifier changes the number of point lights in the area light. Increasing this number results in greater accuracy of lighting at the expense of greater computation time.

**Radius Modifier**

This per-material modifier affects the Region Of Influence (ROI) of the area light.

**Cast No Shadow**

If this check box is selected the material will not cast a shadow, that is this material does not block light.

**Sky**

If this check box is selected this material will block all light except for directional light. This means, for example, a world can be enclosed by sky polygons and not block a directional light representing the sun.

**Flat Shaded**

If this check box is selected surfaces using this material will be flat-shaded using polygon normals, rather than vertex normals.

# Creating and using entity lights

## In this tutorial

We show you how to set up and manipulate entity lights by:

- Creating an entity light.
- Setting the attributes of a light entity.

## Overview

There are two main types of lighting option to consider when designing a level:

### Entity light

Entity lights are created from a “LightMap Light” template. They can be located and manipulated within the game world in a way similar to other entities. This tutorial shows how to create and modify entity lights.

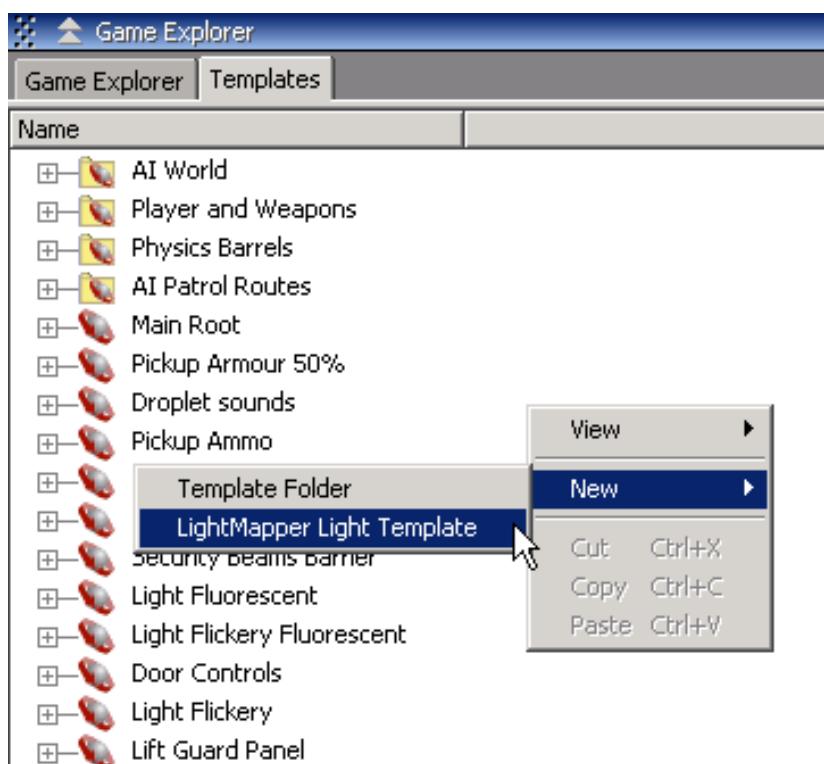
### Area light

Area lights are actually made up of a number of point lights. Area lights are discussed in the [Area lights tutorial](#) (p.149).

## Creating an entity light

Entity lights are set up via templates. If the “LightMap Light” template does not exist it must first be created:

1. Load the RenderWare Studio project GP1\_light.rwstudio that was created in the [Light mapping user interface tutorial](#) (p.133).
2. Right-click in the templates window and a create a new light template by selecting **New ▶ LightMapper Light Template** :

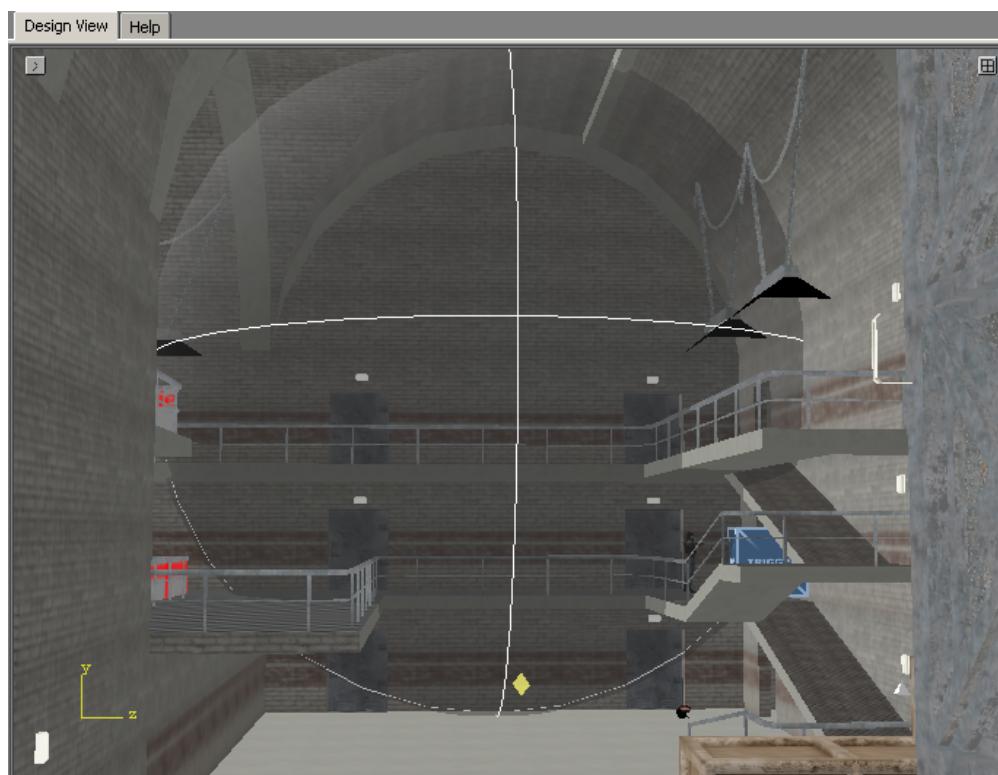


This will create a template called "LightMap Light" and this template will be added to the list of available templates in the **Templates** tab of the **Game Explorer** window.

These light entities can now be dragged and dropped into the scene as shown through the **Design View** window.

**Note:** By default no platform flags are set, so this will not affect the console. Since the light is an entity, this can be overridden as required.

3. Drag the new light template into the **Design View** window as illustrated by the following figure:

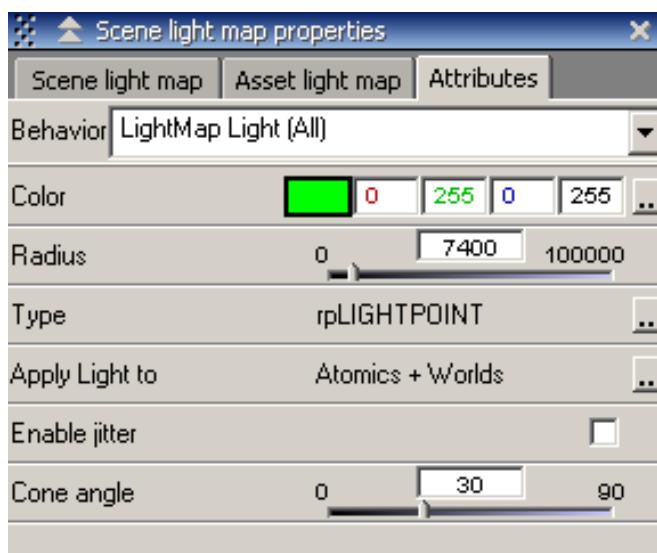


**Caution:** The light mapping algorithm will only be applied to lights found in, or below, the folder where light mapping was enabled. So, for example, the light entity created might be moved, using the Game Explorer to the Scene folder.

## Setting a light's attributes

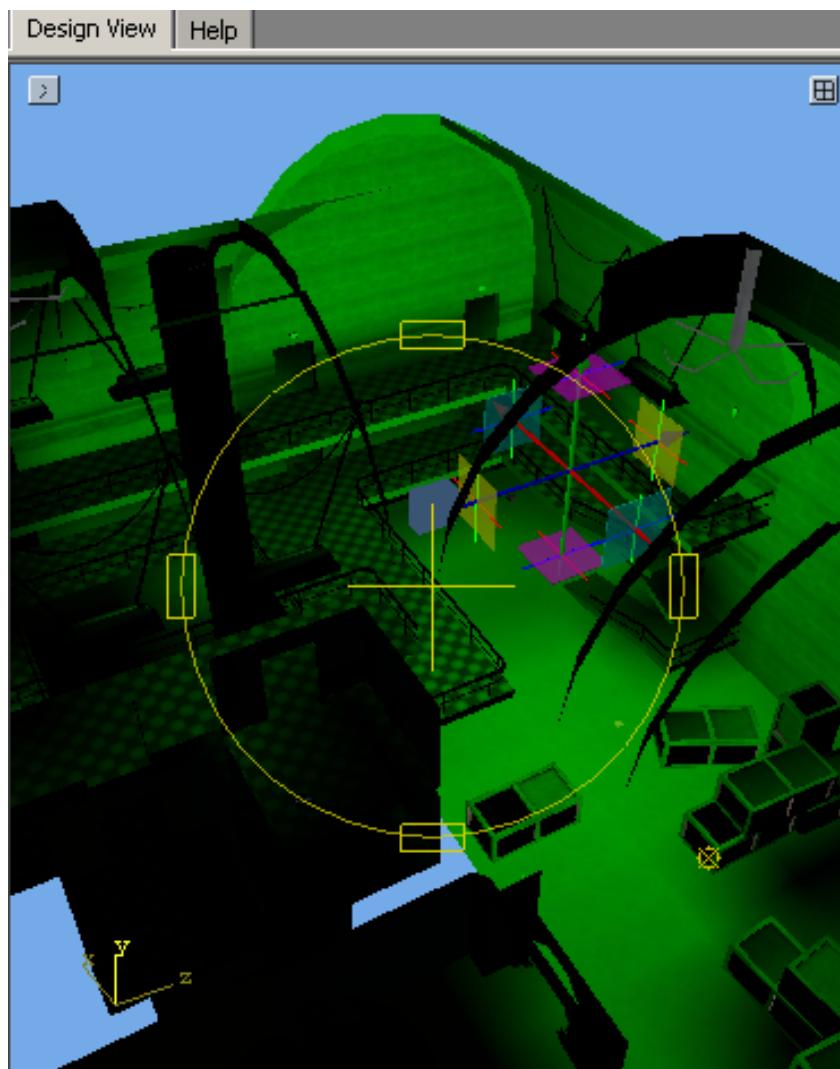
The entity light is a normal entity and as such it has attributes that can be modified. This section of the tutorial shows how to invoke the **Attributes** window.

1. Select the light entity to be modified in the Game Explorer window, and then click on the **Attributes** tabbed window in the **Scene light map properties** window.



Attributes can now be modified as required.

2. For the purposes of this tutorial we will now modify the color of the light.  
By altering the red, green, and blue values, obtain a green light.
3. To see the effects of the new colored light entity in the level, click the Preview button on the light mapping toolbar. The lighting will be generated and displayed in the design view window. The following figure provides an example of what will be seen:



Other changes to light entity attributes can be previewed in this way.

# Area lights

---

## In this tutorial

You learn how to create and manipulate area lights.

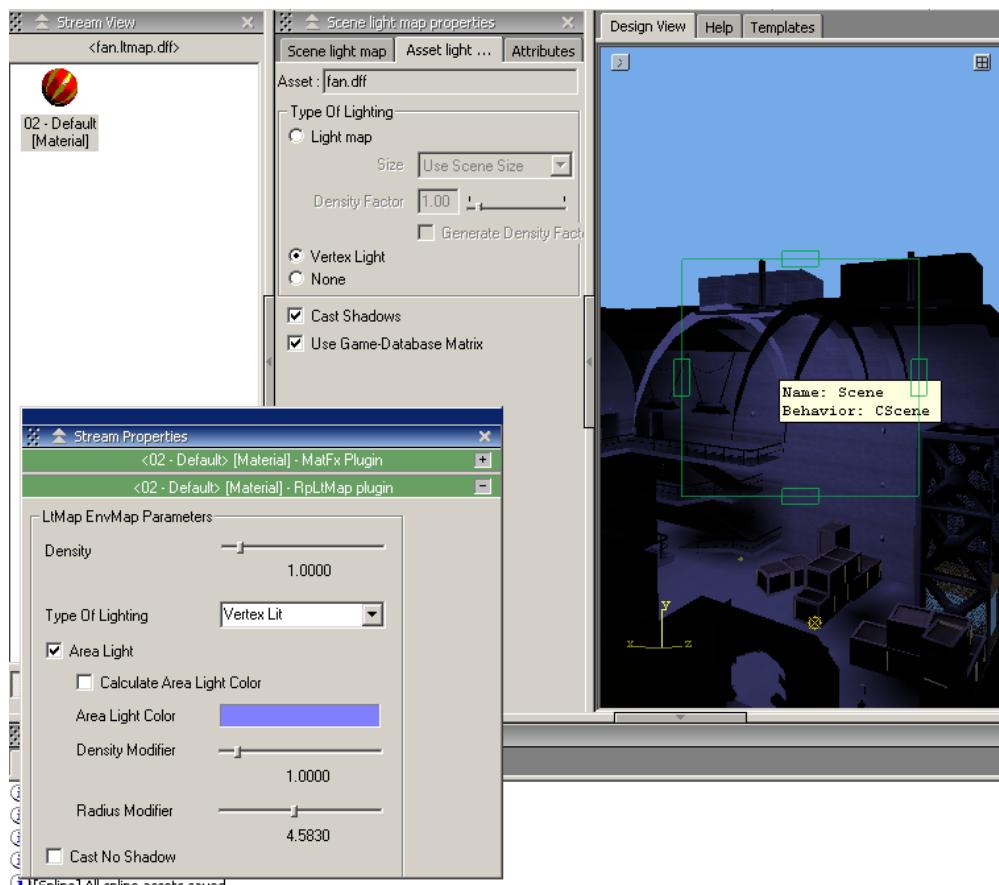
## Overview

An area light is set up by specifying a material as a light source. These materials could represent, for example, fluorescent tubes, skylights and windows. An area light actually consists of a number of point lights. The accuracy of lighting produced by an area light will improve as the number of point lights in an area light increases.

## Area lights

In this mode materials can be tagged as light source emitters such as fluorescent tubes, windows and skylights.

1. Load the RenderWare Studio project GP1\_light.rwstudio that was created in the [Light mapping user interface tutorial](#) (p.133).
2. To tag a material as a light source emitter, switch to material pick mode by selecting **Selection ▶ Pick Materials** from the main menu.
3. Select a material, to which plugin data has been added, to act as a light source. Right-click and select **Show Asset Stream Contents**.
4. Right-click on the material chosen in the previous step. You can then edit the material's properties in the Stream Properties window, as illustrated in the following figure:



5. Click the **Light Map Preview** button to view the effect of the lighting.

The figure above also shows an example view.

Further details on the global area light settings for a scene can be found in the [Scene Light Map window](#) (p.313) topic.

# Light Map Preview Settings window

## In this tutorial

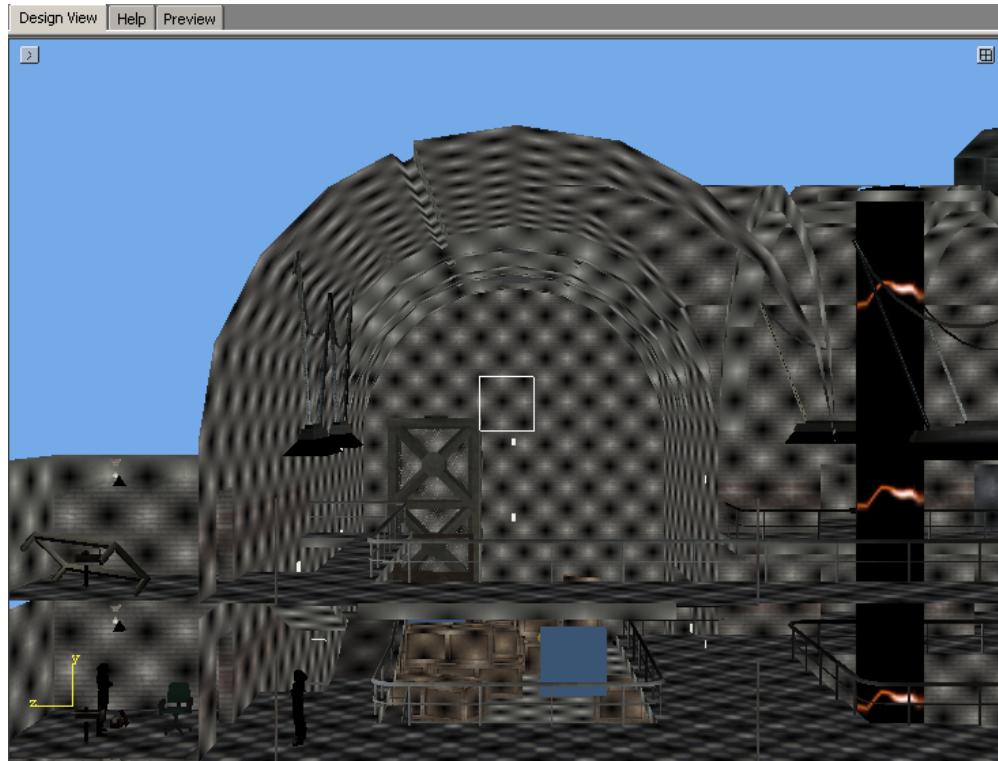
We show you how to modify the light map preview settings.

In the [Light mapping user interface tutorial](#) (p.133) we showed you how to display the **Lightmap Preview Settings** window. For detailed information about this window please see the [Light Map Preview Settings window](#) (p.294) topic.

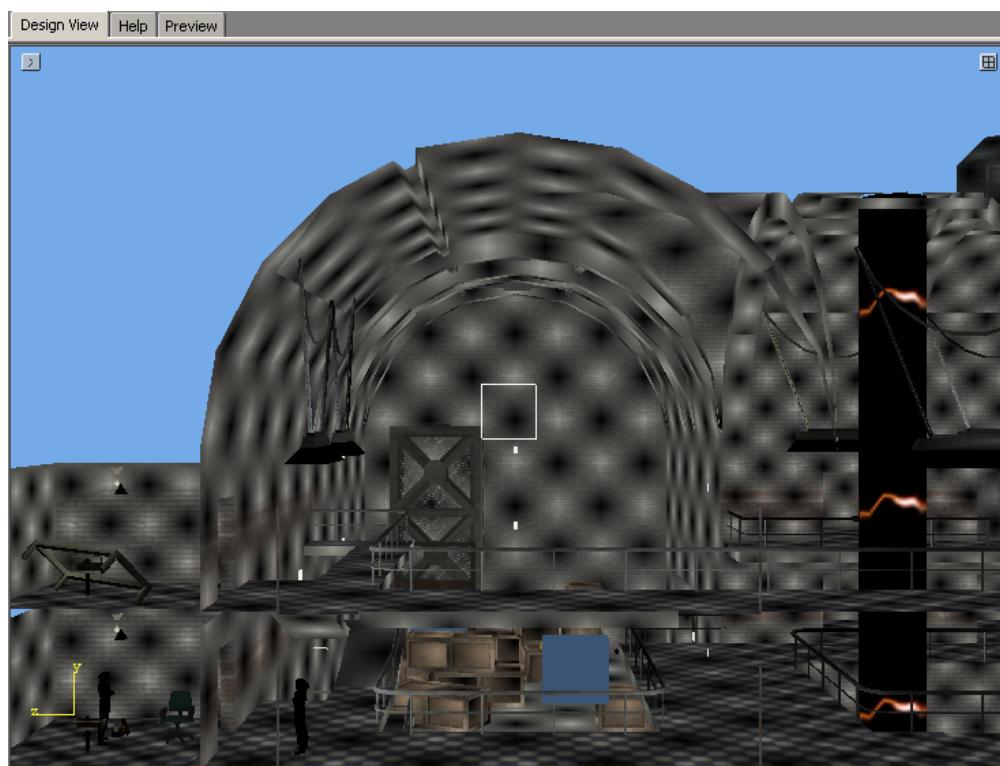
## Allocating light maps

1. Load the RenderWare Studio project GP1\_light.rwstudio that was created in the [Light mapping user interface tutorial](#) (p.133).
2. In the **Lightmap Preview Settings** window select the option **Lightmap allocation only**. Checking this option will cause light maps to be allocated, but the lighting effect of the current settings will not be shown in the **Design View** window. Ensure that the light map size is 256x256 and the density is 1.0.
3. Click **OK**.

4. Now click the **Light Map Preview** button . The following screenshot provides an example of what will appear in the **Design View** window:



5. Now display the **Lightmap Preview Settings** window again and select a lightmap texture size of 128x128 and a Density of 1.0.
6. Click  to view the results. The following screenshot shows an example of what will be seen:



**Note:** The settings in the **Lightmap Preview Settings** window only affect what is seen when is clicked. If the **Generate Light Maps** button is clicked, the scene *settings* are used to generate what is seen in the **Design View** window.

# Display options window

---

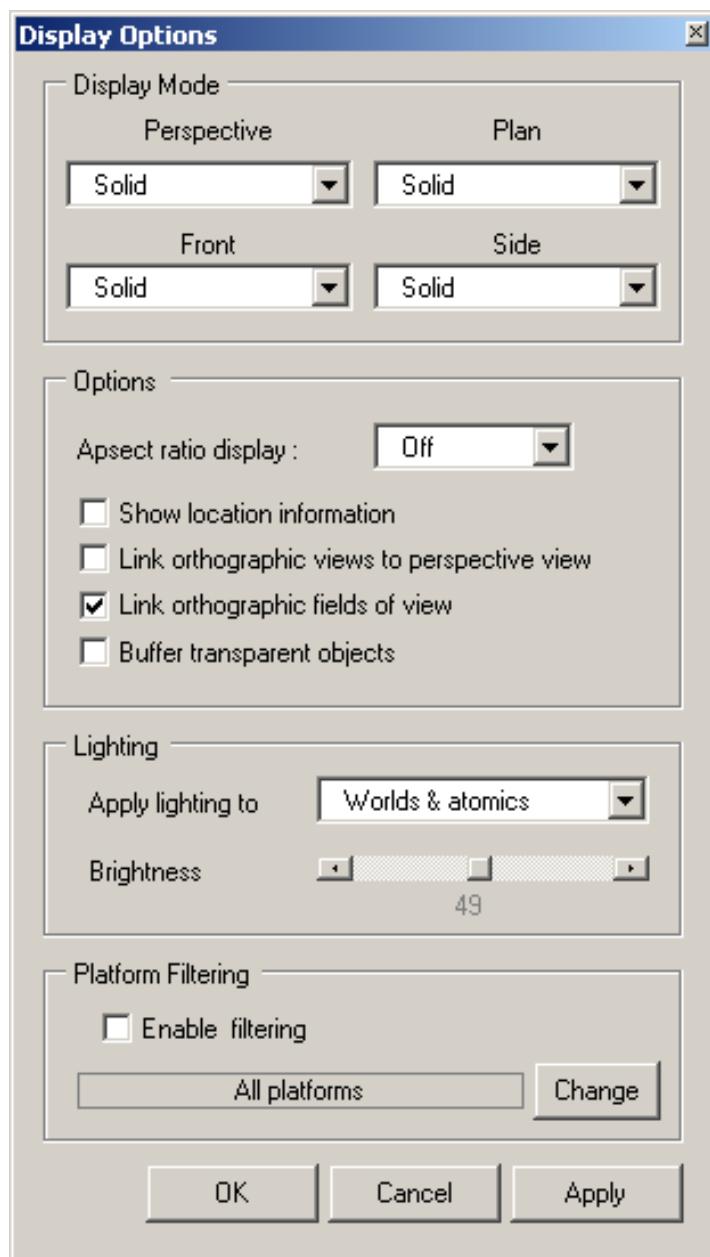
## In this tutorial

We show you how to control light levels in the **Design View** window, through use of the **Lighting** panel.

## The lighting panel

The **Lighting** panel in the **Display Options** window allows global world lighting to be configured. This lighting can override the scene lighting for the purpose of easier level viewing and object placement. Note this is a viewing aid and doesn't affect how the lighting will look in the final game. It assists you because, until lights have been placed, the level will be dark, making it hard to manipulate objects in the **Design View** window.

The **Display Options** window can be accessed in the light mapping layout by selecting **Options ▶ Display** from the main menu:



## Lighting

### Apply lighting to

This drop down list box allows the world lighting to be applied to one of:

- Worlds and atomics.
- Worlds objects only.
- Atomics only.
- No objects.

To see the lighting as it will appear in the game, select **No objects**.

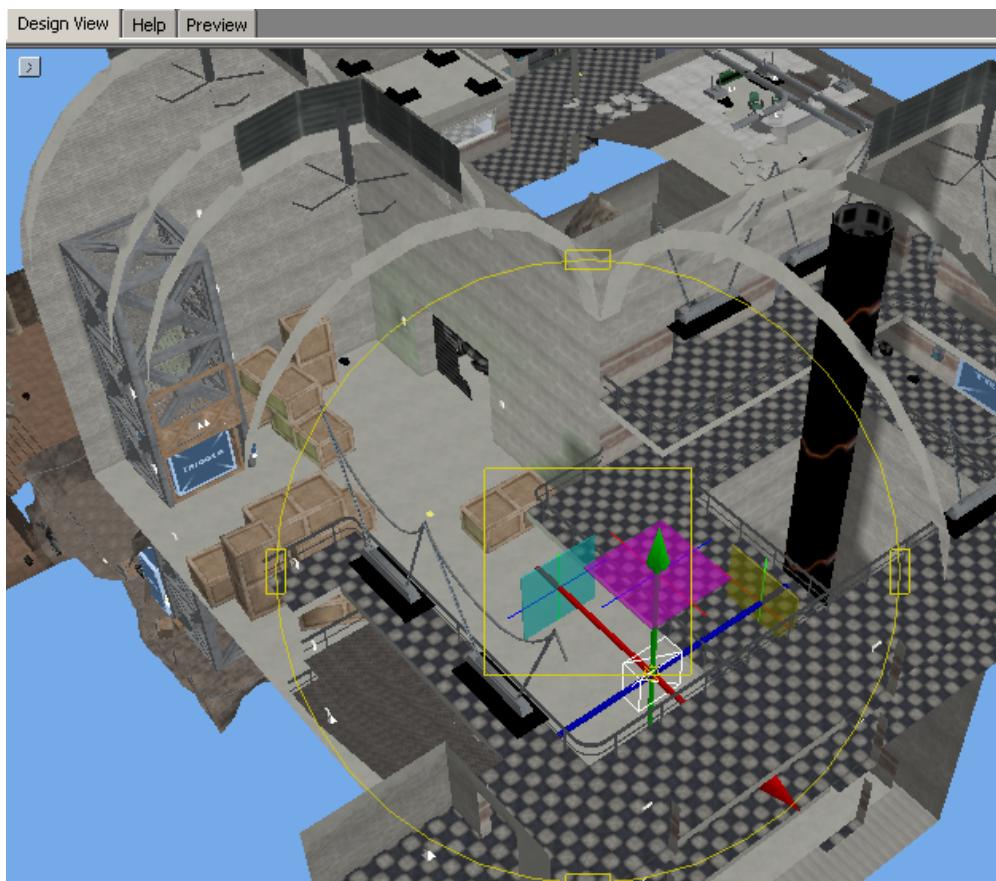
### Brightness

This slider bar allows the general level of brightness of the entire world to be easily adjusted. Reducing the brightness level to zero has the same affect as selecting 'No objects' in the **Apply lighting to** list box. In this case, the level is illuminated by the light objects you have set within the world.

## Adjusting global lighting levels within the world

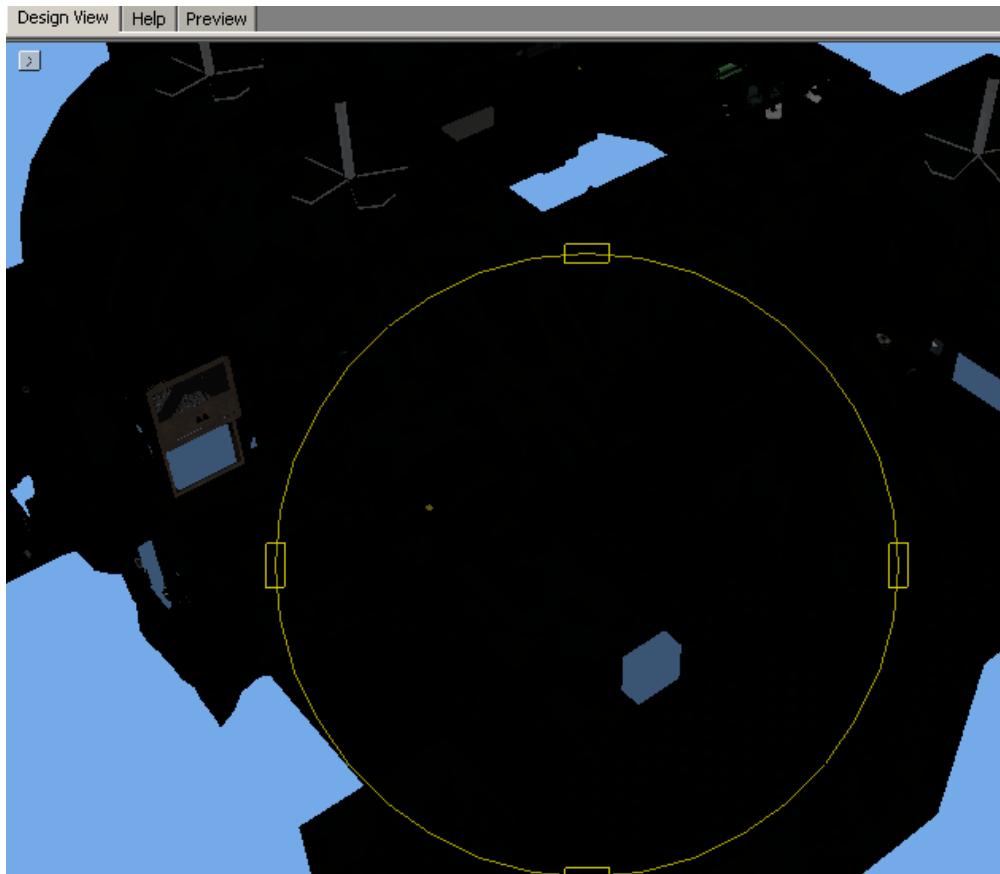
The following procedure demonstrates how to adjust global lighting levels within the game world.

1. Display the **Display Options** window and set the value of **Apply lighting to** to **Worlds and atomics**.
2. Set the **Brightness** slider bar to approximately midway. The screenshot above also illustrates this.
3. Click **OK**
4. Now click . The following screenshot illustrates a typical view:



Note the uniformity of lighting throughout the level, making it easy to see all parts of the world and ensuring easier placement of objects.

By way of contrast, here is the same view with the **Apply lighting to** option set to **None**. Notice how the darkness of the level would make it difficult for you to navigate:



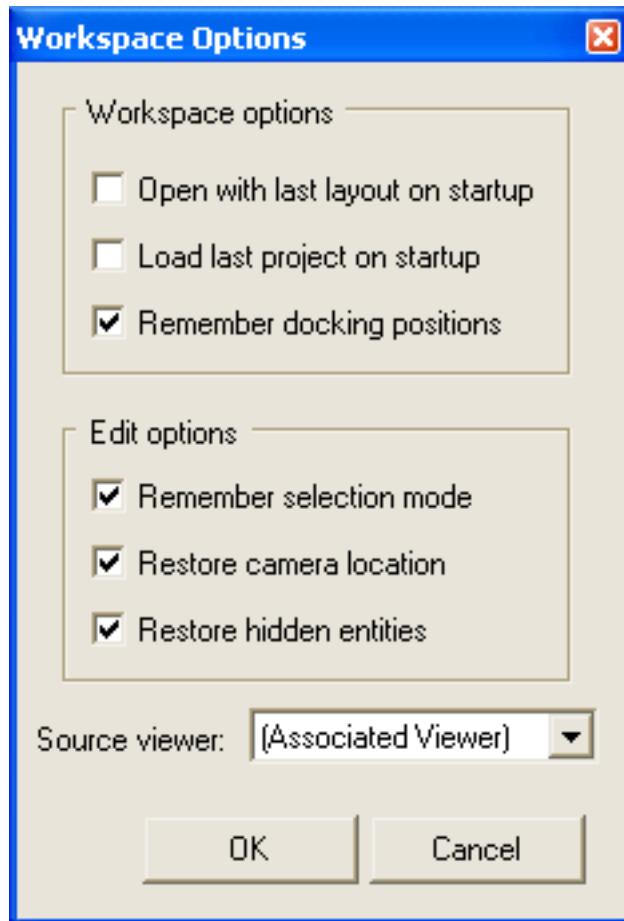
# Workspace Options

---

To set which Workspace options are stored between sessions:

- On the **Options** menu, click **Workspace...**

The Workspace Options dialog appears:



## Workspace Options

Options related to the Workspace layout:

### **Open with last layout on startup**

When RenderWare Studio is restarted, the previous layout used will be restored.

### **Load last project on startup**

When RenderWare Studio is restarted, the previous project used in the Workspace will be reloaded automatically.

### **Remember docking positions**

When RenderWare Studio is restarted, the position of all of the windows will be restored.

## Edit Options

Options related to editing a game:

### **Remember selection settings**

When RenderWare Studio is restarted, the last used pick/flight mode will be restored.

### **Restore camera location**

When RenderWare Studio is restarted, the last level will be made active and the last viewpoint in the Design View window will be restored.

### **Restore hidden entities**

When a project is reloaded, any entities that were marked as hidden during the previous load would be hidden.

### **Source viewer**

Set which viewer/editor is used to edit any source code. This relates to the context menu option, View Source, on a behavior.

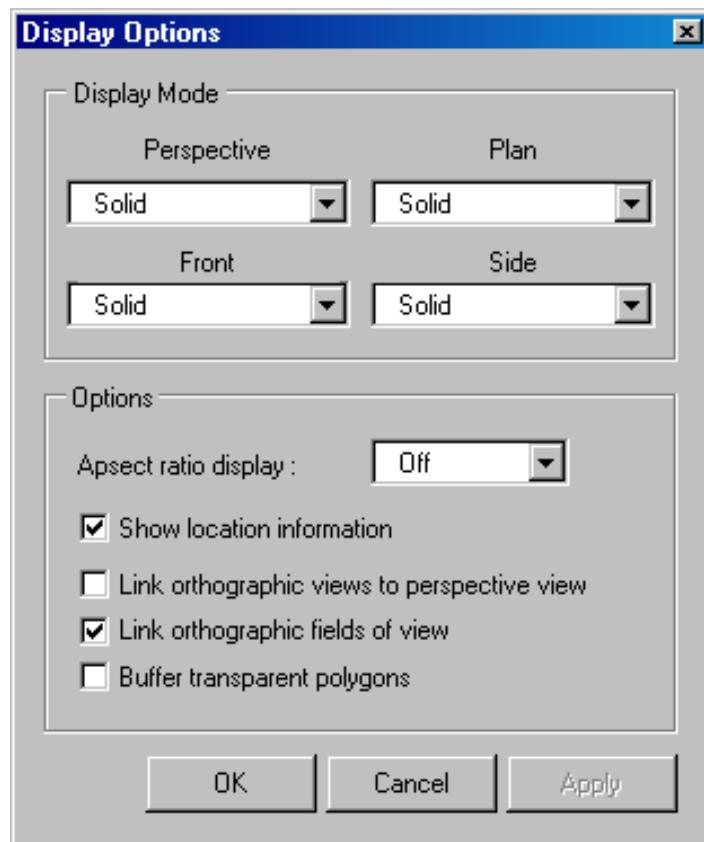
# Display options

---

To set the display properties of the Design View window (such as solid or wireframe rendering):

- On the **Options** menu, click **Display...**

The Display Options dialog appears:



## Display Mode

Sets the rendering display mode in each of the views using the following options from the drop-down lists:

### **Solid**

Draws the scene by filling in the triangles so that it appears solid.

### **Wireframe**

Draws the scene with only the triangle edges drawn as lines so that it appears as a wireframe.

### **Textured wireframe**

Draws the scene as a wireframe using the texture from the polygon's associated material.

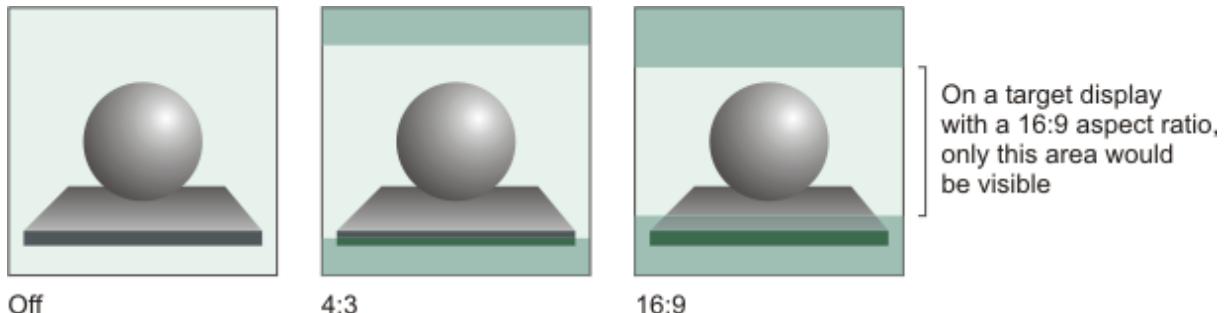
### **Untextured**

Draws the scene as a solid using the base color from the polygon's associated material.

## Options

### Aspect Ratio Display

Overlays a semi-transparent letterbox on the perspective view, indicating the area that would be visible on a display with that aspect ratio:



### Show location information

Displays the current camera location in the perspective view. If you click **Link orthographic views to perspective**, then RenderWare Studio displays the camera location in all views.

### Link orthographic views to perspective

Synchronizes the orthographic views with the perspective view. As you move around the perspective view, the orthographic views automatically pan to show the same area of the game.

### Link orthographic fields of view

Synchronizes the scale of the three orthographic views. As you zoom in or out of one orthographic view, the scale of the others automatically adjust to match.

### Buffer transparent polygons

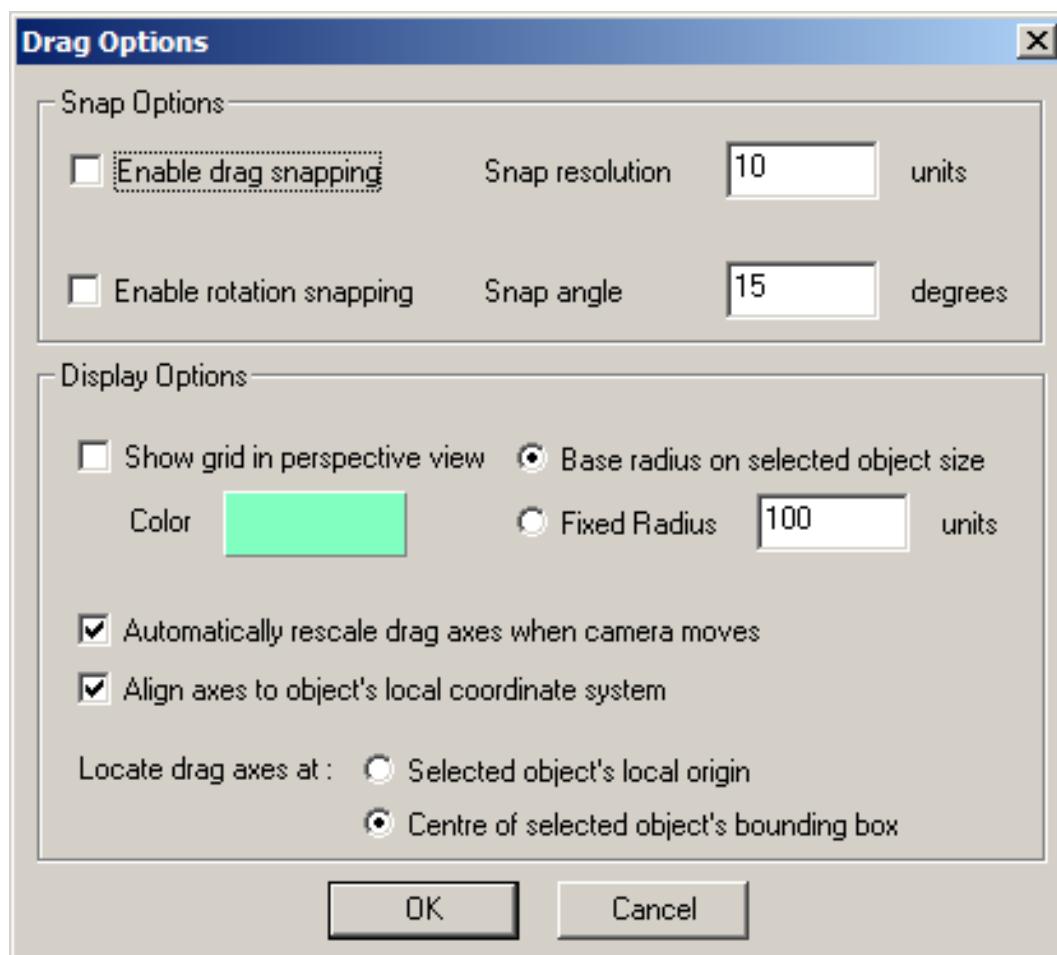
For polygons to have a transparent effect, RenderWare Studio blends them with what has already been drawn on the screen. When you click this option, RenderWare Studio draws the transparent polygons after the other objects in the scene have been drawn. Using this option can have a performance impact on your work.

# Snapping to regular intervals when moving or rotating an entity

To snap to regular intervals when moving, rotating or scaling (p.123) an entity:

- On the **Options** menu, click **Drag...**

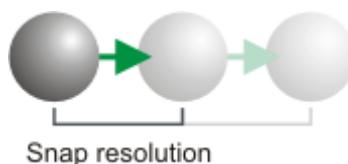
The Drag Options dialog appears:



## Snap options

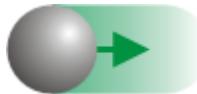
### Enable drag snapping

When you move an entity, it steps between positions whose intervals are defined by **Snap resolution**.



**Note:** These intervals are relative to the current position of the entity, not to the world coordinates.

If this check box is deselected, then you can move entities freely to any position.

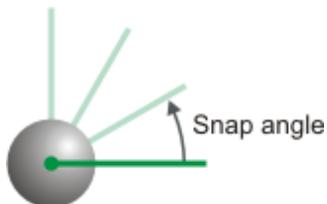


### **Snap resolution**

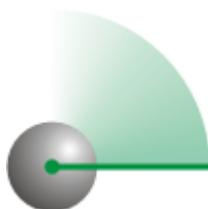
Sets the distance between each snap interval.

### **Enable rotation snapping**

When you rotate an entity, it steps between angles whose intervals are defined by **Snap angle**.



If this check box is deselected, then you can rotate entities freely to any angle.



### **Snap angle**

Sets the interval for rotation snapping, in degrees.

For example, if you enable rotation snapping and set the snap angle to 30, then entities snap to angles of 30°, 60°, 90°, 120° etc.

## **Display options**

### **Show grid in perspective view**

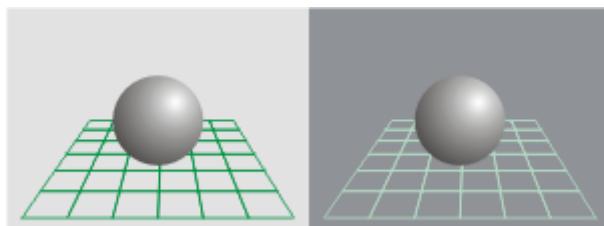
Show a grid around the selected entity. The grid fades as it recedes from the entity, depending on the radius you set.

### **Color**

Sets the grid color. To set the color, click the block of color. This displays the **Color** dialog.

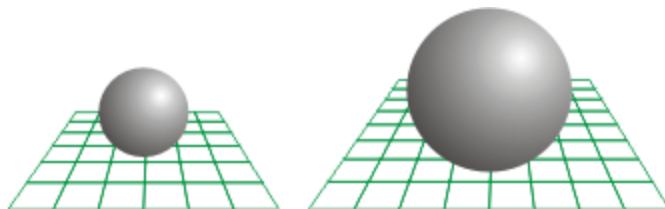


In a game, you sometimes have a very dark or a very light scene. To see the grid clearly, choose a contrasting light or dark color.



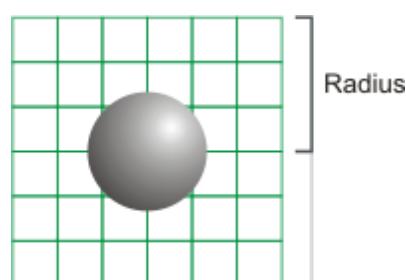
#### Base radius on selected object size

Sets the size of the grid relative to the size of the selected entity.



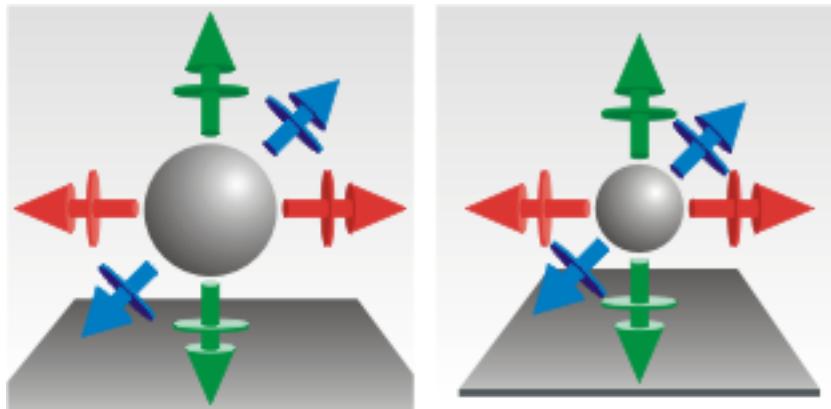
#### Fixed radius

Sets the grid to a fixed size.



**Automatically rescale drag axes when camera moves**

Enlarges the drag axes as you zoom out, and reduces them as you zoom in.



This keeps the drag axes at a usable size, regardless of your distance from the selected entity.

If you select this check box, then you can resize the drag axes by pressing + or - (on the main keyboard, not the numeric keypad).

**Align axes to object's local coordinate system**

Keeps the drag axes aligned to the world grid as you rotate an entity.

If you leave this check box deselected, then the drag axes turn as you rotate an entity.

**Locate drag axes**

Allows you to locate the drag axes at the selected object's local origin, or the center of the selected object's bounding box.

# Organizing your project into folders

---

You can use [folders](#) (p.73) to organize your game database in the same way you use folders in Windows Explorer to organize files on your PC.

## Folders in the Game Explorer window

The Game Explorer window allows you to:

- Create and nest as many folders as you like, letting you organize your game in whatever hierarchy seems logical for its content.
- Copy, move, and create references (shortcuts) using the same mouse operations as in Windows Explorer:
  - Drag to move selected objects between folders.
  - Drag while pressing **Ctrl** to create copies of selected objects.
  - Drag while pressing **Alt** to create [references](#) (p.174) to selected objects.

For example, you could have the child folders of the game corresponding to the levels in a typical video console game. Several of the [examples](#) (p.514) reflect this organization (in the example topics, the terms “folder” and “level” are interchangeable).

You might equally want to organize your folders to correspond to the different parts of a race-track in a driving game. For example, you might have one folder for the crowd and its corresponding entities, another folder for each of the cars involved, and another for the different areas of the track.

To create a folder:

1. In the [Game Explorer](#) (p.288) window, right-click the game icon or an existing folder, and then click **New ▶ Folder as child**.  
A new folder appears with the default name “Folder” followed by a number.
2. Type a name for the folder, and then press **Enter**.

To delete a folder, right-click it, and then click **Delete**.

To rename a folder, right-click it, and then click **Rename**.

**Tip:** To make a folder of references to entities containing a certain asset or behavior, for example to select all entities in a project containing the door asset:

- Create a new folder in the Game Explorer window and call it *Doors*.
- Right click on the door asset file in the Assets window, and then select **Search ▶ Entities Using Asset(s)** from the context menu.
- Go to the [Search Results tab](#) (p.315), and then highlight all the entities that have been found.
- With the **Alt** key held down, drag the selected items to the *Doors* folder.
- Now that you have created this set of references to the entities in one place you can work on them as a group. For example, to make them all impossible to select, right-click the folder icon, and then select **Freeze** from the context menu.

**Tip:** To optimize loading time for a game, it is better to design your [database](#) (p.76) to use a deep folder structure that uses nested folders that contain fewer entities, than to have fewer folders containing large numbers of objects.

## Active and working folders

At any time, a game has one active folder, and one working folder:

### Active folder

The folder whose contents are displayed in the Design View window, and sent to the target console in the game data stream. The active folder must be a child of the game (it cannot be a nested folder). To set the active folder, right-click a folder in the Game Explorer window, and then click **Active Folder**. The name of the active folder is highlighted in **bold**.

### Working folder

If you create a new entity by dragging an asset or template into the Design View window, then the new entity is created in this folder. The working folder is either the active folder (the default) or a descendant of the active folder. To set the working folder, right-click a folder in the Game Explorer window, and then click **Working Folder** (if you do this on the current working folder, then this resets the working folder to the active folder). The name of the working folder is highlighted in **bold**.

## Folders in other windows

You can also create folders in other windows; for example, in the Templates window, you can organize templates into template folders.

Only “normal” folders  (displayed in the Game Explorer window) and asset folders  (displayed in the Game Explorer and Assets windows) are significant to the structure of the game database. Other folders exist only to organize the display of objects in their associated Workspace windows.

# Working with the global folder

---

The global folder is a special type of folder containing objects that are persistent throughout a game (not just one level of a game). A game has one, and only one, global folder. The contents of the global folder are sent to the target console before any other folders. A global folder can contain the same types of object as a normal folder (assets, entities, other folders etc.), but the global folder cannot be nested inside another folder.

The global folder has the same functionality as any other folder in the game database. The only difference to a normal folder is that it has some extra data identifying it as the global folder.

Any assets or entities positioned in the global folder, for example a high score table for keeping a permanent record of the player's scores regardless of the level being played, would remain in memory even when you move to a different level later in the game.

Other important features to note about the global folder are:

- Entities in the global folder are not destroyed when the current game level changes. They are only destroyed when the entire game is destroyed.
- When a new game is created, an empty global folder will be created automatically.
- The global folder is sent once, before any other folders when connecting to a target. It is not re-sent when the active folder is changed.

**Tip:** For a demonstration of a global folder, open the supplied [multiple levels](#) (p.538) example.

# Changing levels in a stand-alone game

---

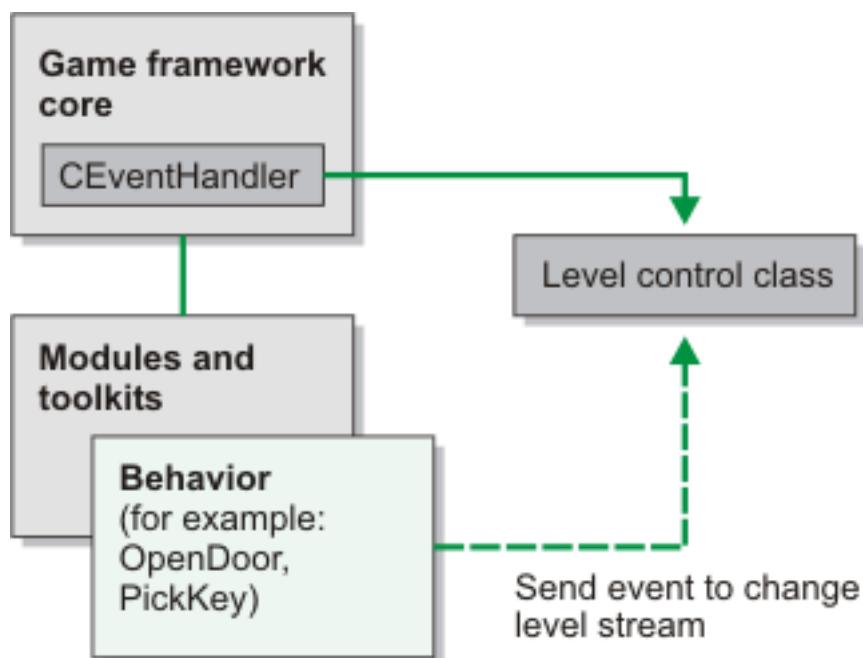
Currently, only the design build configurations of the Game Framework allow you to easily change between levels in your game, by right-clicking the folder in the Game Explorer window of the Workspace, and then clicking **Active Folder**.

In future releases of RenderWare Studio, we plan to include support for changing levels in non-design (stand-alone) build configurations.

For now, however, you need to develop your own code for changing between levels in a stand-alone game.

A good approach is to develop a class that manages the overall control of the levels. This level control class is essentially a state machine that controls which game level folder is loaded, based on certain rules (for example “has player got blue key to exit level?”).

You could make this class inherit from `CEventHandler` and use events to trigger level changes, or you could use direct function calls to trigger the levels:



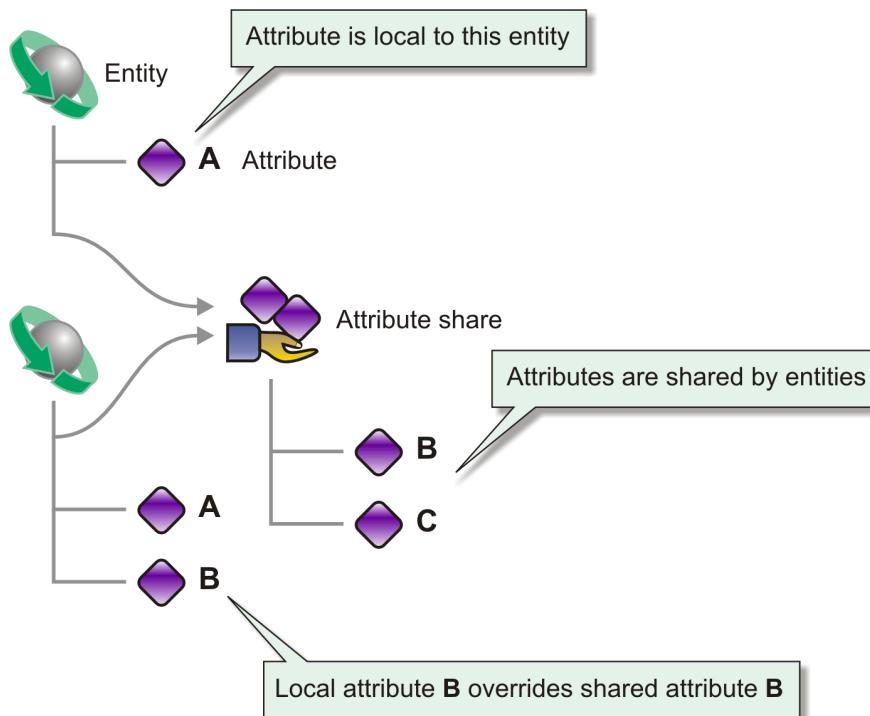
# Sharing attributes between entities

---

By default, attributes are “local”: they apply only to a particular entity, and are stored with that entity. However, you can also share attributes between entities, by creating *attribute shares*.

An attribute share is a set of attributes that can be referred to by many entities. Changing an attribute in an attribute share affects all of the entities that refer to the attribute share.

Entities can contain a mix of local attributes and attribute shares.



**Local attributes override shared attributes.** If an entity has a local attribute, and refers to an attribute share that also contains this attribute, then the Attributes window always edits the local attribute. Similarly, the default rules of the Game Production Manager sends the local attribute to the target console.

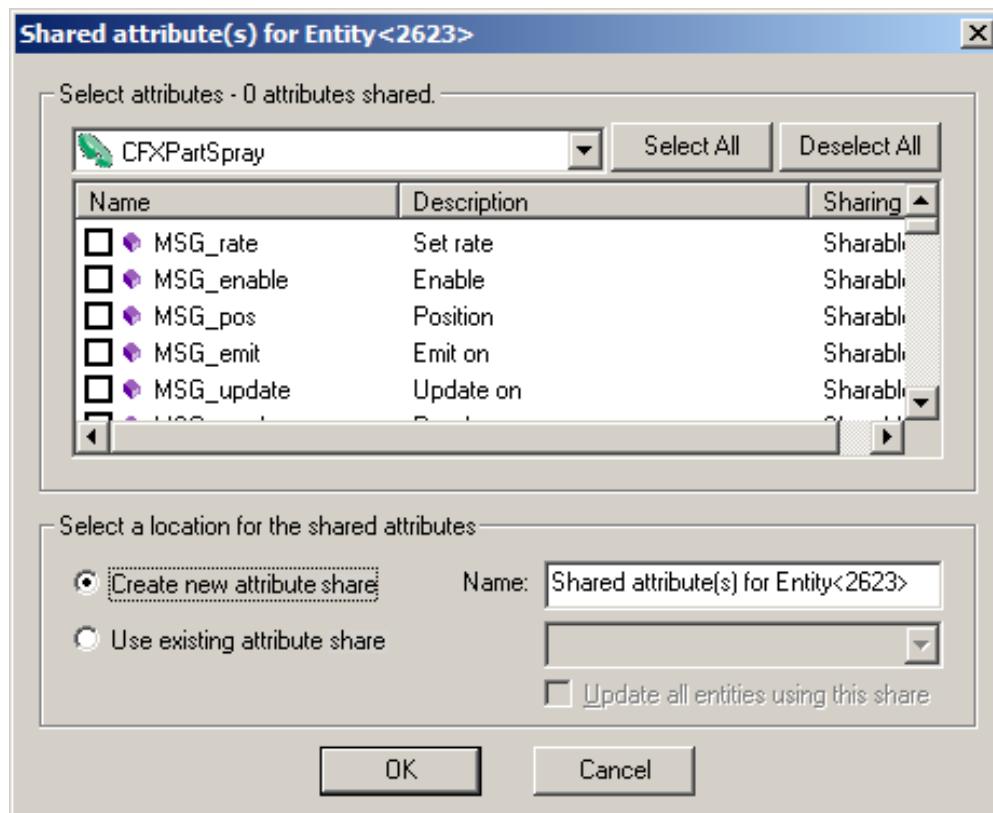
If you want an attribute always to be local, and never shared, then you can declare the attribute as private.

Sharing attributes provides an elegant and natural solution to many game design problems. It allows you to change universal parameters in a single operation. It eliminates duplication and unwanted differences, and reduces the possibility of error. Moreover, the scope of sharing can be at different levels in the same game: for example, two entities in the same folder of the game database hierarchy can share attributes as easily as two entities in different folders.

## Creating an attribute share

1. In the Game Explorer window, right-click an entity containing attributes that you want to share with other entities.
2. Click **Share Attributes**.

The Shared Attributes dialog for the entity appears.



This dialog lists the attributes of this entity that can be shared.

3. In the top drop-down list, select a behavior containing attributes you want to share.
4. Select the attributes. Either:
  - Click the check boxes next to each attribute name  
or
  - Click **Select All**
5. Click **Create new attribute share**.
6. In the **Name** box, type the name of the new attribute share (or leave the default name as-is).
7. Click **OK**.

A new attribute share appears under the entity, replacing the local attributes that you have now shared.

The new attribute share also appears in the [Attribute Shares](#) (p.269) window.

## Adding attributes to an existing attribute share

You can add attributes to an existing attribute share from any entity:

1. In the Game Explorer window, right-click an entity containing attributes that you want to share with other entities.
2. Click **Share Attributes**.
3. Select the attributes to share.
4. Click **Use existing attribute share**, and select the attribute share from the drop-down list.
5. Other entities that use this attribute share might have local attributes for the attributes you have just shared. These local attributes will override the shared attributes. To remove these local attributes, forcing the entities to use the attributes you have just shared, select the **Update all entities using this share** check box. If you leave this check box unselected, then those entities will keep their local attributes.
6. Click **OK**.

## Using shared attributes from an attribute share

If you want an entity to use shared attributes instead of local attributes:

1. In the Game Explorer window, right-click the entity.
2. Click **Use Attributes**.
3. Select the attribute share.
4. Select the shared attributes you want to use.
5. Click **OK**.

If the entity had local attributes for the shared attributes you have just selected, then the Workspace removes those local attributes from the entity.

For any attributes in the attribute share that you did not select:

- If the entity already had a local attribute, then the entity keeps the local attribute.
- Otherwise, the Workspace creates a local attribute with the default value.

If an attribute share contains attributes that do not apply to an entity, then the entity ignores those attributes.

## Overriding a shared attribute with a new local attribute

If an entity is using an attribute share, but you want to override one of its shared attributes with a local attribute:

1. In the Game Explorer window, expand the view of the entity to show the attributes in the attribute share.
2. Copy the attribute from the attribute share to the entity: while pressing **Ctrl**, drag the attribute from the attribute share to the entity.
3. The local attribute appears under the entity. You can now change the attribute, and it will affect this entity only.

## Deleting an attribute from an attribute share

**Note:** This affects all entities that use this attribute share.

To delete an attribute from an attribute share:

1. In the Game Explorer window, expand any occurrence of the attribute share (under any of the entities that use the attribute share).
2. Right-click the attribute, and then click **Delete**.

Under each of the entities that use this attribute share, the previously shared attribute appears as a local attribute.

## Deleting an attribute share

**Note:** This affects all entities that use this attribute share.

To delete an attribute share:

- In the Attribute Shares window, right-click the attribute share and then click **Delete**.

Under each of the entities that used this attribute share, the previously shared attributes appear as local attributes.

## Using attribute shares with templates

Attribute shares act as an extension to [templates](#) (p.127). When you create an instance of an entity using templates, you preserve the entity's assets and attributes so that, each time you drag the template into the Workspace, RenderWare Studio creates a new instance of the entity with its own copy of the attributes of the template. If the template refers to attribute shares, then, when you drag the template into the scene, the new entity shares the attributes of the template.

For example, suppose you had a game with several folders, each containing 50 doors—25 metal and 25 wooden—and you wanted the user to destroy the wooden doors with an axe at a different rate than destroying the metal doors with dynamite. Rather than creating each door entity separately, and then have to go through every folder, selecting every wooden door and changing the “number of axe hits required” attribute, you could create a “wooden door” template that shared this attribute, and create the wooden doors from this template. Changing the attribute for one of the doors changes the attribute for all of the doors based on the template (because they share the attribute).

# Sharing entities and folders between other folders

---

You can share entities and folders by creating *references* to them in other folders:

1. In the Game Explorer window, select one or more entities or folders that you want to share.
2. Press and hold down **Alt**, and then drag the selected entities or folders to another folder.

If an entity or folder has multiple references, then the Game Explorer window displays a small **x** next to each reference.

References are not copies; each reference points to the same entity or folder object in the game database. Changes to one reference are reflected in the others. For example:

- If you create multiple references to a folder, and then you add an entity to one of the references, the entity appears under all of the references.
- If you create multiple references to an entity, and then you change the 3D position of the entity in the scene, then all the references to the entity move.

For a demonstration of entities and folders with multiple references, open the supplied “Multiple Levels” example project.

**Tip:** To create entities that are similar, but not identical, use a [template](#) (p.127). Or simply copy an existing entity (by dragging with **Ctrl** instead of **Alt**).

# Importing from another project

---

You can import game level folders and templates from another project into your current project. You might want to do this if you build game level folders of a game in different projects, and need to merge them together or to use game level folders from one project in another.

**Note:** You could equally import a folder which does not correspond directly to a level if your game database is [organized](#) (p.166) in a different way.

When you import a game level folder, the Workspace copies the game level folder and all of its children (entities, assets and templates) into your current project.

**Note:** If you import a game level folder whose item IDs match items in the current project, then the Workspace assumes that they are the same items, and does not import them. If these are not the same items, and you do want to import them, then rename the item IDs before you import them.

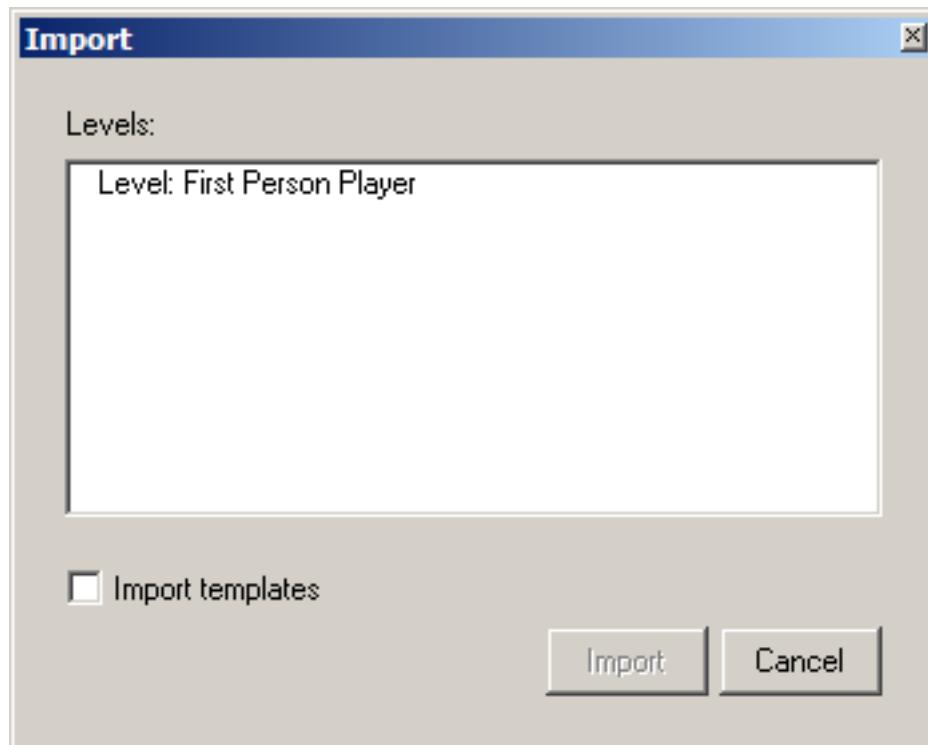
To import one or more game level folders, and optionally import all templates, from another project:

1. On the **File** menu, click **Import**.

The Open RenderWare Studio Project File dialog appears.

2. Browse to the .rwstudio project file from which you want to import items.

The Import dialog appears.



3. Select the game level folders that you want to import.
4. If you want to import the templates from the project, then select the **Import templates** check box.

**Tip:** If you want to import the templates from another project without importing any game level folders, then select the **Import templates** check box, but do not select any game level folders.

Importing game level folders or templates creates game database items in your current project. It does not copy primary asset files to your current project, or even verify that the primary asset files required by imported assets exist in the resource root path of your current project. To verify that the required primary asset files exist in the appropriate locations, use the [Database Validation](#) (p.193) tool.

# Deleting objects versus removing references

---

“Delete” and “remove” are different actions that you can perform on the game database:

## Delete

Erases an object from the game database, and all references to the object.

**Deleting an object in Workspace does not delete its corresponding XML file.** When you save a project, rather than deleting the XML file for a deleted object, Workspace replaces the file contents with a `<Deleted/>` tag. This allows you to use RenderWare Studio with version control systems that do not delete client files (such as NXN alienbrain and Microsoft SourceSafe).

To delete these files, use the PurgeDeleted tool, in  
Studio\Programs\Tools\PurgeDeleted.

## Remove

Erases a reference to an object in the game database (not the object itself). Any other references to the object are not affected.

1. Select the object or reference.
2. Right-click the object or reference, and then click **Delete or Remove**.

If you attempt to delete an object that has multiple references, then a confirmation message appears.

**Note:** An object can still exist in the game database, even if no other objects refer to it.

For example, if you remove a reference to an asset from an entity, then that asset would still exist in the Assets window, and any references to the asset in other folders or windows would also still exist. However, if you delete the asset (from the Assets window), then the asset, and any references to that asset, are deleted from the game database.

# Adding sound to your game

---

## 3D sound

To use 3D audio in Renderware Studio drag an object into the scene and place a AudioGlobalMixer behavior onto it. Then right-click on this in the world lister and select **New ▶ Asset**. Set the asset type to be rwaID\_WAVEDICT, and add the file path to a RenderWare Stream file (.rws). You can now attach an AudioSound3D behavior to an object and tweak its parameters. If you want to add 3DSound functionality to your own behaviors you can either derive from AudioSound3DInterface, or you can contain the Audio toolkit class in your code.

## Streamed audio

To play streamed audio, specify the .vag file in the AudioStreamInterface.cpp file. After the stream loads, you can use the behavior to start and stop the audio.

## Virtual voices

RenderWare Studio supports Virtual Voices. This allows you to have hundreds of voices in a scene, but only map to a few hardware voices. Audio streams cannot use virtual voices, so stream voices are mapped to the last 16 voices in the real voice list.

## Adding screens to the game with Maestro

---

A Maestro animation is a 2D flash 3 animation which is converted to a .anm file using the 2dconvert utility.

All the data required is contained in these .anm files except for the font data (if the Maestro animation uses a font). This font data needs to be exported to a .fnt file to be added as a separate resource in Renderware Studio. Each font file must be saved for the specified platform. If the Maestro animation (.anm) file requires a font then add a font asset (.fnt) to the entity containing the CMaestro class above the Maestro resource so that the font resource gets loaded before the Maestro resource.

**Tip:** You can also convert an animation using the texture converter tool supplied with Maestro.

**Note:** You can only have one font per Maestro object.

# Sequencing entity actions

---

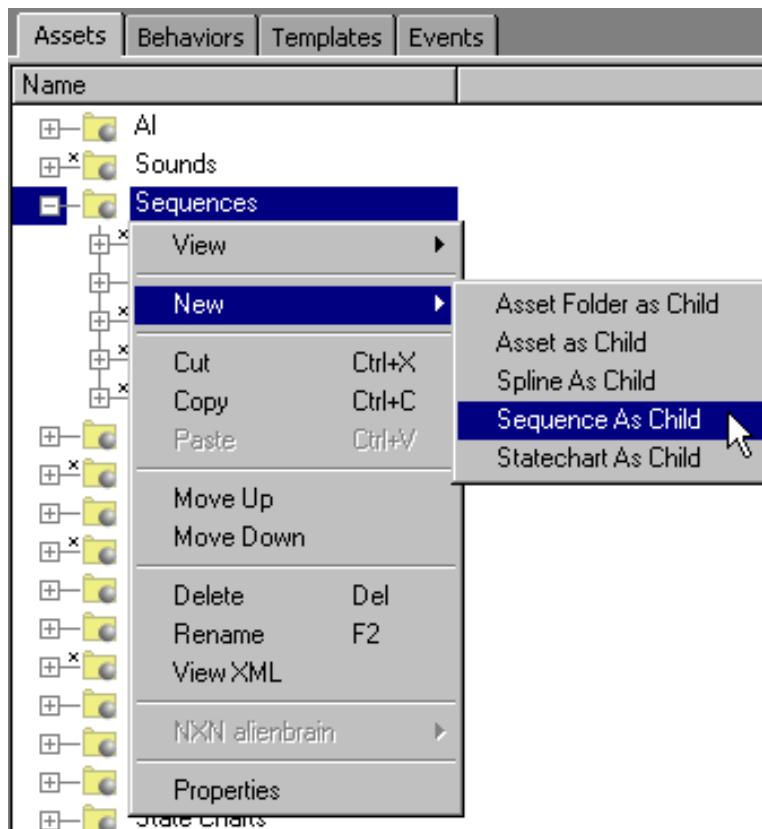
A sequence is a predefined set of actions that always take place in the same order. In a sequence, the attributes of existing entities may be modified together over time, and new entities may be created and controlled. Sequences can be put to uses such as:

- Ending a level, or joining two levels together, with a piece of animation (a “cut scene”) that features the entities in your game.
- Making a game character perform some repeated actions while waiting for the player to provide input.
- Synchronizing the changing attributes of a set of entities over a short period, as in the lift example from the Genre Pack 1 tutorial.

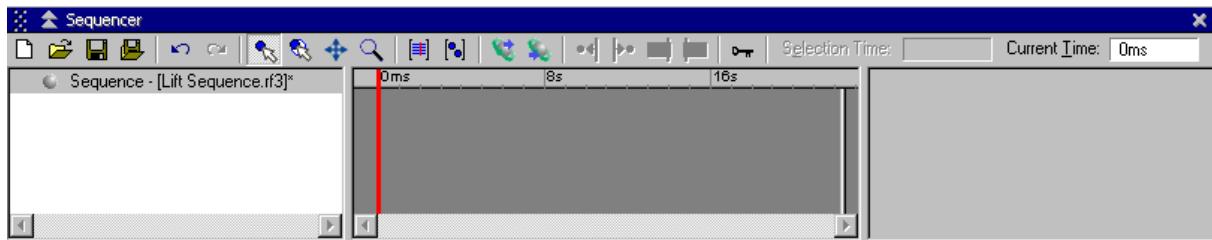
## Creating a sequence asset

In Workspace, you usually create sequences in the Assets window, and edit them in the [Sequencer window](#) (p.317).

1. To create a sequence, right-click in the Assets window of an open project and select **New ▶ Sequence** (or **New ▶ Sequence as Child**). Your new asset will appear in the window.

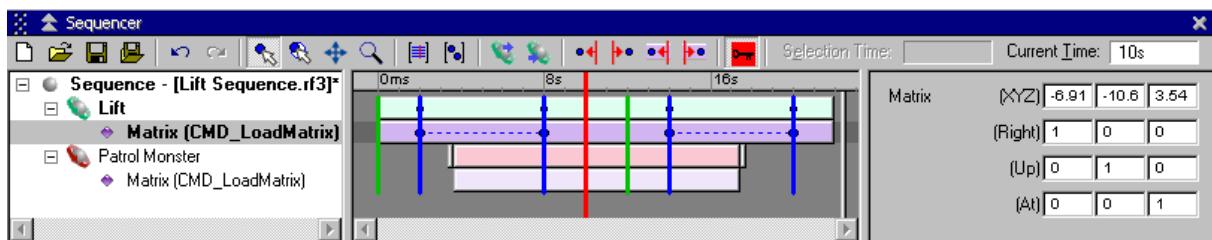


2. Right-click the asset and choose **Edit**. The Sequencer window will come to the foreground, displaying the contents of your (currently empty) sequence.



**Note:** If the Sequencer window is not in the current Workspace layout, it will be opened as a floating window.

3. Drag the entities that you want to appear in the sequence from Game Explorer or the Templates window to the left-hand pane of the Sequencer window.
- For each entity, a dialog will appear asking you to specify which attributes you want to be able to modify during the sequence. You can edit your selections later through the Sequencer window's context menu.
4. Use the features of the [Sequencer window](#) (p.317) to add keyframes, interpolations, and events to the timelines of the attributes you've chosen.

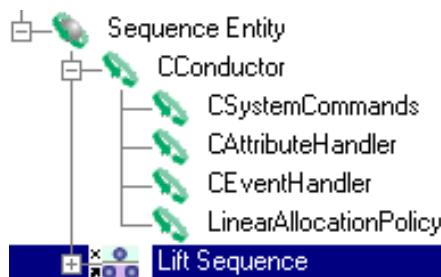


5. Preview your sequence in Design View by right-clicking and enabling **Updates** for every attribute that has a visual representation, and then scrubbing the playhead back and forth along the timeline.

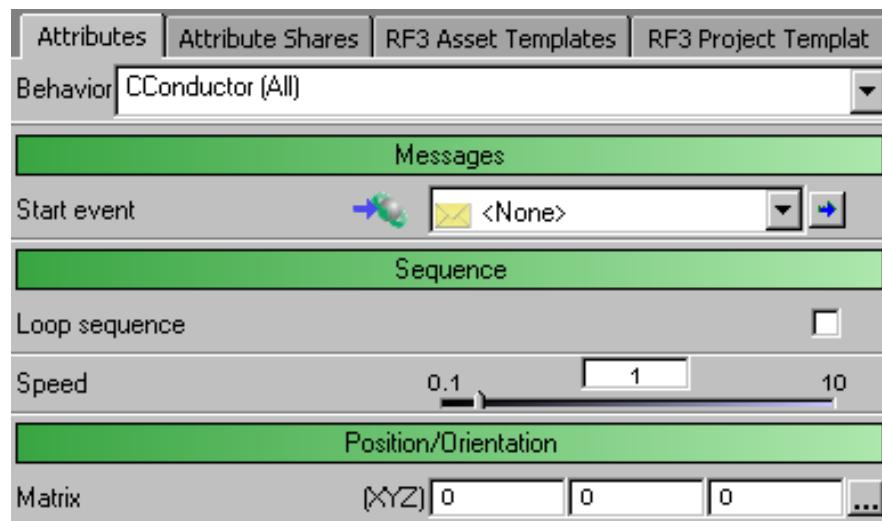
## Using a sequence asset

As soon as you've created a sequence asset, and at any time afterwards, you can incorporate it into your game by attaching a `CConductor` behavior to it.

1. Drag the `CConductor` behavior ([SeqCtrl ▶ CConductor](#)) from the Behaviors window to Game Explorer.
2. Drag your sequence asset to the new entity in the Game Explorer window that was created in the previous step.



3. In the Attributes window for the new sequence entity, specify a **Start event** for your sequence.



Now you can arrange for another entity in your game (a trigger volume, for example) to send this event. When the sequence entity receives it, the sequence will begin to run.

## Saving target data to files

---

You can save data from a target console out to permanent files on your hard disk via a network connection to the Workspace. As an example, you might want to do this to save the lighting data in a game level out to files to be used later as light maps in another level during another session of RenderWare Studio. An ActiveX Control in the Workspace *listens* for the data coming from the console.

RenderWare Studio saves this data as temporary files and displays them in the [Target Files](#) (p.329) window. These temporary files are available only during the current session. Before ending the session, you select which files to write permanently to the hard disk for later use.

### Example of saving lighting data as light map files for later use

1. Open the file:  
`C:\data\Examples\Example Light Mapping.rwstudio`
2. In the Targets window, right-click a local host target (for example, **Local PC - DirectX** or **Local PC - OpenGL**. See note below).

#### DirectX versus OpenGL display drivers

Unless you have an old display adapter, then it is likely that your computer supports both DirectX and OpenGL. However, while the OpenGL display driver is typically supplied and installed with the display adapter, you might have to separately [download](#) ([www.microsoft.com/windows/directx/](http://www.microsoft.com/windows/directx/)) and install the DirectX display driver.

3. Click Launch to start the console emulator application.
4. Click Connect to start sending game data to the target.
5. In the Game Explorer window, click to open the Generating Light Maps level.
6. Click Light Mapper.

The attributes for behavior ATBLightMapper display in the Attributes window.

7. After using the attributes to trigger events, click on the Save World event trigger to save the world model.

RenderWare Studio saves the data as temporary files and displays them in the [Target Files](#) (p.329) window. The temporary files are only available in the current session.

8. In the Target Files window, right-click on a filename and click Save As to a save permanent copy of the file. Or click Delete to delete the temporary file.

# Developing for multiple platforms

---

If you are developing a game for multiple platforms, then, typically, there are some differences between the games on each platform.

For each item in the [game database](#) (p.73) (under the game  item), you can specify the platforms to which the item applies.

The game data stream includes only those items that apply to the target platform. By default, items apply to all platforms.

To specify the applicable platforms for an item:

- In the Game Explorer window, [select](#) (p.288) one or more items.
- Right-click one of the selected items.
- Select or deselect the check box for each platform.

The changes you make apply to all of the selected items.

# Creating a custom target platform

---

The `RWStudio.settings` file in the RenderWare Studio Programs folder contains global settings that apply to all projects, including definitions of target platforms (GameCube, OpenGL etc.). These definitions specify:

- The port number to connect to.
- The byte ordering (big-endian or little-endian) of the target platform.
- Whether strings should be sent using the Unicode or ASCII character set.
- *Platform flags* that identify which objects in the game database apply to this target platform.

The properties for each object in the game database also specify a set of platform flags. RenderWare Studio includes an object in the game data stream only if the platform flags for the object include all of the platform flags for the target.

You can create a custom target platform by editing this settings file. For example, you might want French and English versions of your game to use different art assets. To do this, you would define new “French” and “English” platform flags, and then define new target platforms that require these flags. This is described below.

## Defining new platform flags

The `<OBJECT NAME="Platform Flags">` element in the `RWStudio.settings` file defines the platform flags that your projects can use:

```
<OBJECT NAME="Platform Flags">
  <PARAM NAME="DirectX" VALUE="" />
  <PARAM NAME="GameCube" VALUE="" />
  <PARAM NAME="OpenGL" VALUE="" />
  <PARAM NAME="PS2" VALUE="" />
  <PARAM NAME="Xbox" VALUE="" />
  <PARAM NAME="English" VALUE="" /><PARAM NAME="French" VALUE="" />
/>
</OBJECT>
```

When you define new platform flags, they become available in object property sheets the next time you start RenderWare Studio Workspace:



By default, objects have all property flags selected. When you define new property flags, then existing objects with all property flags selected also have the new flags selected. However, if an existing object has any property flags deselected, then the new flags will not be selected.

After defining new platform flags, you can define custom platforms that require these flags.

## Defining custom platforms

The `<OBJECT NAME="Platforms">` element in the `RWStudio.settings` file defines the target platforms that your projects can use.

Here is the definition of a default Playstation 2 platform:

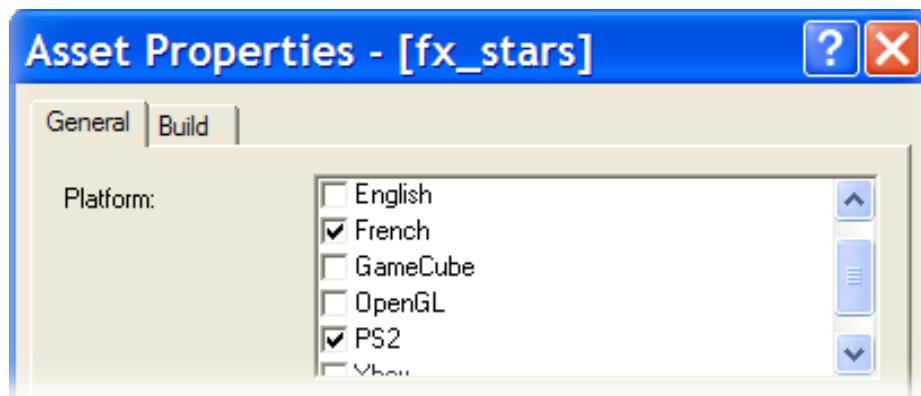
```
<OBJECT NAME="Playstation 2">
  <PARAM NAME="Port" VALUE="5609" />
  <PARAM NAME="Platform Flags" VALUE="PS2" />
  :
</OBJECT>
```

The Platform Flags parameter contains a comma-separated list of the platform flags that objects must have if they apply to this platform. Here are definitions of English and French versions of the Playstation 2 platform:

```
<OBJECT NAME="English Playstation 2">
  <PARAM NAME="Default Port" VALUE="5609" />
  <PARAM NAME="Platform Flags"
    VALUE="PS2,English" />
  :
</OBJECT>
<OBJECT NAME="French Playstation 2">
  <PARAM NAME="Default Port" VALUE="5611" />
  <PARAM NAME="Platform Flags"
    VALUE="PS2,French" />
  :
</OBJECT>
```

The Targets window displays these platforms as “English Playstation 2” and “French Playstation 2” respectively.

The following properties dialog shows an asset that applies only to French Playstation 2 targets:



# Managing team development with alienbrain

---

If you simply store your game database XML files in a shared folder, and you allow multiple users to access the files concurrently, then the users' changes will overwrite each other, and you might lose data.

To overcome this issue, and allow team development, the [NXN alienbrain](#) ([www.alienbrain.com/](http://www.alienbrain.com/)) asset management tool is supplied with RenderWare Studio.

## Using RF3 project template files

All users should refer to the same [RF3 project template](#) (p.104) file names.

## Using alienbrain drive-mapping

If you enable alienbrain drive mapping so that your files appear in (say) the z: drive of your computer (which might map to C:\abdata, for example), then you must also load your project into RenderWare Studio from the z: drive. If you load the same project from C:\abdata, alienbrain integration will fail.

## Adding custom columns to alienbrain's Manager Client window

You can also add custom columns to the NXN Database Explorer window or the alienbrain Manager Client Window based on the Name, Description and Type property tags of a [game database](#) (p.76) XML file:

1. In the Client Manager Window open the "Specify new user-defined column" dialog—See the "Customizing the List View's Title Columns" topic of the alienbrain documentation.
2. Add RWSName, RWSDescription or RWSType to the **Property Name** field.

## Merging file conflicts with alienbrain

Where you have conflicts in a file that has been checked out simultaneously by different users, alienbrain allows you to integrate the conflicting sections of the file—See the "How Do I Perform a Manual Merge?" topic of the alienbrain documentation.

The default application for merging in alienbrain is [Araxis](#) ([www.araxis.com/merge/faq.html](http://www.araxis.com/merge/faq.html)) Merge.

## Managing C++ source

To manage the Game Framework C++ source, use your organization's source control system.

# How RenderWare Studio works with alienbrain

---

If you open a RenderWare Studio project file (`.rwstudio`) that's stored in an alienbrain-managed folder, then Workspace enables the features described below.

**Note:** These features apply only to the project file and the assets for each game database item (`guid.xml`). The project `.settings` file is local to your computer, and is not intended to be managed by alienbrain. When you first import your project onto the alienbrain server, make sure to do so from inside Workspace, using the **File ▶ NxN ▶ Import project to alienbrain** menu option.

**Caution:** The features will only work after you have enabled the **Change set** feature in alienbrain—See the “Enabling and Disabling Change Set Functionality” topic of the alienbrain documentation.

## alienbrain login dialog

If you are not logged in to alienbrain, then Workspace displays the alienbrain login dialog before opening the project.

**Note:** When you log in to alienbrain, you are strongly recommended always to click **Connect**. If you work offline, then you will be working on your own local copies of the XML files. None of the features described below will apply, as you will be unable to save your work directly back onto the server.

## Automatic file check out

There is a two-way link between parent and child XML files in a project. If you perform an action in Workspace that affects the [hierarchical structure](#) (p.73) of an existing [game database XML file](#) (p.76) (say, adding or removing an item) then, before applying your action, Workspace attempts to check out the affected XML files from alienbrain.

For example, if you add a new entity to a folder, then Workspace checks out the XML file for the folder and the entity. This is because adding an entity to a folder involves adding an entity reference to the folder XML file. Similarly, if you are making a change to the structure of an entity, then both the parent entity XML file and the child object file need to be checked out.

However, if you are making a change that does not affect the hierarchy of the game database (say, renaming an object), then that file alone (without any of its associated parents or children) is checked out.

## Newer version on the server

If the file in alienbrain is newer than your local copy, then Workspace does not perform automatic check-outs, but displays a warning message. Before retrying the action, you need to get the latest versions of the game database XML files from alienbrain:

- Select **File ▶ NXN alienbrain ▶ Get Latest Version** (see the description below), or right-click the object in the Game Explorer (or any object-listing window), and then select **Get Latest Version** from the context menu.

## Change sets

Change sets are automatically managed lists of files that are waiting to be checked in to the alienbrain server. Automatic change set management comes into play when you:

- Create any type of new object—for example, an entity, a folder, or an attribute share—in an alienbrain-managed project.
- Make an object reference a resource that is currently unused—for example, a .dff, .bsp, or .rf3 file—and in a managed location.

**Note:** You can have more than one change set waiting for check-in at a time, but all automatic actions take place in the *default* change set.

## Default change set notification

Immediately after loading an alienbrain-managed project, Workspace uses the Version Control Log to display the name of your default change set. If it's not the one you want, you can change it before proceeding.

Optionally, you can configure Workspace *always* to display alienbrain explorer when it has loaded a managed project. This gives users the opportunity to switch default change sets there and then. To enable this functionality, use Enterprise Author to edit the script module called *NXNScriptExtension*. You should remove the comments from the highlighted lines in *Broadcast\_PostLoadProject()*:

```
Sub Broadcast_PostLoadProject(strFilename)
    If VersionControl.IsProjectManaged() Then

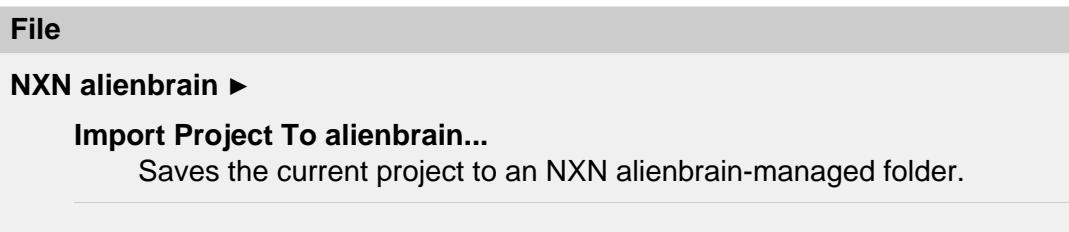
        ' Tell user name of current (default) change set
        Dim DefaultChangeSetName
        DefaultChangeSetName = VersionControl.GetDefaultChangeSetName()

        VersionControlLog.Log "Default change set is '" &
        DefaultChangeSetName & "'.

        ' To ask user if they want to change default change sets, uncomment
        this code.
        '    Dim bManageChangeSets
        '    bManageChangeSets = MsgBox ("Default change set is '" & _
        '                                DefaultChangeSetName & _
        '                                "'.
        '                                Show alienbrain explorer to manage
        change sets?", _
        '                                vbYesNo + vbQuestion, "Version Control")
        '    If bManageChangeSets = vbYes Then
        '        VersionControl.ShowProjectInExplorer
        '    End If
    End If
End Sub
```

## File menu items

The following items are enabled on the **File** menu:



**Get Latest Version**

Gets from alienbrain all of the latest XML files for the project, overwriting your local copies.

**Submit Default Change Set**

Submits and checks in the files in the default change set.

**Undo Default Change Set**

Reverts all the files in the default change set to the state they were in before they were checked out.

**Show NXN Explorer**

Pops up the NXN database explorer window.

**Refresh**

Refreshes the alienbrain status indicators on icons:

- indicates that the object is in an alienbrain-managed folder, but is not checked out.
- indicates that the object is checked out by you.
- indicates that the object is checked out by someone else.
- indicates that the object is due to be imported into the game database.

## Using the context menu in Workspace

Right-click any object (except a behavior) in any of the object lister windows or the Game Explorer window, and you are presented with the following context menu options:

**NXN alienbrain ▶****Check out**

Checks out an object.

**Submit pending changes...**

Saves the current project to an NXN alienbrain-managed folder.

**Undo pending changes**

Undoes the check out of any XML files checked out by you (losing any changes you have made since the files were checked out).

**Get Latest**

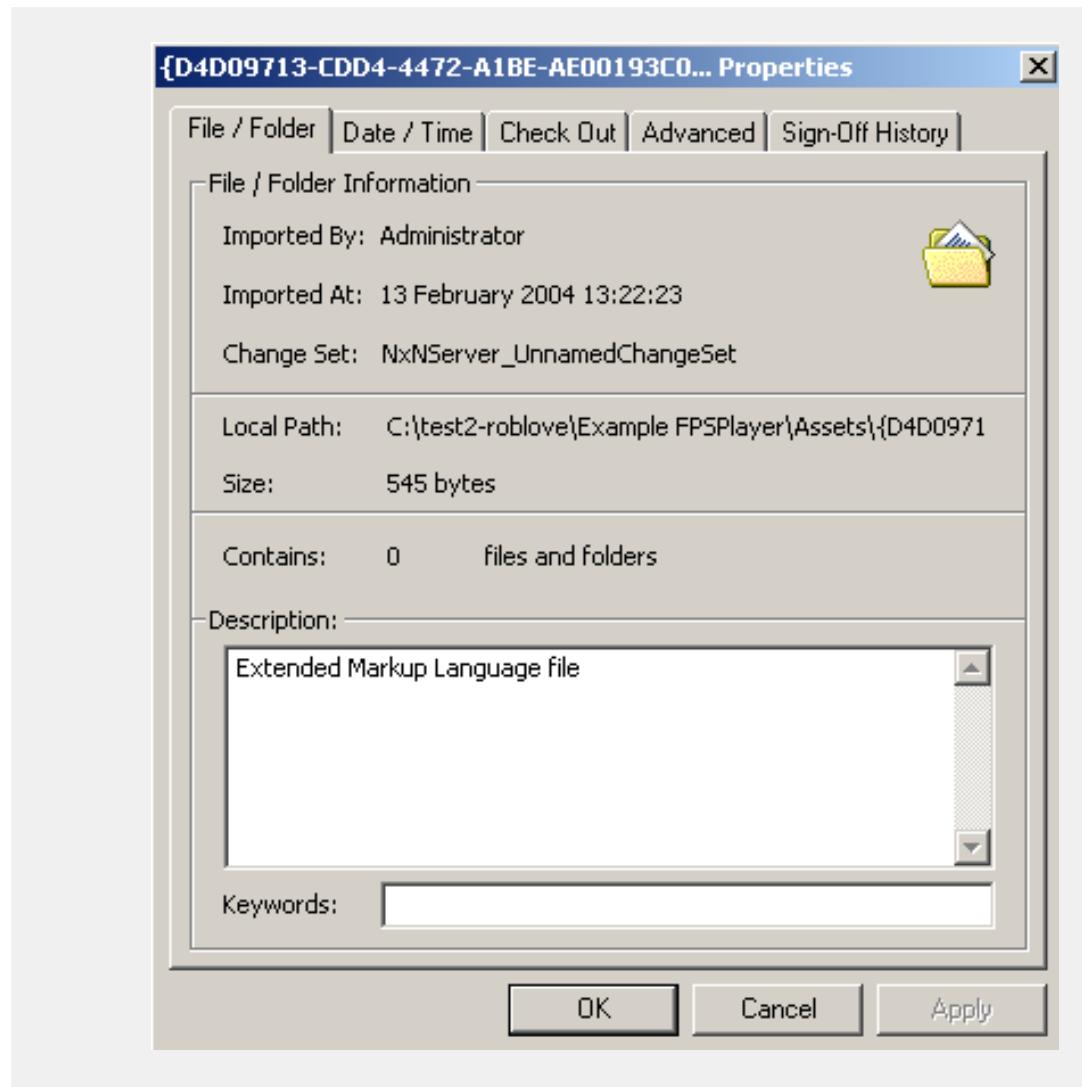
Gets all of the latest XML files for the project from the alienbrain server, overwriting or merging your local copies.

**Show history**

Opens the history window for the selected file, allowing you to see the list of previous versions of the file and difference them against the current version on your local disk.

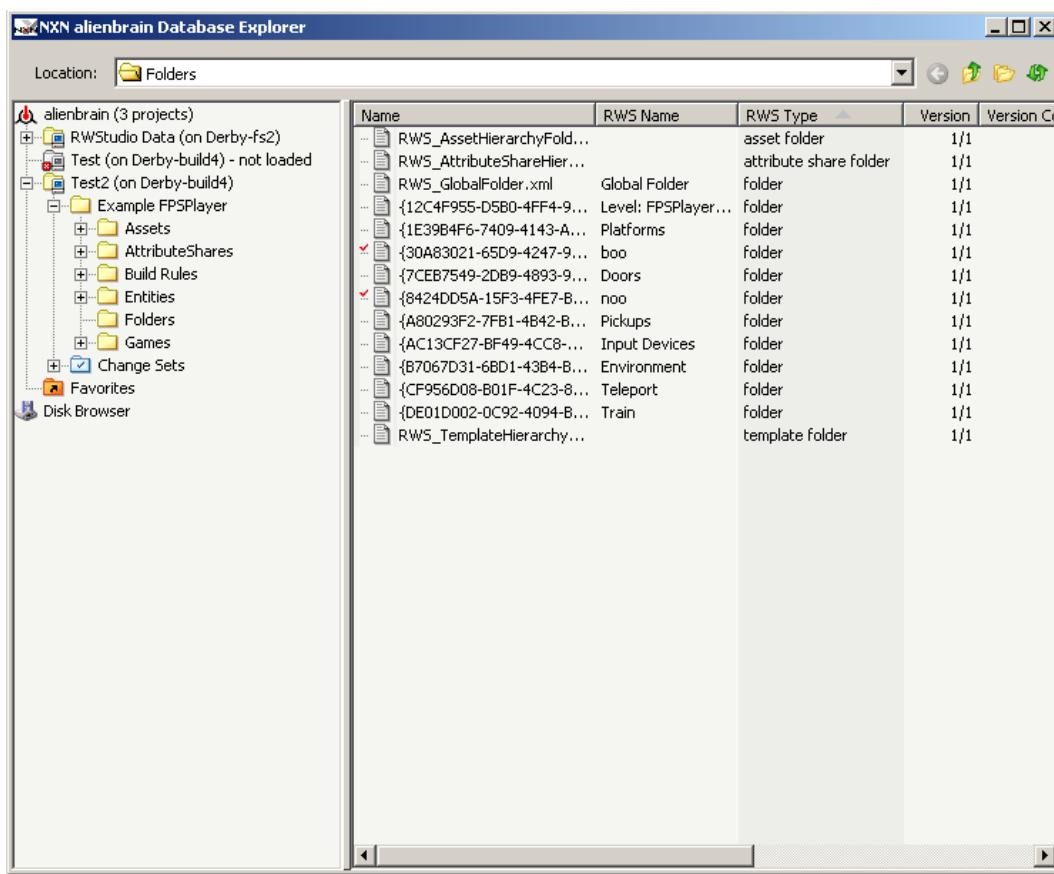
**Properties**

Opens the alienbrain properties window for the selected file.

**Special notes:**

- All of the above context-menu actions can be selected for an individual asset or resource by selecting **Studio Asset ▶** or **Resource ▶** and then the appropriate action from the context menu.
- When you check in the object or resource using this method, the object or resource is taken out of the default change set in alienbrain
- When using alienbrain change sets, you need to be careful that you don't check in an entity that references an asset that is saved on your client, but not on the alienbrain server. Make sure that both the asset and its parent entity file are checked in.

# Using the NXN Database Explorer



This window is a scaled-down version of the NXN manager client window that is hosted by RenderWare Studio. It allows you to perform all the version control operations on your game database that you would in the full alienbrain client manager window.

To launch the window, select **File ▶ NXN alienbrain ▶ Show NXN Explorer**.

## Using the alienbrain client outside of Workspace

If you use the Workspace to check out a file, then use the hosted NXN Database Explorer to check it back in: **do not** use the separate alienbrain client software to check it in.

At any time while you are using Workspace, you can also use NXN Database Explorer to:

- Get latest version
- Rollback to a previous version
- Add a label

If you use the alienbrain client to get the latest version or rollback while a related project is open in the Workspace, then Workspace will not automatically update to reflect changes to your local copies of the files (you will need to re-open the project). Whereas, in Workspace or the NXN Database Explorer, performing a "get latest" does update Workspace to reflect any changes.

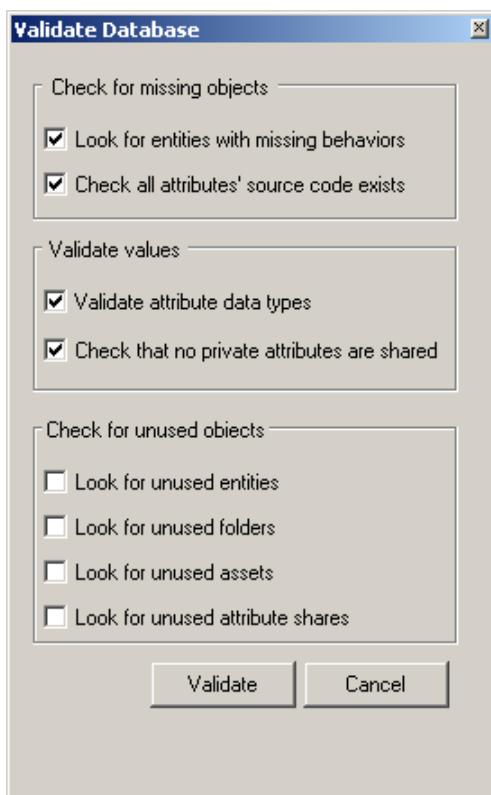
# Validating the database

---

As you develop your game, the source code evolves and your class definitions change. You might add and remove attributes, and sometimes an attribute's data type might change. Thus, any data for an attribute that is already saved in the XML code becomes invalid. Using the **Validate Database** tool, you can check the integrity of attribute data. You can also use this tool to check other parts of the database, such as whether entities or assets are unused in the game.

- On the **Tools** menu, click **Validate Database...**

The Validate Database dialog appears listing the database items that you can check for integrity.



## Look for entities with missing behaviors

Reads all behavior names and inserts them into an STL set. RenderWare Studio then examines all entities in the database and checks their behavior class-name against the constants of the STL set. If any entity class name does not exist, RenderWare Studio generates a warning.

## Validate attribute data types

For each class and entity using the class, RenderWare Studio reads every command and checks its data type string. RenderWare Studio locates the corresponding attribute and checks its data type. When any entity attribute's data differs from that specified in the source code, RenderWare Studio generates a warning.

## Check all attributes' source code exists

It is possible that attributes that were removed from the source code still

exist in the database. This option examines each entity's attributes and determines whether a corresponding command exists for each attribute. RenderWare Studio generates a warning for any attributes it finds which have no corresponding command.

#### **Check that no private attributes are shared**

It is possible that attributes that were shareable might have at some time been made private in the source code. This option checks any attributes within folders to ensure that their corresponding commands are not private. The source parser tags any private attributes.

#### **Look for unused entities**

RenderWare Studio queries the parents of each entity in the database. If an entity has no parent, or is a child of an orphaned entity folder, RenderWare Studio generates a warning.

#### **Look for unused folders**

RenderWare Studio queries the parents of each entity folder. If a folder has no parent, RenderWare Studio generates a warning.

#### **Look for unused assets**

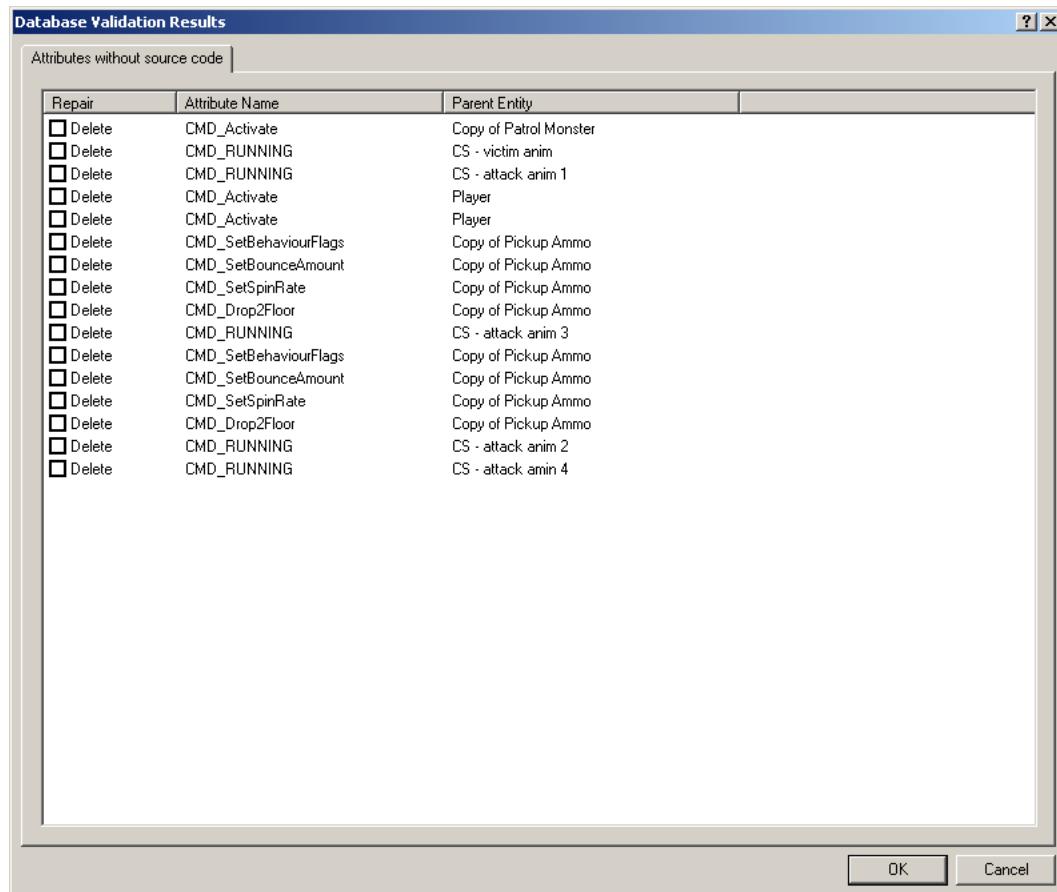
RenderWare Studio queries the parents of each asset in the database. If an asset has no parent, or is a child of an orphaned asset folder, RenderWare Studio generates a warning.

#### **Look for unused attribute shares**

RenderWare Studio queries the parents of each attribute share. If an attribute share has no parent, RenderWare Studio generates a warning.

## **Displaying results of a check and fixing problems**

When all checking has been performed, and if any errors were detected, RenderWare Studio displays the Database Validation Results dialog.



RenderWare Studio groups each error under an appropriate tab, for example, *Attributes without source code*, *Unused items*, and so on. Each error displays under columns for repair and for parameters appropriate to the database item, such as name, type, parent entity, and so forth.

In the case of database items which cause RenderWare Studio to display a **New Behavior** column, you can click the list box to display different behaviors. If this action will correct the error, then use the check box under the **Repair** column to fix it.

Click the box in the **Repair** column if you want to fix the problem. For example, the action to repair might be *Delete*, clicking the box will repair the error by deleting the database item. By default, RenderWare Studio displays all check boxes in this dialog as unchecked.

#### Repairing missing behaviors

Changes missing behaviors to *CEntity* for all affected entities.

#### Repairing attribute data types

Deletes attributes with incompatible data.

#### Repairing private attribute sharing

Removes shared private attributes from their shared parent folder.

#### Repairing unused entities

Deletes unused entities.

#### Repairing unused entity folders

Deletes unused entity folders.

#### Repairing unused assets

Deletes unused assets.

**Repairing unused asset folders**

Deletes unused asset folders.

**Repairing unused attributes**

Deletes unused attributes.

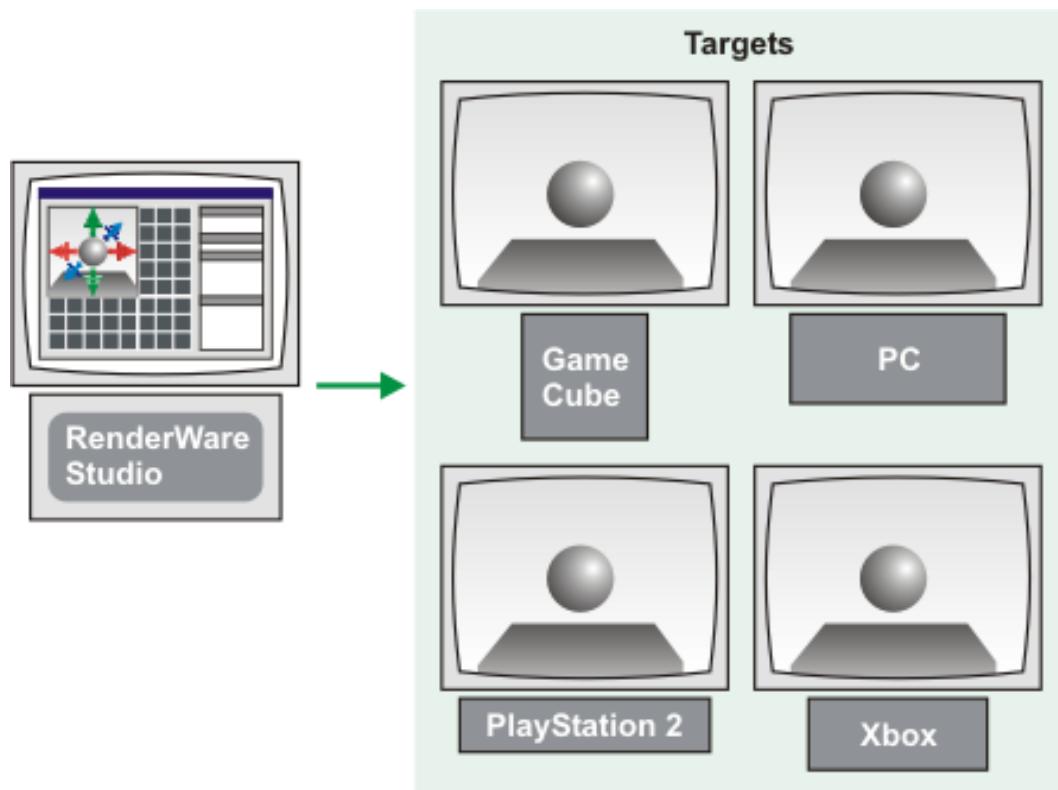
**Repairing unused attribute folders**

Deletes unused attribute folders.

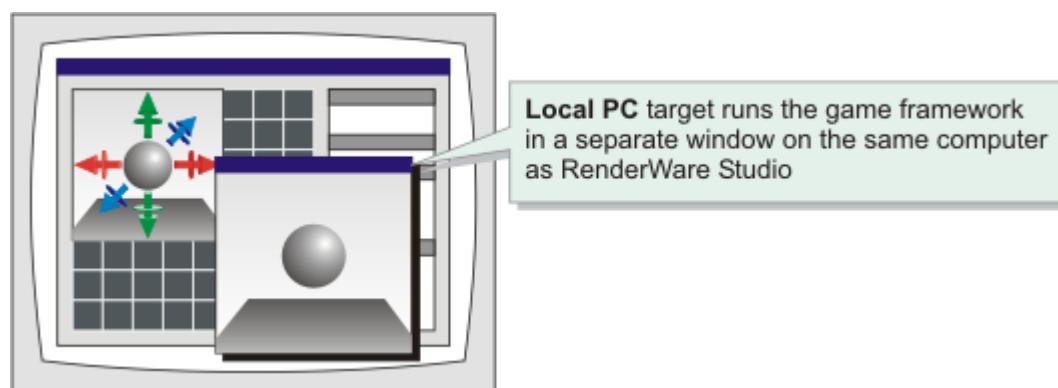
**NXN alienbrain users:** If NXN alienbrain denies access, any attempts at purging database items might fail. For example, the Validation Database tool might try to purge an entity which is checked out by another user. When this happens, RenderWare Studio displays an error message in the Message Log window.

## Viewing your game on a target

While you are developing a game, you can view it simultaneously on several different targets:



You can also view the game in a separate window on the same computer as RenderWare Studio:



While viewing the game on a target, you can continue using RenderWare Studio Workspace to develop the game. Any changes you make (such as adding, moving or deleting entities, or changing behavior attributes) are dynamically reflected on the target.

To view your game on a target:

1. [Create a connection](#) (p.199) to the target.

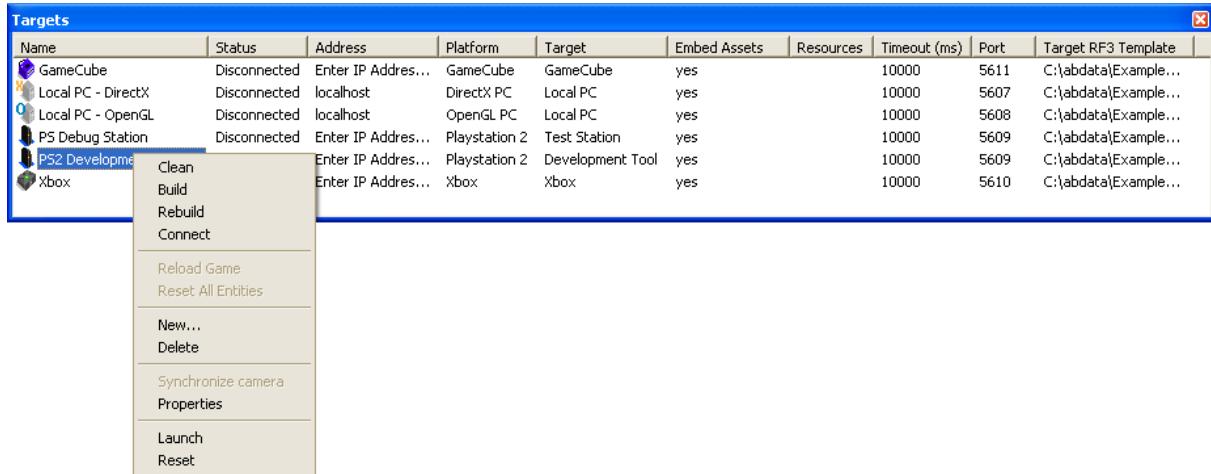
2. (GameCube design builds only.) Start the [GameCube Comms Server](#) (p.210).
3. [Start the Game Framework](#) (p.212) on the target.
4. [Connect](#) (p.213) to the target.

# Creating or editing a target connection

---

To create or edit a connection to a target (a console or PC on which you want to view your game):

1. In the Targets window, right-click a connection name.



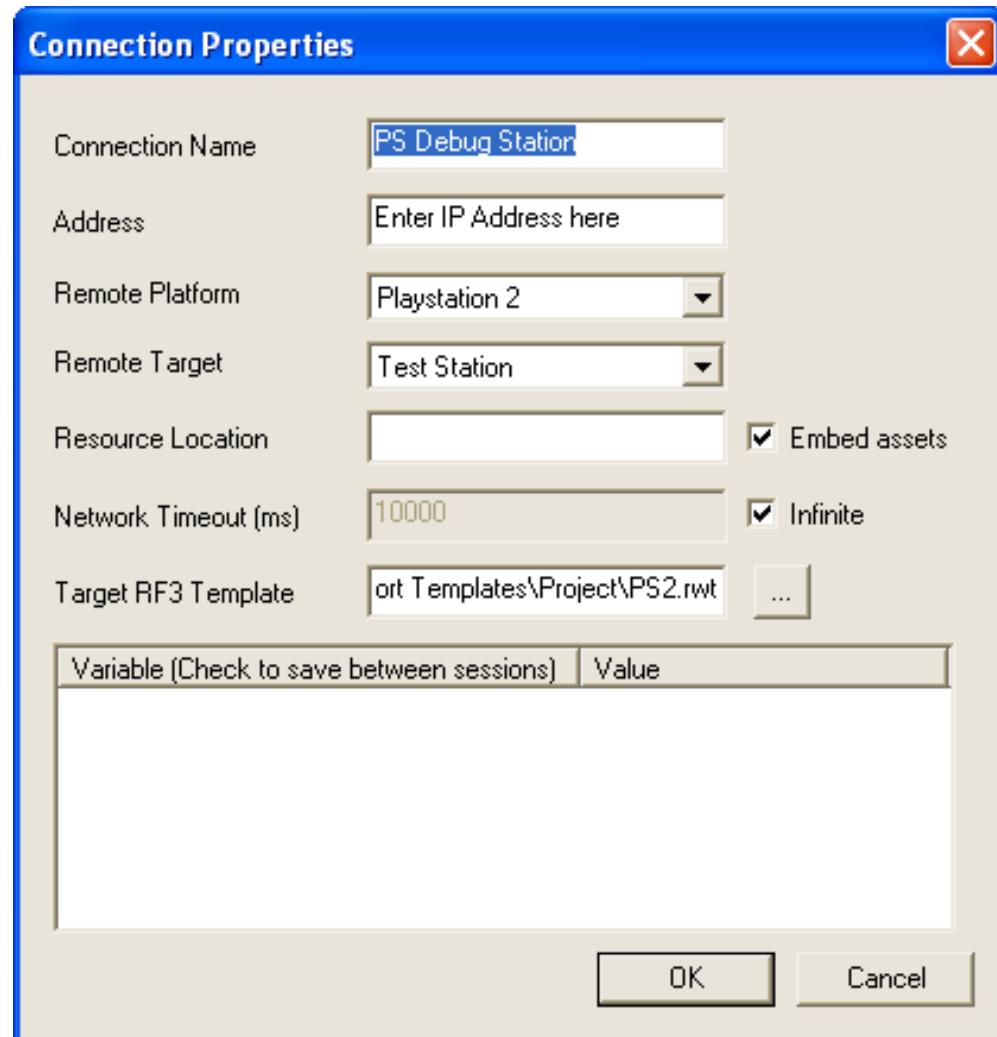
By default, the Targets window contains a connection for each type of target that RenderWare Studio can connect to.

You can choose to edit an existing connection, or create a new connection.

2. Either:

- Click **Properties...** to edit the connection that you right-clicked.  
or
- Click **New...** to create a new connection.

This displays the Properties dialog for the connection.



3. Specify the connection properties:

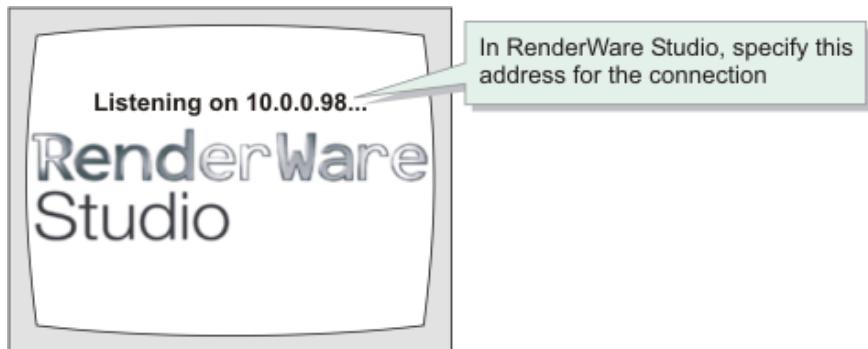
**Connection name**

Description of this connection.

**Address**

The IP address of the target to which you are connecting (or its host name, if it is on the same subnetwork as the computer running RenderWare Studio):

- For a local PC target (to view the game in a separate window on the computer running RenderWare Studio), specify localhost.
- For other targets, specify the address shown when you start the Game Framework:



The IP address is determined by the target:

### **GameCube**

If your LAN has a DHCP server, then, by default, the target uses the dynamic IP address supplied by the DHCP server. However, if your LAN does not have a DHCP server, or you want to override the dynamic IP address and use a static IP address instead, then create an `rwsipcfg.txt` file in the root folder of the DVD (emulation) drive, containing an IP address in dot notation (for example, `10.0.0.45`) at the start of the first line.

If this file does not exist (or exists, but does not contain a valid IP address), then the target uses an IP address supplied by the DHCP server; if no DHCP server exists, then the Game Framework will not initialize.

### **PlayStation 2**

You can either rely on a DHCP server to supply a dynamic IP address or specify a static IP address yourself (for example, in an `ifcnnn` file; for details, see the PlayStation 2 Development Environment documentation).

### **Xbox**

The Game Framework uses the Title IP address (not the Debug IP address).

## **IP port number**

The Workspace sends data to the IP port number defined in the `rwstudio.settings` file (in the RenderWare Studio Programs folder):

```
<OBJECT NAME="platform type">
<PARAM NAME="Default Port" VALUE="port number" />
:
```

The Game Framework listens for data on the platform-specific port number defined in the `network.cpp` file (in the `gfCore` library).

For example:

```
#if defined (_XBOX)
    RwBool bOK = NetStream::Init (5610);
```

To use different port numbers, edit these two files to match the new numbers.

If you use an Internet Connection Firewall (ICF), then you need to ensure that it allows communication via these ports (see your firewall documentation).

## **Remote platform**

The general type of target to which you are connecting:

- GameCube
- DirectX (PC)
- OpenGL (PC)
- PlayStation 2
- Xbox

**Remote target**

The specific type of target, depending on the platform. For example, for a PlayStation 2 target, you can choose Development Tool or Test Station.

**Resource location**

Specifies [how the target gets data](#) (p.204) from primary asset files and their dependencies. (If you select the **Embed assets** check box, then the Workspace embeds asset data in the game data stream; otherwise, it sends only the asset pathnames.)

**Network timeout (ms)**

The number of milliseconds that the Workspace waits for a response from the target before disconnecting. If you select the **Infinite** check box, then the Workspace waits indefinitely.

**Target RF3 Template**

The template that Workspace uses to generate streams from RF3 files, for inclusion in the target's game data stream.

4. Click **OK**.

The Properties dialog closes.

## How targets get asset data

---

Targets get asset data either from the Workspace (via the game data stream) or from files on the target's own file system, depending on the **Embed assets** check box in the target connection [properties](#) (p.199):

- If this check box is selected, then the Workspace sends asset data to the target in the game data stream.
- Otherwise, the properties specify a **Resource location**. This is the absolute path of a folder on the target's file system that contains the primary asset files and their dependencies. The Workspace concatenates this absolute folder path with each asset's relative file path, and sends the resulting pathname to the target in the game data stream. The target uses this pathname to find the file on its own file system.

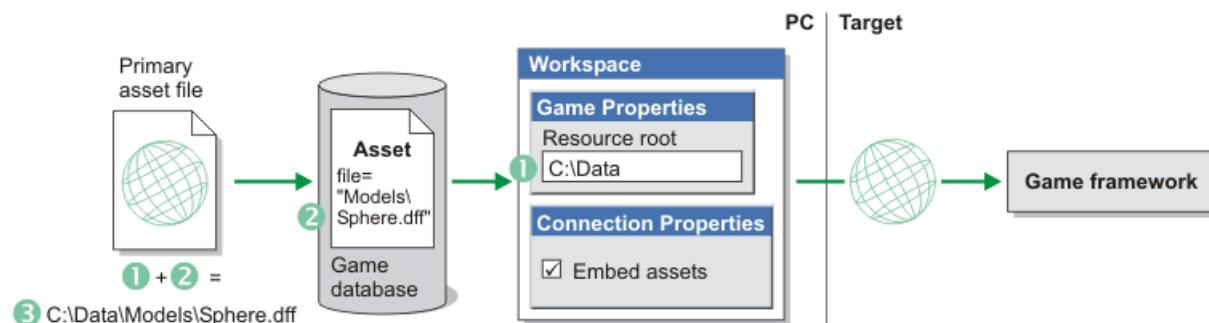
Approach	Description	Advantages	Disadvantages
<b>Embed assets</b>	The Workspace sends asset data in the game data stream.	The primary asset files do not need to be on the target's file system.	Increases the size of the game data stream, increasing the time it takes for your game to start when you connect to a target.
<b>Resource location</b>	The Workspace sends asset pathnames, not data, in the game data stream.	Reduces the time it takes for your game files and to start when you connect to a target.	The primary asset dependencies must be on the target's file system.

## Embedding asset data in the game data stream

To embed asset data in the game data stream, the Workspace needs to locate the primary asset files.

As shown in the diagram below, to find a primary asset file, the Workspace combines:

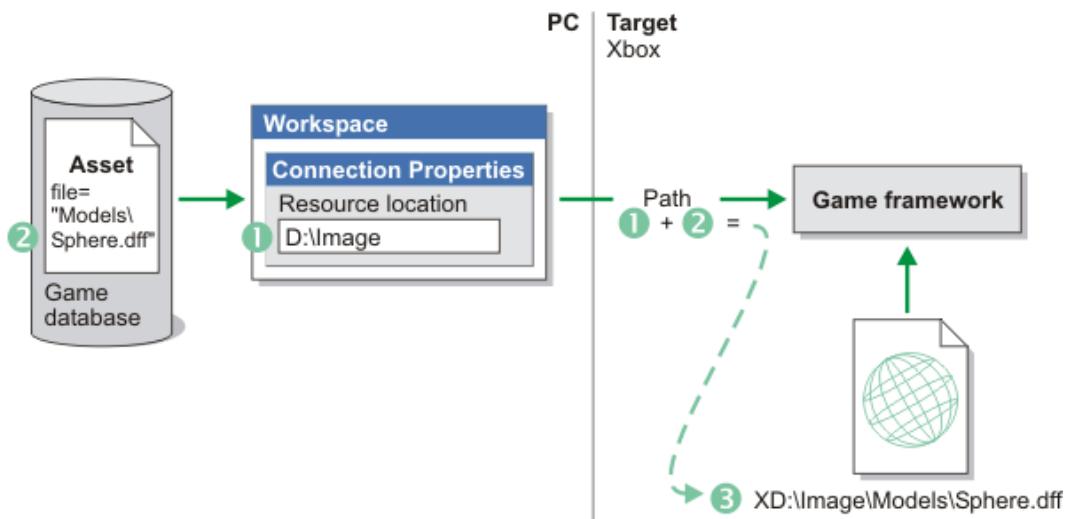
- The **resource root** (p.98) folder of the game (an absolute folder path)
- with
- The **File** property of the asset (a relative file path)



## Including asset pathnames in the game data stream

If you specify the resource location as a path, then the Workspace includes asset pathnames, not asset data, in the game data stream. The target uses these pathnames to find the asset data on its file system.

For example, for an Xbox target:



The Workspace sends to the target:

- The **resource location** for the connection (an absolute folder path) and
- The **File** property of the asset (a relative file path) and the **Dependencies** property of the asset (a relative folder path; not shown in the diagram above).

To find the primary asset file and its dependencies, the Game Framework combines the resource location with these relative paths.

## Specifying the resource location path

How you specify the resource location path depends on the type of target:

Target	Resource location path refers to...
GameCube	A folder on the Optical Disk Emulator Software (ODEM) host PC. For example: C:\Tutorial <b>Note:</b> Do not end this path with a backslash.
DirectX or OpenGL Local PC	A folder on the computer running RenderWare Studio. For example: C:\RW\Studio\Tutorial To use the same asset data for this connection as used in the RenderWare Studio Workspace: 1. In the Game Explorer, right-click the Game node and then click <b>Properties</b> . 2. Double-click the value in the <b>Resource root</b> text box.

This selects the value.

3. Press **CtrlInsert**.

This copies the value to the Clipboard.

4. Click **Cancel**.

5. In the Targets window, right-click the connection and then click **Properties**.

6. Click inside the **Resource Location** text box.

7. Press **ShiftInsert**.

This pastes the contents of the Clipboard into the text box.

---

A folder on the remote computer. In this case,

**DirectX or OpenGL**

**Remote PC**

C:\RW\Studio\Tutorial

refers not to a folder on the computer running RenderWare Studio, but to a folder on the remote PC.

---

Either:

- A folder on the host PC.

Prefix the path with **host0:**, and use forward slashes instead of back slashes. For example:

host0:C:\RW\Studio/Tutorial

or

- A folder on the CD drive of the Development Tool (as per the Debugging Station, below).
- 

A folder on the CD drive of the Debugging Station. (Unlike the Development Tool, the Debugging Station cannot refer to files on a host PC.)

Prefix the path with **cdrom:/**, use forward slashes instead of back slashes, and end the path with a semi-colon. For example:

cdrom:/Tutorial;

Before connecting to the target:

1. Copy the primary asset files and dependencies onto the appropriate folder of a CD.
  2. Insert the CD into the Debugging Station (or Development Tool).
- 

**Xbox**  
Development Kit

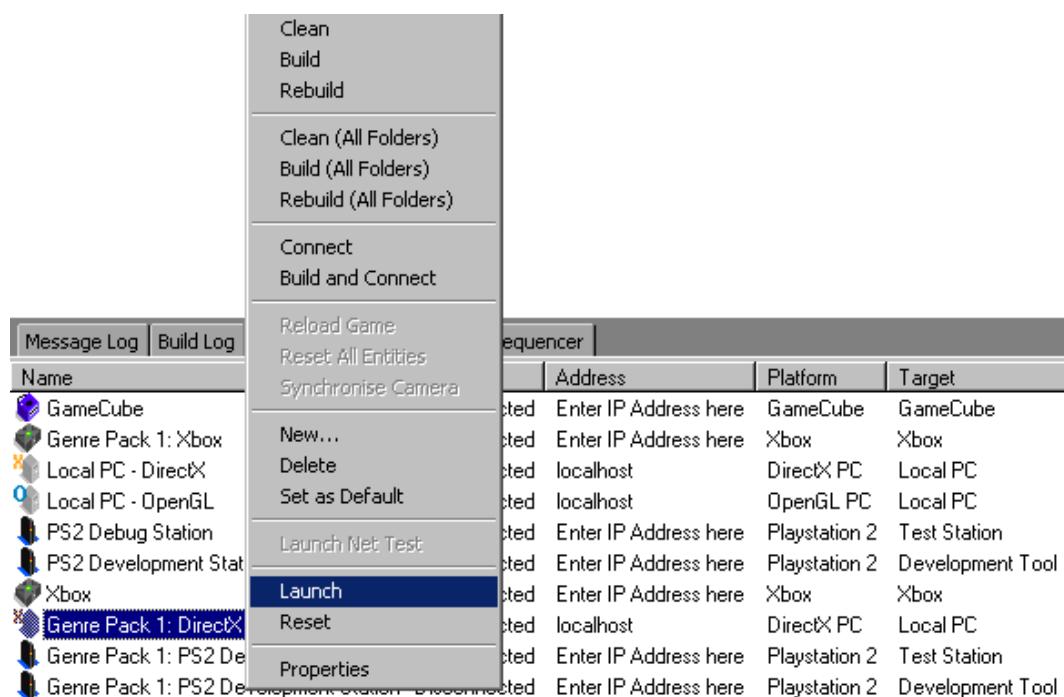
A folder on the Xbox Development Kit, identified by the prefix **d:\** (see the example diagram, above).

---

Before connecting to the target, ensure that the primary asset files and dependencies exist in the location that you have specified.

# Creating a custom action for a target

When you right-click a connection in the Targets window, a context menu appears.



You can add custom actions to this menu for each target platform (similar to the **Launch** action for some targets).

**Caution:** The following procedure involves editing the RenderWare Studio settings file. If you introduce errors into this file, then RenderWare Studio might not correctly initialize some controls.

To add a custom action:

1. Exit RenderWare Studio.
2. Create a backup copy of the `RWStudio.settings` file (stored in the RenderWare Studio Programs folder), so that you can return to this copy later if you introduce errors.
3. Open the `RWStudio.settings` file in a text editor.

This XML file contains an `<OBJECT NAME="TargetLink">` section that defines the platforms and their parameters, the different types of target for each platform, and the actions for each target.

The following excerpt shows the PlayStation 2 platform, its parameters, and its two types of target: Development Tool (DTL-10000) and Test Station. Note that only the Development Tool target has an action defined ("Launch").

```

<OBJECT NAME="TargetLink">
    <OBJECT NAME="Platforms">
        <OBJECT NAME="Playstation 2">
            <PARAM NAME="Port" VALUE="5610">
        </PARAM>
    </OBJECT>

```

```

<PARAM NAME="Platform Flags" VALUE="PS2">
</PARAM>
<OBJECT NAME="Development Tool">
<PARAM NAME="Launch"
VALUE="&quot;C:\RW\Studio\Programs\Tools\RWSCLaunch.exe&quot;,<br>
$(Platform Flags) $(Tool Address) $(elf file)">
</PARAM>
<PARAM NAME="action" VALUE="command" />
</OBJECT>
<OBJECT NAME="Test Station">
</OBJECT>
</OBJECT>
</OBJECT>
</OBJECT>
</OBJECT>
</OBJECT>

```

4. Insert a new `<PARAM>` element, as shown in the listing above. If the executable path of your *command* includes spaces, then insert `&quot;` before and after the path (when RenderWare Studio reads the command, this entity reference gets resolved into a quotation mark).

**Tip:** If you insert a `/` before the `>` of the `<PARAM>` tag, then you can (must) omit the closing `</PARAM>` tag.

### Passing values to the command

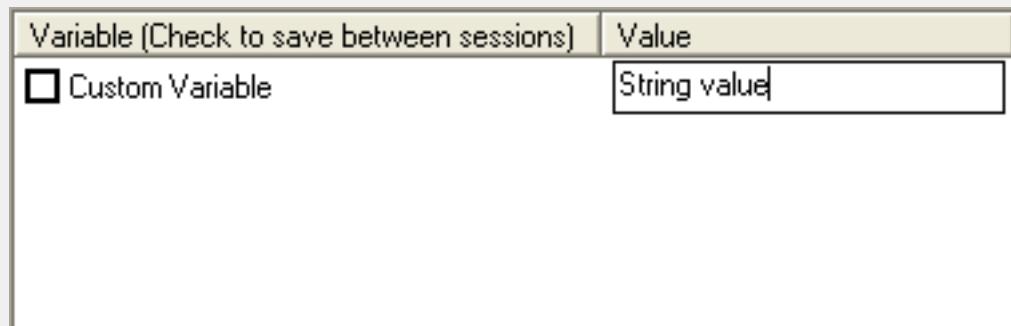
You can supply parameters to your command in the form:

`$(variable)`

where *variable* can be either:

- The `uid` attribute of a `<column>` element (for details, see the `<TargetLink>` element in the settings file)
- or
- An arbitrary name

If you specify an arbitrary name—for example, `$(Custom Variable)`—then the connection properties dialog shows it like this:



To specify a value, click in the **Value** column next to the variable name, and then enter a string value. If you select the check box, then the value is saved in the settings file.

5. Save the `RWStudio.settings` file.

The next time you start RenderWare Studio, the new item appears on the context menu for all connections of that target type.

# Starting the GameCube Comms Server

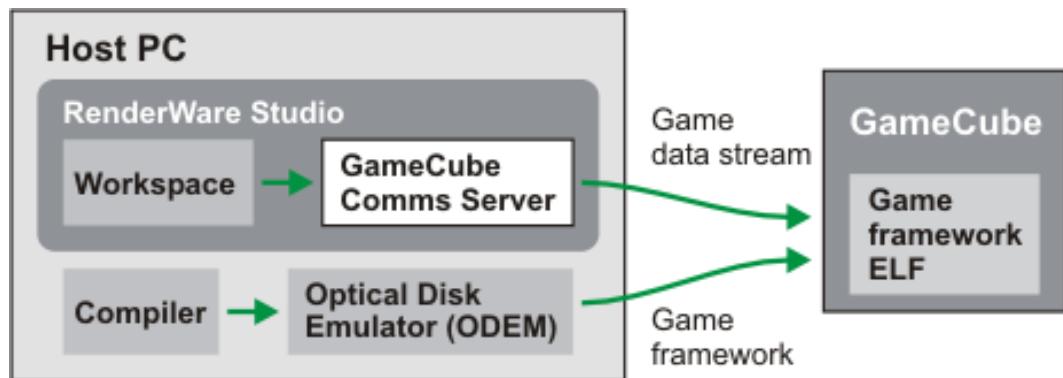
---

This topic applies only if you are *not* using a [broadband adapter](#) (p.215).

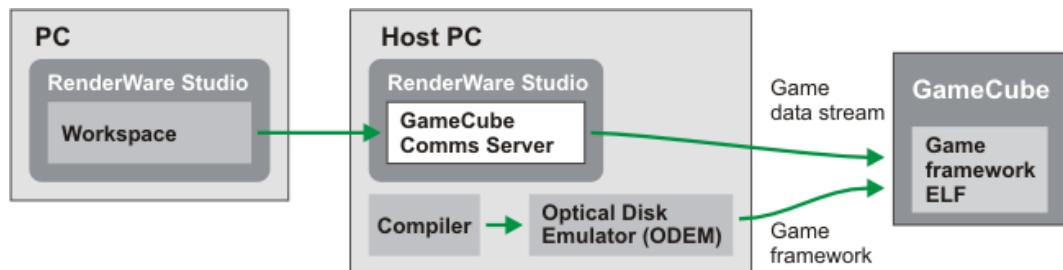
Before you can view a design build of your game on a GameCube, you need to start the GameCube Comms Server on the PC that is connected to the GameCube (the “host” PC).

The GameCube Comms Server (supplied with RenderWare Studio) enables the Workspace to send the game data stream to the GameCube.

You can run the Workspace either on the same PC as the GameCube Comms Server:



or on a separate PC (connected via an IP network):



## Note:

- Either way, you need to install RenderWare Studio on the host PC.
- You can also run the compiler on a separate PC (so far, we have only tested this with ProDG).

## To start the GameCube Comms Server:

- Click **Start ▶ Programs ▶ RenderWare Studio ▶ Target ▶ GameCube CommsServer**

A GameCube icon appears in your system tray (typically, this is at the bottom right of your screen):

If there is a problem with the GameCube Comms Server, then a red cross appears on the icon. Resetting the server usually fixes the problem: either right-click the icon and then click **Reset**, or double-click the icon.

If you start the Game Framework on the GameCube before starting the GameCube Comms Server, then the console screen will be blank. To fix this, start (or reset) the GameCube Comms Server.

If you attempt to connect to the GameCube, but the GameCube Comms Server is not running, then the “Connecting to Console...” progress bar will not complete, and will eventually time out. Start (or reset) the GameCube Comms Server, and then try connecting again.

## Starting the Game Framework

---

The Game Framework is an executable file that you:

1. Compile in a PC-based C++ development environment.
2. Transfer to a target (console or PC).
3. Start (run) on the target.

The Game Framework manages the entities, their behaviors, and communication between behaviors.

# Connecting to a target

---

This topic assumes that you have already [created a connection](#) (p.199) to the target, and that you have [started](#) (p.212) the Game Framework on the target.

If you want to connect to a design build on a GameCube, then you also need to start the [GameCube Comms Server](#) (p.210).

To connect to a target:

1. In the Targets window, right-click a connection name.
2. Click **Connect**.

After a few seconds, the game appears on the target.

# Connecting to a GameCube SN-TDEV

---

To use an [SN-TDEV](#) ([www.snsys.com/GameCube/SN-TDEV.htm](http://www.snsys.com/GameCube/SN-TDEV.htm)) with RenderWare Studio:

## Install the hardware and software

1. Install the SN-TDEV hardware and host software as described in the manuals supplied with the SN-TDEV.
2. Attach a broadband adapter to the SN-TDEV and connect this to your LAN.
3. Ensure that you have the latest “Network Base Package” and “Socket Library Package” from the Nintendo [developer relations site](#) ([www.warioworld.com](http://www.warioworld.com)).
4. Ensure that you have the latest “Build Tools” from the SN Systems [support site](#) ([www.snsys.com](http://www.snsys.com)).

## Compile the Game Framework

5. After installing all of the above, compile the Game Framework using a [GDEV Broadband](#) (p.408) build configuration.

## Connect RenderWare Studio to the SN-TDEV

6. Run the .elf file created in the previous step.
7. Open a RenderWare Studio project.
8. In the Targets window, right-click the GameCube target, and then select **Properties....**
9. In the **Address** box, type the target [IP address](#) (p.199) (displayed on the target screen), and then click **OK**.
10. In the Targets window, right-click the GameCube target again, and then select **Connect**.

Your project will now be sent via the broadband adapter to the SN-TDEV.

# Connecting to a GameCube DDH or GDEV via a broadband adapter

---

To connect the Workspace to a GameCube DDH or GDEV via a broadband adapter:

1. Attach a broadband adapter to the DDH or GDEV and connect this to your LAN.
2. Ensure that you have the latest “Network Base Package” and “Socket Library Package” from the Nintendo [developer relations site](http://www.warioworld.com) ([www.warioworld.com](http://www.warioworld.com)).
3. Compile the Game Framework using a **Broadband** (p.408) build configuration.
4. Run the `.elf` file created in the previous step.
5. Open a RenderWare Studio project.
6. In the Targets window, right-click the GameCube target, and then select **Properties....**
7. In the **Address** box, type the target [IP address](#) (p.199) (displayed on the target screen), and then click **OK**.
8. In the Targets window, right-click the GameCube target again, and then select **Connect**.

Your project will now be sent via the broadband adapter to the DDH or GDEV.

# Running Game Framework under AtWinMon (PlayStation 2)

---

This topic describes the steps you need to complete to run the RenderWare Studio Game Framework under **AtWinMon** on a PlayStation®2 Test Station with a USB-to-ethernet adapter.

You can download AtWinMon from the [PS2 Devnet](#) ([www.ps2-pro.com/projects/gt-at-monitor/](http://www.ps2-pro.com/projects/gt-at-monitor/)) website (requires user ID and password). You will need the latest version of the **alternative monitor** and the Windows client tools. At the time of this writing, the Window client tools include **atwinmon113v2** and **atwin070**.

For an explanation on how to use AtWinMon and how to build and burn a CD, see the readme file (ReadMe.txt) located in the directory where AtWinMon was installed. In addition, you might find the following notes useful.

- The Game Framework was tested with the Inet version of AtWinMon; this is the most current version. The older *configurable* version should also work but has not been tested.
- The version of the IOP BIOS and .irx modules used by AtWinMon should be the same as the ones used by your executable (the precompiled version in atwinmon113v2 use the 2.54 versions). When you download AtWinMon, there is a batch file called getiop.bat which simplifies this process.
- The network settings used by AtWinMon can be set in two ways:
  1. By hard-coding the settings in config.c and rebuilding AtWinMon.
  2. By burning a CD with AtMon on (the original Sony monitor program) and running it. It will allow you to set the network configuration when running on the console, and save the settings to a memory card. If this memory card is plugged in when AtWinMon is used then the settings in it will override any hard-coded settings in config.c.
- All of the files to be put on the CD for AtWinMon are put into the Target directory. The batch files, getiop.bat and put.bat does this automatically.

## AtWin Client Tools

For an explanation on how to use the Windows client tools, refer to the readme file (ReadMe.txt) in the directory where AtWin was installed.

The file \exe\default.txt lists in this order: the IP addresses of the host PC and the target PS2 Test Station / T10K Devkit. You will need to enter your correct IP addresses.

**Note:** AtWinMon will show the PS2 address on the screen when it is running. To get the IP address of your PC, ask your network administrator, or if you use Windows 2000, enter ipconfig in a command prompt.

## Compiling the Game Framework to run with AtWinMon

To build a PS2 game\_framework executable (.elf) to run with AtWinMon, refer to the readme file (ReadMe.txt) in the Atdev directory where AtWinMon was

installed.

Complete the following steps to run the Game Framework with AtWinMon.

1. If using the Inet version of AtWinMon, add ATWINMON to the list of project defines for the gfCore (Game Framework core) project.
  - For SN Systems ProDG, add ATWINMON to the list of preprocessor defines for the gfCore project in Visual C++.
  - For Codewarrior, add the line below to Prefix\_PS2\_RW.h:

```
#define ATWINMON
```
2. **Note:** This define stops the Game Framework from loading up the Inet modules because AtWinMon has already loaded them.  
Replace the project crt0.s file with the one in the Atdev directory.
  - For SN Systems ProDG, the crt0.s file will be in the same directory as the game\_framework Visual C++ project file. Make a backup of the original before copying the crt0.s file.
  - Metrowerks Codewarrior uses Sony's crt0.s file located in `usr\local\sce\ee\lib`. Make a backup copy of this before copying the AtWinMon crt0.s file.
3. Do a full rebuild of the game\_framework project.

## Running the Game Framework with AtWinMon

1. Burn a CD with AtWinMon on and setup the network settings as appropriate.
2. Re-compile the Game Framework so that it will work with AtWinMon.
3. Insert the AtWinMon CD into the PS2 and allow it to start. AtWinMon displays a screen containing general and network information.
4. Set the host PC and target PS2 IP address in the `exe\default.txt` file where the client tools are located.
5. Run **mcpclient.exe**.
6. Run **rfsv.exe**.
7. In the **mcpclient window**, enter the following: **exec your game .elf file**.

For example:

```
exec
C:\RW\Studio\Console\game_framework\SN_Sky\bin\Design_Metrics\game_framework_ps2.elf
```

8. Connect to the target with RenderWare Studio as you usually do.

## Tips on compiling and running Game Framework with AtWinMon

- If you need to access files, rather than hardcoding the root file server (e.g. `host0:` or `cdrom0:`), use `RWS::GetFileServerRoot()` instead. The file server root when using AtWinMon depends on the IP address of the client that is connected to AtWinMon. (The syntax is: `atfile:xxx.xxx.xxx.xxx,`).
- An .elf compiled with the crt0.s file from in the Atdev directory also works as a standalone .elf which you can run on a T10K Devkit. However, it is not

TRC-compliant, so for a submission build, be sure to use the original.

- Debug builds of the Game Framework are likely to run out of memory on a PS2 Test Station, especially if they load large game level folders from RenderWare Studio.

# Debugging your game

---

To debug your game, build the Game Framework using one of the debug [build configurations](#) (p.408).

**Note:** RenderWare Studio displays debug messages when you connect to a debug console. If you work with a debugger, such as the Visual Studio debugger, then RenderWare Studio still displays debug messages in the Message Log message box.

## Overlaying message names on the game display

To see which entities are sending event messages, and what the messages are:

1. On the target, start a debug build of the Game Framework.
2. In the Game Explorer window, drag a **CDebugTools** behavior from the Behaviors folder to the current folder.

As entities send event messages, the message names appear next to the entity, overlaid on the game.

To change the debugging information generated by the CDebugTools behavior, see the [Attributes](#) (p.270) window for the CDebugTools behavior.

## Selecting a build configuration to run on a PC

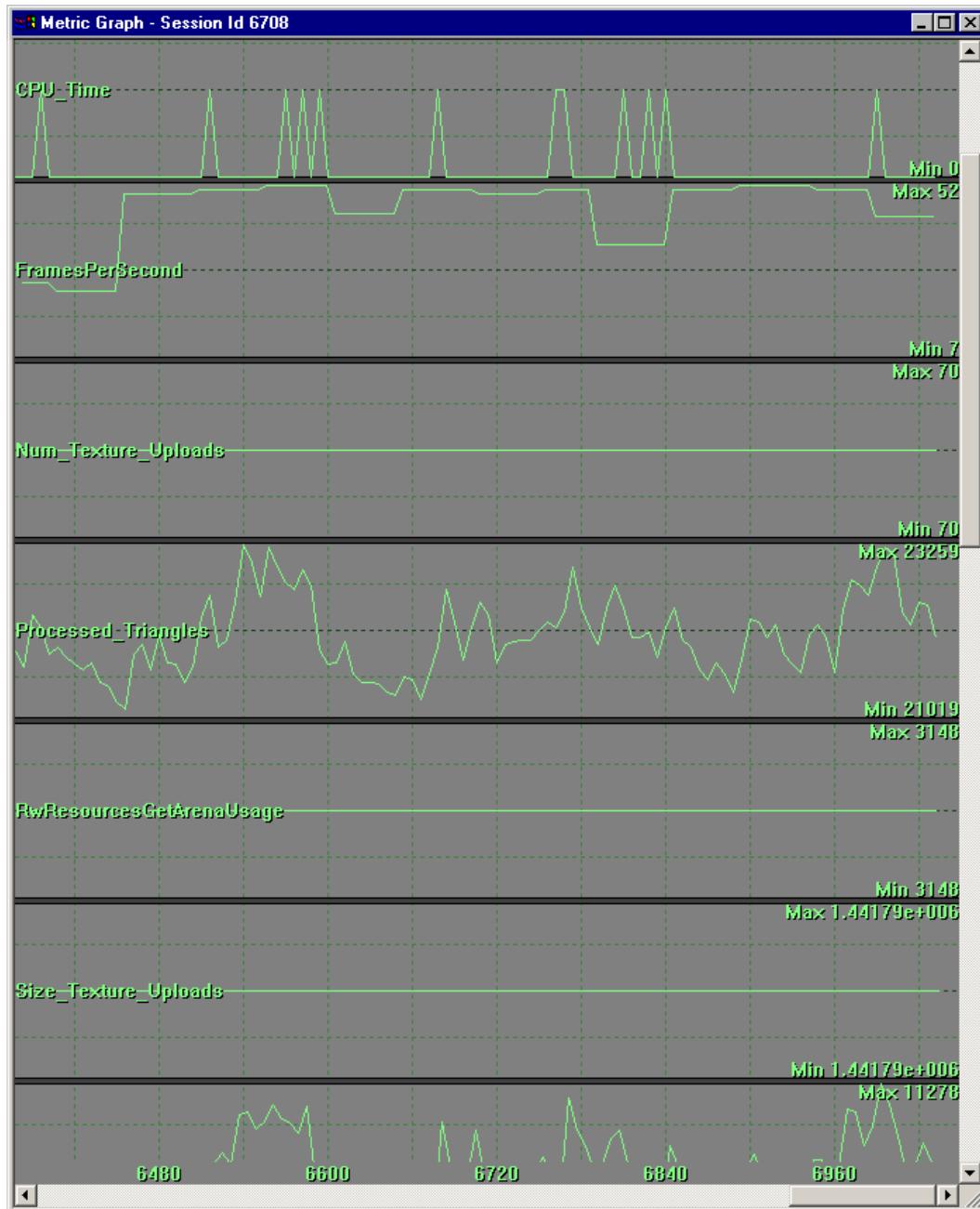
If you are running the Game Framework on a PC target, then you can prebuild several different Game Framework executables (for example, using different build configurations), and then choose which executable you want to run:

1. In the Targets window, right-click the PC target connection.
2. Click **Launch**.
3. Edit the path to refer to the location of the Game Framework executable that you want to run.

# Graphing variables over time

---

The Metric Graph shows variable values over time.



**Note:**

- The horizontal axis shows the frame number.
- The vertical axis automatically scales to include the minimum and maximum values.

## Adding variables to the Metric Graph

For each variable that you want to graph, you need to insert an

RWS\_TRACE\_METRIC macro call in your Game Framework source. For example:

```
RWS_TRACE_METRIC( 6 , "CPU_Time" , Time );
```

This sends the value of the variable Time to a buffer on every sixth call. The value appears on the Metric Graph with the caption “CPU\_Time” (captions cannot include spaces or tabs).

Each call to RWS\_TRACE\_METRIC sends the value once only (or rather, on every *n*th call, according to the first macro parameter). You should insert the RWS\_TRACE\_METRIC macro calls in sections of code that are called every frame.

You can include more than one variable on the same captioned graph. For example:

```
RWS_TRACE_METRIC( 6 , "CPU_Time" , Time2 );
```

This shows the value of the variable Time2 on the same “CPU\_Time” graph as Time (from the previous example). If you include more than one variable on a graph, then, rather than showing a line, the graph shows a band, indicating the range of values.

Another macro, RWS\_SEND\_TRACE\_METRICS, sends the buffer contents to the Metric Graph, and then clears the buffer:

```
RWS_SEND_TRACE_METRICS( 60 );
```

The parameter on this macro specifies the number of times the macro is called before it sends and clears the buffer. If the buffer fills before this number is reached, then the parameter is ignored, and the macro sends and clears the buffer immediately. In the example above, if the Game Framework is running at 60Hz, then the Metric Graph receives updated values once every second.

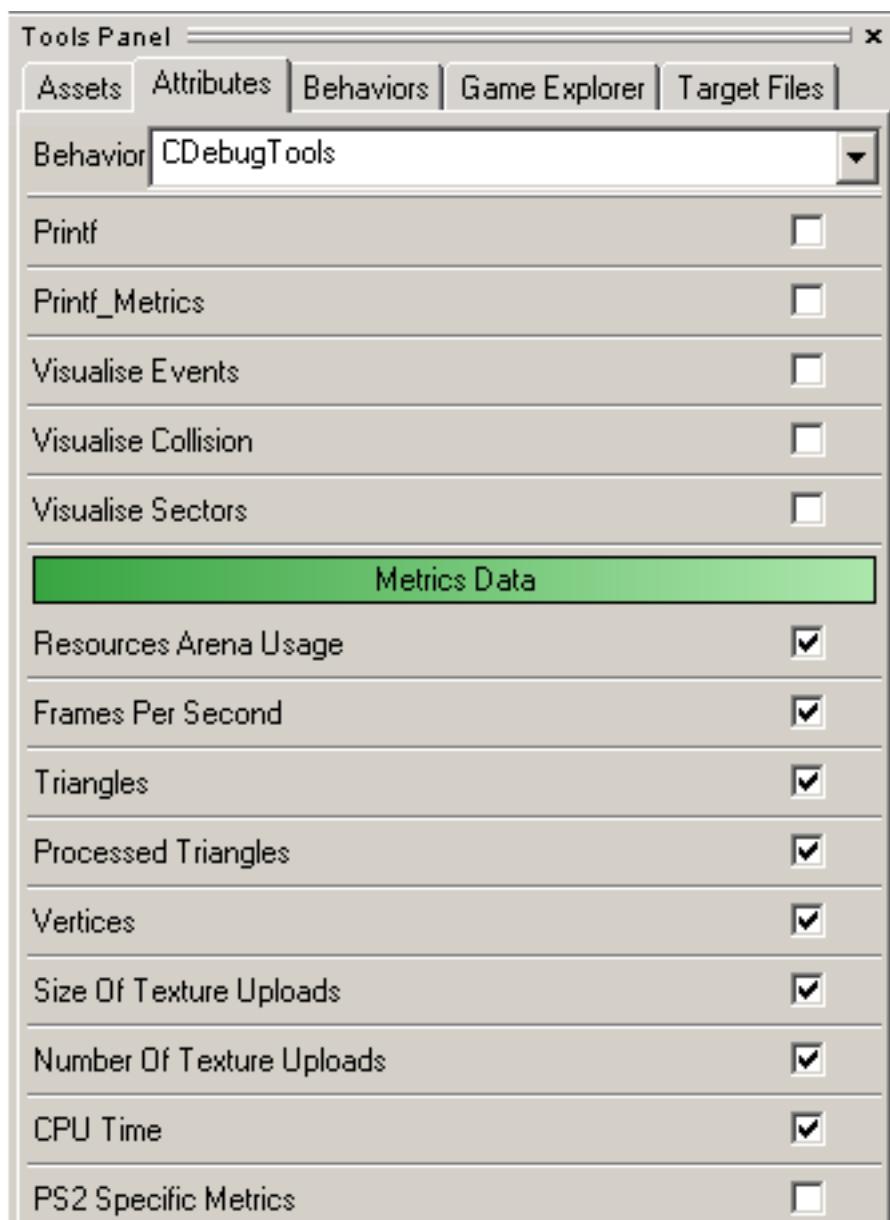
**Note:**

- Sending too much metric data too frequently can congest your network.
- The supplied Game Framework already contains a call to RWS\_SEND\_TRACE\_METRICS, in mainloop.cpp (in the core library).

## Adding debug-related variables to the Metric Graph

If you add a **CDebugTools** behavior to a game level folder, then, instead of inserting RWS\_TRACE\_METRIC macro calls, you can use an easier method to add certain debug-related variables on the Metric Graph:

1. Select the entity to which the CDebugTools behavior is attached.
2. Click Attributes.
3. In the Behavior list, select CDebugTools.
4. Under the Metrics Data heading, select the variables that you want to display on the Metric Graph:

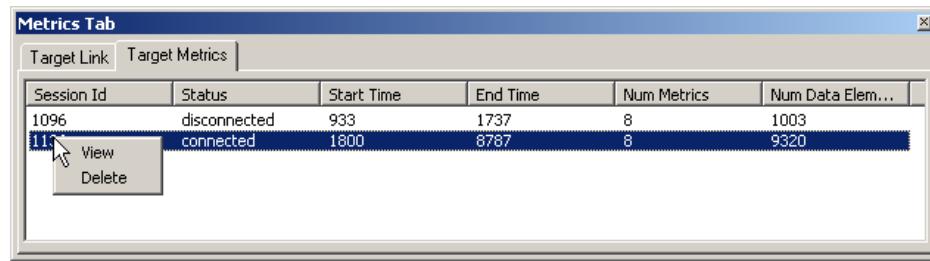


(Selecting these check boxes is equivalent to inserting  
RWS\_TRACE\_METRIC macro calls for these variables.)

## Viewing the Metric Graph

To view the Metric Graph:

1. In your compiler:
  - a. Build the Game Framework using a metrics build configuration.
  - b. Start the Game Framework on the target.
2. In the Workspace:
  - a. Connect to the target.
  - b. Switch to the Metrics layout:
    - On the **View** menu, select **Layouts ▶ Metrics Layout**.
  - c. Click the Target Metrics tab.



The Target Metrics tab displays a row for each target connection that you have created since starting the Workspace. Each row contains the following columns:

**Session Id**

Unique number that identifies the connection.

**Start time**

Frame number when the connection started.

**End Time**

Frame number when the connection ended.

**Num Metrics**

Number of graphs shown in the Metric Graph window.

**Num Data Elements**

Number of times that the RWS\_TRACE\_METRIC macro has sent a value to the buffer.

- d. On the Target Metrics tab, right-click the connection that you have just created, and then click **View**.

The Metric Graph for that connection appears.

# Viewing RenderWare streams with NetTest

---

The NetTest tool displays a text description of the contents of a RenderWare stream. NetTest can either:

- Listen to an IP port, and describe a stream as it is being sent to a target. This is especially useful for debugging target connections, so that you can see exactly what data is being sent to the target across the network.  
or
- Open an existing stream file (.rws or .stream).

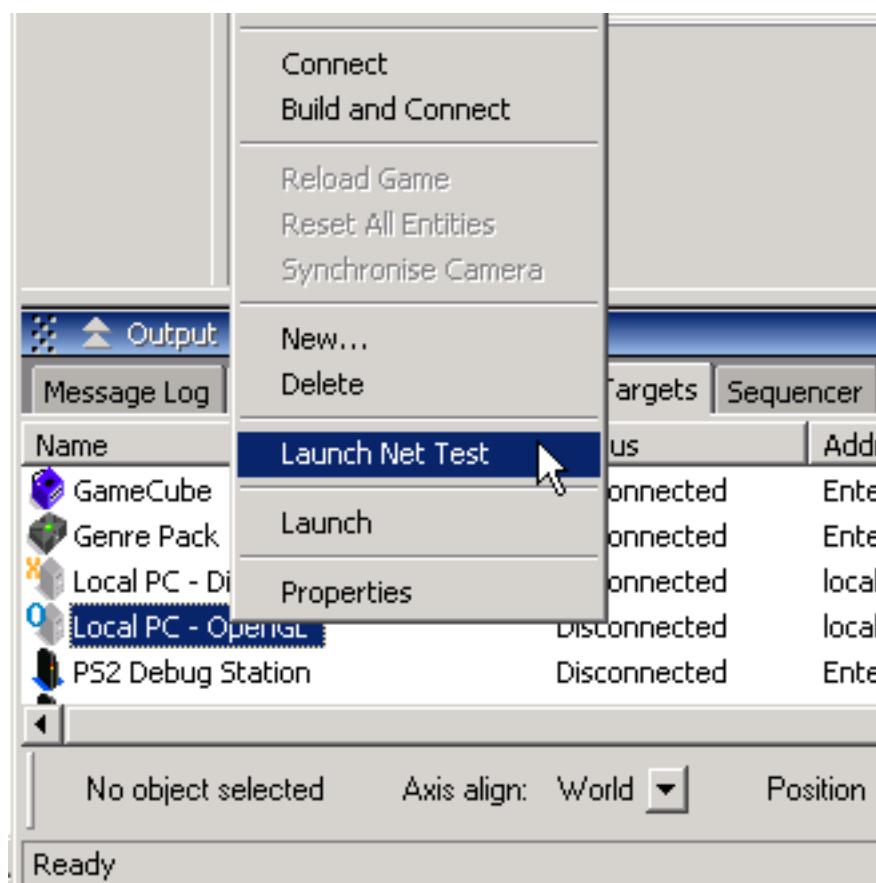
In Workspace, before connecting to a target, you can start NetTest, then connect to a target, and see NetTest describe the game data stream as it is being sent to the target. You can also start NetTest from standalone scripts.

## Listening to an IP port

To start NetTest from Workspace, before you connect to a target:

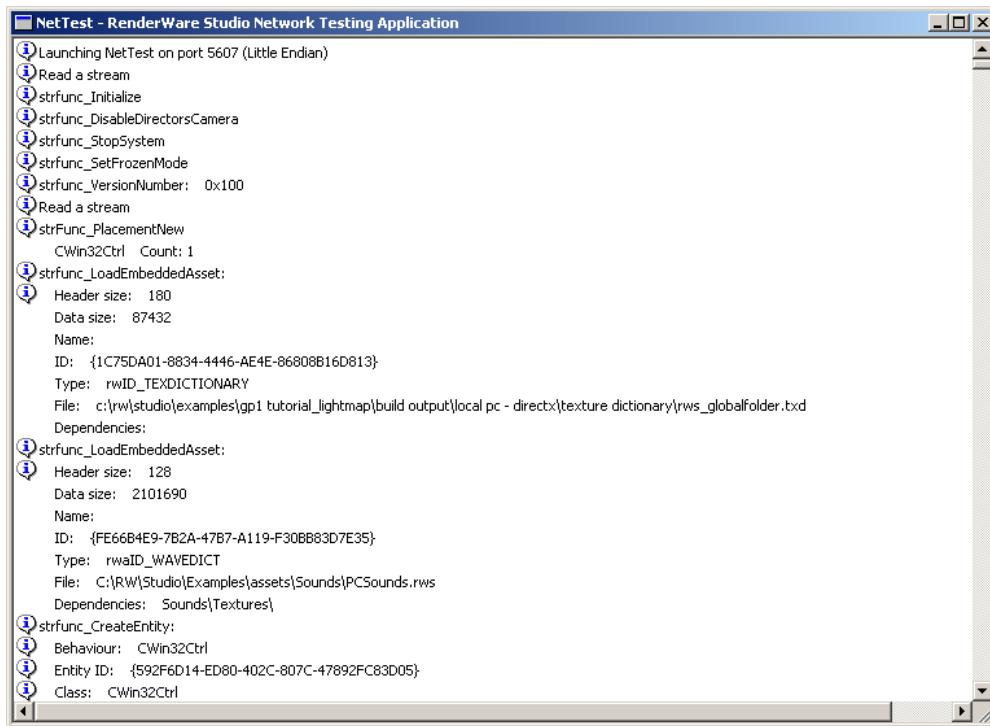
1. In the Targets window, right-click a target, and then select **Launch NetTest**.

**Note:** The NetTest window will only appear if you have edited the [connection properties](#) (p.199) so that the **Address** of the Target is *localhost*.



NetTest appears in a new floating window, and begins listening on the IP

- port specified for that target.
2. Right-click the target again, and then select **Build and Connect**.  
NetTest displays information about the stream as it is sent to the target:



To start NetTest listening without running Workspace, run the following Windows script file:

```
C:\RW\Studio\Programs\Tools\NetTest\Script\NetTest.wsf
[/port:value] [/bigEndian]
```

For example:

```
NetTest.wsf /port:5608
```

starts NetTest listening on port 5608 for a little-endian stream.

By default, this script file listens on port 5607 (the default port for the DirectX - Local PC target).

## Viewing an existing stream file

To view an existing stream file, run the following Windows script file:

```
C:\RW\Studio\Programs\Tools\NetTest\Script\ViewStream.wsf
[/stream:path] [/bigEndian] [/unicode]
```

Stream files have the file extension .stream or .rws. An .rws file contains a single asset or entity and a .stream file usually contains a number of entities.

When you build the game data stream for a target, the Game Production Manager writes stream files to a Build Output directory for that target, under the project folder:

```
C:\RW\Studio\Examples\Example Dynamic Lights\Build Output\Local PC
- OpenGL\Folder\guid.stream
```

## Analyzing NetTest data streams

The information that can be viewed for a particular entity includes:

- Assets
- Behaviors
- Attributes

Assets refer to data files, for example, textures, models, sound files. Behaviors are C++ class files that inherit functionality from parent classes, for example, CAttributeHandler. Attributes are properties of the entity that can be modified in Workspace.

## Entity count data

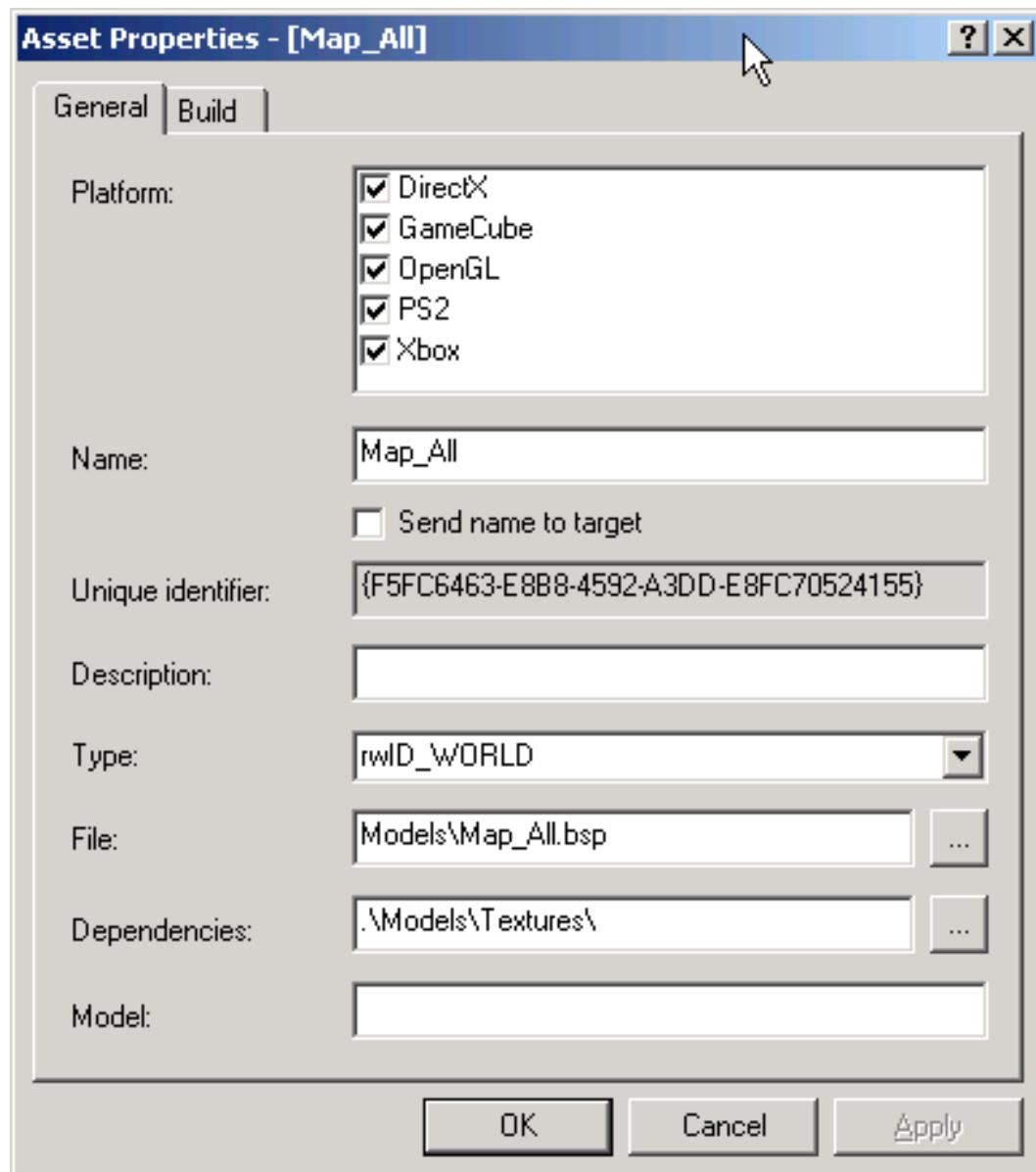
At the start of the NetTest data, is a list of entities contained within that stream file:

```
Read a file stream
strFunc_PlacementNew

FPSRender Count: 1
FPSPlayer Count: 1
CWin32Keyboard Count: 3
CFXLight_2Stage Count: 1
CFXLight_Spline Count: 1
CFXSpinLight Count: 1
CFXWaveLight Count: 1
CFXColorLight Count: 1
CDirectorsCamera Count: 1
CDebugTools Count: 1
```

This list shows entity types and the number of entities contained within the stream. As already mentioned, each entity may consist of one or more assets and behaviors. Each asset in the stream file is then listed in more detail in the following sections of the NetTest output.

**Tip:** While reading the following sections it might be useful to view a typical asset properties dialog in Workspace. To do this right-click on an asset in **Asset Window** of Workspace and select the **Properties** menu item. You will see something similar to the following figure:



## Asset data

The output produced by NetTest when examining a stream file will include information about assets:

```
strfunc_LoadEmbeddedAsset:
  Header size: 120
  Data size: 903280
  Name: Map_All
  ID: {F5FC6463-E8B8-4592-A3DD-E8FC70524155}
  Type: rwID_WORLD
  File: C:\RW\Studio\Examples\Models\Map_All.bsp
  Dependencies: .\Models\Textures\
```

### **strfunc\_LoadEmbeddedAsset**

Specifies the function that will be used to load this particular asset from the data stream. Note the asset in this case is actually embedded into the data stream. Alternatively, assets may be loaded from the target's file system. Other asset types and behaviors may use different functions to load them. Typically these loader functions begin with `strfunc_`.

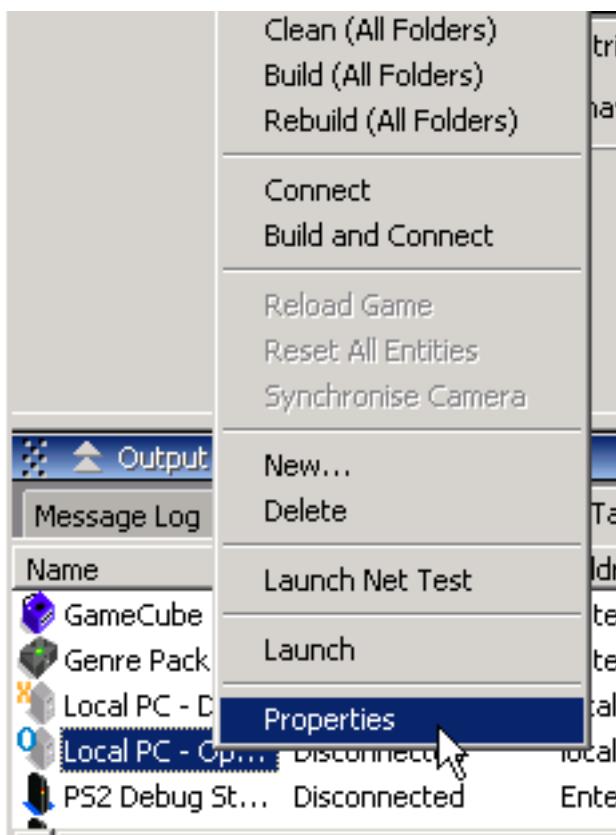
### Header size

Is the size of the instructions (strfunc commands and their data) in the stream that informs (in this case) the target to load an asset that's embedded in the stream. Other strfuncs may, for example, request an entity to be created or update certain attributes.

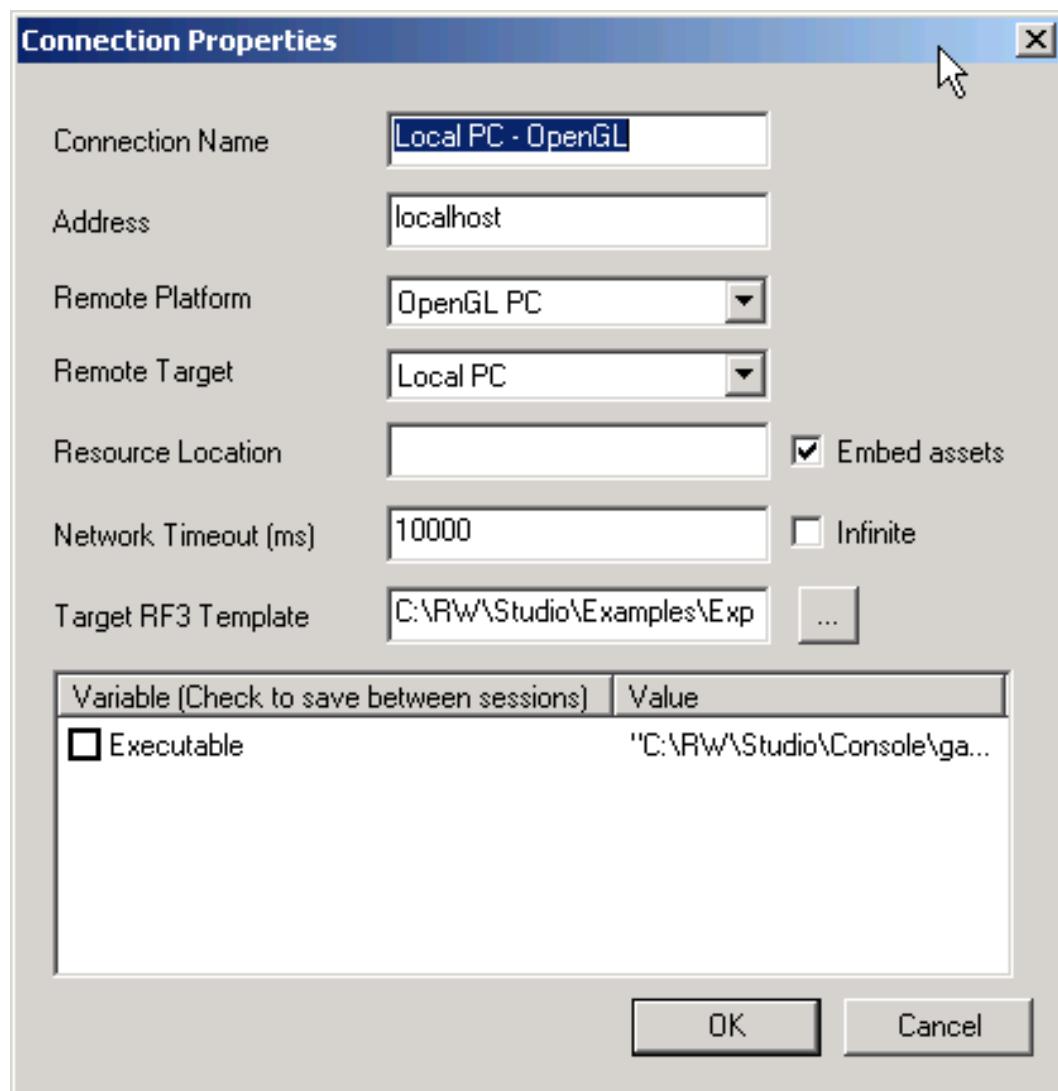
### Data size

Is the size in bytes of the asset. In this example the `bsp` file asset is embedded into the output data stream. In Workspace it is possible to deselect the "embed assets" checkbox in the connection properties dialog. In this case only the `Name` would be sent via the stream, relying on a filesystem to load the requested files.

To access the **Connection Properties** dialog, first select the target **Properties**, by right clicking on the desired target, as shown in the following figure:



This displays the **Connection Properties** dialog:



The **Embed assets** check box can be seen.

#### Name

Is the asset's name as specified in Workspace. In the asset properties dialog, there's a checkbox which allows the user to specify whether to send the name or not.

#### ID

This is the same as the **Unique Identifier**, as can be seen on the asset's properties dialog.

#### Type

This field is an identifier for assets (this can change this in the asset's properties dialog). The type string tells the resource manager in the Game Framework which function to use to load the data from the stream. The type strings for known RenderWare object types are generally filled in automatically when a RenderWare stream is imported into Workspace.

#### Dependencies

Refers to a directory where a target might locate dependent files for the asset. The resource handler code in the Game Framework has code to determine correct handling of this information.

## Entity data

The following shows data for a lighting behavior, CFXLight\_Spline:

```

strfunc_CreateEntity:
    Behaviour: CFXLight_Spline
    Entity ID: {79C1E544-7AAD-4386-AC21-10F542C27D02}
    Class: CSystemCommands
        Attach asset, ID:
{8B0D9A81-6525-4140-8B53-042BA3885238}
        Class: CFXLight_Spline
            Attribute 0: [iMsgStartSystem][69 4D 73 67 53 74
61 72 74 53 79 73 74 65 6D]
                Attribute 1: [ ] [[00 00 00 FF]
                Attribute 2: [ ] [[00 7F 00 FF]
                Attribute 3: [ ] [[7F 7F 00 FF]
                Attribute 4: [ ] [[7F 00 00 FF]
                Attribute 5: [ ] [[00 00 00 FF]
                Attribute 6: [o ] [[6F 12 03 3B]
                Attribute 7: [ ] [[00 00 00 00]
                Attribute 8: [ ] [[00 00 00 00]
                Attribute 9: [ ] [[00 00 00 00]
        Class: CFXBaseLight
            Attribute 0: [ ] [[01 00 00 00]
            Attribute 1: [ B ] [[00 00 34 42]
            Attribute 2: [ zD ] [[00 00 7A 44]
            Attribute 3: [ ] [[00 00 00 00]
        Class: CSystemCommands
            Attribute 1: [ ] [[00 00 80 3F 00 00
00 00 00 00 00 00 00 00]
                Attribute 2: [ ] [[FF FF FF FF]
                Attribute 3: [ ] [[FF FF FF FF]
                Attribute 4: [ ] [[FF FF FF FF]
        Class: CAttributeHandler
            Attribute 0: [ ] [[00 00 00 00]

Global Flag: FALSE

```

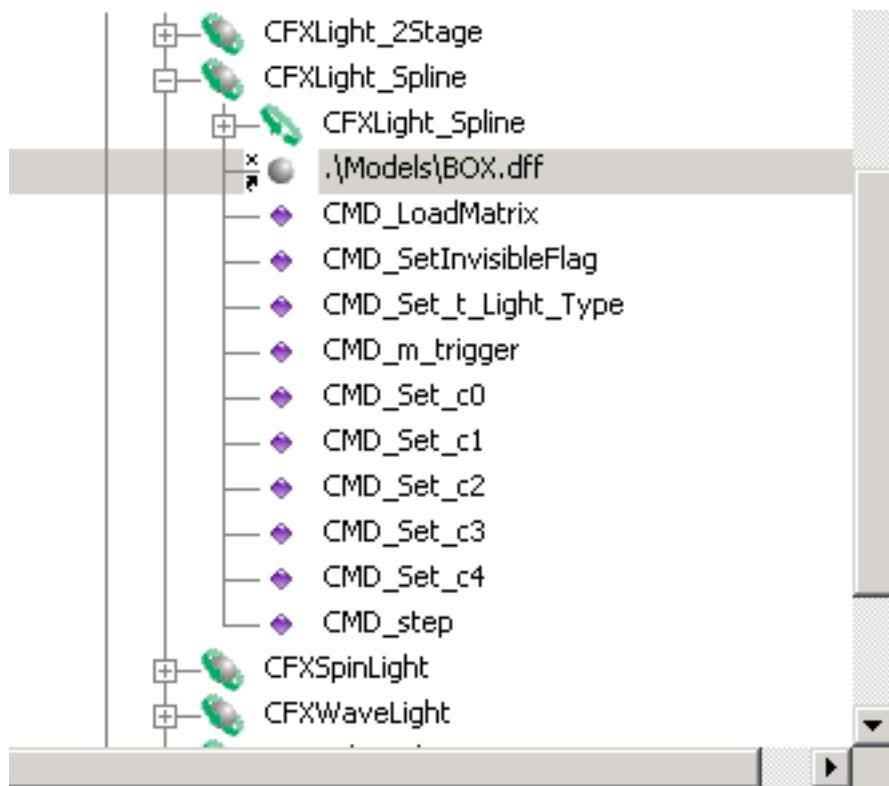
The CFXLight\_Spline entity will be created, via the strfunc\_CreateEntity function, from its constituent assets and behaviors. The required asset can be seen specified using it's unique ID, as the following snippet from the above output shows:

```

Class: CSystemCommands
    Attach asset, ID:
{8B0D9A81-6525-4140-8B53-042BA3885238}

```

The fact that CFXLight\_Spline also requires an asset can be seen in the Workspace environment. This can be seen in the following extract from the **Game Explorer** window:

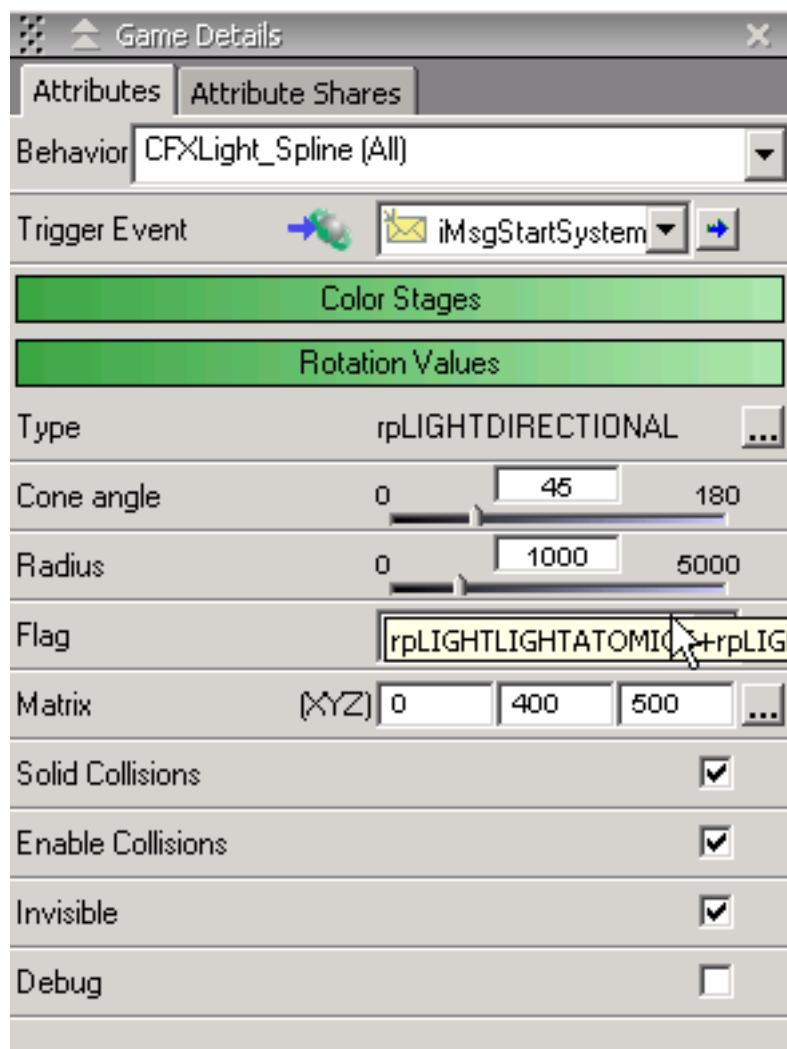


You can see that `CFXLight_Spline` uses several other classes, including:

- `CSystemCommands`
- `CAttributeHandler`
- `CFXBaseLight`

The `CSystemCommands` and `CAttributeHandler` classes are discussed in lesson 1 of the [creating behaviors tutorial](#) (p.341) and will not be discussed further here. The `CFXBaseLight` class provides `CFXLight_Spline` with basic lighting functionality which it extends.

**Note:** Each class has associated with it a number of attributes, or properties, that can be modified in the Workspace environment. The `CFXLight_Spline` class has 10 attributes that can be modified in Workspace. The following figure shows an example of this:



**Note:** Each of these attributes and its data value is also shown in the NetTest output.

## Debugging with NetTest

If the build system rules are changed this may affect what data is generated in the output stream and the ordering of data. NetTest can be used to check the results of any changes that are made to the build rules.

NetTest can also provide a starting point if a target starts to crash using new data. Checking whether it is possible to connect to the NetTest target, can help identify the source of the problem.

# Distributing your game

---

To distribute your game as a standalone application, you need:

- A [production build](#) (p.408) of the Game Framework executable file. The production build excludes the code that enables the Game Framework to receive a data stream from the Workspace. This makes the executable smaller and faster.
- The game data stream files (described below): typically—depending on how you organize your game—one for the global folder and one each for the other “root” folders (directly under the game node).
- Any other files, not specifically related to RenderWare Studio, that are required to run the game on that platform.

## Creating the game data stream files

For each root folder (that is, the folders directly under the game node):

1. In the Game Explorer window, right-click the folder, and then click **Active Folder**.
2. In the [Targets](#) (p.330) window, right-click a connection for the target platform, and then click **Build**.

This creates two platform-specific RenderWare stream files:

```
MyGame\Build Output\target name\Folder\active folder guid.stream
MyGame\Build Output\target name\Folder\global folder guid.stream
```

where:

- *MyGame* is the folder where your game database is stored (under the folder where your *MyGame.rwstudio* project file is stored).
- *target name* is the target that you right-clicked.
- *active folder guid* and *global folder guid* are the unique identifiers of these folders. (To see the unique identifier of a folder, right-click the folder in the Game Explorer window, and then click **Properties**.)

These stream files are the same data that the Workspace sends when you connect to that target. (The stream file for the global folder will be the same, regardless of the active folder.)

## Renaming the game data stream files

You need to rename these `.stream` files to match the names expected by your game code.

By default, the Game Framework expects the “boot-up” stream file—containing the data that you want to use at the start of the game—to be named `bootup.dff`.

If you want to use a different file name, or you want to change the path where the Game Framework expects to find this file, then you need to edit the startup file for the target platform.

Each startup file defines `RWS_BOOTUP_FILE`, which identifies the location of the boot-up stream file:

```
#define RWS_BOOTUP_FILE "path";
```

The startup files are stored in the `gfCore` library. The table below lists the startup file names for each platform, and the default expected location of the boot-up stream file.

Platform	Startup file name	Default location of boot-up stream file
<b>GameCube</b>	<code>gcn.cpp</code>	Current folder (the same folder as the Game Framework executable).
<b>Local PC</b>	<code>win.cpp</code>	Current folder.
<b>PlayStation 2</b>	<code>sky.cpp</code>	For a CDROM build configuration, the root folder of the CD: <code>cdrom0:\</code> Otherwise, the root folder of the ODEM host PC: <code>host0:\</code>
<b>Xbox</b>	<code>xbox.cpp</code>	<code>D:\</code> The root folder of the data region associated with the D: drive letter.

If you use the global folder, then you also need to add code to load its stream file in the startup file (similar to the existing code for the boot-up stream file).

# Distributing a PlayStation 2 game on CD

---

To distribute a PlayStation 2 game on CD:

1. [Build](#) (p.408) the Game Framework using one of the two CD-ROM build configurations:

## Design Release CDROM

Allows you to test your game on a Debugging (“Test”) Station, while continuing to use Workspace to change the game data.

(Unlike the more expensive Development Tool, the Debugging Station can read an executable from CD only.)

## Release CDROM

The “master” version for testing and final distribution.

**Caution:** Before using ProDG to compile any PlayStation 2 build, you must add the PlayStation 2 include paths to your list of include directories in Visual Studio (**Tools ▶ Options ▶ Projects ▶ VC++ Directories**). For example:

```
c:\usr\local\sce\common\include  
c:\usr\local\sce\ee\include
```

2. Copy to CD the files listed in the table below.

- In the table below, the highlighted files are required only in some situations (for details, see the Description column).
- Except where noted in the table below, you must copy the files to the path identified by `IOP_MODULEPATH_CDROM`, defined in `skyiop.cpp` (in the Game Framework `gfCore` library). By default, this path (and the other paths mentioned in the table below) is the CD root folder.
- The table below lists the files required by RenderWare Studio. Your game might require additional files (for example, your game might use other IRX files).

Files	Description
<code>iopr254.img</code>	OIP image.
<code>sio2man.irx</code>	Pad IRX files.
<code>padman.irx</code>	
<code>mtapman.irx</code>	
<code>system.cnf</code>	Boot-up configuration file.
<code>sles_000.00</code>	Game Framework executable (must match the name in the boot-up configuration file, above).
<code>bootup.dff</code>	<a href="#">Bootup stream file</a> (p.233).
<code>padding.fil</code>	Padding.
<code>inet.irx</code>	Network IRX files for connecting to the Workspace computer via the USB Ethernet adapter.
<code>netcnf.irx</code>	
<code>inetctl.irx</code>	Required only for the Design Release CDROM build.
<code>usbd.irx</code>	
<code>rwscomms.irx1</code>	To connect to the Workspace computer via the

**an986.irx<sup>1</sup>** Broadband Ethernet adapter, you will also need (in addition to the files on the left):

dev9.irx  
smap.irx

---

**dev000.cnf** Network configuration files.  
**ifc000.cnf** Required only for the Design Release CDROM build.  
**net000.cnf**  
**icon.sys**  
**sys\_net.ico**

---

**libsdl.irx** Audio IRX files.  
**sdrdrv.irx** Required only if your game uses RenderWare Audio.  
**rwa.irx<sup>2</sup>**

---

<sup>1</sup> Copy this file to the CD path identified by RWS\_COMMSPATH\_CDROM.

<sup>2</sup> Copy this file to the CD path identified by RWS\_AUDIO\_MODULEPATH\_CDROM.

---

# Loading resources with custom resource handlers

---

When writing your own custom resource handlers, it is important to make sure you always adjust the position of the stream pointer to the end of the stream. The end of the data chunk may be padded, so you need to make sure that this padding is skipped for the pointer to properly jump to the next ID. If you don't do this, then you can get stream corruptions during your level loading which can cause the framework to assert when it comes across an invalid stream chunk header.

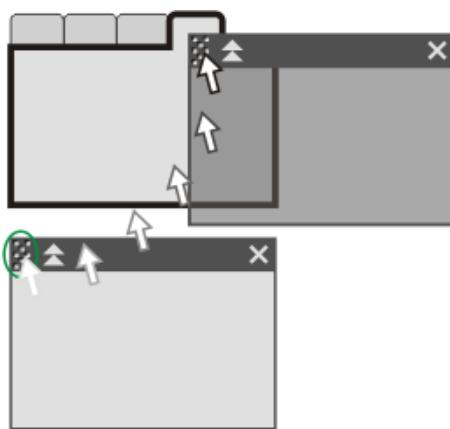
# Workspace user interface

---

The Workspace user interface consists of a set of dockable [windows](#) (p.262) that are children of the Workspace application window. The application window itself is a “docking frame” that is responsible for displaying its child windows.

## Arranging window layouts

You can arrange the Workspace child windows according to your own preferences. For example, to move a window between stacker or tabber containers, drag its tab or grabber (█). A black outline shows where the window will appear if you drop it:



If you change your mind while dragging a window, press **Esc** to return the window to its previous position. If you change your mind *after* dropping the window, simply drag the window back to its previous position.

Layout changes are saved when you close Workspace. You can also [save different layouts](#) (p.335), and switch between them while using Workspace (select **View ▶ Layouts**). To restore the supplied layouts to their original state, select **View ▶ Restore Default Layouts**.

## Showing hidden windows

If you cannot see the window you want, select **View ▶ Windows**. This shows a list of all Workspace windows, whether or not they are in the current layout. When you select a window from this list:

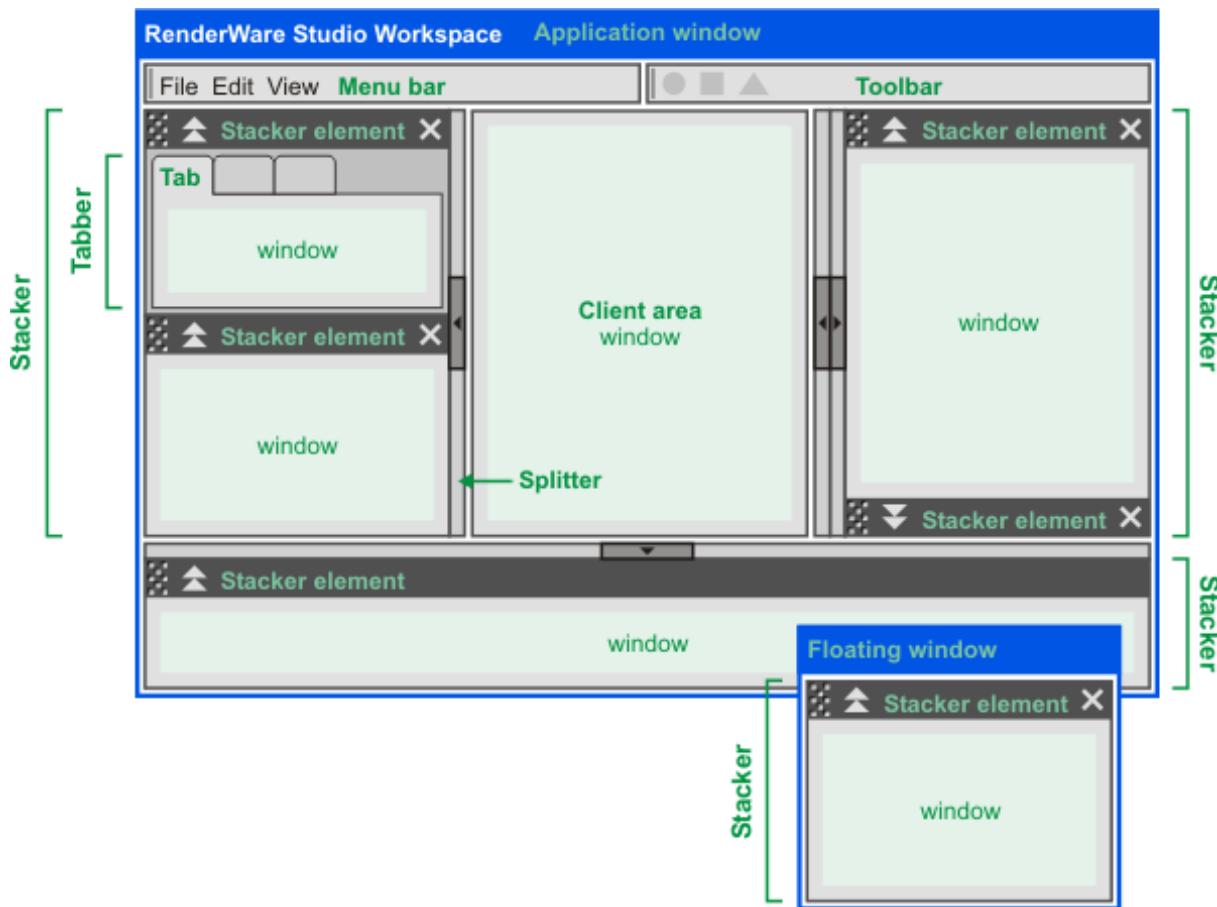
- If the window is in the current layout, then Workspace ensures that it is visible. For example, if the window is in a tab inside a collapsed stacker, then the stacker expands, and the tab appears at the front.
- If the window is not in the current layout, then the window appears in a new floating stacker element. To dock the stacker element, drag it by its grabber to a new position inside the Workspace application window.

Similarly, if you cannot see the menu or toolbar you want, select **View ▶ Toolbars**. If it is not in the current layout, the menu or toolbar that you select appears docked to the top of the application window (you cannot float menus or

toolbars).

## Elements of the Workspace user interface

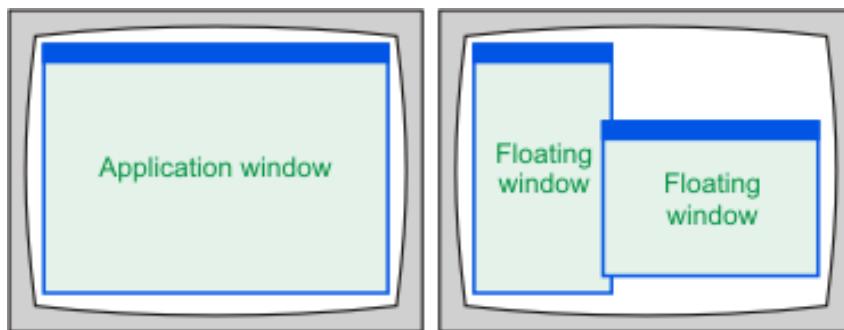
When arranging windows in Workspace, it's useful to know the elements of the Workspace user interface:



- A stacker contains one or more stacker elements in a vertical column. The diagram above shows four stackers: one on either side of the work area, one below the work area, and one in a floating window.
- A tabber contains one or more tabs.
- Tabbers appear inside a stacker element, unless the tabber is in the work area.
- The work area occupies the space left by the stackers.
- Stackers adhere ("dock") to the nearest edge of the application window.
- Stackers can appear side-by-side only when they are on the same row as the work area. Stackers above or below the work area occupy the full width of the application window.

## Arranging windows on multiple monitors

You can arrange windows across multiple monitor screens. Right-click the title bar of a stacker element, click **Float Window**, and then drag the floating window to another monitor:

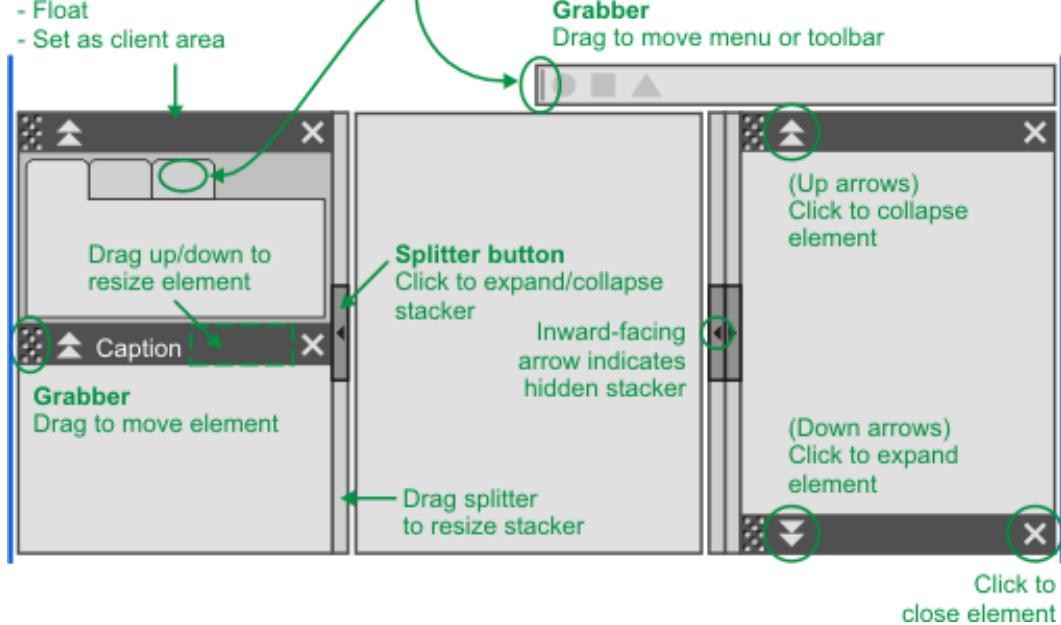


## How to arrange windows

The diagram below summarizes how to arrange windows.

Right-click title bar to:

- Rename caption
- Convert tabber to stacker
- Convert single window to a tab
- Float
- Set as client area



The table below describes each of these actions in detail.

To...	Do this...
Expand or collapse a stacker	<p>Click the splitter button next to the stacker. The stacker expands or collapses in the direction of the arrow on the splitter button. Stackers always collapse towards the edge of the application window.</p> <p>A collapsed (hidden) stacker is indicated by a splitter button whose arrow points inwards, away from the edge of the Workspace application window.</p> <p><b>Tip:</b> To find out what windows a collapsed stacker contains, move your mouse over the splitter button.</p>
Resize a stacker	Drag the splitter that is next to the stacker.
Expand or collapse a stacker element	Click the expand or collapse button in its title bar.
Change the height of a stacker element	Drag up or down in the blank area of its title bar (between the caption and the close button).
Move a stacker element between stackers	Drag the stacker element by its grabber.
Move a tab between tabbers	Drag the tab.
Move a stacker element to a tabber, as a new tab	Drag the stacker element by its grabber.
Move a tab to a stacker, as a new stacker element	Drag the tab.
Move a stacker element or a tab to a new stacker beside the work area	Drag it to a stacker that is next to the work area. Before you drop, ensure that the black outline shows a full rectangle.
Move a stacker element or a tab to a new stacker in a new row	Drag it to a splitter. If the splitter is vertical, then dropping near the top of the splitter creates a new stacker above the splitter; dropping near the bottom creates a new stacker below.
Float a stacker element	Right-click its title bar, and then click <b>Float Window</b> .
Dock a floating stacker element	Drag it, by its grabber, to the Workspace application window.
Set a stacker element as the work area	<p>Right-click the title bar of the stacker element, and then click <b>Set as client</b>.</p> <p>This moves the previous work area to a new floating window.</p>
Re-order tabs inside a tabber	Drag them.
Convert a tabber to a stacker	Right-click the title bar of the stacker element

that contains the tabber, and then click **Convert Items to Stackers**.

**There is no quick method to undo this:** to convert the stacker elements back to tabs, you need to convert one of the stacker elements to a tabber (see below), and then drag the other elements to the tabber.

---

Convert a stacker element to a tabber with a single tab	Right-click the title bar of the stacker element that contains the tabber, and then click <b>Convert to Tabs</b> .
Rename the caption of a stacker element	<ol style="list-style-type: none"><li>1. Right-click the existing caption.</li><li>2. Click <b>Rename Caption</b>.</li><li>3. Type the new caption.</li><li>4. Press <b>Enter</b>.</li></ol> <p><b>Note:</b> You cannot currently rename the caption of a tab.</p>
Close a menu or toolbar	Right-click its grabber, and then click <b>Close</b> .
Close a tab	Right-click it, and then click <b>Close</b> .

---

# Main Menu

---

## File

### [New Project](#) (p.98)

Creates a new project.

### **Open Project...**

Opens an existing project file (.rwstudio).

### **Save Project**

Saves the current project.

### **Save Project As...**

Saves the current project to a different set of XML files.

**File ▶ Save As** copies the XML files from their current location to the new location (relative to the new .rwstudio project file). On the **File** menu, click **Save As**, and enter the new file name (*NewGame*). RenderWare Studio creates the new project file and folder structure, and copies the related XML files.

### **Import**

Allows you to [import](#) (p.175) game level folders and templates from another project into your current project.

### **NXN alienbrain ▶** (p.98)

The following menu items are enabled only if you open a project that is stored in a folder managed by [alienbrain](#) (p.187).

#### **Import Project To alienbrain...**

Saves the current project to an NXN alienbrain-managed folder.

#### **Get Latest Version**

Gets from alienbrain all of the latest XML files for the project, overwriting your local copies.

#### **Submit Default Change Set**

Submits and checks in the files in the default change set.

#### **Undo Default Change Set**

Reverts all the files in the default change set to the state they were in before they were checked out.

#### **Show NXN Explorer**

Pops up the NXN database explorer window.

#### **Refresh**

Refreshes the alienbrain status indicators on icons:

- indicates that the object is in an alienbrain-managed folder, but is not checked out.

- ☒ indicates that the object is checked out by you.

- ☒ indicates that the object is checked out by someone else.
- ➔ indicates that the object is due to be imported into the game database.

---

**Recent Projects ▶**

Lists projects that you have recently opened. To open one of these projects, click its file name in the list.

---

**Exit**

Closes RenderWare Studio.

**Edit****Undo**

Undoes actions that affect game data. Actions that you can undo include:

- Adding and deleting objects in the game database (such as entities and assets).
- Changing properties.

You cannot undo:

- Changes that you have made to behavior attributes in the Attributes window. This includes parameter changes, changes to event handlers, attribute handlers and system command variables.
- Changes to the Workspace layout, such as resizing or closing a docker.

**Redo**

Redoes the action that was previously undone (see the exceptions for **Undo**, above).

**Show Undo Stack**

Displays in the Undo Stack dialog, the operations from the current session which you can undo and redo.

**Search in Game...**

Displays the search dialog allowing searches through the whole database.

**View****Windows** (p.238) ▶

Shows or hides each of the dockable windows in the current user interface layout.

**Toolbars** (p.254)

Lists the toolbars available in Workspace and toggles them on and off when each is selected.

**Layouts** (p.238) ▶

Switches between user interface layouts. Each layout consists of mostly the same windows, organized into different containers.

**Clone Current Layout**

Copies the current layout and suggests “Copy of <layout name>” as the new layout name.

**Rename Current Layout**

Renames the current layout

**Delete Current Layout**

Deletes the current layout

**Restore Current Layout**

Restore the original settings of the layout is one of the original layouts supplied.

**Restore to Default Layouts**

Deletes all customized layout settings files leaving only the original layouts.

**Refresh Event View** (p.284)

Redraws the flowchart displayed in the Event Map window.

**Show all entities in the active folder**

Shows entities in the active folder. Shown entities are noted as *Visible* in the Visibility column of the Game Explorer window.

**Hide all entities in the active folder**

Hides entities from view in the active folder. Hidden entities are noted as *Hidden* in the Visibility column of the Game Explorer window.

**Selection****Flight Camera** (p.111)

Fly the camera by pointing in the view, and then pressing and holding down a mouse button. This camera mode has separate controls for the perspective view and the orthographic views.

The following **...Camera** (p.108) menu items switch between the modes for controlling the camera in the Design View window:

**Orbit Camera**

Orbit the camera around a point by dragging.

**Pan Camera**

Pan the camera by dragging.

**Zoom Camera**

Zoom (dolly) the camera in and out by dragging.

**Pick only** (p.123)

Allows you to **select** (p.121) entities by clicking them.

**Pick and Move** (p.123)

Allows you to **select** (p.121) entities by clicking and then moving them.

**Pick and Rotate** (p.123)

Allows you to **select** (p.121) entities by clicking and then rotating them.

**Pick and Scale** (p.123)

Allows you to **select** (p.121) entities by clicking and then scaling them.

**Pick and Drag** (p.123)

Allows you to **select** (p.121) entities by clicking and then dragging them.

**Pick Materials** (p.325)

Allows you to select the materials that form the texture of an asset, and see their properties in the stream viewer.

**Lock Selection**

Toggles between the selection lock being on and off when you select an entity in pick mode in the **Design View window** (p.276). When the lock is on, it stops you selecting other entities if you click on them by accident.

**Orbit Only**

Camera orbits a selection where an object is selected in pick mode, otherwise it orbits the world origin.

**Orbit Objects**

Camera orbits a selection where an object or objects are selected in pick mode, otherwise it orbits in camera arc mode.

### **Camera Arc**

The camera tilts and turns around its own axis.

### **Toggle Pick/Camera**

Toggles between the last Pick and Camera menu item clicked

### **Next Mode**

Cycles forward to the next camera mode.

### **Previous Mode**

Cycles back to the previous camera mode.

### **Locate**

Moves the camera to show a close-up of the selected entity.

### **Aim At**

Moves the camera to aim at the selected entity but does not show close-up of entity.

## **Target**

### **All Targets ►**

### **Reload Game**

Restarts the Game Framework on the target and resends all assets and entities.

### **Reset All Entities**

Resends entities to the Game Framework. This is a quicker way to reset a game when assets have not changed.

### **Director's Camera**

Overrides any camera control behaviors in the Game Framework. Instead, the console view follows the movements of the camera in Workspace.

### **Pause Mode**

Pauses the Game Framework on the target.

### **Clean**

Deletes the stream files and other intermediate files created by the Game Production Manager (in the `Build Output\connection name\...` folders under your project).

### **Build**

Updates any out-of-date intermediate files and then (if there were any updates), creates new stream files (`Build Output\connection name\Folder\*.stream`) for the global folder and the active folder.

### **Rebuild**

Cleans, then builds.

### **Clean (All Folders)**

**Note:** The following menu options will only work *after* one of the target consoles have been **Set as Active** in the context menu of the **Targets** (p.330) window.

Cleans all folders in the project, not just the global folder and the active folder.

### **Build (All Folders)**

Builds all folders in the project, not just the global folder and the active folder.

### **Rebuild (All Folders)**

Rebuilds all folders in the project, not just the global folder and the active folder.

### **Connect**

Connects to the target using the last built Renderware stream.

### **Build and Connect**

Builds, then sends the stream files for the global folder and active folder to the target. Any subsequent changes are also sent to the target, until you disconnect.

**Tip:** Double-clicking the word “Connected” or “Disconnected” in the **Status** column has the same effect as selecting this option.

### **Disconnect**

Stops sending game data to the target.

### **Reload Game**

Stops and restarts the game, and then resends the game data stream.  
(Equivalent to disconnecting, and then connecting.)

### **Reset All Entities**

Sends all entities and their attributes, but without sending their assets. This is a quick method of restarting the game if no assets have changed.

### **Synchronize camera** (only enabled for connected targets)

Ensures that the camera position shown on the target matches the position of the camera in the RenderWare Studio Workspace.

### **Launch**

Starts the console emulator application.

### **Properties**

Sets the target [properties](#) (p.199).

**Tip:** Double-clicking in the **Target** column has the same effect as selecting this option.

## **Options**

### [\*\*Flight\*\*](#) (p.115)

Sets how finely the camera responds to mouse and keyboard movement, and the minimum frame rate for camera motion.

### [\*\*Flight Keys\*\*](#) (p.117)

Sets the keys used for keyboard flight.

### [\*\*Drag\*\*](#) (p.162)

Sets various options for dragging entities, such as whether entities snap to regular intervals when you move or rotate them.

### [\*\*Display\*\*](#) (p.160)

Sets the display properties of the Design View window (such as solid or wireframe rendering).

**Workspace** (p.158)

Selects whether or not Workspace remembers various user interface settings, such as how you have arranged the dockable windows. Also sets whether or not Workspace automatically opens the last project when it starts.

**Light map** (p.294)

Allows you to adjust various parameters that affect how light map previews will appear in the Design View window.

**Tools****Parse All Source**

Parses all source for the current game.

**Validate Database...** (p.193)

Displays a dialog to select which parts of the database to validate. You can then correct any problems.

**Help****Workspace Help**

Displays the Workspace Help.

**About Workspace**

Displays information about RenderWare Studio, such as the installed version.

# Toolbars

---

## Standard

Button	Description	Equivalent menu option
	<b>New Project</b> Creates a new project and folder structure.	File ▶ New Project...
	<b>Open Project</b> Opens an existing project file (.rwstudio).	File ▶ Open...
	<b>Save Project</b> Saves the current project.	File ▶ Save
	<b>Undo</b> Undoes actions that affect game data. Actions that you can undo include: <ul style="list-style-type: none"> <li>Adding and deleting objects in the game database (such as entities and assets).</li> <li>Changing properties.</li> </ul> You cannot undo: <ul style="list-style-type: none"> <li>Changes that you have made to behavior attributes in the Attributes window. This includes parameter changes, changes to event handlers, attribute handlers and system command variables.</li> <li>Changes to the Workspace layout, such as resizing or closing a docker.</li> </ul>	Edit ▶ Undo
	<b>Redo</b> Redoes the action that was previously undone (see the exceptions for <b>Undo</b> , above).	Edit ▶ Redo
	<b>Show Undo Stack</b> Displays the operations from the current session that you can undo and redo in the Undo Stack dialog.	Edit ▶ Show Undo Stack
	<b>Parse All Source</b> Parses all source for the current game.	Tools ▶ Parse All Source
	<b>Refresh Event View</b> Refreshes the event view.	View ▶ Refresh Event View

## Selection

Button	Description	Equivalent menu option
	<p><b>Flight</b> Fly the camera by pointing in the view, and then pressing and holding down a mouse button. This camera mode has separate controls for the perspective view and the orthographic views.</p>	<b>Selection ▶ Flight</b>
	<p><b>Orbit</b> Orbit the camera around a point by dragging.</p>	<b>Selection ▶ Orbit</b>
	<p><b>Pan</b> Pan the camera by dragging.</p>	<b>Selection ▶ Pan</b>
	<p><b>Zoom</b> Zoom (dolly) the camera in and out by dragging.</p>	<b>Selection ▶ Zoom</b>
	<p><b>Pick only</b> Allows you to <a href="#">select</a> (p.121) entities by clicking them in the Design View window. This is known as “pick” mode.</p>	<b>Selection ▶ Pick only</b>
	<p><b>Pick and Move</b> Allows you to <a href="#">select</a> (p.121) entities by clicking them in the Design View window. This is known as “pick” mode.  After selecting an entity, you can <a href="#">move</a> (p.123) the entity.</p>	<b>Selection ▶ Pick and Move</b>
	<p><b>Pick and Rotate</b> Allows you to <a href="#">select</a> (p.121) entities by clicking them in the Design View window. This is known as “pick” mode.  After selecting an entity, you can <a href="#">rotate</a> (p.123) the entity.</p>	<b>Selection ▶ Pick and Rotate</b>
	<p><b>Pick and Scale</b> Allows you to <a href="#">select</a> (p.121) entities by clicking them in the Design View window. This is known as “pick” mode.  After selecting an entity, you can <a href="#">scale</a> (p.123) the entity.</p>	<b>Selection ▶ Pick and Scale</b>
	<p><b>Pick and Drag</b> Allows you to select an object and then drag it within the Design View window, as you would when dragging a new object into the scene in the Design View window for the first time.</p>	<b>Selection ▶ Pick and Drag</b>
	<p><b>Pick Materials</b> Allows you to select a material in the Design View window. All assets using the same material are highlighted yellow in the Design view the material object is highlighted automatically in the <a href="#">Stream Viewer</a> (p.325) window.</p>	<b>Selection ▶ Pick Materials</b>
	<p><b>Selection Lock</b> Toggles <a href="#">selection locking</a> (p.121).</p>	<b>Selection ▶ Selection Lock</b>
	<p><b>Orbit only</b> Camera orbits a selection where an object is selected in pick mode, otherwise it orbits the world origin.</p>	<b>Selection ▶ Orbit only</b>
	<p><b>Orbit objects</b> Camera orbits a selection where an object or objects are</p>	<b>Selection ▶ Orbit objects</b>

selected in pick mode, otherwise it orbits in camera arc mode.

	<b>Camera arc</b> The camera tilts and turns around its own axis.	<b>Selection ► Camera Arc</b>
---	--	-------------------------------

## Active Target

**Note:** The following toolbar buttons are enabled only *after* one of the target consoles have been **Set as Active** in the context menu of the **Targets** (p.330) window.

Button	Description	Equivalent menu option
	<b>Clean</b> Deletes the stream files and other intermediate files created by the Game Production Manager (in the <code>Build Output\connection name\...</code> folders under your project).	<b>Target ► Clean</b>
	<b>Build</b> Updates any out-of-date intermediate files and then (if there were any updates), creates new stream files ( <code>(Build Output\connection name\Folder\*.stream)</code> for the global folder and the active folder).	<b>Target ► Build</b>
	<b>Rebuild</b> Cleans, then builds.	<b>Target ► Rebuild</b>
	<b>Connect</b> Connects to the target using the last built Renderware stream.	<b>Target ► Connect</b>
	<b>Build and connect</b> Builds, then sends the stream files for the global folder and active folder to the target. Any subsequent changes are also sent to the target, until you disconnect.	<b>Target ► Build and connect</b>
	<b>Launch</b> Starts the console emulator application.	<b>Target ► Launch</b>
	<b>Synchronize Camera</b> Ensures that the camera position shown on the target matches the position of the camera in the RenderWare Studio Workspace.	<b>Target ► Synchronize Camera</b>
	<b>Reset All Entities</b> Resets all entities in the game on the target.	<b>Target ► All entities ► Reset Game</b>
	<b>Director's Camera</b> Overrides any camera control behaviors in the Game Framework. Instead, the console view follows the movements of the camera in Workspace.	<b>Target ► All entities ► Director's Camera</b>
	<b>Pause Game</b> Pauses the Game Framework on the target.	<b>Target ► All entities ► Pause Mode</b>

## Light Map

Button	Description
	<b>Light Map Preview</b> Calculates lighting according to the settings in the Light Map Preview window. The results of applying the lighting calculations to the scene are displayed in Design View.
	<b>Light Map Generate</b> Calculates lighting according to the scene settings for density and area, rather than the settings in the Light Map Preview window.
	<b>Light Map Settings</b> Displays the Light Map Preview window.

## Preview window

Button	Description
	<b>Play animation</b> Plays an animation in the <a href="#">Preview window</a> (p.300) as it will appear in the game.
	<b>Pause animation</b> Pauses the animation that is playing in the Preview window.
	<b>Stop animation</b> Stops the animation that is playing in the Preview window.
	<b>Repeat animation</b> Sets the animation in the Preview window to be played on a continuous loop until the stop button is pressed or the Preview window is closed.
	<b>Display skeleton</b> Displays the clump's skeleton and bone axes in the Preview window.
	<b>Display bone axes</b> Displays the constituent bones and joints of an asset's skeleton and the orientation of those joints at any given point in an animated sequence.
	<b>Show object</b> Hides the clump so that only its skeleton and axes are visible.
	<b>Change background color</b> Changes the background color that you are previewing a clump against.
	<b>Reset object</b> Resets the clump to the original size and orientation it was in when it was first dragged to the Preview window

from the Assets window.



#### **Display object statistics**

Displays the number of atomics, vertices, triangles, and bones that make up the clump and the duration (in seconds) of the animated sequence.



#### **Render world**

Toggles rendering of the world asset file.



#### **Render world sectors**

Toggles rendering of the sectors of the world asset file.



#### **Render world in wireframe**

Toggles rendering of the world asset file in wireframe mode.



#### **Render world vertex normals**

Toggles rendering of the world asset file's vertex normals.



#### **Next sector**

Allows you to cycle forward through the sectors of the world asset file.



#### **Previous sector**

Allows you to cycle backwards through the sectors of the world asset file.



#### **View all world sectors**

Stops viewing of world sectors one at a time and reverts to viewing all world sectors.



#### **Enable Z-testing**

Toggles between Z-testing enabled and disabled - displays all wireframes, sectors and vector normals overlaid over the world asset.

## Spline

Button	Description
	<b>New nodes</b> Creates a new node at the point where the user clicks in the workspace and completes the segment between that node and the previously inserted node.
	<b>Insert node</b> Inserts a node exactly half way between the two currently selected nodes.
	<b>Delete node</b> Deletes the currently selected node.
	<b>Open node</b> Opens the currently selected closed spline asset allowing further nodes to be inserted, thereby extending the loop.
	<b>Close node</b> Closes a set of spline control points, thereby making a closed spline asset.

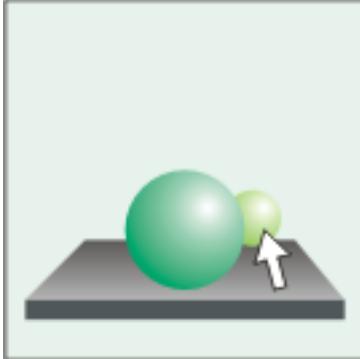
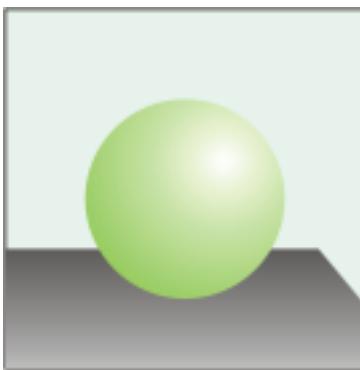
## Volume

Button	Description
	<b>Create axis-aligned box volume</b> Defines a box volume whose edges are aligned with the x-, y-, and z-axes of the game area.
	<b>Create box volume</b> Defines a box volume whose edges are parallel with and orthogonal to the current viewing direction.
	<b>Center-to-corner</b> Specifies that when creating a box volume, the original point clicked should be in the center of the bottom (or top) face.
	<b>Corner-to-corner</b> Specifies that when creating a box volume, the original point clicked should be in one corner of the bottom (or top) face.
	<b>Texture selection</b> Displays a drop-down list allowing selection of a texture for the surface of the box created by the other tools in this toolbar.

# Keyboard shortcuts

---

## General

Key	Action
<b>F1</b>	<p>Displays Help.</p> <p>Equivalent to clicking <b>Workspace Help</b> on the <b>Help</b> menu.</p>
<b>F3</b>	<p>Moves the camera in the <a href="#">perspective view</a> (p.276) to show a close-up of the selected entity.</p> <p>Either:</p> <ul style="list-style-type: none"> <li>Click an entity name in the Game Explorer window, and then press <b>F3</b> to see a close-up of that entity in the perspective view.</li> </ul> <p>or</p> <ol style="list-style-type: none"> <li>Pick an entity that you can already see in the perspective view.</li> </ol>  <ol style="list-style-type: none"> <li>Press <b>F3</b> to see a close-up.</li> </ol>  <p>This is equivalent to right-clicking an entity in the Game Explorer window, and then clicking <b>Locate in view</b>.</p>
<b>Shift+F3</b>	<p>This is similar behavior to that of “Locate in view” without moving the camera in to show a close up of the entity selected, the camera is simply turned so that the entity is now in the center of its view but the distance to the entity remains the same.</p>

---

This is equivalent to right-clicking an entity in the Game Explorer window, and then clicking **Aim at in view**.

---

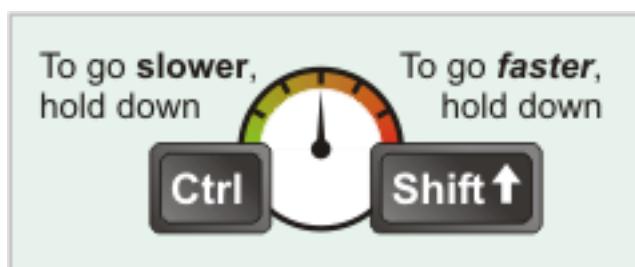
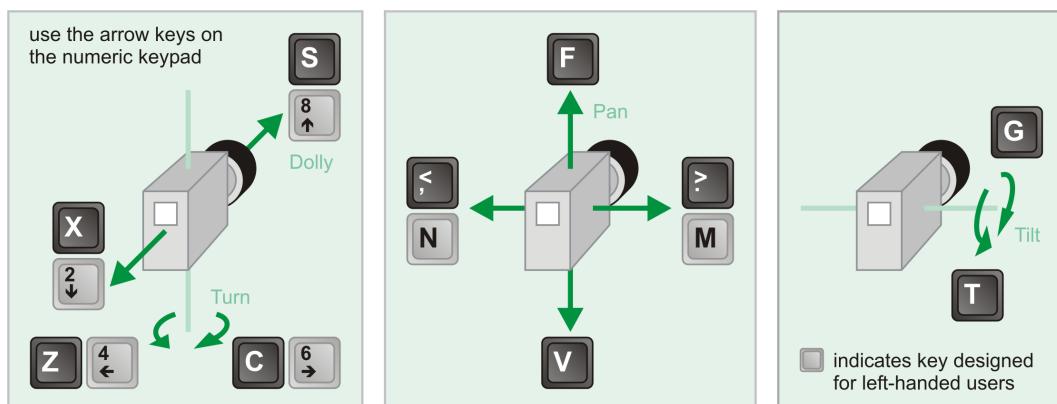
<b>F9</b>	Toggles between <a href="#">pick</a> (p.123) and <a href="#">camera</a> (p.108) modes.
<b>Esc</b>	Deselects any selected entities.
<b>Space bar</b>	Toggles between the selection lock being on and off when you <a href="#">select</a> (p.121) an entity in pick mode. When the lock is on, it stops you selecting other entities if you click on them by accident.

---

## Target

Key	Action
<b>Ctrl+D</b>	Toggles the Director's Camera on or off.  Selecting the Director's Camera overrides any camera control behaviors in the Game Framework. Instead, the console view follows the movements of the camera in Workspace. For example, in a “first person” game, the camera typically follows the movements of the player. If you select the Director's Camera, then, although you can still use your console's control pad to move the player, the console view moves with the Workspace camera, not the player.
<b>Ctrl+P</b>	Pauses the Game Framework, or continues running it after pausing.

## Controlling the camera in the perspective view



**Note:** The keys for left-handed users and right-handed users are all active at the same time.

Key	Action	
Left-handed	Right-handed	
S	↑	Dollies the camera into the view (moving the camera forwards, level with the "ground").
X	↓	Dollies the camera out of the view (moving the camera backwards).
Z	←	Turns the camera left (around its Y axis).
C	→	Turns the camera right.
F		Pans the camera upwards (along its Y axis).
V		Pans the camera downwards.
>	D	Pans the camera right (along its X axis).
<	A	Pans the camera left.
T		Tilts the camera up (around its X axis).
G		Tilts the camera down.
Shift		Increases camera flight speed: <ul style="list-style-type: none"> <li>Press and hold down <b>Shift</b> while pressing one of the flight control keys (listed above).</li> </ul>
Ctrl		Decreases camera flight speed: <ul style="list-style-type: none"> <li>Press and hold down <b>Ctrl</b> while pressing one of the flight control keys (listed above).</li> </ul>

## Picking entities

Key	Action
Q	Switches to <a href="#">Pick only</a> (p.123) mode.
W	Switches to Pick and move mode.
E	Switches to Pick and rotate mode.
R	Switches to Pick and scale mode.

## Event Map

Key	Action
F5	Refreshes the <a href="#">Event Map</a> (p.284) flowchart. Equivalent to clicking <b>Refresh Event View</b> on the <b>View</b> menu.
W	Scrolls the flowchart upwards.
S	Scrolls the flowchart downwards.
A	Scrolls the flowchart left.
D	Scrolls the flowchart right.

**+** Zooms in on the flowchart.

(see note)

**-** Zooms out of the flowchart.

(see note)

**Note:** Use the **+** and **-** keys on the numeric keypad, not the keys on the main keyboard.

## Game Explorer window

Key	Action
<b>F2</b>	Allows you to rename the selected object.
<b>↑</b>	Selects the object above the current object.
<b>↓</b>	Selects the object below the current object.
<b>←</b>	Contracts the current object if it is expanded, otherwise selects its parent object.  <b>Tip:</b> Use the <b>←</b> key to traverse upwards through the hierarchy, contracting objects as you go.
<b>→</b>	Expands the current object, if it has children.
<b>Shift+click</b>	Selects consecutive objects:  1. Click the first object. 2. Press and hold down <b>Shift</b> . 3. Click the last object.  You can select multiple objects only of the same type, and with the same parent.
<b>Ctrl+click</b>	Selects objects that are not consecutive:  1. Press and hold down <b>Ctrl</b> . 2. Click each object.  You can select multiple objects only of the same type, and with the same parent.
<b>Ctrl+drag</b>	Copies the selected objects to another location in the game database.  (Dragging without holding down <b>Ctrl</b> moves the selected objects.)
<b>Alt+ ↑</b> or <b>Alt+ ↓</b>	Promotes or demotes the position of the current object in the game data stream, relative to its siblings.  The Game Explorer window shows objects in the order in which they are sent to the target in the game data stream. You need to ensure that objects are sent to the target in the order that your game requires them.
<b>Alt+drag</b>	Creates a reference to the selected object in the drop destination of the Game Explorer window.

## Design View window

Key	Action
+	Increases the size of the <a href="#">drag axes</a> (p.123).
-	Decreases the size of the drag axes.
Space bar	Toggles <a href="#">selection locking</a> (p.121).

# Windows

---

## Use of the term “window”

In Workspace, you can switch between user interface layouts that present different arrangements of (mostly) the same elements. What appears in one layout as a tab might appear in another layout as a separate window. This documentation refers to all such user interface elements as “windows”.

Below is a list of the windows supplied with Workspace. Your organization may have [customized Workspace](#) (p.423) so that it contains additional windows not listed here. To see a complete list of Workspace windows available to you, select **View ▶ Windows**.

### [Assets](#) (p.264)

Lists the assets that you have imported into your project. You can use these assets to create entities.

### [Asset Light Map](#) (p.267)

Allows the type of lighting to be specified for an asset. Can also be used to specify whether the asset casts shadows and whether or not to use the Game-Database Matrix.

### [Attribute Shares](#) (p.269)

Lists the attribute shares that you created in the Game Explorer window.

### [Attributes](#) (p.270)

Allows you to edit the values of the behavior attributes for the selected entity. Also allows you to send messages from the entity, for testing purposes.

### [Behaviors](#) (p.273)

Lists the behaviors that you can attach to an entity. The behaviors are organized in folders by category.

### [Design View](#) (p.276)

Displays four camera views of the current game: one perspective view and three orthographic views (front, plan and side).

### [Event Map](#) (p.284)

Displays a flowchart of events and the entities that transmit or receive them.

### [Events](#) (p.286)

Lists events sent and received by entities in the active folder.

### [Game Explorer](#) (p.288)

Lists the game database hierarchy, and the objects in the game database.

### [Help](#) (p.291)

Provides context-sensitive help for behaviors.

### [Light Attributes](#) (p.292)

Displays information about a light entity. Allows properties such as the color, radius and type of light to be modified.

### [Log windows](#) (p.297)

#### Message Log

Displays messages about activities in the current RenderWare Studio session.

**Version Control**

Displays NXN alienbrain version control information.

**Build Log**

Displays build progress messages from the Game Production Manager.

**Object Information** (p.299)

Displays information about the currently selected object. This information changes dynamically depending on the pick mode selected from the selection toolbar.

**Preview** (p.300)

Allows you to preview RenderWare clumps, atomics, animations and worlds in close-up without adding them as entities to the Design View.

**RF3 Asset Templates Window** (p.309)

Allows you to add and edit RenderWare asset exporter templates.

**RF3 Project Templates Window** (p.311)

Allows you to add and edit RenderWare Studio project exporter templates.

**Scene Light Map** (p.313)

Allows use of light mapping to be selected for a world scene. Various light mapping parameters can also be modified in this panel, along with the characteristics of area lights.

**Search Results** (p.315)

Lists the objects you have searched for in the Game Explorer, Assets, or Behaviors windows.

**Sequencer** (p.317)

Allows the construction and configuration of predefined sets of actions that always take place in the same order. In a sequence, the attributes of existing entities may be modified together over time, and new entities may be created and controlled. Displays a timeline with keyframes and interpolations.

**Stream Viewer** (p.325)

Provides a way to look inside RenderWare streams, and attach custom editors to handle the different types of data, such as MatFx and RpToon data, that have been attached as plug-ins to assets.

**Target Files** (p.329)

Lists files that the target has “saved” (sent via the network) to Workspace.

**Targets** (p.330)

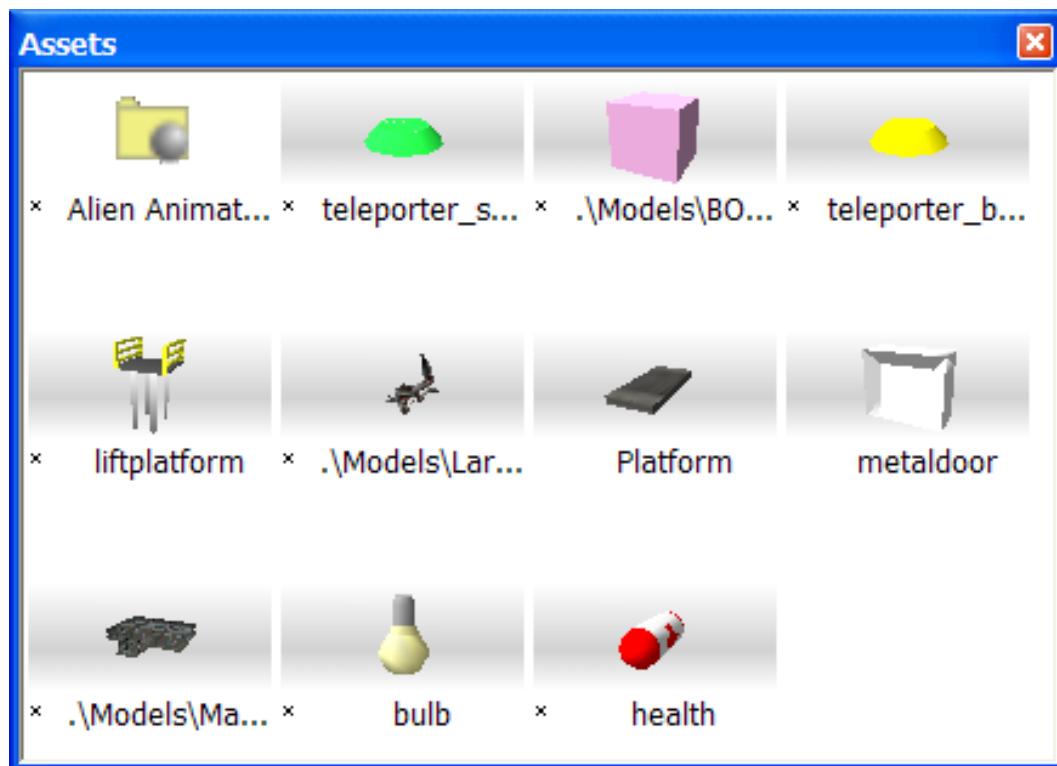
Lists the targets (consoles or PCs) on which you can run your game. You can use this window to build the game data stream for a target, then connect (send the game data stream) to the target, so that you can dynamically view the game as you develop it.

**Templates** (p.334)

Lists the templates that you have created in the Game Explorer window.

## Assets window

Lists the assets that you have imported into your project. You can use these assets to create entities.



A small **x** indicates that the asset or asset folder is used in the project.

## Organizing assets into asset folders

You can organize your assets into asset folders, so that you can add a set of assets to an entity or folder in a single step, rather than one-by-one. To create an asset folder, right-click an empty area in the window (or an existing asset folder), and then click **New ▶ Asset Folder** (or **Asset Folder as Child**).

To move an existing asset into a folder, simply drag it. To create a new copy of an asset in an asset folder, drag while pressing **Ctrl**.

## Context menu

To view the actions that you can perform on an object in the Assets window, right-click an object. The following context menu appears:

### **View ▶**

Sets how the window represents the assets:

#### **Small icons**

Lists asset names (with a small 2D icon) across, then down the available space in the window. (Compare with **List**.)

**Large thumbnails** or **Small thumbnails** (shown above)

Displays asset names under a large or small 3D view of the primary asset file.

(For assets whose primary asset file cannot be represented in 3D, a default icon appears instead.)

**Tip:** If your project contains many assets, then it can take a long time to load and refresh the thumbnails. To reduce load times, use a different view option.

#### Detail

Lists asset names and other properties under column headings.

#### List

Lists asset names (with a small 2D icon) down, then across the available space in the window. (Compare with **Small icons**.)

#### Tree

Shows the hierarchy of asset folders and assets. This is especially useful for viewing [RF3-based assets](#) (p.104).

Lists asset names and other properties under column headings.

---

### New ►

(This option appears only if you right-click in an empty area of the window.)

#### Asset Folder

Creates a new asset folder.

#### Asset

Creates a new asset. Displays a blank [properties](#) (p.101) dialog for the new asset.

**Tip:** Alternatively, you can create a new asset by simply dragging a primary asset file (such as a .dff) from Windows Explorer to the Assets window.

#### Spline

Creates a new spline asset. Displays a dialog where you specify the path to store the spline asset file (.spl).

---

### Show Asset Stream Contents

Displays the contents of the [asset stream](#) (p.325).

---

#### Cut

Cuts the object to the Clipboard.

#### Copy

Copies the object to the Clipboard.

#### Paste

Pastes the object from the Clipboard.

---

#### Move Up

(Only for objects with previous siblings.) Moves the object above its previous sibling.

#### Move Down

(Only for objects with following siblings.) Moves the object below its following sibling.

---

**Delete**

[Deletes](#) (p.177) the object from the game database.

**Note:** Deleting an asset from the game database does not delete the associated primary asset file or textures from your hard drive.

**Rename**

Renames the object.

**Activate Preview**

(Asset in thumbnail view only.) Allows you to rotate and zoom the thumbnail.

To rotate, drag the thumbnail.

To zoom, hold down both mouse buttons while dragging up or down.

**Reload**

(Assets only.) Reloads the data from the primary asset file.

**View XML**

Views the object as XML code in a browser window.

**Search ▶**

(Assets only.) Displays several options for identifying which entities use this asset (or type of asset). The results are displayed in the [Search Results](#) (p.315) window.

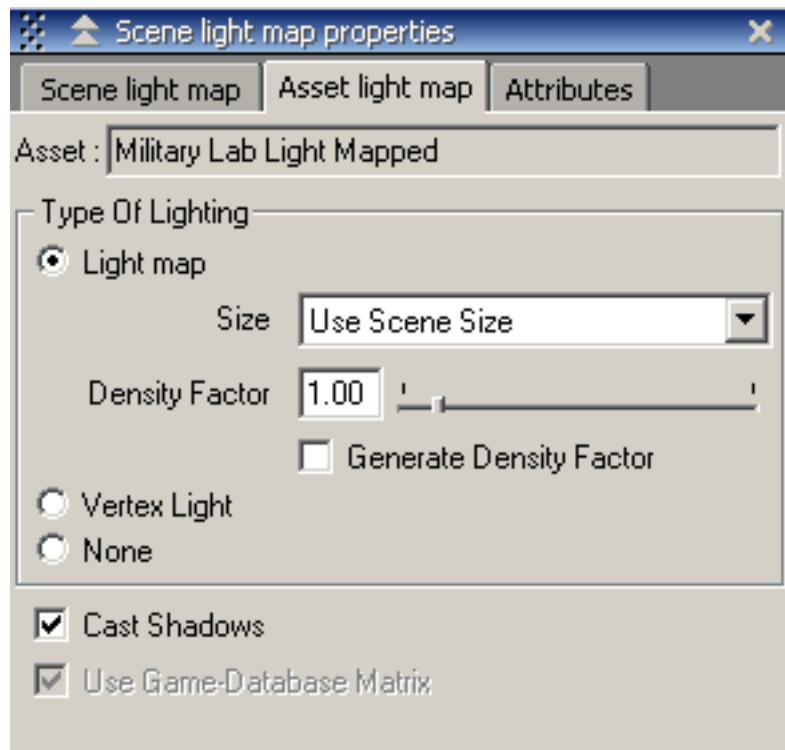
**Properties**

(Assets only.) Sets the asset [properties](#) (p.101).

# Asset light map window

---

Allows the type of lighting to be specified for an asset. Can also be used to specify whether the asset casts shadows and whether or not to use the Game-Database Matrix.



## Asset

This shows the asset file name.

## Type of lighting

Select asset lighting type.

### Light map

Enables light mapped lighting. **Size**, **Density Factor**, and **Generate Density Factor** can then be selected or modified as required.

### Size

Size of light map texture created in pixels. Light maps are always square.

### Density factor

Refers to texel density, this is the average number of pixels in a light map that are applied to a polygon. The higher the density the greater the detail of the light map.

### Generate Density Factor

Specifies that the density factor is to be calculated automatically.

### Vertex light

Selects vertex lighting, as opposed to light mapped lighting.

### None

Selects no lighting for this asset.

**Cast Shadows**

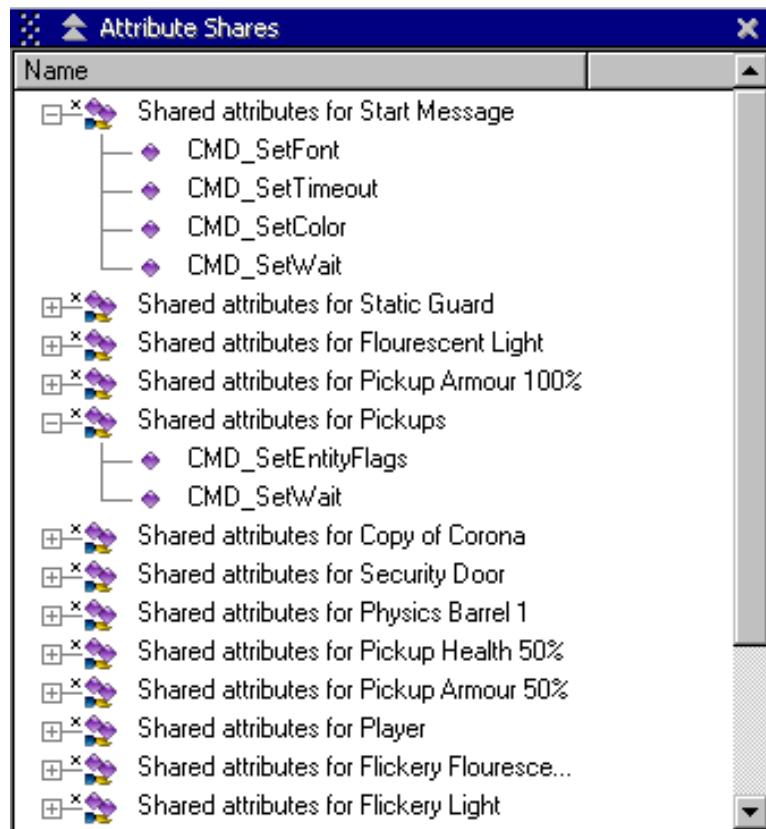
Selects whether this asset will cast shadows or not.

**Use Game-Database Matrix**

Causes lighting to be calculated as per the game world, as opposed to using any pre-existing lighting information.

## Attribute Shares window

Lists the attribute shares that you created in the Game Explorer window.



## Attributes window

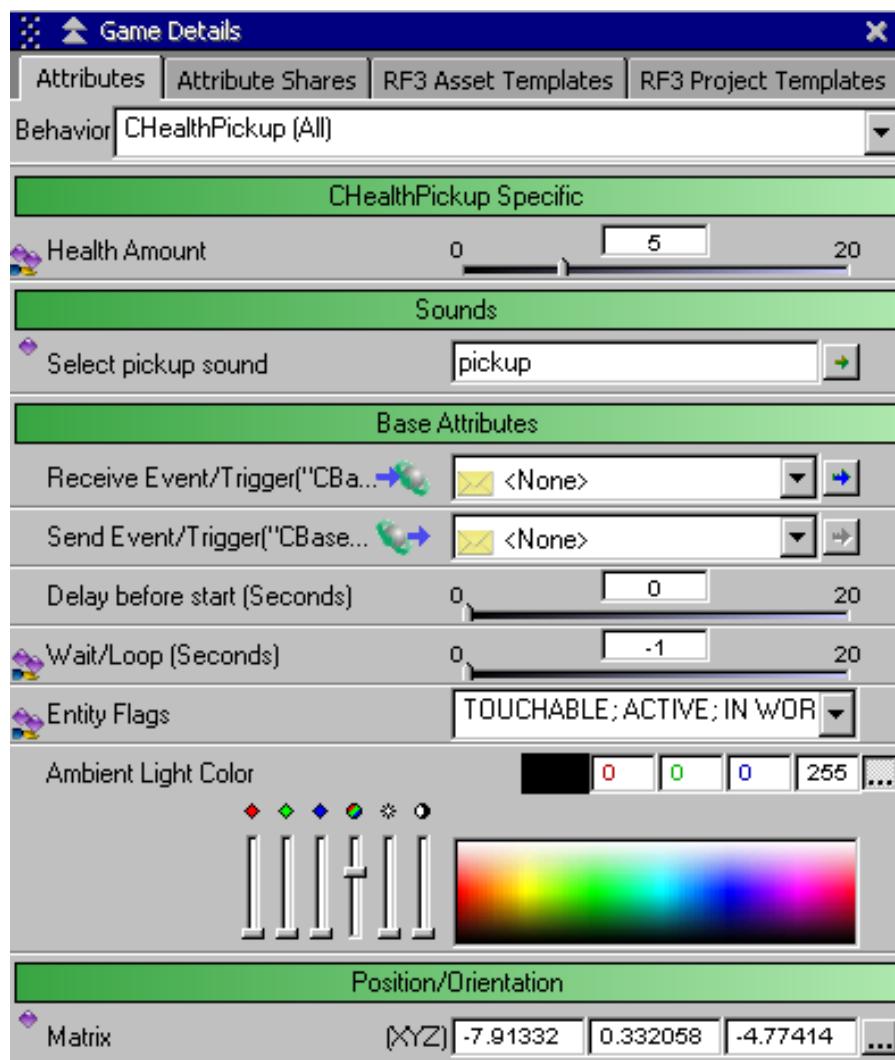
---

Allows you to edit the values of the behavior attributes for the selected entity. Also allows you to send messages from the entity, for testing purposes.

The **Behavior** list (at the top of the window) shows the behavior class attached to the selected entity, and each of the classes from which it inherits. If you select “*Behavior class(All)*”, then the Attributes window shows a combined view of the attributes for all of these classes. If you select a particular class, then the Attributes window shows only the attributes defined by that class.

If you have more than one entity selected:

- If all of the selected entities have the same behavior, then the Attributes window shows a combined view of their attributes. Any changes you make affect all selected entities. If the selected entities have different values for an attribute, then the attribute control is shown grayed out; you can still use the control, and set a common value for this attribute.
- If the selected entities have different behaviors, then the Attributes window shows only the attributes of the last selected entity.



## Editing attributes

The example window above shows a behavior that demonstrates many of the different types of controls that you can use to edit behavior attributes.

- To show or hide the attributes under a separator, click the green separator bar
- To expand a separated section and see the extended controls available for a control, as shown above in the color picker control for the *Ambient Light Color* attribute, click the  button.

**Note:** For some attributes, such as the color value input boxes for the color picker, you will need to ensure that the window is wide enough to view them fully. For example, if the window shown were not wide enough, then you would only see a single blank box instead of the five boxes shown.

To view a description of an attribute:

- Right-click the attribute name, and then click **Help**. Alternatively, hover over the attribute name, and a tooltip containing its description will appear.

## Attribute symbols

In addition, each attribute in the window can have a symbol displayed alongside it.

- If there's no symbol next to the attribute name, then that attribute has not yet been given a value.
- If there's a purple lozenge next to the attribute name (), then that attribute has been given a value.
- If there's a hand holding two purple lozenges next to the attribute name (()), then that attribute is **shared** (p.170).

## Help for programmers

The Workspace generates the attribute editor interface for each behavior by parsing RWS\_ATTRIBUTE, RWS\_SEPARATOR, and RWS\_MESSAGE macros in the C++ source header files.

To view the source code of an attribute:

- Right-click the attribute name, and then click **View Source**. The header file for the behavior containing that attribute is automatically opened in the **Source viewer** (p.158).

```
class AllControls : public CSysCommand, public CAttributeHandler, ↴
public CEEventHandler
{
public:
    RWS_MAKENEWCLASS(AllControls);
    RWS_DECLARE_CLASSID(AllControls);
    RWS_CATEGORY("Tutorial");
    RWS_DESCRIPTION("AllControls", "");

    // Bitfield - @ preceding the item name specifies "checked" as default
    // List      - @ preceding the item name specifies the item is the
    default selection
```

```

RWS_BEGIN_COMMANDS
    RWS_ATTRIBUTE( CMD_MyBitfield, "BITFIELD", "Tip", ↴ BITFIELD,
RwUInt32, LIST(Item A|@Checked Item B|Item C))
    RWS_ATTRIBUTE( CMD_MyBoolean, "BOOLEAN", "Tip", ↴ BOOLEAN, RwBool,
DEFAULT(0))
    RWS_ATTRIBUTE( CMD_MyColor, "COLOR", "Tip", ↴ COLOR,
RwUInt32, DEFAULT(0xFF0000))
    RWS_ATTRIBUTE( CMD_MyList, "LIST", "Tip", ↴ LIST,
RwUInt32, LIST(Item A|Item B|@Default Item C))
    RWS_ATTRIBUTE( CMD_MyMatrix, "MATRIX", "Tip", ↴ MATRIX,
RwMatrix, DEFAULT(0))
    RWS_ATTRIBUTE( CMD_MyVector, "VECTOR", "Tip", ↴ VECTOR, RwV3d,
RANGES((-360,0,360), (-360,0,360), (-360,0,360)))

    RWS_SEPARATOR("Edits", 0)
    RWS_ATTRIBUTE( CMD_MyEdit1, "EDIT1", "Tip", ↴ EDIT, STRING,
DEFAULT(Default value))
    RWS_ATTRIBUTE( CMD_MyEdit2, "EDIT2", "Tip", ↴ EDIT, RwInt32,
DEFAULT(-10))
    RWS_ATTRIBUTE( CMD_MyEdit3, "EDIT3", "Tip", ↴ EDIT, RwUInt32,
DEFAULT(10))
    RWS_ATTRIBUTE( CMD_MyEdit4, "EDIT4", "Tip", ↴ EDIT, RwReal,
DEFAULT(10.0))
    RWS_ATTRIBUTE( CMD_MyEdit5, "EDIT5", "Tip", ↴ TEXTEDIT, RwReal,
DEFAULT(Default value))

    RWS_SEPARATOR("Messages", 0)
    RWS_MESSAGE(MSG_MyMessage, "MESSAGE1", "Tip", ↴ RECEIVE,
"RwUInt32*", 0)
    RWS_MESSAGE(MSG_MyMessage, "MESSAGE2", "Tip", ↴ TRANSMIT, 0,
"Whatever")

    RWS_SEPARATOR("Sliders", 0)
    RWS_ATTRIBUTE( CMD_MySlider1, "SLIDER1", "Tip", ↴ SLIDER, RwReal,
RANGE(0.01, 1.0, 100.0))
    RWS_ATTRIBUTE( CMD_MySlider2, "SLIDER2", "Tip", ↴ SLIDER, RwUInt32,
RANGE(0, 180, 360))
    RWS_ATTRIBUTE( CMD_MySlider3, "SLIDER3", "Tip", ↴ SLIDER, RwInt32,
RANGE(-360,0,360))
    RWS_END_COMMANDS;

AllControls(const CAttributePacket& attr);
~AllControls(void);

virtual void HandleEvents(CMsg &pMsg);
virtual void HandleAttributes(const CAttributePacket& attr);

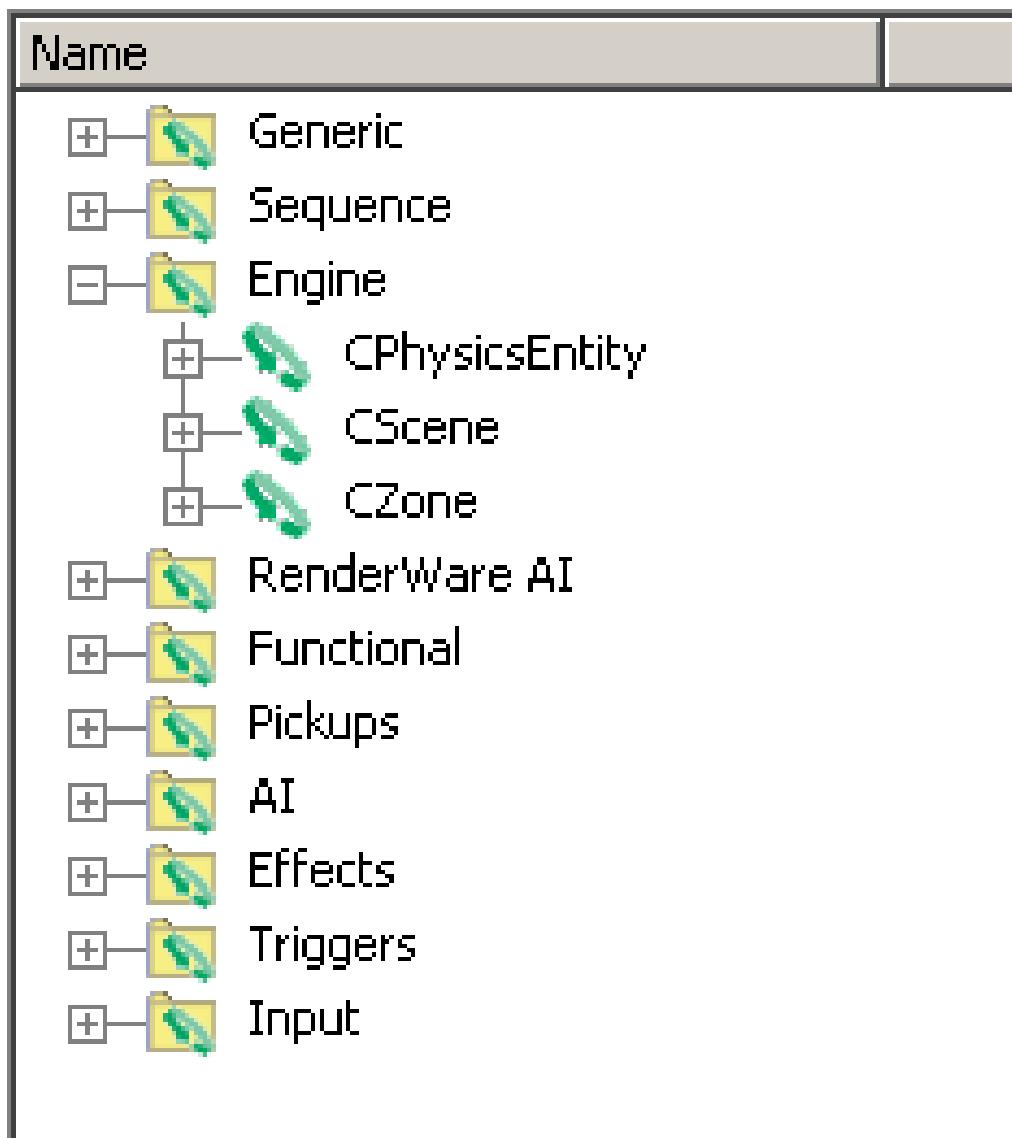
protected:
    RwMatrix *m_mat;          /* Matrix used for rotating object */
    CAtomicPtr m_pAtomic;    /* Behavior's Atomic */
    RwReal m_rot[3];         /* RwReal array for storing rotation values*/
} ;

```

For more information about these macros, see Automated Form Generation in the Game Framework reference.

## Behaviors window

Lists the behaviors that you can attach to an entity. The behaviors are organized in folders by category.



To check the Game Framework source code for changes, and update the Behaviors window accordingly:

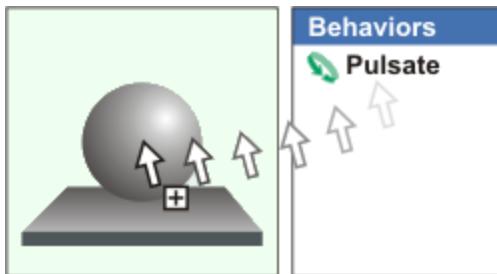
- Right-click an empty area of the Behaviors window, and then click **Parse Source**.

## Attaching a behavior to an entity

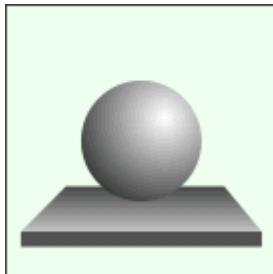
By default, Workspace attaches the CEntity behavior to new entities.

To attach a different behavior to an entity:

1. Drag the behavior from the Behaviors window to the entity in the Design View window.



2. If you connect to a target, then you can dynamically view the behavior in the game.



After attaching a behavior, you can set its attributes (for example, the speed and color of the pulsation) in the Attributes window.

For descriptions of the behaviors supplied with RenderWare Studio, see the Game Framework Help.

### Help for programmers

To view the source code for a behavior:

- Right-click the behavior, and then click **View Source**.

The source code appears in the editor set in Workspace Options.

To view user documentation in the Workspace for Genre Pack 1-related behaviors:

- Right-click the behavior, and then click **View Help**.

The [Help window](#) (p.291) appears displaying the help for the selected behavior.

## Context menu

To view the actions that you can perform in the Behaviors window:

- Right-click a behavior.

The following context menu appears:

**View ►**

Sets how the window represents the behaviors:

**Detail**

Lists behavior names and other properties under column headings.

**List**

Lists behavior names (with a small 2D icon) down, then across the available space in the window.

**Hierarchy**

Lists behavior names in hierarchical order under column headings.

---

**Copy**

Copies a behavior. This allows you to paste the behavior onto an entity.

---

**View Source**

Displays the source code for the behavior in the editor chosen in [Workspace Options](#) (p.158) under “Source Viewer”.

**Parse All Source**

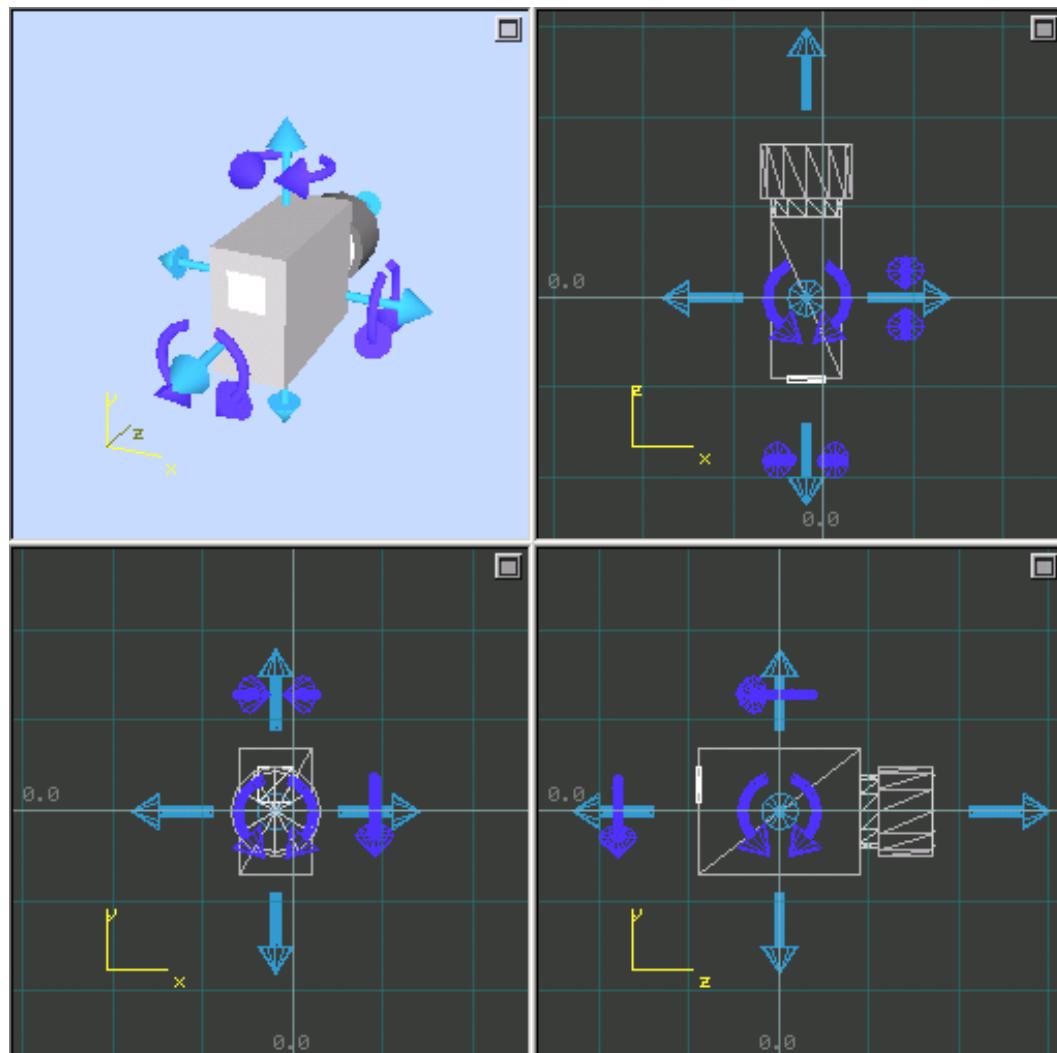
Creates a new behavior or changes an existing behavior by parsing the source code and refreshing the display.

**Search**

(Behaviors only.) Displays several options for identifying which entities use this behavior. The results are displayed in the [Search Results](#) (p.315) window.

## Design View window

Displays four camera views of the current game: one perspective view and three orthographic views (front, plan and side).

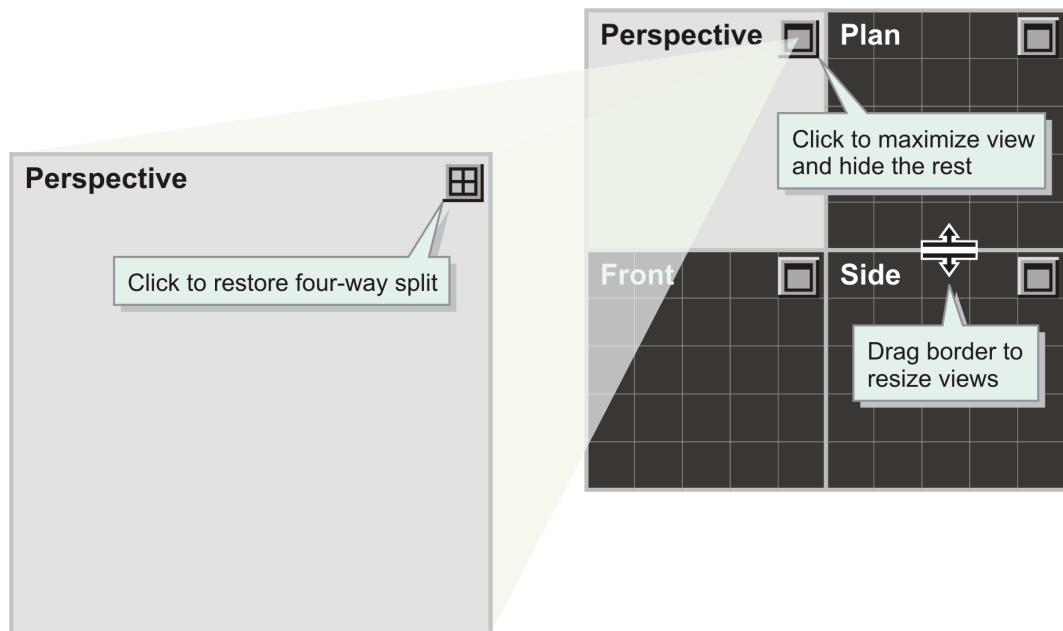


You can set various [options](#) (p.160) for how the Design View window displays your game.

### Resizing the views

To resize the views, drag the border between them.

To maximize one of the four views, click its  icon.



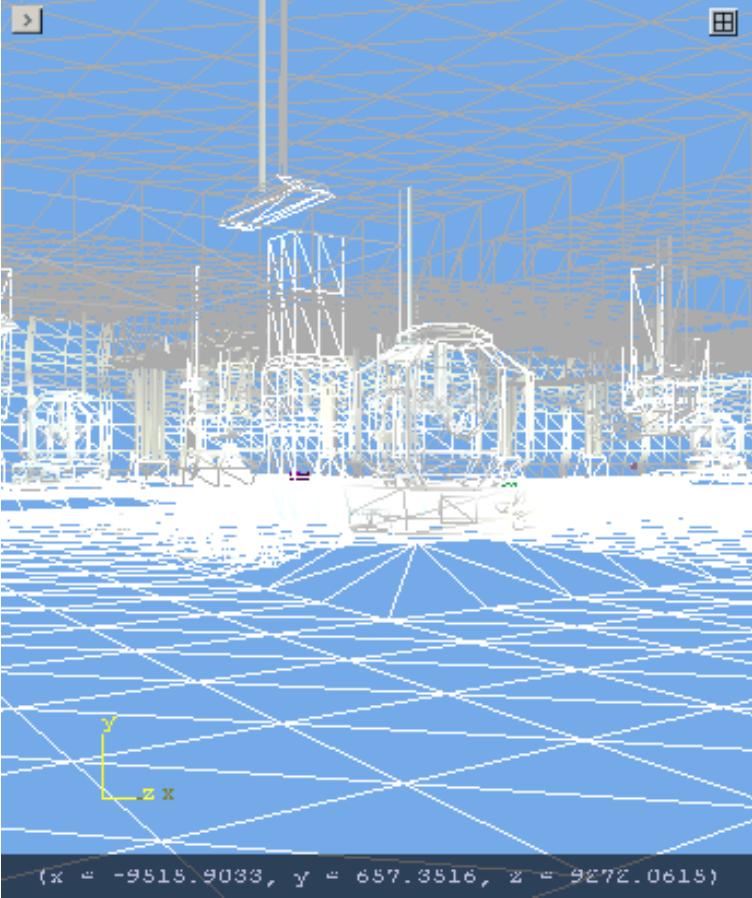
## Design View toolbar

---

To display this toolbar click on the expander button  in the top left-hand

corner of any of the four views in the [Design View](#) (p.276) window.

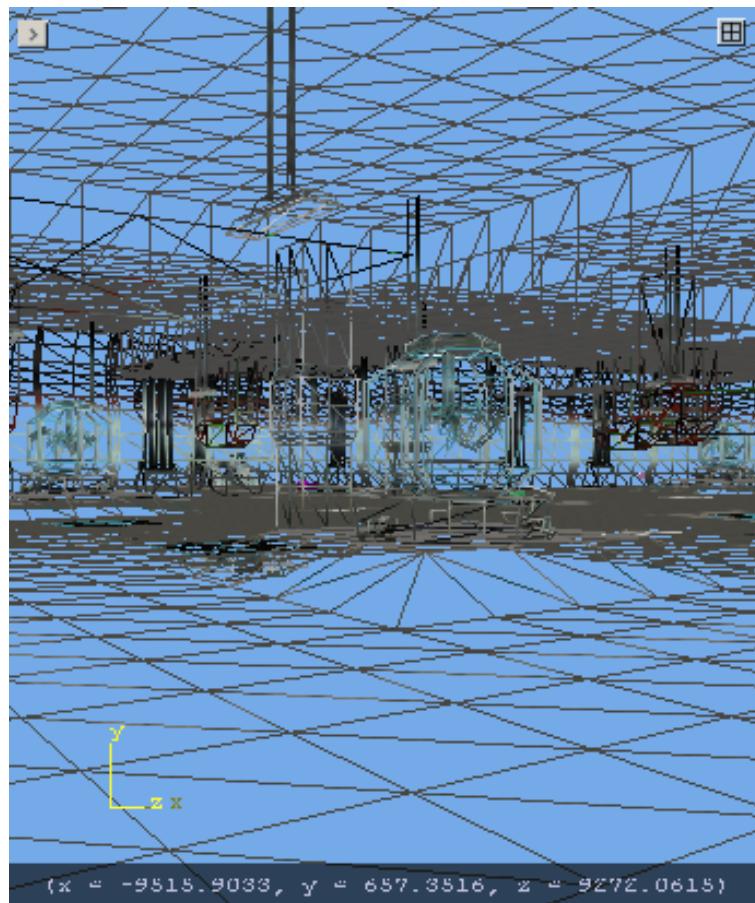
The top four buttons change the view mode as follows:

Button	View	Result when view button selected
	<b>Wireframe view</b> Displays the scene in wireframe view.	 <p>(x = -9515.9033, y = 657.3516, z = 9272.0615)</p>



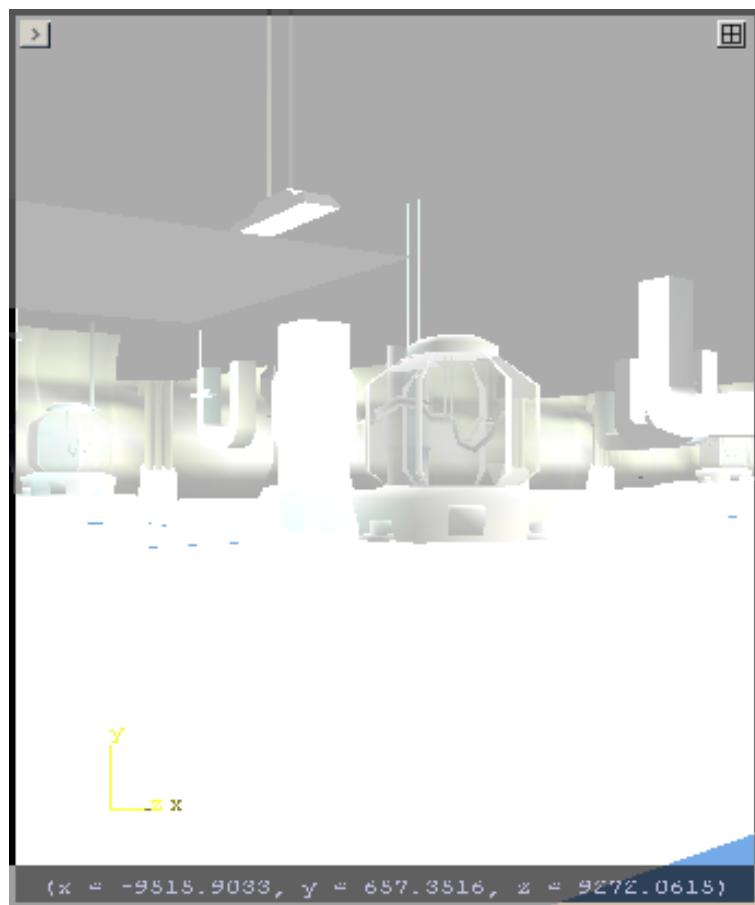
### Wireframe with textures

Displays the scene in wireframe view with the wireframe colored where surfaces are textured.



### Solid without textures

Displays the solid surfaces of the scene without textures.



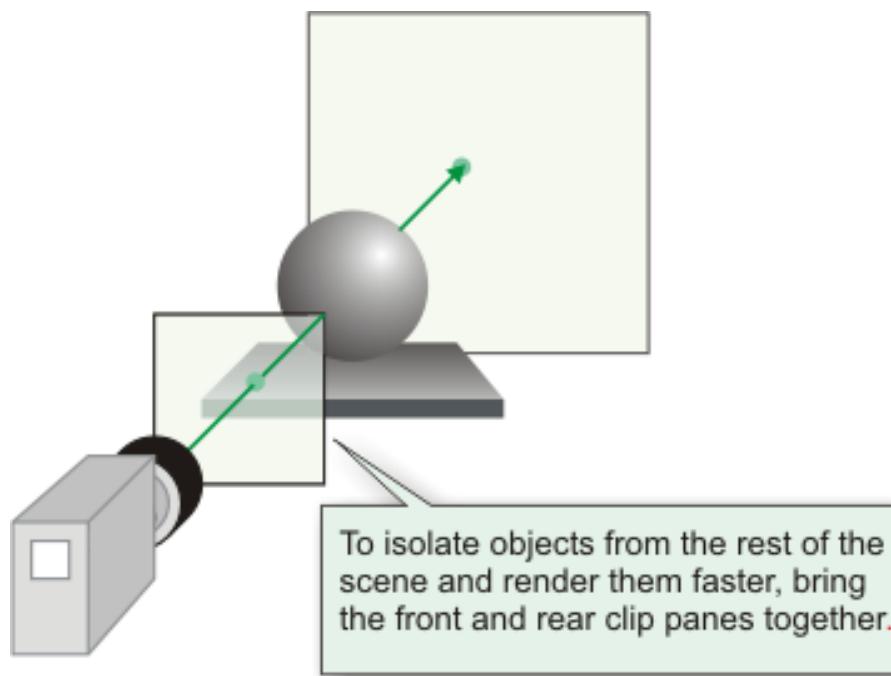
**Default**

Displays the scene with solids frames and textures.



## Clip pane tools

The clipping pane is a barrier to determine the loading and display points of 3d models and textures. There is a rear clipping pane in between the scene and the camera and a far clipping pane behind the scene.



By default the clip panes are set automatically so that all the textures and models in the camera's view are rendered. In certain cases you may want to isolate objects in a view because they are very close together or exclude objects that are expensive to render in processor terms.

To do this:

1. Click on the View mode icon  in the Design View window to display the Design View toolbar.
2. Click the Auto button  to toggle between the default clip pane mode and the user-defined mode.
3. Click and drag the slider buttons below the  button to move the clip panes.
  - To move the back pane, click the back button  and drag upwards to move the pane away from the camera out from the scene; and drag downwards to move the pane nearer the camera into the scene. The result for the scene pictured above would be:

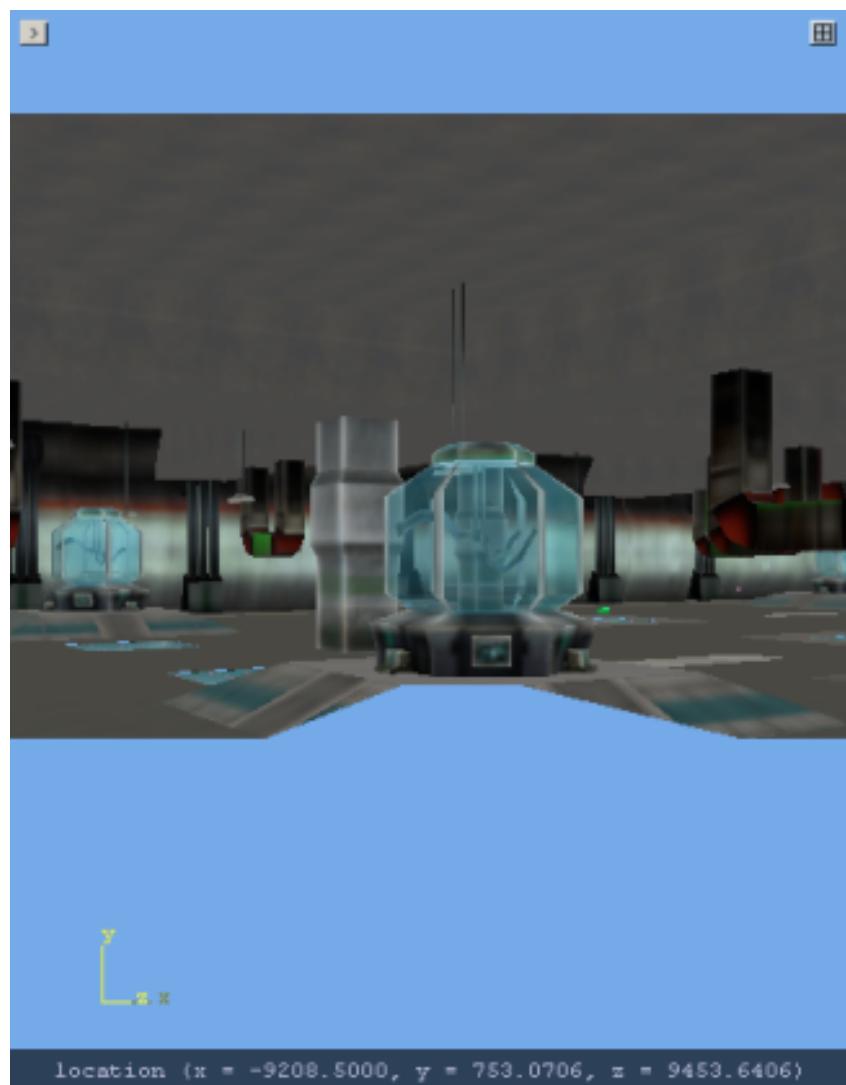


- To move the front pane, click the front button  and drag upwards to



Front

move the front clip pane away from the camera into the scene; and drag downwards to move the pane nearer the camera away from the scene. The result for the scene pictured above would be:



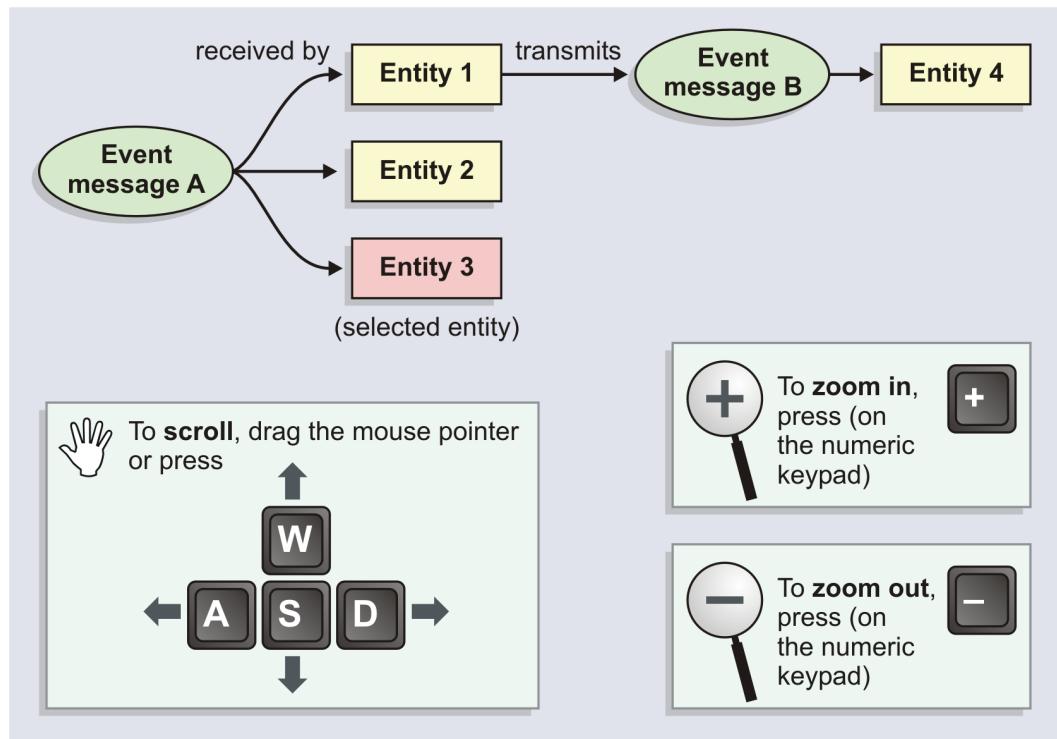
**Note:** You may need to drag the buttons a long way before changes in the scene become apparent.

# Event Map

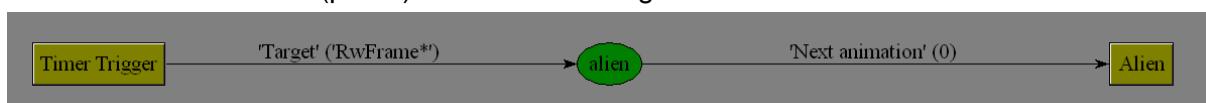
Displays a flowchart of events and the entities that transmit or receive them.

In the simplified example below:

- Event A is received by entities 1, 2 and 3.
- Entity 1 transmits event B, which is received by entity 4.



Selecting an item in the Event Map highlights the item, its connecting attribute names and their connecting nodes. The picture below - taken from the [Alpha Sort](#) (p.515) example shows the “alien” event that was transmitted from the Timer Trigger entity and received by the Alien entity. The text in brackets “RwFrame” shows the [Data format](#) (p.286) of the event being transmitted.



## Tips

- If you are using a mouse with a middle-mouse wheel and Microsoft™ Intellipoint software installed, then you can also zoom in and out of the Event Map window by rotating the mouse wheel.
- If you select an entity in either the [Game Explorer](#) (p.288) or the [Design View](#) (p.276), then it is highlighted in the Event Map and vice versa.
- If you select an entity in the Event Map, its attributes are highlighted in the [Attributes window](#) (p.270).
- If you have a game database containing a large volume of data, the Event Map may take some time to refresh. Therefore, instead of automatically

refreshing itself, the background color of the map changes signifying that the flowchart data has changed and needs to be updated. To refresh the Event Map, and see the most recent changes either:

- On the **Edit** menu, click **Refresh Event View**.
- or
- Press **F5**.

#### Tip for programmers

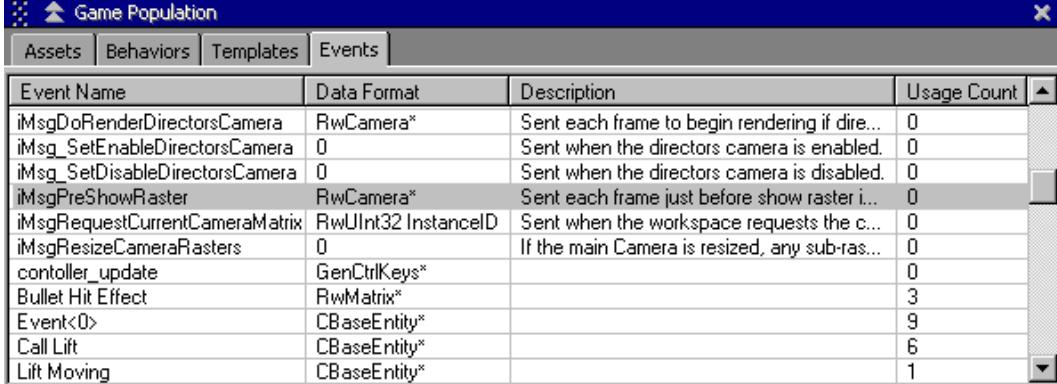
The Event Map builds the flowchart from the RWS\_MESSAGE macros in the source code. It is important that the TRANSMIT and RECEIVE keywords for an event are set correctly for the direction of the event to be shown accurately in the Event Map

**Note:** If the keyword for an event macro is set to TRANSMIT, then the  icon will appear in the Entity Attributes window beside the attribute described by the RWS\_MESSAGE macro, if the keyword is set to RECEIVE, then the  icon will appear beside the attribute.

# Events window

---

Lists events sent and received by entities in the active folder.



Event Name	Data Format	Description	Usage Count
iMsgDoRenderDirectorsCamera	RwCamera*	Sent each frame to begin rendering if dire...	0
iMsg_SetEnableDirectorsCamera	0	Sent when the directors camera is enabled.	0
iMsg_SetDisableDirectorsCamera	0	Sent when the directors camera is disabled.	0
iMsgPreShowRaster	RwCamera*	Sent each frame just before show raster i...	0
iMsgRequestCurrentCameraMatrix	RwUInt32 InstanceID	Sent when the workspace requests the c...	0
iMsgResizeCameraRasters	0	If the main Camera is resized, any sub-ras...	0
controller_update	GenCtrlKeys*		0
Bullet Hit Effect	RwMatrix*		3
Event<0>	CBaseEntity*		9
Call Lift	CBaseEntity*		6
Lift Moving	CBaseEntity*		1

The columns list the following information:

- The event name which is used by the behavior.
- The data format of any data associated with the event.

**Note:** The data format of an event governs its compatibility with other entities and which event you can drag to the Entity Attributes window from the Events window.

For example, if you were working with the events in the “Level: Particle Effects 5” folder of the [FX Particle Spray example](#) (p.527) this means that you could drag the T1 , T2 or T3 events interchangeably between the different “Timers used to Position Fountain” behaviors, because they all have the same “Data format”. You could not however drag a INQ\_ACTN\_TURN event onto one of these behaviors because it has a different data type.

- A description of the event.
- The number of times the event is referenced by “Transmits” and “Receives” of data from other entities.

**Tip:** If you select an event, then it is automatically highlighted in the Event Map and vice versa.

## Creating a new event

To create a new event:

1. Right click in the Events window and select **New Event**.
2. Right click the new event and select **Rename**, and then overtype the default name of the new event.
3. Select the entity with which you want the event to be associated.
4. Drag the new event into the Attributes window, beside the appropriate attribute line.

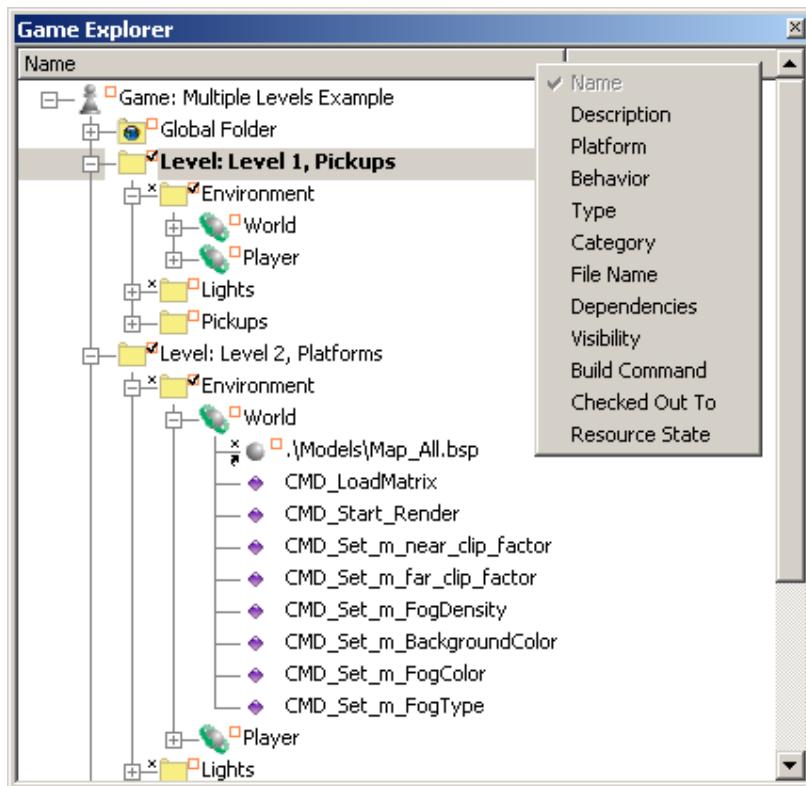
When the event is dragged to the Attributes window, the event name listed

within that attribute is changed to the name of the dragged event and a new or modified event chain is displayed in the Event Map.

# Game Explorer window

---

Lists the game database hierarchy, and the objects in the game database.



This window can display various columns of information; to select which columns are displayed, right-click the column header. (For example, the “Checked out to” column displays the user name of the person who has that object checked out in NXN alienbrain.)

If an object has multiple references, then this window displays a small **x** next to each reference.

**Tip:** To find an entity in the [Design View](#) (p.276) window, right-click the entity name in the Game Explorer window, and then click **Locate in view**.

## Changing the order of objects in the game data stream

The default build rules of the Game Production Manager send sibling objects (for example, the assets of an entity) to the target console in the order shown in this window. You need to ensure that objects are sent to the target in the order that your game requires them.

To change the order of an object relative to its siblings:

1. Select the object.
2. Press and hold **Alt**, and then press **↑** or **↓**.

## Context menu

To view the actions that you can perform in the Game Explorer window, right-click an object. The following context menu appears:

### **View ►**

Changes between the several possible views of the Game Explorer window.

### **New ► Folder as child**

(Folders only.) Creates a new child folder.

### **Cuts**

Cuts the object to the Clipboard.

### **Copy**

Copies the object to the Clipboard.

### **Paste**

Pastes the object from the Clipboard.

### **View Source**

(Behaviors, commands, or attributes.) Views the associated C++ source for this object.

### **Move Up**

(Only for objects with previous siblings.) Moves the object above its previous sibling.

### **Move Down**

(Only for objects with following siblings.) Moves the object below its following sibling.

### **Delete or Remove**

[Deletes or removes](#) (p.177) the object.

### **Rename**

Renames the object.

### **Show all Entities and Hide all Entities**

(Folders only.) Shows or hides all entities in the folder (see **Hide in View** and **Show in View**, below).

**Tip:** You can achieve the same effect by right-clicking in the Design view, and then selecting **Show all entities** from the context menu.

### **Hide from view**

(Entities only.) Hides the entity in the Design View window. In the Visibility column, the entity is marked as Hidden.

**Tip:** You can achieve the same effect by selecting the entity in Design View, and then selecting **Hide selection** from the context menu.

### **Show in view**

(Entities only.) Displays the entity in Design View window. In the Visibility column, the entity is marked as Hidden.

### **Freeze all Entities and Unfreeze all Entities**

(Folders only.) Freezes or unfreezes all entities in the folder (see **Freeze**

and **Unfreeze** below.)

#### **Freeze**

(Entities only). Prevents you from being able to select the object in the Design View window. If you click over the frozen object, the objects behind it will be selected instead.

**Tip:** You can achieve the same effect by right-clicking the object in the Design View window, and then selecting “Freeze selection” from the context menu.

#### **Unfreeze**

(Entities only). Allows the user to select the object again.

---

#### **Share Attributes**

(Entities only.) Displays the [Share Attributes](#) (p.170) dialog. This dialog shows the attributes of this entity that can be shared.

#### **Use Attributes**

(Entities only.) Displays the [Use Attributes](#) (p.170) dialog. This dialog shows the attribute shares, and the attributes in those attribute shares that you can use for this entity.

#### **Locate in view**

(Entities only.) Locates and selects the entity in the Design View.

#### **View XML**

Displays the XML code for the database object in a browser window.

#### **Create Template**

(Entities and folders only.) Creates a [template](#) (p.127) entity, or (if you clicked a folder), a template folder containing a template for each of the entities under the folder. The new templates and template folders appear in the Templates window.

#### **Properties**

Displays the object properties dialog.

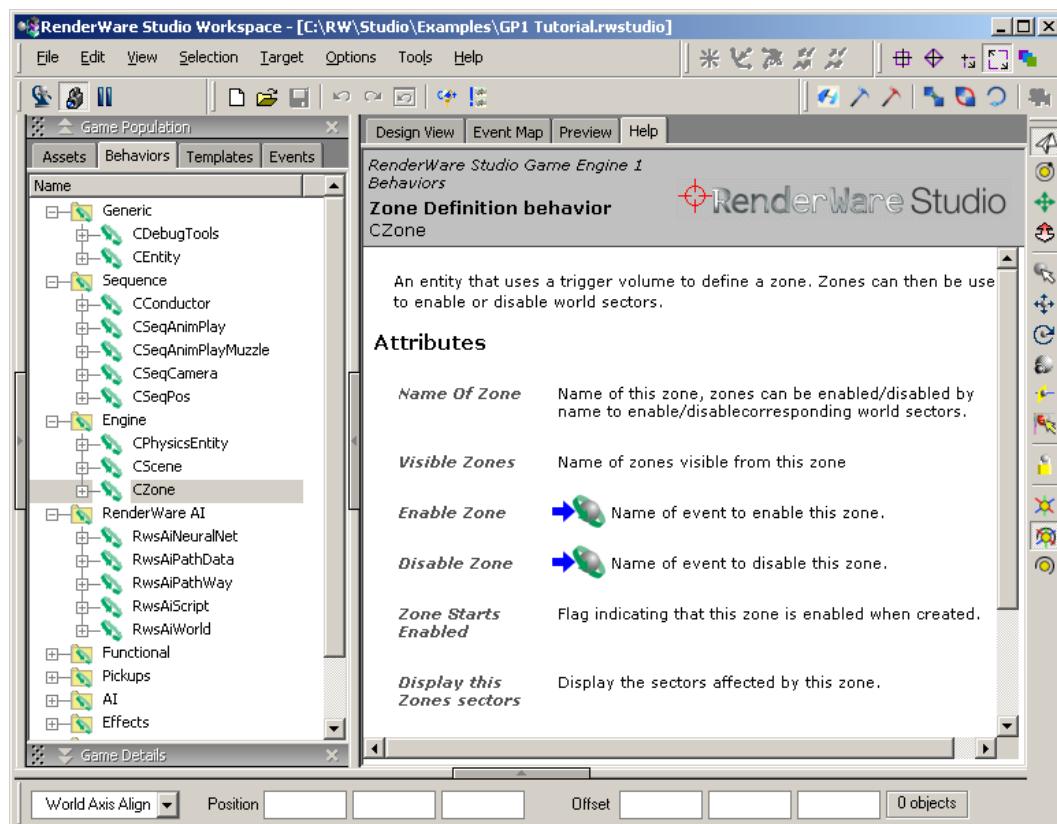
#### **Reset or Reset all Entities**

(Entities or folders only.) Resends the selected entity (or entities in the selected folder) and attributes to any connected target consoles, without resending assets. This is a quick method of resetting the game if no assets have changed.

# Help window for behaviors

---

Provides context-sensitive help for behaviors.



To view help for a behavior, right-click the behavior in the [Behaviors](#) (p.273) window, and then select **View Help** from the context menu.

**Note:** This behavior help is displayed in Workspace because it is aimed at designers as well as programmers. Currently, only the behaviors in Genre Pack 1 are supplied with this type of help.

To create help for your own behaviors:

1. Create an HTML file with the same name as the behavior and a file extension of .htm (for example, CMyBehavior.htm).
2. Save the HTML file in the help folder under your project's source root folder (if this help folder does not exist, then create it):

```
source root\help\behavior class.htm
```

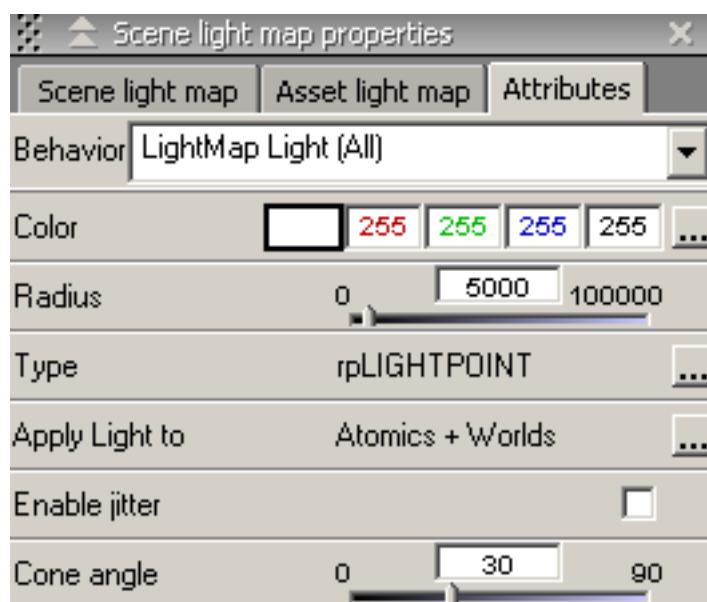
See the example behavior help supplied with Genre Pack 1, in:

```
C:\RW\Studio\GenrePack1\console\game_engine_1\source\help
```

# Light Attributes window

---

Displays information about a light entity. Allows properties such as the color, radius and type of light to be modified.



## Behavior

Displays the currently selected behavior of the entity. Also allows you to select the behavior of the entity from a list of options.

## Color

Allows you to modify the color of the light using red, green, blue, and alpha values. Note, the alpha value is *not* used in the lighting calculations.

## Radius

Sets the sphere of influence of a point or spot light.

**Note:** Light levels drop off exponentially as distance increases. If the radius is set at too low a value, it is possible that a light may not display as expected. For example, a wall may not appear affected by a spot light pointed at it, if the radius of the spot is set at just a little beyond the wall. The reason being the light intensity has dropped significantly at the extents of the radius. The solution, in this case, is to increase the radius of the spot light to well beyond the wall being lit.

## Type

Allows you to select the type of light from one of Directional, Point, Spot, Soft Spot.

### Directional

This type of light is used to represent a source assumed to be at a large distance from the world. For example sunlight and moonlight could be represented by directional lights. Setting the radius and cone angle has no affect on this light type.

### Point

Represents a point of light. Light is emitted in a sphere from the source,

which has a radius.

**Spot**

Represents a source where a cone of light is emitted with a certain cone angle and radius.

**Soft spot**

Similar to a spot light, but with a softer edge.

**Note:** Spots have both a cone angle and a radius, the radius being the sphere of influence of the light.

**Apply Light to**

One of Atomics and Worlds, Worlds only, Atomics only.

**Enable jitter**

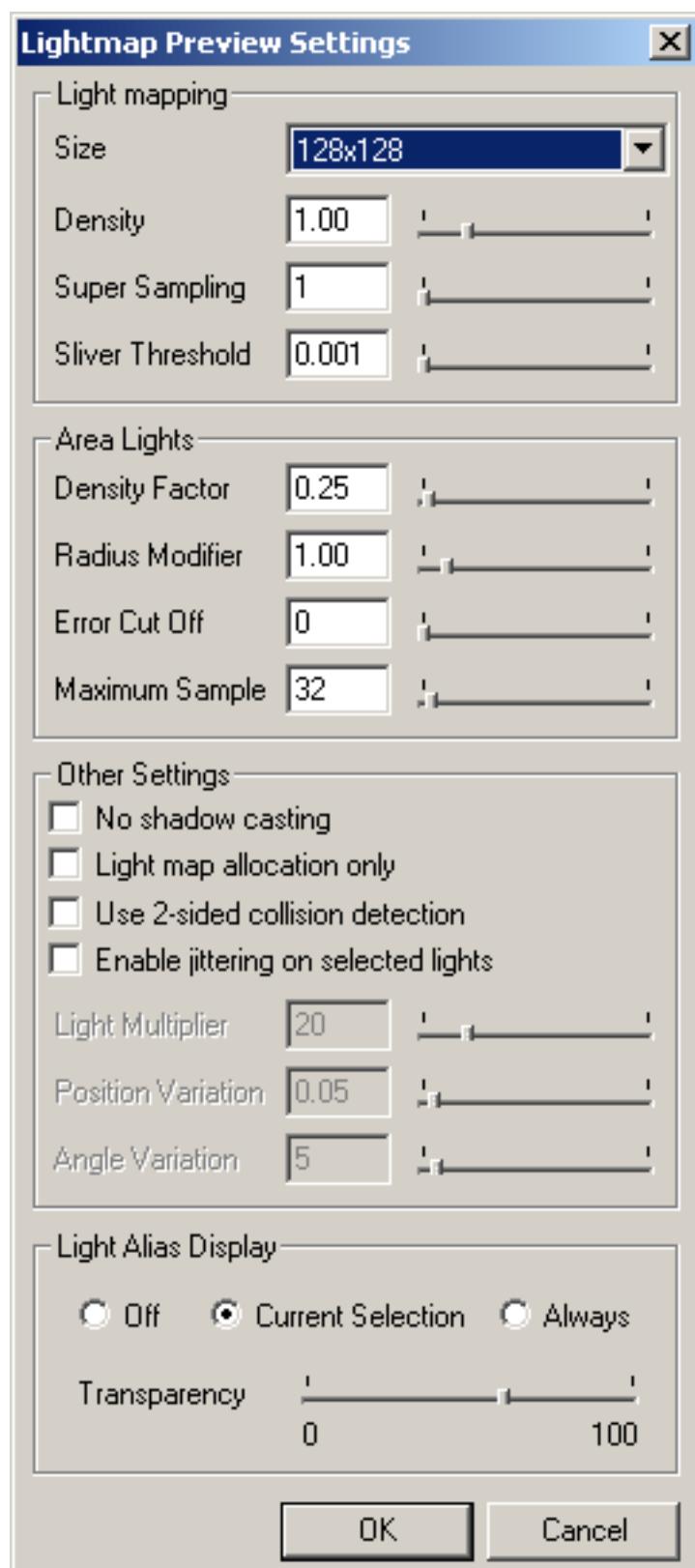
This enables “jittering” of lights. In this context, jitter means to process the light as if it occupied a small range of positions or angles. The purpose of this is to soften shadows cast by these lights, the softness being proportional to the distance from the occluding object casting the shadow.

**Cone angle**

For spot lights, allows you to modify the cone of influence. Changes to this value can be viewed immediately and adjusted interactively in the **Design View** window, by ensuring that **Light Alias Display** in the **Light Map Preview Settings** window has been set to either **Current Selection** or **Always**.

## Light Map Preview window

Allows you to adjust various parameters that affect how light map previews will appear in the Design View window.



## Light mapping

General light mapping options.

### Size

Size of light map texture created in pixels. Light maps are always square.

### Density

Refers to texel density, this is the average number of pixels in a light map that are applied to a polygon. The higher the density the greater the detail of the light map.

### Super Sampling

This is a technique for generating light maps of better quality. If the light map was 128x128 and the supersampling was 2, the light map would be generated at 256x256 and scaled down to 128x128.

### Sliver Threshold

Any polygon that is less than the area specified here will be ignored by the light map generation process. This avoids spending time calculating light mapping for polygons that will be ignored anyway.

## Area Lights

Options related to area lights.

### Density Factor

This alters the number of point lights for a given area light. The full formula used to calculate density factor is:

`LightMapping.density×AreaLights.densityfactor×material.arealight.densityfactor`

### Radius Modifier

This will change the radius of all area lights used in a scene. The radius of an area light is:

`material.arealight.radius × Lightmapping.radius_modifier`

### Error Cut Off

This allows the user to sharply cut off the influence of an area light. The larger the value, the sooner the cut off point will be reached. This is generally used to reduce light map generation times.

### Maximum Sample

This is the maximum number of point lights used to emulate an area light.

## Other Settings

Miscellaneous light-related parameters.

### No shadow casting

Disables generation of shadows.

### Light map allocation only

Selecting this check box will mean that light maps will be created, but will not be shown in the **Design View**.

### Use 2-sided collision detection

During the light map calculation, rays are cast from each sample point to each light. In the default case this process is carried out for one side of a triangle. However, if this check box is selected both sides of the triangle are taken into consideration. This results in greater detail being possible, at the

expense of greater processing time.

### **Enable jittering on selected lights**

This enables “jittering” of lights. In this context jitter means to process the light as if it occupied a small range of positions or angles. The purpose of this is to soften shadows cast by these lights, the softness being proportional to the distance from the occluding object casting the shadow.

## **Light Alias Display**

Settings associated with drawing additional geometry for lights in the **Design View** window.

### **Off**

No additional geometry will be drawn.

### **Current Selection**

This is the default setting. Additional geometry will only be drawn for the object currently selected in the **Design View** window.

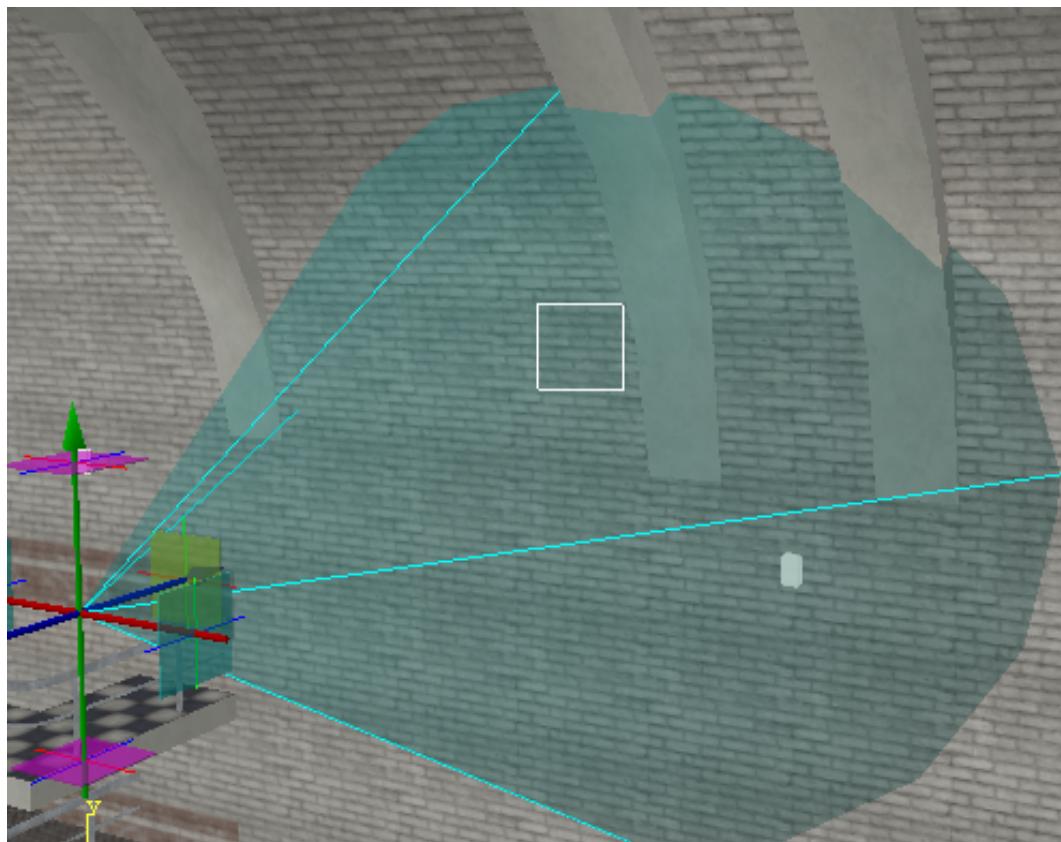
### **Always**

Additional geometry will be drawn for all light objects, regardless of whether currently selected or not.

### **Transparency**

This slider control allows the transparency of the sphere of influence of the light alias to be adjusted.

An example of the additional geometry being drawn in the **Design View** window, in this case for a spot light, is shown in the following screenshot:

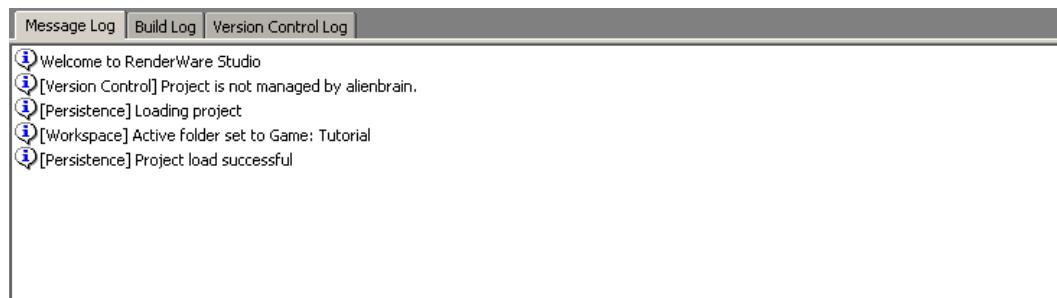


# Log windows

---

## Message Log

Displays messages about activities in the current RenderWare Studio session. Each message begins with a graphical symbol (indicating whether it is information, a warning, or error) followed by the name of the part of RenderWare Studio that sent the message (enclosed in square brackets). For example, in the screen capture below, the [Local PC - DirectX] messages are from the Game Framework (running on a DirectX local PC target) as it reads a game data stream sent by the Workspace.



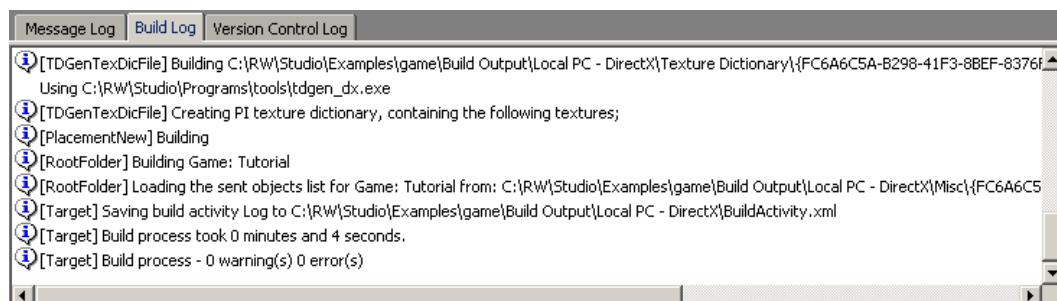
Messages include details of:

- XML parser process updates
- Status of the connection to the target console
- Asset import progress when opening a project
- RF3 Asset import progress

These messages also include debug messages. RenderWare Studio displays debug messages when you connect to a debug console. If you work with a debugger, such as Visual Studio debugger, RenderWare Studio still displays debug messages in the Message Log window.

To view the options that you can set for any of these Log windows, right-click inside the window.

## Build Log



Displays build progress messages from the Game Production Manager.

Messages include details concerning:

- The build of all RenderWare Studio objects within the global and active folders—assets, asset folders etc.
- The build of all rule-dependent tools, for example texture dictionaries
- The sent objects list
- Additions to the Build Activity Log file
- Custom build rules

Messages are displayed in the order in which the game is built. Messages relating to the global folder are listed first, followed by messages relating to the active folder.

## Version Control Log



Displays NXN alienbrain version control information.

Messages include:

- More detail on version control messages listed in the Message Log window in Workspace
- The messages echo the contents of the Message Log window in the full alienbrain Manager Client window

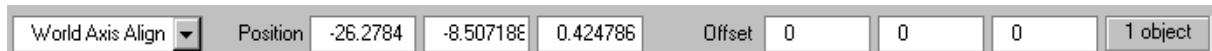
## Copying messages to the clipboard

To copy the entire contents of any Log window to the clipboard:

1. Click a message in the Log window (this highlights the message).
2. Press **Ctrl+A** (this highlights all messages in the window).
3. Press **Ctrl+C** or right-click inside the Message Log window, and then select **Copy** from the context menu.

# Object information bar

---



Displays information about the currently selected object. This information changes dynamically depending on the pick mode selected from the selection toolbar.

The information which can be edited will be one of three types - position, orientation or scale.:

### **Position**

The position of the object in the world relative to the rest of the world.

### **Orientation**

The rotational position of the object.

### **Scale**

The scale of the object relative to its size when the object was first dragged into the world.

**Tip:** To position, scale or rotate an object precisely, select the object with the appropriate pick tool, and then change the numerical values in the boxes by small increments.

When you rotate an object, the axes around which it rotates are aligned with the axes of the world object by default. If you select **Local** from the drop down list box, then these rotational axes move with the object as it rotates.

The offset box shows what has changed since you started to alter an object. Initially, the values in the box will be set to 0, 0, 0 (or 1, 1, 1 for scaling). As you start to drag with the mouse, these values will increase or decrease relative to the total starting values. When you release the mouse key, these values are reset to 0, 0, 0.

**Tip:** You can also type directly into the offset box. For example, to move an object from wherever it is by 50 units horizontally along its X axis, type 50 in the first offset input box, then press **Enter**. Conversely to move it left along its horizontal axis type -50. This saves you having to work out the absolute positional unit totals yourself and allows you to "nudge" objects when positioning them precisely.

# Preview window

---

Allows you to preview RenderWare clumps, atomics, animations and worlds in close-up without adding them as entities to the Design View.

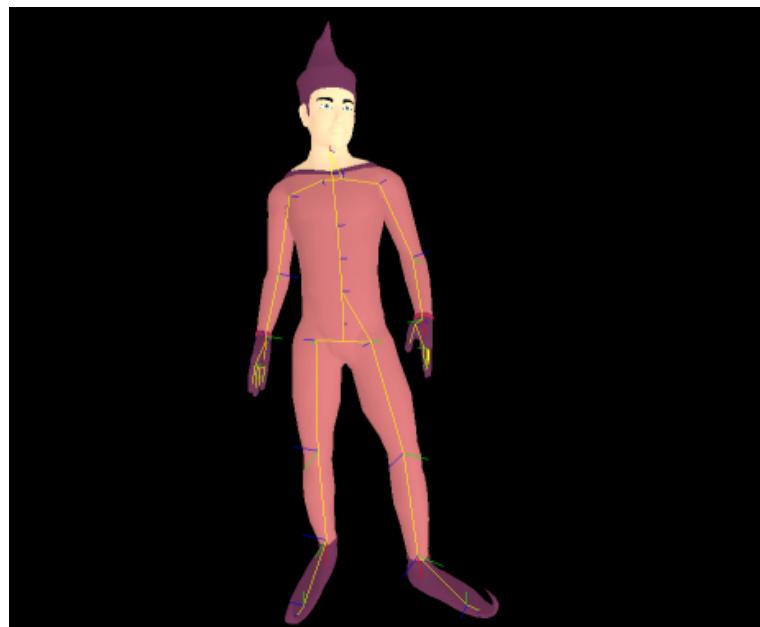
You can only preview assets that have a RenderWare chunk ID of `rwID_CLUMP`, `rwID_ATOMIC`, `rwID_HANIMATION`, or `rwID_WORLD`.

## Previewing an asset

To preview an asset, either:

- Drag an asset from the Assets window to the Preview window.
- or
- Drag an asset file directly from Windows Explorer to the Preview window.

An image of the asset appears in the Preview window:



**Tip:** If the asset does not display clearly against the background, click the Change background color button .

To pan around the Preview window, click and drag with the middle mouse button. To orbit the asset, hold down the **Alt** key and then drag with the middle mouse button. To zoom in on the asset, hold down the **Ctrl** and **Alt** keys and then drag with the middle mouse button. You can also click the orbit, pan, and zoom toolbar buttons in exactly the same way as you would when navigating in Design View, and then drag with the left mouse button.

**Tip:** To reset the object's original orientation and size (before you began navigating around it), click the Reset object button .

## Animating the asset

To view an animation:

1. Drag the animated asset's atomic or clump to the Preview window.
2. Drag an animation asset—with a chunk ID of `rwid_HANIMANIMATION`—from the Assets window to the Preview window.  
The animation starts playing.
3. Click the **Animation toolbar** (p.251) buttons to  play,  pause,  stop and  repeat the animation.

**Tip:** You can stop the animation at any point using the slider control directly under the Preview window. You can also drag the slider control slowly along the slider bar to see how the asset is positioned at any particular time in the sequence.

## Viewing the skeleton and joints of an animated asset

If the asset has a skeleton, you can click the following toolbar buttons:

1. Display skeleton button .
2. Display bone axes button  to view the constituent bones of an asset's skeleton and the orientation of where those joints are at any time in an animated sequence.
3. Show object button  to view the asset's skeleton without rendering the atomic or clump.

## Viewing asset statistics

To view statistics about the asset, click the Display object statistics button .

The following statistics are displayed in the Preview window:

### Atoms

The number of separate objects which make up an asset. For example, the asset shown above has ten different constituent objects which make up its limbs, arms, head etc.

### Vertices

The number of points—in three-dimensional space—that make up the faces of the asset's geometry.

### Triangles

The number of triangles used to define the surfaces of an object.

### Bones

The constituent parts of a skeleton or frame hierarchy—sometimes called joints.

### Animation duration

The total duration—in seconds—of an animation.

### Animation position

The current point in time where the animation is at a given moment, this can be fine-tuned using the slider bar.

## Preview window automatically restarts after a crash

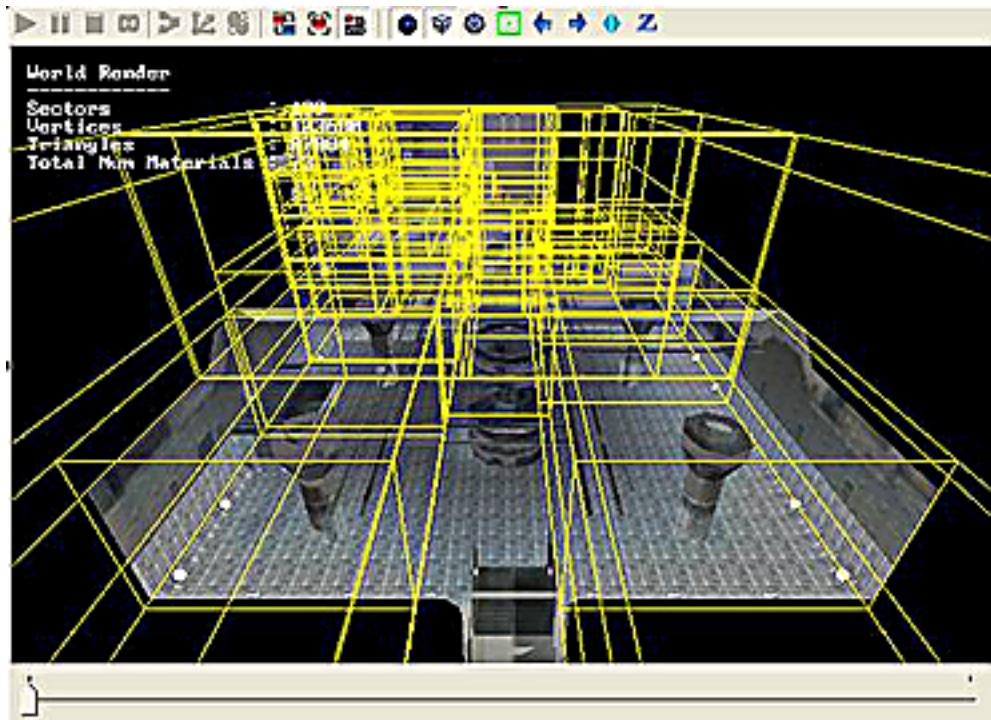
Errors in the asset data can cause the Preview window to crash. If the Preview window crashes, then Workspace detects this, and automatically restarts the window.

## Previewing world asset files

You can preview world .bsp files by dragging them to the Preview window in the same way as clumps and atomic files. To position the camera in the Preview window, use the same toolbar buttons as described above for clumps and atomics (the remaining toolbar buttons that apply only to clumps and atomics will be grayed out). The [quick entry modes](#) (p.108), using modifier keys with the middle mouse button to navigate will also work as they do in the Design View window.

To view a world's sectors, drag the world .bsp file from the asset window to the Preview window, and then click the **Render world's sectors** button .

You will initially see the entire world asset rendered with bounding boxes representing all the world sectors. The statistics for its sectors, vertices and materials are displayed in the top left-hand corner of the window.

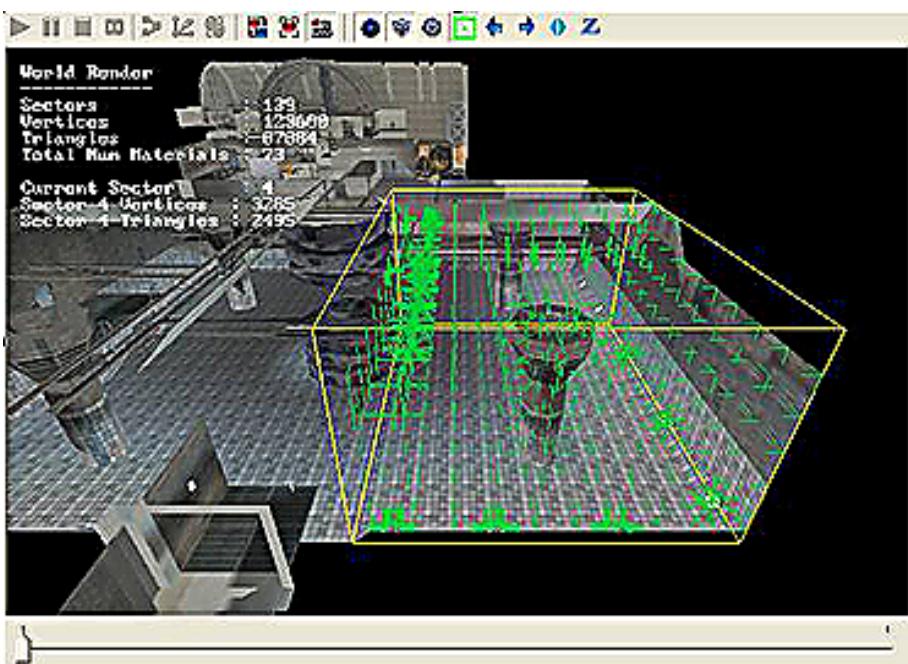


To toggle the rendering of the world, click the **Render world** button .

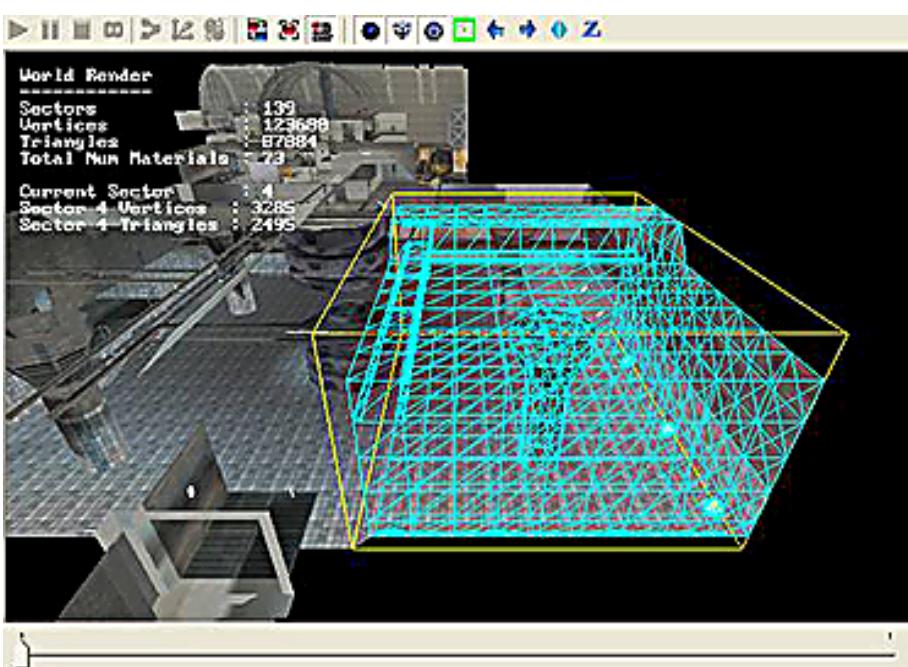
To step through the various sectors in the world, click either the **Next sector**  or **Previous sector**  buttons. To view all the world sectors again, click .

For each sector in the world you can:

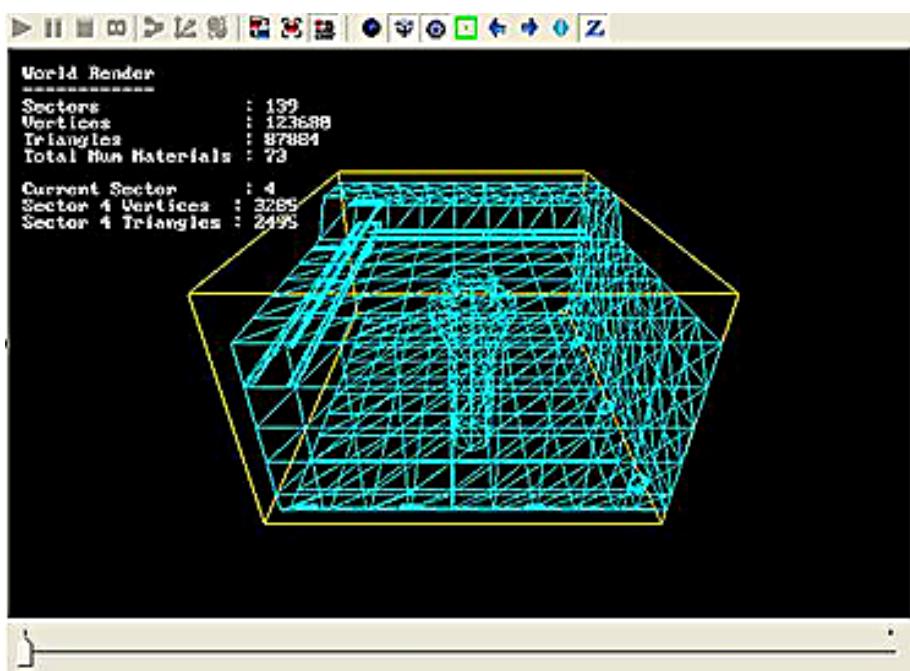
- View the vertex normals of that sector by clicking .



- View the wireframe overlaid on the world by clicking .



- View just the wireframe of the sector without its textures by clicking  again to toggle world rendering off.



## Z-testing

To toggle Z-testing, click . Usually wireframes are overlaid over the surfaces of objects. With Z-testing enabled, the wireframe of objects will be occluded by the textured surfaces of objects.

# Profiling Tools

---

## Memory Stats window

Shows details of the Game Framework's memory allocation, including information such as the total amount used, and the locations where memory is allocated and freed.

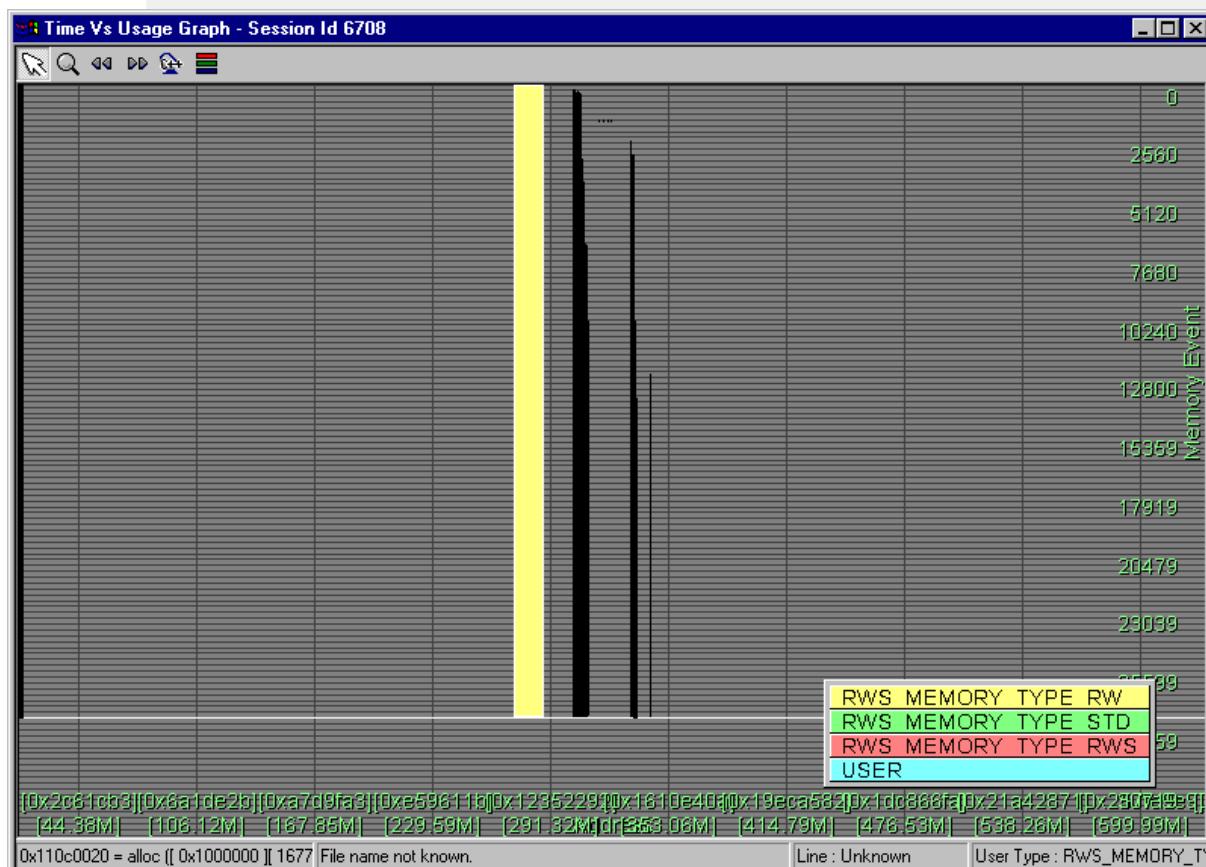
Metrics	Profiler	Memory Stats			
Session Id	Status	Currently Allocated Memory	Total Allocated Memory	Total Freed Memory	
6708	connected	28927 K Bytes	40026 K Bytes	11099 K Bytes	

Like the other profiling tools' windows, Memory Stats contains a list of the sessions under Workspace's control. Simultaneous connections to multiple targets result in multiple items in the list.

Each list item gives a summary of the memory allocated and freed by a session, and there's a context menu that provides four options:

### View Time Vs Usage

Displays a graph of time against memory usage for this session.



**Delete**

Removes this session from the Memory Stats window.

#### Delete All

Removes all sessions from the Memory Stats window.

#### Stop Data Capture

Ends the data capture process for this session.

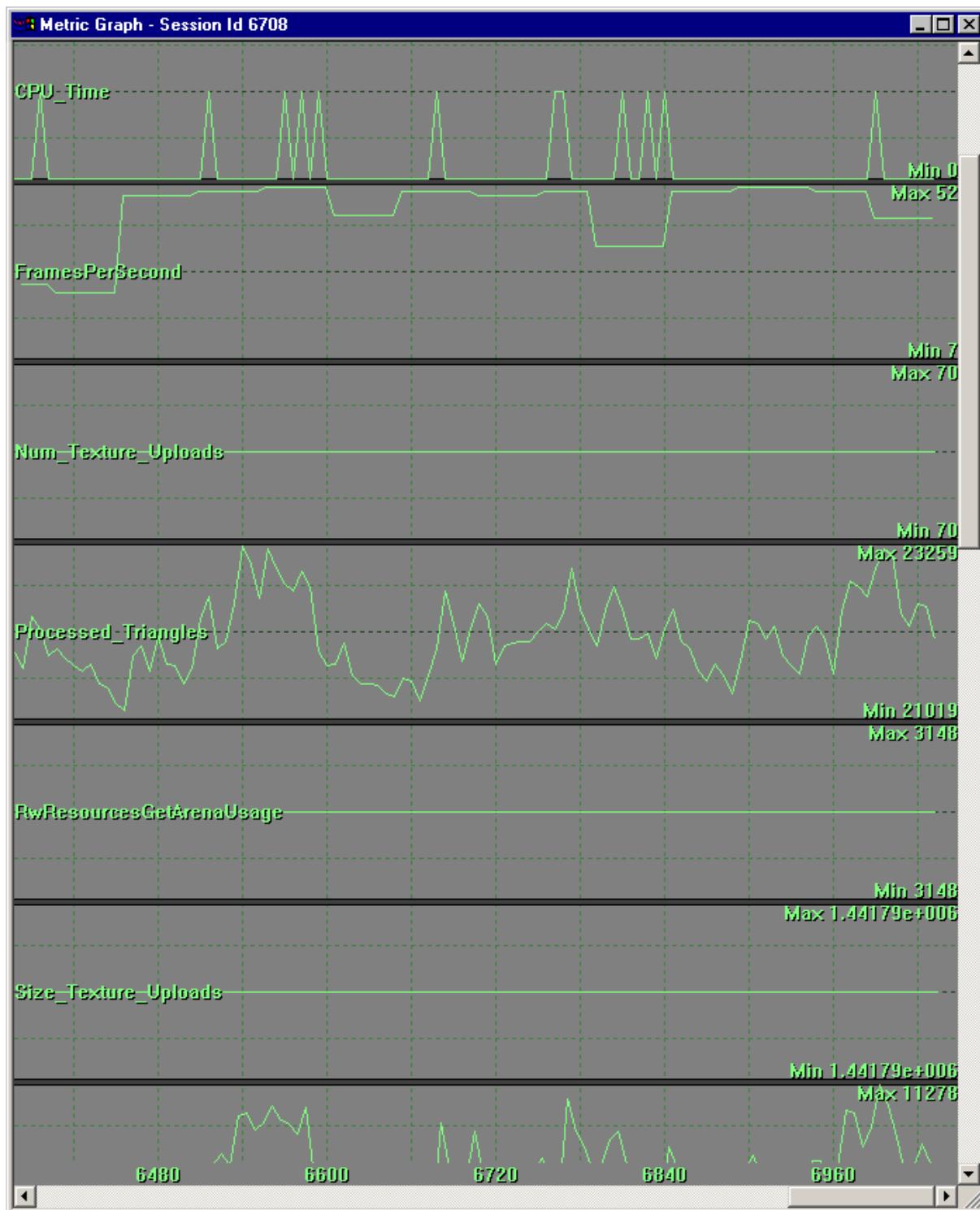
The Time Vs Usage Graph window enables you to find out more about any memory allocation event in the current session, including type, timing, and position in the game's source code.

## Metrics window

Displays custom data (usually, the properties of entities) sent back to Workspace by the game code. Game programmers use macros to specify what data should be sent, and how frequently.

Metrics	Profiler	Memory Stats			
Session Id	Status	Start Time (Frames)	End Time (Frames)	Num Metrics Items	Num Data Elements
6708	connected	0	0	0	0

The Metrics window is driven by the inclusion of [RWS\\_TRACE\\_METRIC\(\)](#) and [RWS\\_SEND\\_TRACE\\_METRICS\(\)](#) macros (p.220) in the game source code. The list view provides a summary, while right-clicking a particular session and selecting **View** displays the Metric Graph window.



## Profiler window

Provides a view of what functions are being called in the game code, and how long each one takes to execute. Game programmers use macros to specify which functions can be monitored in this way.

Metrics	Profiler	Memory Stats
Session Id	Status	
6708	connected	

The Profiler window itself contains only a list of session numbers. Right-clicking and selecting **View**, however, displays the Session Profiler.

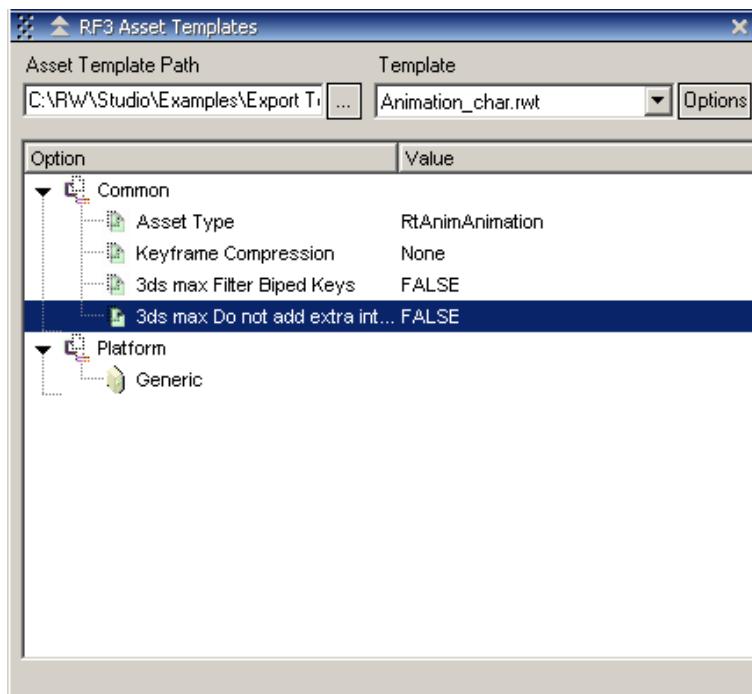
Session Profiler - Session Id 6708 Platform = 'DirectX' Parent = 'Root' Frame = 5930 Total Time = 146.828003					
Function	Calls	This Node Time (Seconds)	Childs Time (Seconds)	Total Time (Seconds)	% Total Time
RWS::MainLoop::Poll	5920	96.318	49.826	146.144	99.53
RWS::InputDevices::WinMouseInput	48	0.001	0.000	0.001	0.00
RWS::Win::IsActive	5925	0.011	0.000	0.011	0.01
MainWndProc	19	0.000	0.042	0.042	0.03
RWS::MainLoop::Open	1	0.000	0.206	0.206	0.14
RegisterImageLoaders	1	0.000	0.000	0.000	0.00
SelectVideoMode	1	0.002	0.000	0.002	0.00
AttachPlugins	1	0.001	0.000	0.001	0.00
GE1::Sound::RegisterAudioObjects	1	0.000	0.000	0.000	0.00

This window displays output from the Game Framework's function profiler, which sends function timings back from the game to Workspace so that programmers can identify bottlenecks in the code. Double-clicking one of the functions in the list displays a further list of functions called by the first function. In this way, you can walk the call stack of the running game.

## RF3 Asset Templates window

---

Allows you to add and edit RenderWare asset exporter templates.

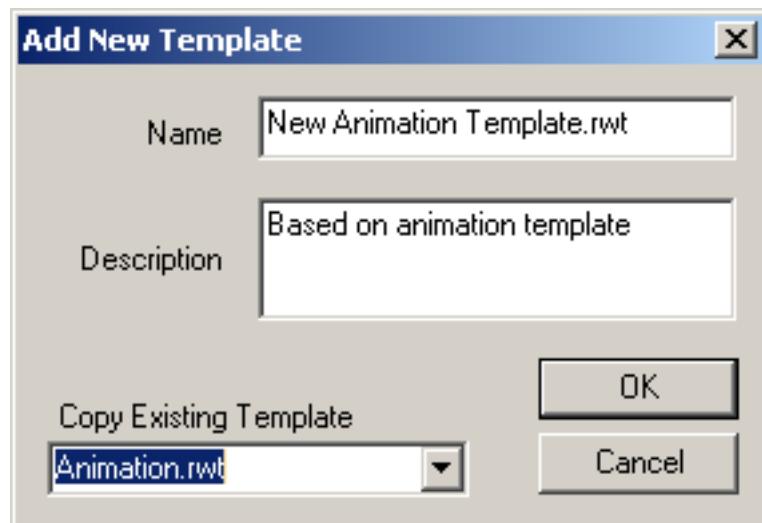


The default path for the exporter templates shown in the **Asset Template Path** is the folder that the RF3 asset templates are installed in when you installed RenderWare Studio. This path can be changed as desired.

For further information on this window, consult the RenderWare Graphics user guide.

**Note:** Changes to the .rf3 files implemented by the RF3 asset templates take precedence over those implemented by the [RF3 project templates](#) (p.311).

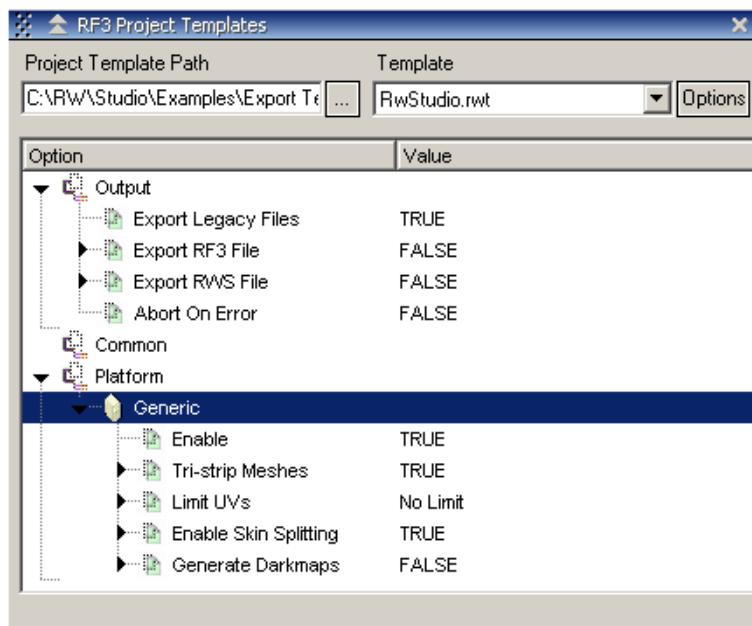
- To edit an existing template:
  - Select one of the items in the **Templates** drop-down list, ensure that the file is not set to read-only.
  - Click over an **Option** to highlight the line.
  - Select one of the values from the drop-down list in the **Values** list.
- To add to the existing templates:
  - Click the **Options** button.
  -



- Type a new template filename and description in the resulting dialog.
- **Copy an existing** template by selecting one from the drop-down list if you want to use one as a starting point.

## RF3 Project Templates window

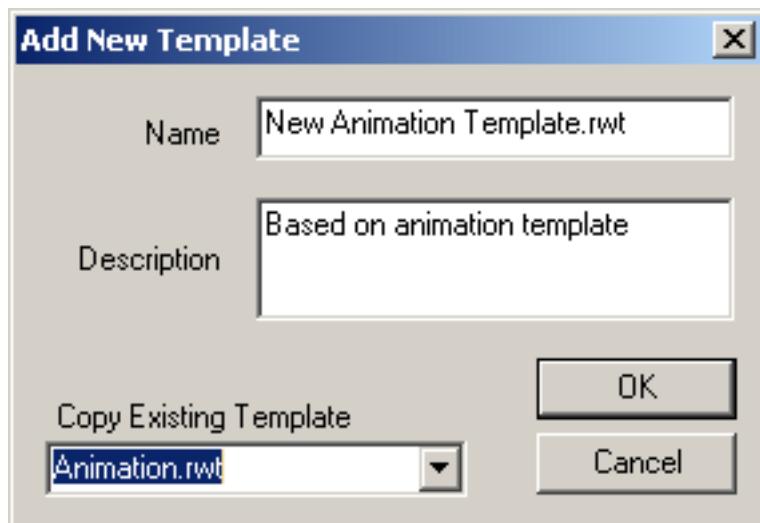
---



The default path for the exporter templates shown in the **Project Template Path** is the folder that the RF3 project templates are installed in when you installed RenderWare Studio. This path can be changed as desired.

For further information on this window, consult the RenderWare Graphics user guide.

- To edit an existing template:
  - Select one of the items in the **Templates** drop-down list, ensure that the file is not set to read-only.
  - Click over an **Option** to highlight the line.
  - Select one of the values from the drop-down list in the **Values** list.
- To add to the existing templates:
  - Click the **Options** button.
  -

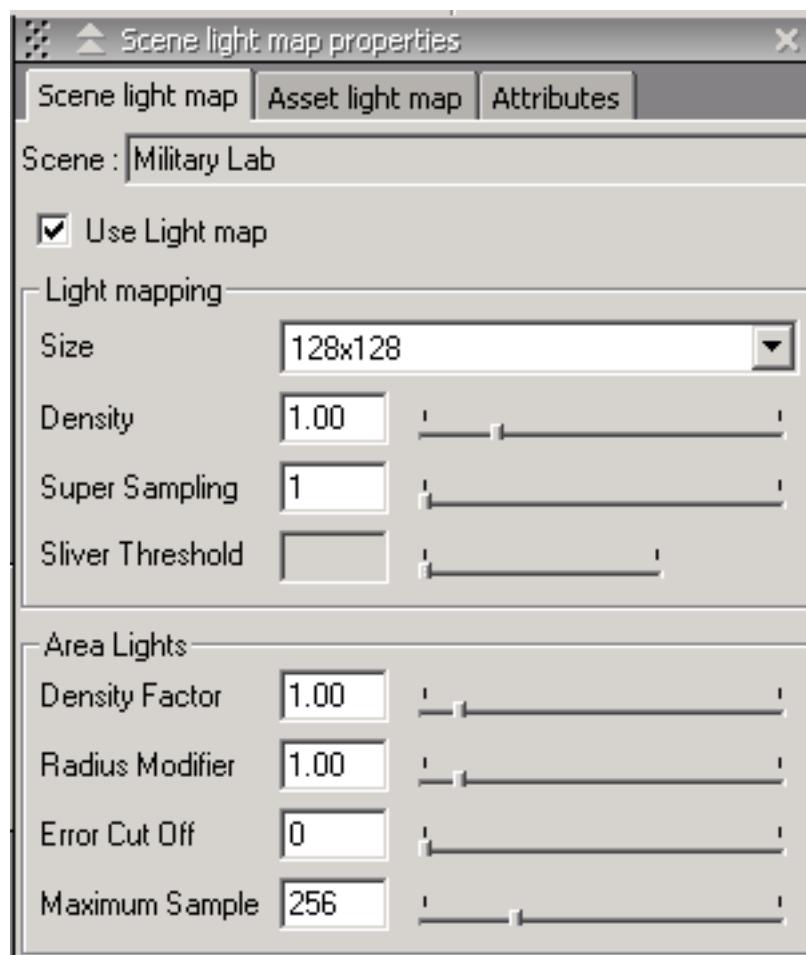


- Type a new template filename and description in the resulting dialog.
- **Copy an existing** template by selecting one from the drop-down list if you want to use one as a starting point.

## Scene Light Map window

Allows use of light mapping to be selected for a world scene. Various light mapping parameters can also be modified in this panel, along with the characteristics of area lights.

The following figure shows the scene light map window:



### **Scene**

This is the name of the scene to which light mapping will be applied.

### **Use Light map**

Check this to enable scene light mapping.

### **Light mapping**

#### **Size**

Size of light map texture created in pixels. Light maps are always square.

#### **Density**

Refers to texel density, this is the average number of pixels in a light map that are applied to a polygon. The higher the density the greater the detail of the light map.

#### **Supersampling**

This is a technique for generating lightmaps of better quality. If the light map

was 128x128 and the supersampling was 2, then the light map would be generated at 256x256 and scaled down to 128x128.

### **Sliver threshold**

Any polygon that is less than the area specified here, will be ignored by the light map generation process. This avoids spending time calculating light mapping for polygon that will be ignored anyway.

### **Area Lights**

When the light map editor finds an area light, it will create several small point lights, within the area light surface, in order to emulate a true area light.

### **Density factor**

This alters the number of point lights for a given area light. The full formula used to calculate the density factor is:

`LightMapping.density × AreaLights.densityfactor ×  
material.arealight.densityfactor`

### **Radius modifier**

This will change the radius of all area lights used in a scene. The radius of an area light is:

`material.arealight.radius × Lightmapping.radius_modifier`

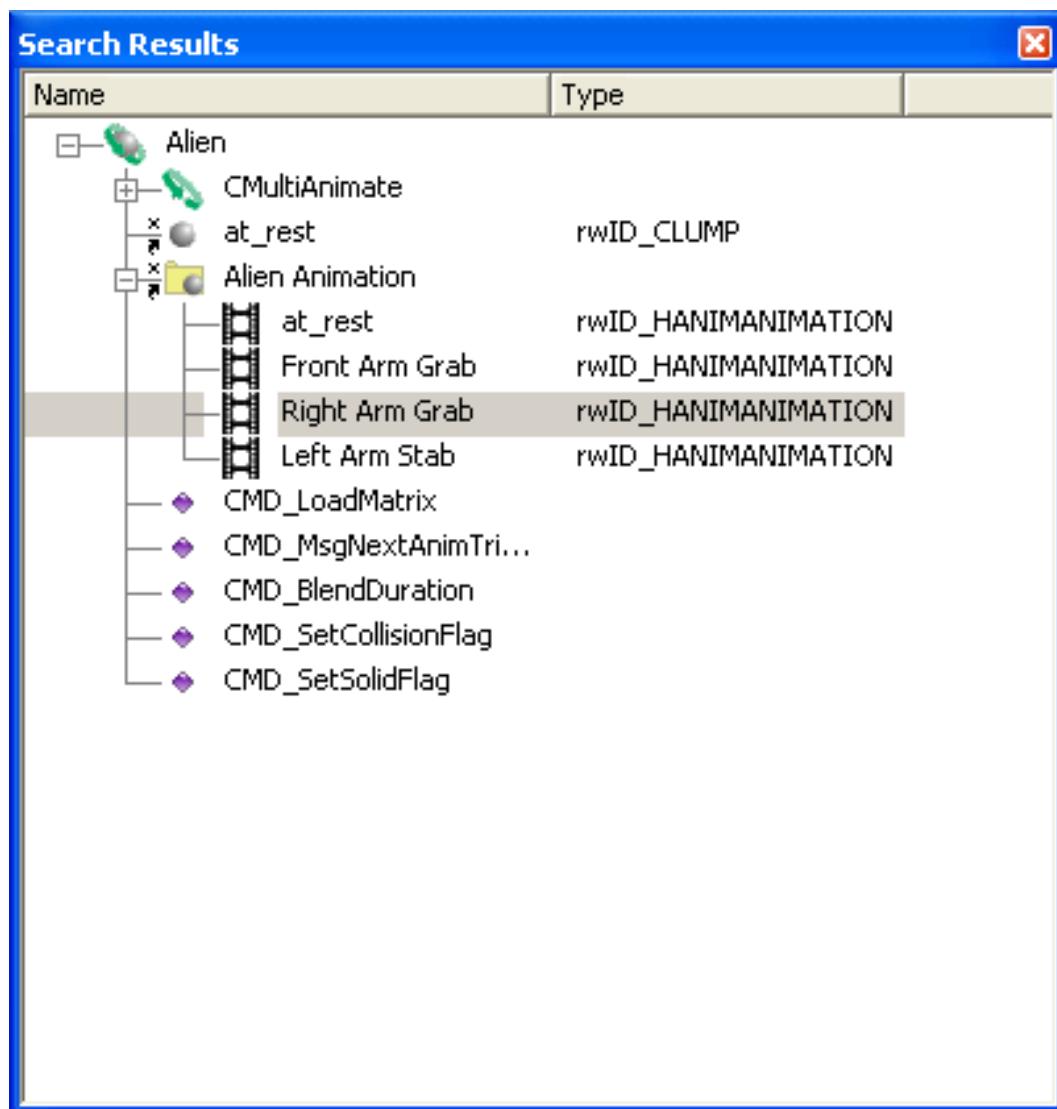
### **Error Cut Off**

This allows the user to sharply cut off the influence of an area light. The larger the value, the sooner the cut off point will be reached. This is generally used to reduce light map generation times.

### **Maximum Sample**

This is the maximum number of point lights used to emulate an area light.

## Search Results window



Lists the objects you have searched for in the Game Explorer, Assets, or Behaviors windows.

To search for entities using a particular object:

1. Right-click in the relevant window on the object (asset or behavior) that you wish to find and select **Search**. Depending on the context you can then choose to search for:
  - all entities using the object
  - only those entities which are “active” - being used in the Design View window - using the object.

**Note:** When searching for assets, you can search for the asset itself or the asset type.

**Author:** Ask Jon what asset types you can search for looks as though sound assets cannot be searched for.

2. Switch to the Search Results window and view the entities listed. (The number of entities found is listed in the Message Log window once the

search is complete.)

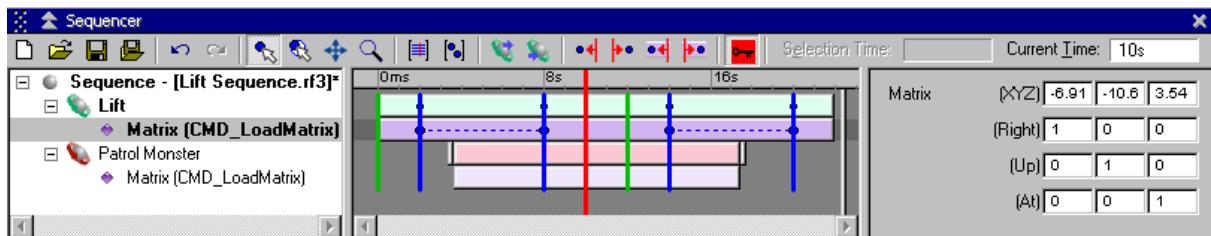
#### Tips

- To refine your search you can right-click on an asset or behavior in the Search Results window and search again. This could be useful where you have searched for an asset type and then wish to refine the results to a particular asset within the asset type.
- You can also locate the entity in the Design View by right-clicking the entity and selecting **Locate in view** or **Aim at in view** from the context menu.

# Sequencer window

---

Allows the construction and configuration of predefined sets of actions that always take place in the same order. In a sequence, the attributes of existing entities may be modified together over time, and new entities may be created and controlled. Displays a timeline with keyframes and interpolations.



The Sequencer window's user interface consists of four elements:

- The left-hand pane, which allows you to control what entities and attributes feature in the sequence
- The central pane, which displays a graphical representation of the current sequence
- The right-hand pane, which shows the “current” value of the selected attribute
- The toolbar, which provides load/save functionality, editing options, and event control features

## Left-hand pane

The left-hand pane of the Sequencer window contains a list of the entities involved in the current sequence. Each entity in the list can be expanded to show the attributes that may be modified during the sequence:

- Entities with green bands were dragged into the pane from Game Explorer. They existed in the game before the sequence began, and will continue to exist after the sequence has finished.
- Entities with red bands were dragged into the pane from the Templates window. They are *temporary* entities that exist only while a sequence is being executed. Such entities are created and destroyed by the sequence, and can have lifetimes shorter than that of the sequence.

In addition, entity names in the left-hand pane can appear in **bold** type to represent that “updates” are enabled. When this is the case, scrubbing the playhead in the central pane causes changes that take place in the sequence to be reflected immediately in Design View. If a sequence changes an entity’s position, for example, you can use this feature to preview that movement in Workspace.

## Central pane

The central pane contains shaded horizontal bars that indicate the lifetimes of the

entities in the sequence:

- Green bars represent green-banded entities. They are always the same length as the sequence itself.
- Pink bars represent red-banded entities. They are created with the same length as the sequence, but have handles at each end that allow for resizing.
- Purple bars represent the attributes of both entity types. An entity can have one or more of its attributes changed over the course of a sequence, so there can multiple attribute bars beneath a single entity bar. These bars can be overlaid with further symbols:
  - Blue circles mark the *keyframes* on each bar, where the designer dictates that an attribute should have a particular value at a particular time.  
A large blue circle on an attribute bar is echoed by a smaller blue circle on an entity bar, so that the location of the keyframe is clear even when the display is collapsed.
  - Dashed lines represent interpolations (transitions) from one keyframe to the next, during which an attribute's value changes smoothly from one setting to another.

The central pane can also contain four kinds of vertical line that all have to do with time and timings:

- The gray line marks the end of the sequence. It can be dragged left or right to make the sequence shorter or longer.
- The red line is the *playhead*, which represents the instant in the sequence that's currently under scrutiny.
- Vertical blue lines show the positions of events being sent by the sequence, to be received by any entity in your game.
- Vertical green lines show positions where the sequence will pause and wait to receive an event from another game entity.

## Right-hand pane

The right-hand pane provides one way of displaying and editing the value of the attribute that's currently selected in the left-hand and central panes.

## Toolbar

The Sequencer window's toolbar is divided by function into seven sets of buttons, starting with the standard “new, open, save, save as” set that's typical in Windows applications. These buttons allow for sequence assets to be edited independently of the project currently open in Workspace.

The next two buttons in the row also perform familiar operations. They are



**Undo** and



**Redo**, with the latter button not becoming active until the former

has been clicked at least once.

The four buttons that come next allow you to set the behavior of the Sequencer window's central pane:



**Edit Mode (Move)** is the default mode, and the one you'll use most of the time. It allows you to make changes to the content and character of a sequence, as described in *Editing keyframes*, below.



**Edit Mode (Scale)** offers similar functionality to **Edit Mode (Move)**, with different behavior under certain circumstances. Again, this is described in *Editing keyframes*.



**Scroll Mode** switches off the Sequencer window's editing features, so that clicking and dragging in the central pane simply scrolls the sequence in the view.



**Zoom Mode** also switches off editing. With this button selected, clicking and dragging up and down in the central pane changes the scale of the timeline, to show more or less detail.

**Tip:** You can also change the scale of the timeline in either of the edit modes, by placing the cursor anywhere in the central pane and using the mouse wheel to "zoom" in and out.

The next two buttons provide a quick way of zooming the display in the central pane to a sensible setting:



**Zoom Extents** zooms the view so that the whole sequence is visible in the central pane.



**Zoom Keyframe/Event Extents** zooms the view so that all keyframes and events are visible in the central pane.

The next two buttons are linked to RenderWare Studio's event-handling mechanism and allow you to set points in a sequence where events should be sent or received:



**New Transmit Event** opens the Event Editor dialog, where you can type the name of the event you wish the sequence to send.



**New Wait For Receive Event** also opens the Event Editor dialog, this time so that you can specify the name of the event you wish the sequence to wait for before proceeding.

After the event buttons comes a block of four that make moving the playhead in the central pane quicker and more accurate:



**Previous Keyframe/Event** moves the playhead to the time of the previous keyframe or event, regardless of the timeline that contains it.



**Next Keyframe/Event** moves the playhead to the time of the next keyframe or event, regardless of the timeline that contains it.



**Previous Keyframe from Selected Timeline** moves the playhead to the time of the previous keyframe in the selected timeline.



**Next Keyframe from Selected Timeline** moves the playhead to the time of the next keyframe in the selected timeline.

**Tip:** The functionality of the first two buttons can also be achieved by pressing **Ctrl** while dragging the playhead. Rather than moving smoothly, the playhead “snaps” to keyframes and events.

The last button in the toolbar is a toggle that can prevent you from creating keyframes accidentally in some circumstances:



**Toggle Design View Keyframe Generation** controls whether a keyframe is generated automatically when you change the value of an attribute that's under the control of the Sequencer. Usually, you'll want this behavior; when you don't, this button is the way to deactivate it.

## Creating and editing a sequence

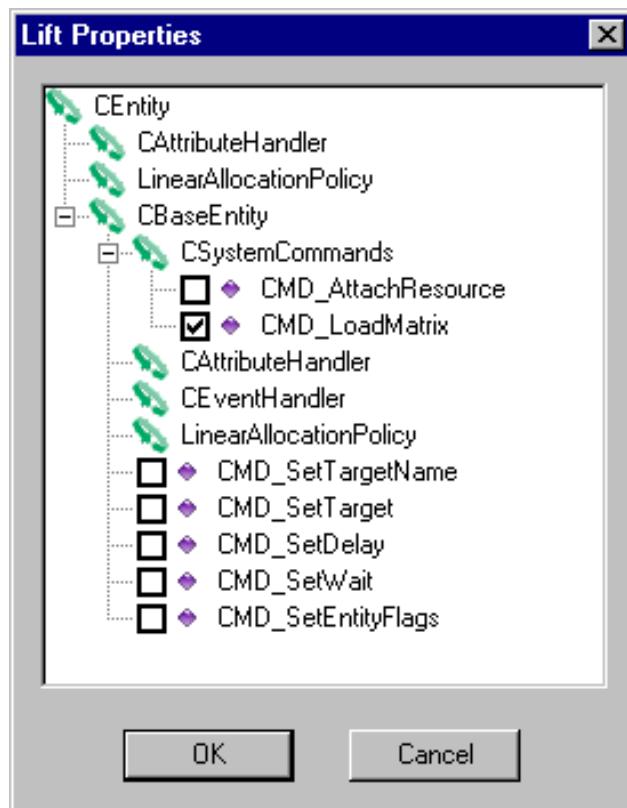
With a game project open in Workspace, a typical set of actions for creating a sequence and opening it for editing is:

1. Create a sequence asset using the context menu in the Assets window.
2. Create an entity from that asset.
3. Edit the asset by right-clicking in Game Explorer.

At this stage, with the sequence open for editing in its window, Sequencer offers a range of operations.

## Adding entities and attributes

To have an entity (or a template entity) take part in the sequence, drag it from the [Game Explorer](#) (p.288) (or the [Templates window](#) (p.334)) to the left-hand pane of the Sequencer window. This displays a property window that allows you to choose which of the entity's attributes may be modified during the sequence:

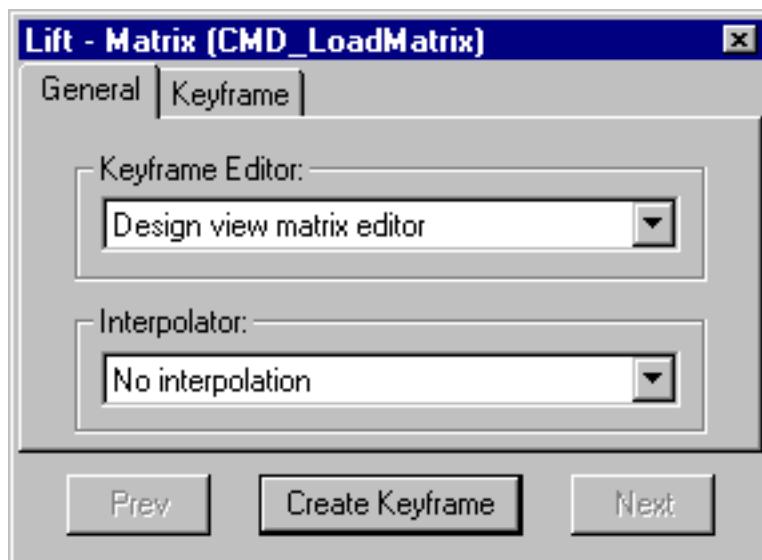


If you decide that you need to remove from or add to this list later in the process, right-click the entity's name in the left-hand pane and select **Properties** to display the same window.

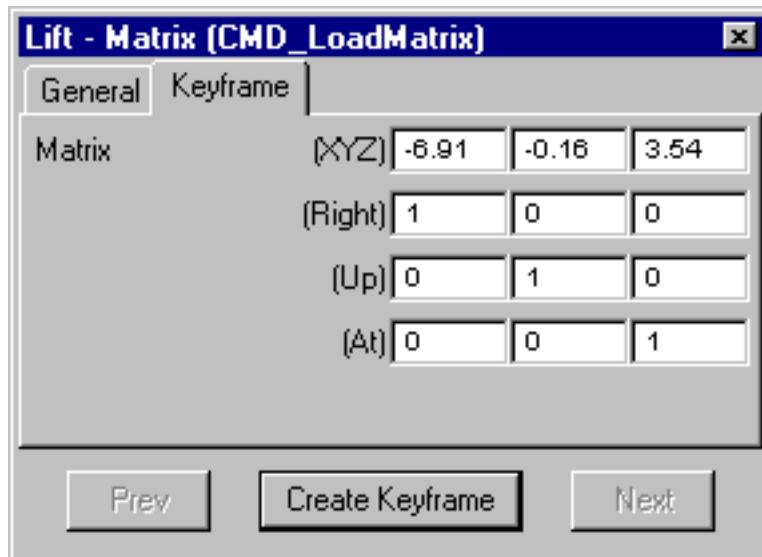
## Creating keyframes

Sequencer provides several ways of creating keyframes, all of which require you to select the **Enable/Disable Keyframe Generation** button on its toolbar, and then to drag the playhead to your preferred position in the central pane. (Alternatively, you can type a value into the **Current Time** box.) At this point, with an attribute selected in the left-hand pane:

- Type (or otherwise select) values in the right-hand pane. As soon as you make a change, a keyframe is created using the default keyframe editor.
- (For positional attributes) select the entity that contains your attribute in Design View and move it around. A keyframe is created in which the attributes have their values at the end of the move.
- Double-click the attribute name in the left-hand pane, or right-click it and select **Properties**. In the dialog that appears, choose the **Keyframe Editor** to use for this attribute, and the type of **Interpolator** to be used between this keyframe and the next:



To create a keyframe with the attribute's current value, click **Create Keyframe** at this point. Otherwise, select the Keyframe tab, and provide a new value:



What you see on this tab depends on the attribute you're editing. In this example, we're controlling the position of the entity in the game by adjusting its *Matrix* attribute.

- Right-click on an attribute's timeline in the central pane and select **Create Keyframe**. A keyframe is created at the playhead's position, with the attribute's current value.

## Editing keyframes

Editing a keyframe can mean changing its position, its value, or its relationship with other keyframes.

- Sequencer provides for keyframes to be cut, copied, pasted, and deleted—alone or in groups, through multiple selection. First, you must select the keyframe(s) you want to manipulate by clicking or dragging, so

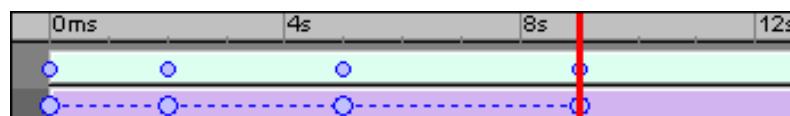
that the blue circle (●) becomes shaded (○). Then:

- Cut the keyframe by pressing **Ctrl+X** or right-clicking and selecting **Cut**.
- Copy the keyframe by pressing **Ctrl+C** or right-clicking and selecting **Copy**.
- Paste a previously cut or copied keyframe by selecting an attribute, placing the playhead, and then pressing **Ctrl+V** or right-clicking and selecting **Paste**.
- Delete the keyframe by pressing **Delete** or right-clicking and selecting **Delete**.
- Reschedule the keyframe by dragging it along the timeline, or typing the new time into the **Selection Time** box.
- To change the value of an attribute at an existing keyframe, you start by putting the playhead at the location of the keyframe you want to edit. This is done through the **Current Time** box, the toolbar buttons, the **Prev** and **Next** buttons of the attribute's edit dialog, or by **Ctrl**-dragging the playhead.

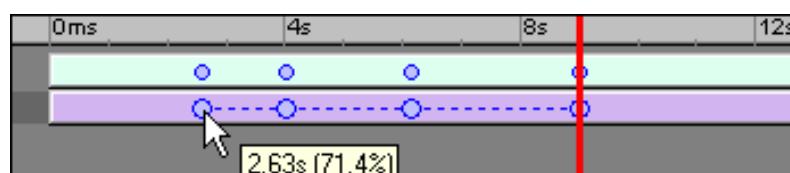
When you have the right attribute selected and the playhead in the right place, *changing* the value follows the same model as *setting* the value when you created the keyframe in the first place.

- Relationships between keyframes can be divided into two categories: their separation, and the interpolation (if any) that takes place from one to the next.
  - To define an interpolation between two keyframes, move the playhead to any point between them. Then, you can set the **Keyframe Editor** and **Interpolator** separately by right-clicking and selecting the items with those names in the central pane's context menu. Alternatively, you can set them together, on the General tab of the attribute's edit dialog.
  - To change the separation of two or more keyframes, drag them along the timeline. The effect this has can depend on whether you've chosen **Edit Mode (Move)** or **Edit Mode (Scale)**.

In “move” mode, all selected keyframes (together with any interpolations) are simply translated along the timeline; the separation between selected keyframes does not change. In “scale” mode, on the other hand, the current playhead location is used as an anchor point for the operation.



The selected keyframes act as though they're on a piece of elastic. Dragging any one of them causes the others to move in proportion to their distance from the anchor.



## Previewing a sequence

By default, when you have a sequence open for editing in the Sequencer, and the attributes you're affecting have a visual manifestation (such as an entity's position), scrubbing the playhead back and forth previews any changes in the Design View window.

You can switch off this behavior on an attribute-by-attribute (or entity-by-entity) basis by right-clicking in the left-hand pane and selecting **Disable Updates**. If you want to focus on one or two attributes, ensure that only those attributes have updates enabled.

## Sending and receiving events

To make your sequence send or wait for an event, drag the playhead to your preferred position and click the **New Transmit Event** or **New Wait For Receive Event** button.



The combo box on this dialog allows you to select an event from a list, or to type a new name of your choosing.

The vertical bars that represent events in Sequencer's central pane can be manipulated in the same way as the circles that represent keyframes. They can be cut, pasted, moved, scaled, and so on.

The topic on [adding a sequence to your game](#) (p.180) provides further information about sequences, including an explanation of how to set a sequence in motion.

## Stream viewer window

---

Provides a way to look inside RenderWare streams, and attach custom editors to handle the different types of data, such as MatFx and RpToon data, that have been attached as plug-ins to assets.

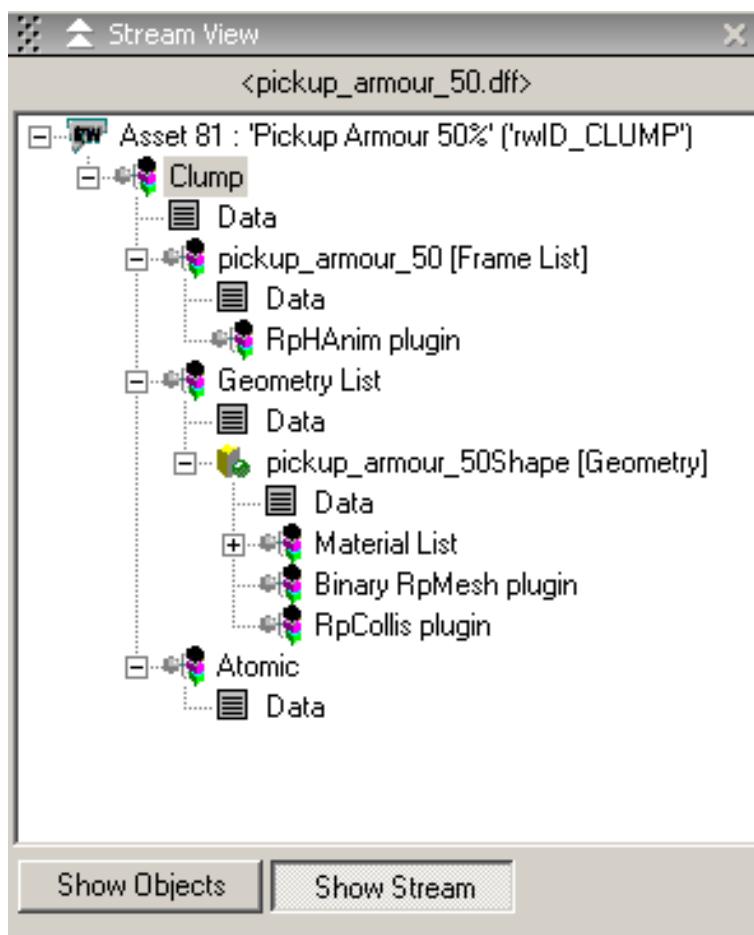
Edits made to the stream data are saved in the asset's XML file, allowing them to be re-applied if the artwork for the asset is exported again.

To see the stream viewer in action:

1. Right-click an asset in the Game Explorer, Design View or Assets window, and then select **Show Asset Stream Contents** from the context menu. The Stream View window appears showing the contents of the asset's file or the contents of the files which together make up a .rf3 file.

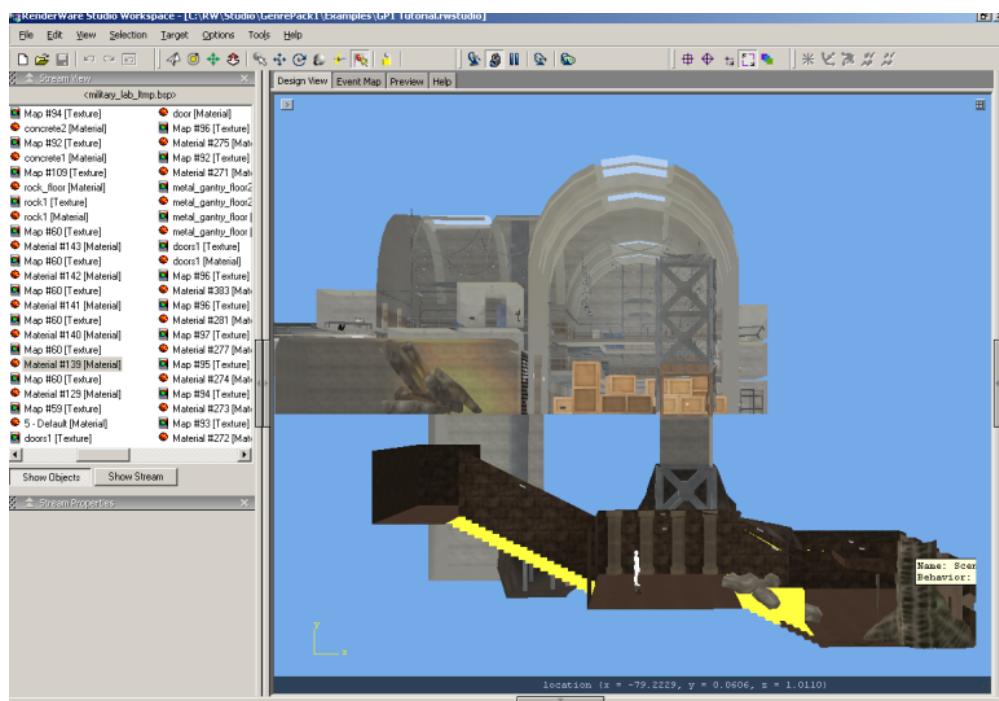


2. Right-click in the Stream View window and select **Show Object Type**. You can then choose from a drop down list to filter the objects displayed by object type. To show only those objects which were given identifiers when the asset was exported from the art package, right-click and select **Hide unnamed objects**.
3. Click the **Show Stream** button to see the objects arranged in the same hierarchy as in the asset file.



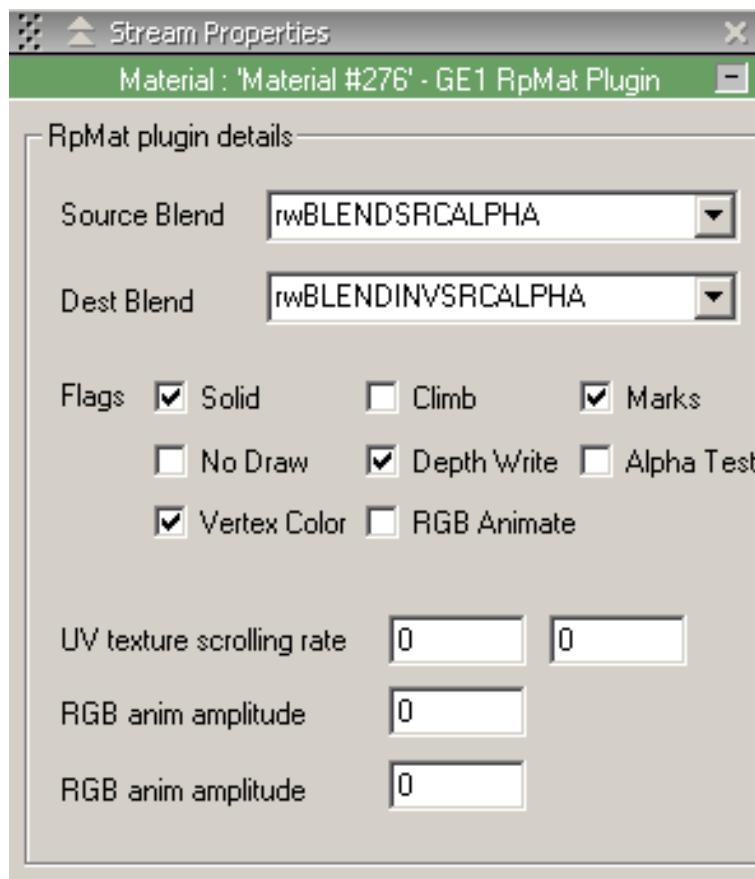
Where an asset has too many material-type objects to be certain of the exact name of the material that you wish to edit, you can click on it in the Design View window using the **Pick Materials** toolbar button

The selected material will then be highlighted in yellow in the Design View window and automatically located in the Stream View window. Once you are using the **Pick Materials** button, the contents of the Stream View window are automatically updated when different assets are selected in Design View, there is no need to select **Show Asset Stream Contents**.



This also works in reverse so that when you select a material in the Stream View window, any visible geometry displaying that material in the Design View will be selected.

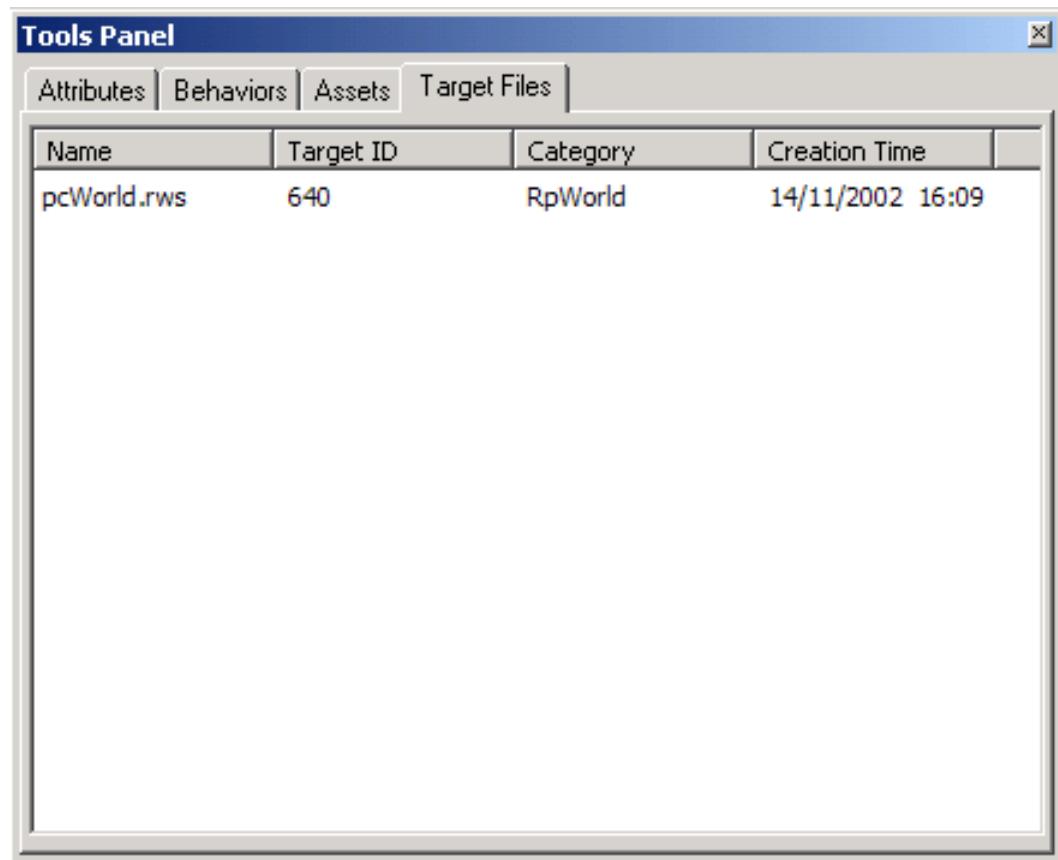
4. To edit an object, select it in the Stream View window, and then edit it in the Stream Properties window.



**Note:** The edits to the asset's plug-in data that you have made will be stored in the asset's XML reference file in the game database. This means that each time the game is built and sent to the console those edits will be included "on the fly" as the asset file is included in the stream that is sent to the console.

## Target Files window

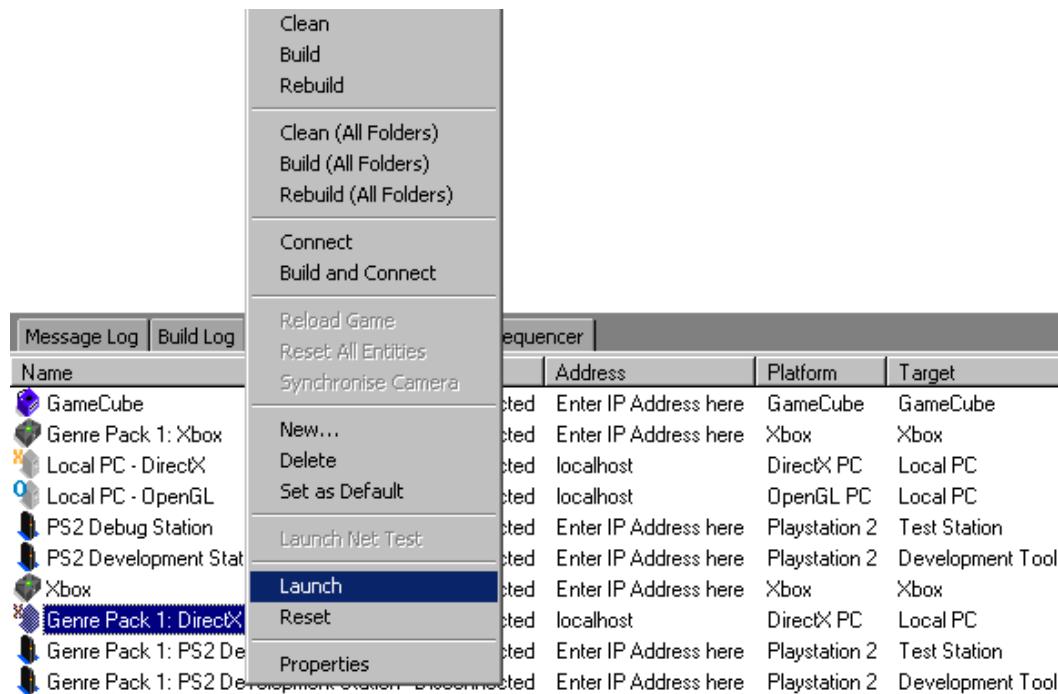
Lists files that the target has “saved” (sent via the network) to Workspace. To save one of these files to a permanent file on your hard disk, right-click it and then select **Save As....** Otherwise, these files last only for the current session.



## Targets window

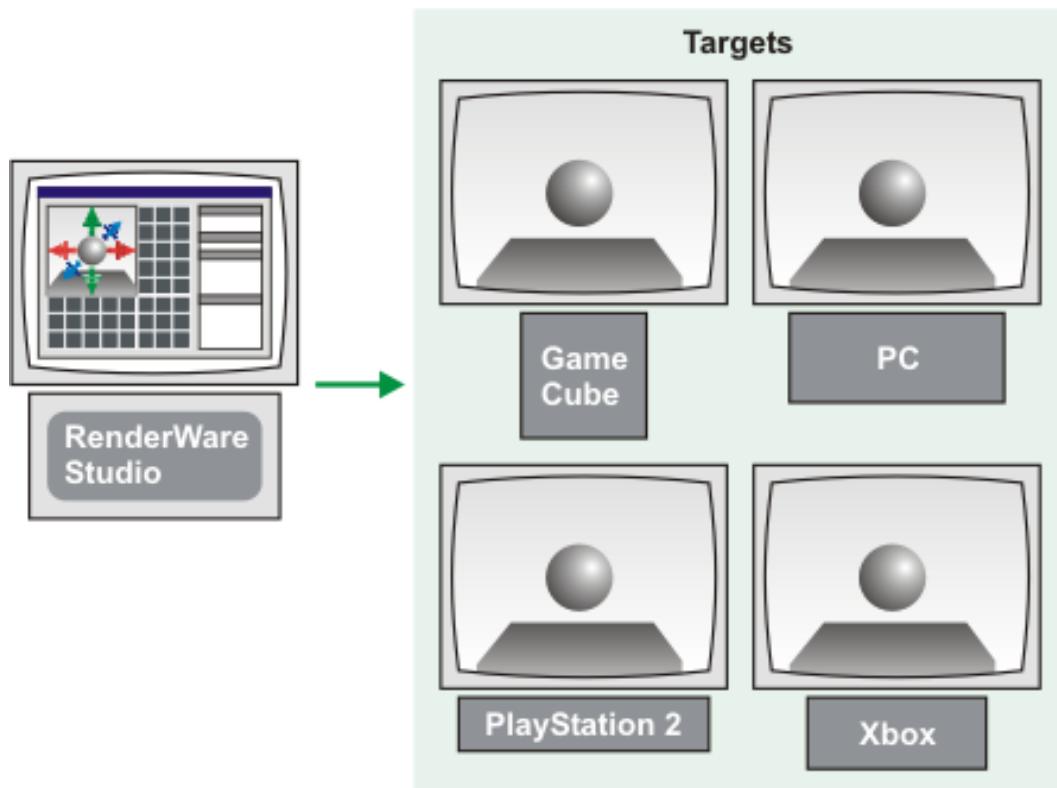
---

Lists the targets (consoles or PCs) on which you can run your game. You can use this window to build the game data stream for a target, then connect (send the game data stream) to the target, so that you can dynamically view the game as you develop it.

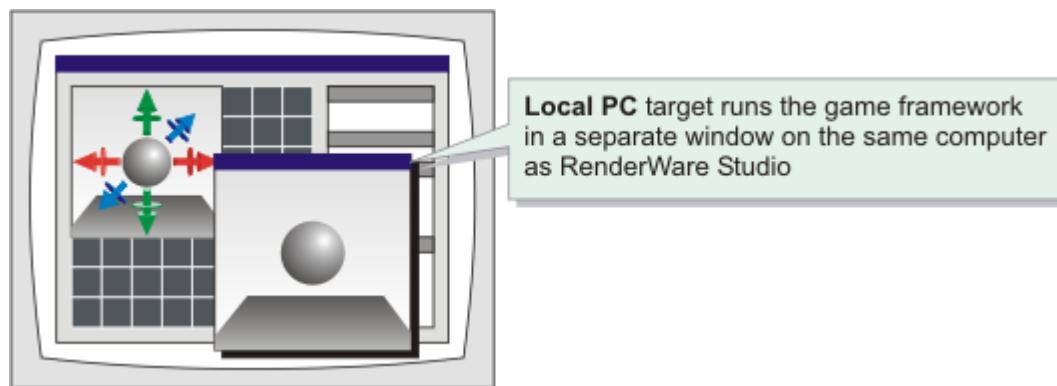


The information displayed in this window is defined in the [RWStudio.settings](#) (p.185) file, in the RenderWare Studio Programs folder. Each column heading in this window is defined by a `<column>` element; each target is defined by a `<row>` element, with `<field>` elements defining values for each column.

While you are developing a game, you can view it simultaneously on several different targets:



You can also view the game in a separate window on the same computer as RenderWare Studio:



While viewing the game on a target, you can continue using RenderWare Studio Workspace to develop the game. Any changes you make (such as adding, moving or deleting entities, or changing behavior attributes) are dynamically reflected on the target.

## Context menu

To view the actions that you can perform in the Targets window, right-click one of the connection names. The following context menu appears:

### Clean

Deletes the stream files and other intermediate files created by the Game Production Manager (in the `Build Output\connection name\...` folders under your project).

### Build

Updates any out-of-date intermediate files and then (if there were any updates), creates new stream files (`Build Output\connection name\Folder\*.stream`) for the global folder and the active folder.

### **Rebuild**

Cleans, then builds.

---

### **Clean (All Folders)**

Cleans all folders in the project, not just the global folder and the active folder.

### **Build (All Folders)**

Builds all folders in the project, not just the global folder and the active folder.

### **Rebuild (All Folders)**

Rebuilds all folders in the project, not just the global folder and the active folder.

---

### **Connect**

Connects to the target using the last built Renderware stream.

### **Build and Connect**

Builds, then sends the stream files for the global folder and active folder to the target. Any subsequent changes are also sent to the target, until you disconnect.

**Tip:** Double-clicking the word “Connected” or “Disconnected” in the **Status** column has the same effect as selecting this option.

### **Disconnect**

Stops sending game data to the target.

---

### **Reload Game**

Stops and restarts the game, and then resends the game data stream.  
(Equivalent to disconnecting, and then connecting.)

### **Reset All Entities**

Sends all entities and their attributes, but without sending their assets. This is a quick method of restarting the game if no assets have changed.

### **Synchronize camera** (only enabled for connected targets)

Ensures that the camera position shown on the target matches the position of the camera in the RenderWare Studio Workspace.

### **Director's Camera**

Overrides any camera control behaviors in the Game Framework. Instead, the console view follows the movements of the camera in Workspace. For example, in a first person game, the camera typically follows the movements of the player. If you select the Director's Camera, then, although you can still use your console's control pad to move the player, the console view moves with the Workspace camera, not the player.

---

### **New...**

Creates a new target, and displays a dialog where you can set the target properties.

### **Delete**

Deletes the selected target.

**Set as Active**

Sets the current target as the default to launch when you select **Launch** from the **Target**[main menu](#) (p.244) or the build toolbar.

**Launch Nettest**

Launches the [Nettest tool](#) (p.224) (only if the **Address** of the Target is *localhost*).

**Launch**

Starts the console emulator application.

**Properties**

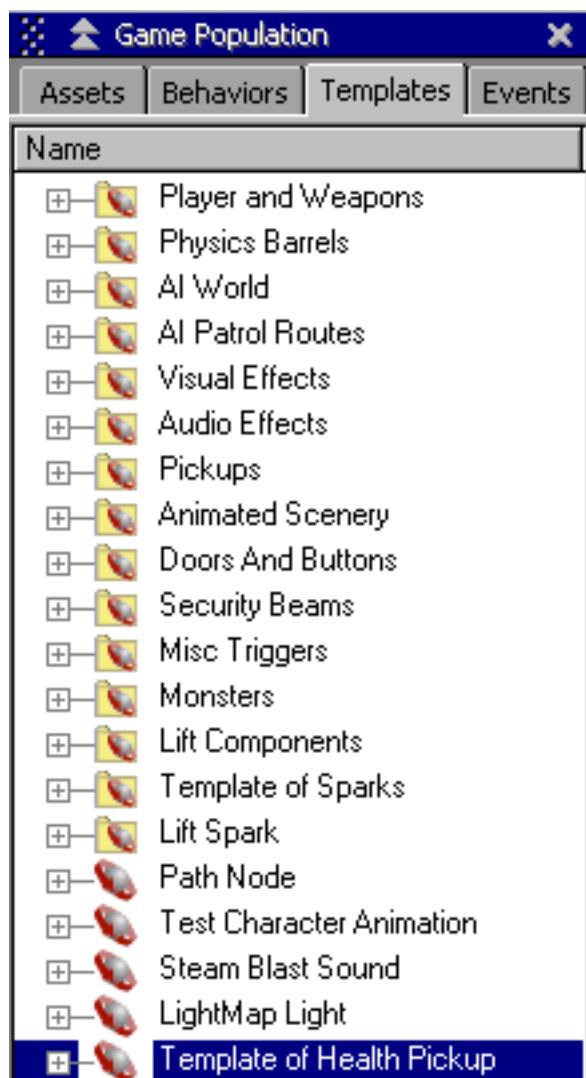
Sets the target's properties.

**Tip:** Double-clicking in the **Target** column has the same effect as selecting this option.

You can define additional, [custom context menu items](#) (p.208) for a target platform.

## Templates window

Lists the templates that you have created in the Game Explorer window.



# Saving layouts

---

You can create a [layout](#) (p.238) for each of the different ways you use Workspace, and switch between layouts while using Workspace.

For example, when importing artwork and creating assets in Workspace, you might want a different arrangement of windows than when you are testing a running game. Creating different layouts enables you to customize Workspace for these tasks.

To create a new layout:

1. Select **View ▶ Clone Current Layout**.
2. Enter a name for the new layout.

This is now the current layout; any changes that you make to the window arrangement affect only this layout.

3. [Arrange the windows](#) (p.238).

The new layout is saved only when you close Workspace.

To switch between layouts, select **View ▶ Layouts**.

To restore the current layout to its most recently saved state, select **View ▶ Restore Current Layout**.

To restore all supplied layouts to their original states, select **View ▶ Restore to Default Layouts**.

## Where layouts are saved

Layouts are saved in `.Layout` XML files in the `Programs/RenderWareStudio` folder (and any of its subfolders) under your RenderWare Studio installation folder. For details, see the Enterprise Author documentation.

Each time you close Workspace, the `.Layout` files are updated with any changes that you made during that session.

## Distributing layouts to other users

To make your layout available to another RenderWare Studio user, copy your `.Layout` file to their `Programs/RenderWareStudio` folder. The next time they start Workspace, the layout will appear in their **View ▶ Layouts** menu.

To allow users to undo changes they make to your layout, copy your `.Layout` file to their PC with a `.LayoutDefault` file extension. Any changes they make to your layout will be saved to a new `.Layout` file in the same folder as the `.LayoutDefault` file. To restore your original layout, they can select **View ▶ Restore Default Layouts**.

# Tutorials

---

These tutorials step you through typical tasks:

**First steps** (p.86)

Uses the tutorial project supplied with RenderWare Studio to demonstrate basic tasks. If you have not used RenderWare Studio before, then this is a good place to start.

**Creating behaviors** (p.337)

Shows you how to add new behaviors to the Game Framework. (Requires knowledge of C++ programming.)

**Creating new attribute editor controls** (p.382)

Shows you how to create a new user interface control for editing attributes. (Requires knowledge of C++ programming.)

# Creating behaviors

---

## In this tutorial

We show you how to create RenderWare Studio behaviors. A behavior provides logic that you can attach to entities in your game. We will also see how to create events and trigger them, and how to expose a behavior's attributes in the RenderWare Studio Workspace.

This tutorial consists of nine lessons:

1. Creating an “empty” behavior.
2. Making the behavior perform an action (in this case, spin its associated entity).
3. Editing the behavior attributes in real-time, using the RenderWare Studio Workspace.
4. Registering and sending a custom event.
5. Receiving and processing a custom event.
6. Sending and receiving a custom event within one behavior.
7. Using all the behavior attribute editor controls.
8. Setting the message in a derived class to be handled by the base class.
9. Writing a behavior “from scratch”, without using the CAtomicPtr object.

If you want to begin with a complete technical understanding of behaviors, then you might want to complete [lesson 9](#) (p.366) first. Lessons 1-8 create behaviors that rely on a CAtomicPtr object. This “smart pointer” handles some of the functionality required to implement a behavior, hiding the underlying RenderWare Graphics processing. You might want to understand the underlying processing first, and then move on to using CAtomicPtr in lesson 1.

## What this tutorial covers

- The base classes from which you must derive.
- Handling events.
- Processing attributes.
- Creating interface controls for the RenderWare Studio workspace so that they can be edited in real time from a PC.
- Signalling custom events.
- Acting upon custom events.

## What you need to know

This tutorial assumes that you are familiar with C++ and that you have set up your RenderWare environment (see the RenderWare documentation for how to do this).

This tutorial does not describe how to use the RenderWare Studio Workspace, so

you need to be comfortable with the process of placing entities in the world, assigning behaviors to them and altering their attributes.

## Location of files for this tutorial

The files that you will need for this tutorial are supplied with your RenderWare Studio distribution in the following folders:

Description	Path
Primary asset file for world	C:\RW\Studio\Examples\Tutorial\maps\fountain\fountain.bsp
Primary asset file for “spinner” entity	C:\RW\Studio\Examples\Tutorial\objects\spinner.dff
Project file	C:\RW\Studio\Examples\Tutorial\Tutorial.rwstudio
C++ source	C:\RW\Studio\console\game_framework\source\modules\Examples\Tutorial\Tutorial.cpp C:\RW\Studio\console\game_framework\source\modules\Examples\Tutorial\Tutorial.h

**Note:** C:\RW\Studio is the default location for RenderWare Studio. You might have installed RenderWare Studio in a different folder.

## Before you begin

RenderWare Studio extracts all the information for displaying a behavior's interface controls from the source code of that behavior. The C++ parser in RenderWare Studio looks through the source files for class declarations, notes the classes they are inherited from, and recognizes a few key framework classes. When a class is recognized as being framework-compatible, it is scanned for special macros that define the commands that the behavior can accept (we'll use these in lessons 3 and 7).

Each of the supplied C++ source files reflect the code as it should be at the end of a particular lesson.

You can use this tutorial in two ways. Either:

- Create new compilable source files based on the code in this tutorial.  
**Note:** This tutorial contains relevant code snippets only. For the complete code, see the supplied source files.  
or
- Use the tutorial as a reference for browsing the supplied source files.

If you plan to create new source files, then create a new folder under the Tutorials folder (for example, **Walkthrough**). This is the folder into which you should save the source files you are going to create. The behaviors you write must be compiled into the target project, so you need to add them to the project (for example, in CodeWarrior or Visual C++).

During the lessons, it is a good idea to refer to the source code files at each step, as they contain the fully working behaviors for your reference.

In each lesson, we will create a new behavior called CTutorialx, some of which will be based on the code we created in the previous lesson.

Start the tutorial now: [Lesson 1. Creating a behavior](#) (p.341).

# Lesson 1. Creating a behavior

---

## In this lesson

We create a basic behavior that does nothing more than appear in the RenderWare Studio behavior list. You can attach this behavior to an entity in the world, but it won't do anything yet.

**Note:** This lesson creates a behavior that relies on a `CAtomicPtr` object. This “smart pointer” handles some of the functionality required to implement a behavior, hiding the underlying RenderWare Graphics processing. [Lesson 9](#) (p.366) demonstrates how to create a similar basic behavior, without relying on a `CAtomicPtr` object.

For your behavior to be picked up by RenderWare Studio, its class must be derived from `CAttributeHandler`. The `CAttributeHandler` class defines the interface that you must implement, and provides the necessary logic to integrate your class into the framework.

If you want the behavior to interact with other behaviors, then you can also inherit from `CEventHandler`.

If you want the behavior to process position updates and resource attachments from RenderWare Studio, then you can also inherit from `CSystemCommands`.

`CEventHandler` provides the mechanism to route event messages to the correct destination for processing. Any object that is derived from `CEventHandler` can receive events because of the pure virtual `HandleEvent` method. When writing a behavior, you must override `HandleEvent` to implement your logic.

`CAttributeHandler` allows an object to receive updated attributes from the framework. It contains a pure virtual member function `HandleAttributes` that receives a list of attributes that should be updated.

`CSystemCommands` defines the basic commands (usually, a command sets an attribute) that RenderWare Studio requires to manipulate the entity within the game world.

Our first behavior, `CTutorial1`, is declared like this:

```
class CTutorial1 : public CSystemCommands, public
CAttributeHandler, public CEventHandler, public
LinearAllocationPolicy
{
public:
    RWS_MAKENEWCLASS(CTutorial1);
    RWS_DECLARE_CLASSID(CTutorial1);
    RWS_CATEGORY("Tutorial");
    RWS_DESCRIPTION("Tutorial1", "Tutorial 1");

    CTutorial1(const CAttributePacket& attr);
    ~CTutorial1(void);

    virtual void HandleEvents(CMsg &pMsg);
    virtual void HandleAttributes(const CAttributePacket& attr);

protected:
```

```
CAtomicPtr m_pAtomic;
};
```

Source: tutorial1.h

`RWS_MAKENEWCLASS(classname)` declares the functions that allow the class factory to construct a new `CTutorial1` object.

`RWS_DECLARE_CLASSID(classname)` declares the member variables required to identify the object.

Our constructor takes a `CAttributePacket` as its parameter. When an object is created, a `CAttributePacket` is provided that contains the initial set of attributes for the object.

`HandleEvents` is required as it is a pure virtual method of `CEventHandler`.

`HandleAttributes` is required as it is a pure virtual method of `CAttributeHandler`.

We'll discuss access to `m_pAtomic` in a moment. The first definition in the `CTutorial1` source file is:

```
RWS_IMPLEMENT_CLASSID (CTutorial1)
```

This is the other half of the

`RWS_DECLARE_CLASSID/RWS_IMPLEMENT_CLASSID` pair.

`RWS_DECLARE_CLASSID` declares some static class data required to use `CAttributeHandler`, and `RWS_IMPLEMENT_CLASSID` implements the static class data declared by `RWS_DECLARE_CLASSID`. The implementation of the `CTutorial1` constructor is simply:

```
CTutorial1::CTutorial1(const CAttributePacket& attr)
    : InitCEventHandler(&m_pAtomic)
{
    RWS_FUNCTION( "RWS::Tutorial::CTutorial1::CTutorial1" );

    m_pAtomic = CreateAtomicInWorldFromResource(attr, this);

    RWS_ASSERT(m_pAtomic, "Failed to create atomic");

    RWS_RETURNVOID();
}
```

Source: tutorial1.cpp

Here, we pass the `attr` parameter to the `CEventHandler` constructor. We use the `InitCEventHandler` macro, rather than calling the `CEventHandler` constructor directly, since the parameters to the `CeventHandler` constructor differ between release and debug versions. This is because the event visualization is compiled out in a final build.

Inside the constructor, the `CreateAtomicInWorldFromResource` function creates an atomic (the basic node in the RenderWare scene graph). This function extracts the information it needs (that is, the type of entity, its initial position and transformation) from the attributes used to construct the atomic.

`CAtomicPtr` is a smart pointer (it's actually more than that, as we shall see later), and will automatically delete the clone of the atomic when our entity is destroyed; therefore, the `CTutorial1` destructor is empty for now. Our basic `HandleEvents` method does nothing yet, either, as we will create the behavior's action in lesson 2:

```
void CTutorial1::HandleEvents(CMsg &pMsg)
{
```

```
RWS_FUNCTION( "RWS::Tutorial::CTutorial1::HandleEvents" );
RWS_RETURNVOID( );
}
```

Source: tutorial1.cpp

The final method to implement is `HandleAttributes`. Although our class does not itself have any attributes that can be set, `CSystemCommands` does specify a set of attributes that `CAtomicPtr` needs to process; and since `CTutorial1` is derived from `CSystemCommands`, we need to process them. This is simply done by passing the commands on to our `m_pAtomic` member:

```
void CTutorial1::HandleAttributes(const CAttributePacket& attr)
{
    RWS_FUNCTION( "RWS::Tutorial::CTutorial1::HandleAttributes" );

    CAttributeHandler::HandleAttributes(attr);

    // handle system commands, e.g. if object moved from within
    // RenderWare Studio, then
    // move them on console

    m_pAtomic.HandleSystemCommands(attr);

    RWS_RETURNVOID();
}
```

Source: tutorial1.cpp

`HandleSystemCommands` can process changes to the matrix, visibility state and collision state of the entity. These are the attributes that allow RenderWare Studio to interactively manipulate an entity in the scene.

After saving and compiling, you can see `CTutorial1` in the RenderWare Studio Workspace. Place an entity in the world, and then apply the behavior by dragging the `CTutorial1` from the Behavior List to the entity in the world.

At the moment, the entity doesn't do anything, as we have not defined an action for the behavior. We will do this in the next lesson.

## Lesson 2. Adding an action

---

### In this lesson

We make our behavior do something. Our new behavior, CTutorial2, will rotate the entity slightly every tick.

First, CTutorial2 needs a few more members:

```
protected:
    RwMatrix *m_mat;
    CAtomicPtr m_pAtomic;
    RwReal m_rot[3];
```

Source: tutorial2.h

These members store the information for our transformation. In the CTutorial2 constructor, we must now initialize these members:

```
m_mat = RwMatrixCreate();
m_rot[0] = 0.0f;
m_rot[1] = rwPI; // Rotation on the Y axis.
m_rot[2] = 0.0f;
```

Source: tutorial2.cpp

And in the destructor, we need to free up the matrix:

```
RwMatrixDestroy (m_mat);
```

Let's move on to the event setup. For our behavior to respond to the tick, it needs to be notified when a tick occurs. The Game Framework has certain events that it sends at predefined points in its logic. The event we are interested in is iMsgRunningTick.

The Game Framework sends an event only to behaviors that have specifically requested notification of that type of event. Telling the framework that a behavior wishes to receive an event is called "linking" to the event.

To link our behavior to the iMsgRunningTick event, the CTutorial2 constructor calls:

```
LinkMsg (iMsgRunningTick, 0);
```

To unlink from the event, the destructor calls:

```
UnLinkMsg (iMsgRunningTick);
```

The framework manages the lifetime of events in the system; before removing an event, the framework needs to know when the event is no longer in use. That is, when no behaviors have the event either registered (explained in a later [lesson](#) (p.348)) or linked.

Each time the main loop starts a new frame, it dispatches an iMsgRunningTick event to all behaviors that are linked to it. To process this event, CTutorial2 implements HandleEvents:

```
void CTutorial2::HandleEvents(CMsg &pMsg)
{
    RWS_FUNCTION( "RWS::Tutorial::CTutorial2::HandleEvents" );
```

```
if (pMsg.Id == iMsgRunningTick)
{
    RpFrame* pFrame = RpAtomicGetFrame(m_pAtomic.ptr());
    RwMatrixSetIdentity(m_mat);
    RwMatrixRotate(m_mat, &XAxis, m_rot[0], rwCOMBINEPRECONCAT);
    RwMatrixRotate(m_mat, &YAxis, m_rot[1], rwCOMBINEPRECONCAT);
    RwMatrixRotate(m_mat, &ZAxis, m_rot[2], rwCOMBINEPRECONCAT);

    RwFrameTransform(pFrame, m_mat, rwCOMBINEPRECONCAT);
}
```

Source: tutorial2.cpp

After confirming that the message received is the message we want to process, we call the RenderWare functions to transform our entity. In the next step, we're going to make it possible to edit the rotation values in real time, so we'll need this code.

`HandleAttributes` does not need any changes from `CTutorial1`, as we haven't changed the attributes of the behavior.

The `CTutorial2` behavior is now complete. When you attach it to an entity in the RenderWare Studio Workspace, the entity begins to rotate when viewed on the target.

## Lesson 3. Editing behavior attributes

---

### In this lesson

We define attributes for our behavior, that the user can manipulate in real time from RenderWare Studio.

Our CTutorial2 behavior had an array of three `RwReal` variables (floating point numbers) called `m_rot`, which specify the angle, in radians, to rotate the entity around the x, y, and z axes each frame. To allow RenderWare Studio to manipulate the `m_rot` attributes, we have to tell it about them.

The public section of the `CTutorial3` class declaration contains the following:

```
RWS_BEGIN_COMMANDS
    RWS_ATTRIBUTE( CMD_rot_x, "X Rotation", "Specify the x axis rotation",
SLIDER, RwReal, RANGE(0,0,360))
    RWS_ATTRIBUTE( CMD_rot_y, "Y Rotation", "Specify the y axis rotation",
SLIDER, RwReal, RANGE(0,180,360))
    RWS_ATTRIBUTE( CMD_rot_z, "Z Rotation", "Specify the z axis rotation",
SLIDER, RwReal, RANGE(0,0,360))
RWS_END_COMMANDS;
```

**Source:** tutorial3.h

RenderWare Studio reads everything between `RWS_BEGIN_COMMANDS` and `RWS_END_COMMANDS`. Each `RWS_ATTRIBUTE` macro registers an attribute.

For more information about the `RWS_ATTRIBUTE` macro, see Automated Form Generation in the Game Framework Help.

As we saw in `CTutorial1`, when a behavior is constructed it is passed a `CAttributePacket`. This packet contains information needed to initialize our atomic. After the object has been constructed, `HandleAttributes` is called to set all the attributes to the default values expected by RenderWare Studio. This should mean that you don't have to set initial values for your attributes, as they will all be set by the `HandleAttributes` call after the object has been constructed.

At the top of our `CTutorial3::HandleAttributes` implementation we should still have the code from `CTutorial1`:

```
m_pAtomic.HandleSystemCommands (attr);
```

After this, we can process changes to our attributes. To do this, we add some code to `HandleAttributes`:

```
CAttributeCommandIterator attrIt(attr,
RWS_CLASSID_OF(CTutorial3));

while (!attrIt.IsFinished())
{
    switch (attrIt->GetCommandId())
    {
        case CMD_rot_x:
            attrIt->GetCommandData(m_rot[0]);
            m_rot[0] = RWDEG2RAD(m_rot[0]);
            break;
```

```

case CMD_rot_y:
    attrIt->GetCommandData(m_rot[1]);
    m_rot[1] = RWDEG2RAD(m_rot[1]);
    break;

case CMD_rot_z:
    attrIt->GetCommandData(m_rot[2]);
    m_rot[2] = RWDEG2RAD(m_rot[2]);
    break;

}
++attrIt;
}

```

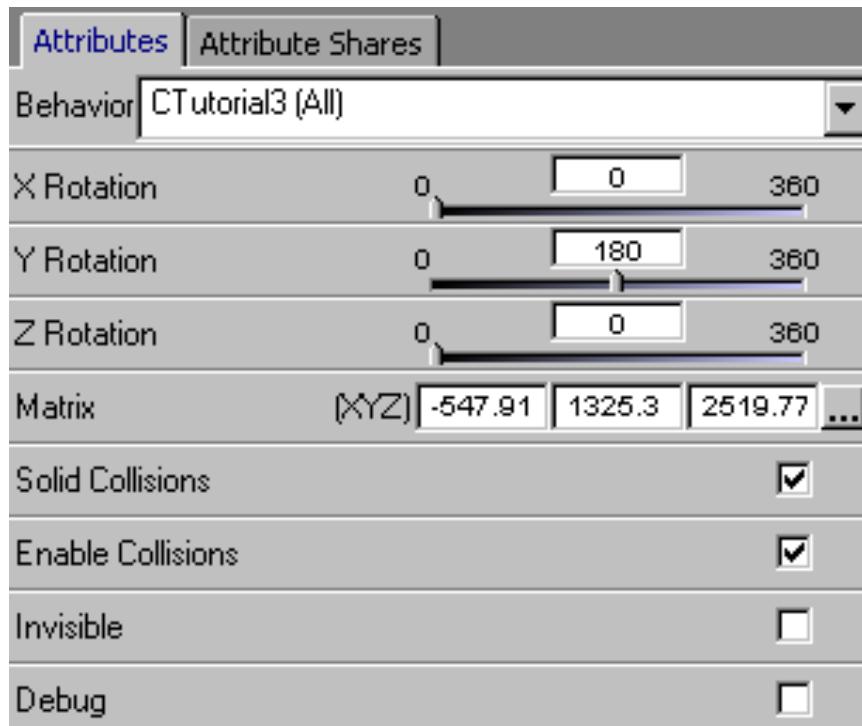
Source: tutorial3.cpp

A CAttributePacket contains a list of attributes to be set. The CAttributeCommandIterator class is a helper class that allows you to run through the attributes one at a time. CAttributeCommandIterator also ensures that you only iterate attributes that are relevant to your class.

The CAttributeCommandIterator class allows you to retrieve the value of the attribute using the GetCommandData method. In this example, once we have the value we convert it to radians (as the sliders are specified in degrees).

CAttributeCommand has many inline helper functions for converting the data to whatever type you need, so in the above example we could have used GetAs\_RwReal, but the helper methods are not as type-safe.

CTutorial3 is now complete. When you attach the new behavior to an entity in the world, you will now be able to adjust how much the entity is rotated each frame by moving the sliders in the RenderWare Studio Workspace:



## Lesson 4. Creating an event

---

### In this lesson

In previous lessons, our behavior has not communicated with the rest of the game framework. The event system lets the game send an event to any behavior (specifically, any `CEventHandler`) anywhere in the framework, allowing behaviors to communicate.

In `CTutorial4`, we create a new behavior that registers a new event with the Game Framework, and sends this event when the user changes the state of a check box in the RenderWare Studio Workspace.

The header file for `CTutorial4` contains the following protected member:

```
CEventId m_trigger;
```

This will hold the event ID that we will send when the user changes the state of the check box.

We need to add the check box as an attribute, so in the public section of the class declaration you need to add:

```
RWS_BEGIN_COMMANDS
    RWS_ATTRIBUTE(CMD_TriggerMessage, "Send Event", "", BOOLEAN, RwUInt32,
    DEFAULT(0))
RWS_END_COMMANDS;
```

Source: `tutorial4.h`

The event we wish to send is registered in the `CTutorial4` constructor by the `RegisterMsg` function. `RegisterMsg` accepts three parameters:

```
RegisterMsg(m_trigger, "trigger", "RwUInt32");
```

The first is a reference to our `CEventId` that will contain the event ID for the new message that we have registered. The second parameter is the unique name for the event we wish to register. If this event has already been registered somewhere else then `m_trigger` will be set to the ID of the original event with that name. The third parameter is for type checking.

Along with every message you send, you can supply a parameter of type `void*` (but any value that fits in the size of a `void*` can be cast to a `void*`), which is passed through the event mechanism. Since this casting breaks the built-in type safety of C++, the framework registers a message using both its name and its parameter type; this is because we cannot perform the type checking that would be done at compile time, because we are making run-time links. If another behavior had registered a trigger event with a `char*`, then it would not have the same ID as the event we have registered, making sure that we do not cast a `char*` to an `RwReal`, or (even worse) cast a `float` to a `char*`.

In our destructor, we should unregister our interest in the event. To do this, we call `UnRegisterMsg`:

```
UnRegisterMsg (m_trigger);
```

The actual attribute handling takes place in `HandleAttributes`. As in

CTutorial3, `m_pAtomic` first attempts to handle the attributes, then we use a `CAttributeCommandIterator` to iterate through all the commands defined by our class:

```
m_pAtomic.HandleSystemCommands(attr);

CAttributeCommandIterator attrIt(attr,
RWS_CLASSID_OF(CTutorial4));

while (!attrIt.IsFinished())
{
    switch (attrIt->GetCommandId())
    {
        case CMD_TriggerMessage:

            // Receive message from RenderWare studio to send message
m_trigger

            CMsg message(m_trigger,
reinterpret_cast<void*>(attrIt->GetAs_RwUInt32()));
            SendMsg(message);
            break;
    }

    ++attrIt;
}
Source: tutorial4.cpp
```

In response to a `CMD_TriggerMessage` (which the Workspace sends whenever the check box changes state), we construct a new message with an event ID of `m_trigger` (the event we registered). The second parameter to the `CMsg` constructor specifies the message parameter. In this case, the function `attrIt->GetAs_RwUInt32()` retrieves the parameter as an `RwUInt32` (it will be zero if the check box is cleared, non-zero otherwise), but then casts it back to a `void *` (which is what `CMsg` expects). This is done purely for demonstration purposes to show that a `BOOLEAN` attribute is accessed as an `RwUInt32`.

After constructing the message, `SendMsg` sends the message to the event system. All `CEventHandler`-derived classes that are linked to our message will be woken up to handle the event.

Currently, no objects are linked to our trigger event. In the next lesson, we will create a behavior that responds to the event.

# Lesson 5. Receiving an event

---

## In this lesson

We create a CTutorial5 behavior that listens for the trigger event that CTutorial4 sends in response to the check box being clicked in the Workspace. When CTutorial5 receives this event, it animates the entity.

Receiving a custom event is similar to the way in which we received the iMsgRunningTick message in CTutorial2; the only difference is that we must perform an extra step to get the ID of our custom message (iMsgRunningTick is a global CEventId, so we didn't have to get its ID).

To do this, we use RegisterMsg in the CTutorial5 constructor. Since RegisterMsg returns the CEventId of a message with that name and parameter if one is already registered, this will either get or create the CEventId that we need:

```
RegisterMsg(m_trigger, "trigger", "RwUInt32");
LinkMsg(m_trigger, 0);
```

Alternatively, LinkMsg could have been written like this:

```
LinkMsg(m_trigger, "RwUInt32");
```

In which case, the LinkMsg call would have checked that the parameter was an RwUInt32, but since we registered the event ourselves there is no need to do this here. If you ever link to a message ID that is registered by another behavior, then it is recommended that you use the second parameter to LinkMsg to ensure that a change to the original message registration does not cause a surprise type mismatch. This applies also to globally registered events, to ensure that the data these events expect to send or receive is the same as you intended.

As in CTutorial2, when our behavior is destroyed we need to make sure we unlink the event from our behavior. To do this, we use UnLinkMsg in the CTutorial5 destructor:

```
UnLinkMsg(m_trigger);
```

We're also going to be using the iMsgRunningTick event again (although we won't link to it in our constructor because we don't want to be receiving it all the time), so we need to make sure we unlink that in our destructor. There is no harm in unlinking an event that you did not link to (similarly, there is no harm in unregistering an event that you did not register).

```
UnLink (iMsgRunningTick);
```

Finally, we have to remember to unregister the m\_trigger event:

```
UnRegisterMsg (m_trigger);
```

CTutorial5 rotates the entity whenever the attribute of the CTutorial4 entity is checked. All of this logic is done in the HandleEvents function of CTutorial5:

```
RwFrame* pFrame = RpAtomicGetFrame(m_pAtomic.ptr());
```

```

if (pMsg.Id == iMsgRunningTick)
{
    RwMatrixSetIdentity(m_mat);
    RwMatrixRotate(m_mat, &YAxis, 0.5f, rwCOMBINEPRECONCAT);
    RwFrameTransform(pFrame, m_mat, rwCOMBINEPRECONCAT);
}
else if (pMsg.Id == m_trigger)
{
    RwUInt32 checked = reinterpret_cast<RwUInt32>(pMsg.pData);

    if (checked == 1)
    {
        if (IsLinked(iMsgRunningTick) == (RwBool)FALSE)
        {
            LinkMsg(iMsgRunningTick, 0);
        }

        RWS_TRACE( "Message received, rotating object" );
    }
    else
    {
        if (IsLinked(iMsgRunningTick) == (RwBool)TRUE)
        {
            UnLinkMsg(iMsgRunningTick);
        }

        RWS_TRACE( "Message received, stop rotating object" );
    }
}

```

Source: tutorial5.cpp

If we receive an `iMsgRunningTick` message, we rotate the object. The next test, `message.Id == m_trigger` checks to see if we have received the trigger event that is sent by `CTutorial4`. When we receive this event with the check box checked, we make sure that we are linked to `iMsgRunningTick`, and when we receive the message with the check box unchecked we make sure that we are not linked to `iMsgRunningTick`.

To test `CTutorial5`, first ensure `CTutorial4` is working. Run the RenderWare Studio Workspace, place an entity in the world and then drag the `CTutorial4` behavior to it. Now place another entity in the world and apply the new `CTutorial5` behavior to that. If everything worked, then changing the check box attribute of the `CTutorial4` entity should cause the `CTutorial5` entity to start or stop spinning.



# Lesson 6. Sending an event

---

## In this lesson

We demonstrate how behaviors can transmit and receive user-defined messages.

The CTutorial6 header file defines two messages:

### Receive Event

The user-defined message ID that we want the behavior to receive. When the behavior receives this event, it will start or stop rotating its associated entity.

### Transmit Event

The user-defined message ID that we want the behavior to transmit when the user changes the state of the **Send Event** check box (below).

and one attribute:

### Send Event

A boolean value, displayed in the Workspace as a check box. When the user changes the state of this check box, the behavior sends the **Transmit Event** (above).

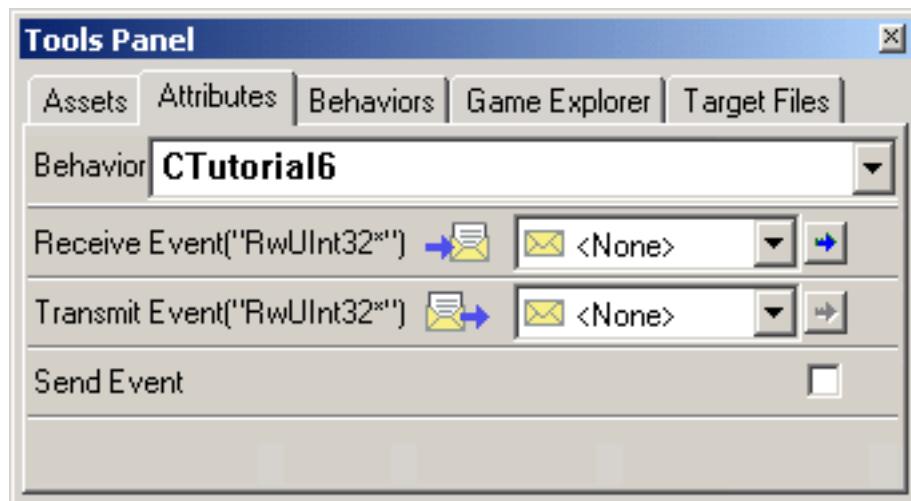
These messages and attribute are defined in the public section of the CTutorial6 class declaration:

```
RWS_BEGIN_COMMANDS
    RWS_MESSAGE(CMD_targetname, "Receive Event", "", RECEIVE,
    "RwUInt32*", 0)
    RWS_MESSAGE(CMD_transname, "Transmit Event", "", TRANSMIT,
    "RwUInt32*", 0)
    RWS_ATTRIBUTE(CMD_TriggerMessage, "Send Event", "", BOOLEAN,
    RwUInt32, DEFAULT(0))
RWS_END_COMMANDS;
```

Source: tutorial6.h

**Note:** The RWS\_MESSAGE macro is explained in [lesson 7](#) (p.337).

As a result of these macros, the RenderWare Studio Workspace will display the following interface for a CTutorial6 entity:



(That is, you can enter values for **Receive Event** and **Transmit Event**, and change the state of the **Send Event** check box.)

The protected section of the CTutorial6 class declaration defines the following items:

Item	Description
RwMatrix *m_mat	The matrix for rotating the m_pAtomic object (below).
CAtomicPtr m_pAtomic	A pointer to the behavior's atomic (as described in earlier lessons).
CEventId m_incoming	Stores the <b>Receive Event</b> message ID.
CEventId m_outgoing	Stores the <b>Transmit Event</b> message ID.

Let's look at how the message and attribute values affect the behavior.

In the HandleAttributes function, look at the CMD\_targetname handler:

```
case CMD_targetname:
    // Receive message from RenderWare Studio to get the target
    // name for the message.
    // the following steps can be shortened to...
    //
    // ReplaceLinkedMsg(m_incoming, attrIt->GetAs_char_ptr(),
    "RwUInt32*");
    //
    // ...but for clarity the individual steps have been shown.
    //
    UnLinkMsg(m_incoming);
    UnRegisterMsg(m_incoming);
    RegisterMsg(m_incoming, attrIt->GetAs_RwChar_ptr(),
    "RwUInt32*");
    LinkMsg(m_incoming, "RwUInt32*");
    break;
```

Source: tutorial6.cpp

First, this handler unlinks and unregisters the message from the system. This is necessary when you modify an attribute or message ID in the Workspace. It is

quite safe to unlink and unregister a message if the message has not currently been linked and registered. When this has been done, the message is then registered and linked.

All of the above steps can be performed using just one function call:

```
ReplaceLinkedMsg(m_incoming, attrIt->GetAs_RwChar_ptr(), "RwUInt32");
```

**Note:**

- The long-hand method is used in the supplied source code for illustrative purposes only.
- To extract the value of the **Trasmit Message** attribute from the RenderWare Studio Workspace, the RegisterMsg macro refers to the attrIt packet (attrIt->GetAs\_RwChar\_ptr()).

Next, we handle the **Send Event** check box attribute:

```
case CMD_TriggerMessage:
    // Receive message from RenderWare studio to send message
    "m_outgoing".
    RwUInt32 temp = attrIt->GetAs_RwUInt32();
    CMsg message(m_outgoing, static_cast<void*>(&temp));
    SendMsg(message);
    break;
```

Source: tutorial6.cpp

In response to a CMD\_TriggerMessage, which the Workspace sends when the check box changes state, we construct a new message with event ID m\_outgoing. In this case, attrIt->GetAs\_RwUInt32() retrieves the parameter as an RwUInt32 (zero if the check box is cleared, non-zero otherwise), but then casts it back to the void\* expected by CMsg. This is done to show that a BOOLEAN attribute is accessed as an RwUInt32.

After constructing the message, SendMsg sends the message to the event system. All CEventHandler derived classes that are linked to our message will be woken up to handle the event.

The next step is to look at how the events are handled for this behavior. The event will behave in a similar manner to the previous lessons; when the event is triggered, the behaviors will link to or unlink from the iMsgRunningTick message. Every time the main loop starts a new frame, it dispatches an iMsgRunningTick event to all objects that are linked to it. CTutorial6 can process this event by implementing HandleEvents.

After confirming that the message we have received is indeed the message we intend to process, we get to the RenderWare code to transform our object.

```
if (pMsg.Id == iMsgRunningTick)
{
    RpFrame* pFrame = RpAtomicGetFrame(m_pAtomic.ptr());
    RpMatrixSetIdentity(m_mat);
    RpMatrixRotate(m_mat, &YAxis, 0.5f, rwCOMBINEPRECONCAT);
    RpFrameTransform(pFrame, m_mat, rwCOMBINEPRECONCAT);
}
else if (pMsg.Id == m_incoming)
{
    // When the 'incoming' message is received the link to the
    iMsgRunningTick system message made
    // depending on the state of the incoming data. If there is no
```

```

data then state is toggled. The
// toggle is needed as when a message is 'test fired' the data
parameter will be zero (NULL).
RwUInt32 var;

if (pMsg.pData)
{
    var = *static_cast<RwUInt32*>(pMsg.pData);
}
else
{
    var = !IsLinked(iMsgRunningTick);
}

// Change the state...
if (var == 1)
{
    if (IsLinked(iMsgRunningTick) == (RwBool)FALSE)
    {
        LinkMsg(iMsgRunningTick, 0);
    }
}
else
{
    if (IsLinked(iMsgRunningTick) == (RwBool)TRUE)
    {
        UnLinkMsg(iMsgRunningTick);
    }
}
}

```

Source: tutorial6.cpp

As shown in the code above, when we handle `m_incoming` we extract its data to test whether the check box has been selected or not and accordingly link or unlink the `iMsgRunningTick` message.

## Testing this behavior

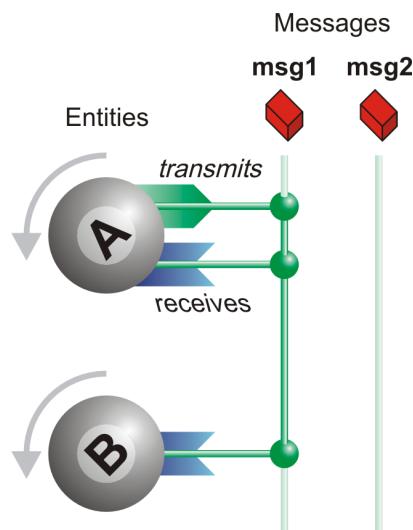
To test CTutorial6:

1. Start the RenderWare Studio Workspace.
2. **Create** (p.119) two entities (we will refer to these as “Entity A” and “Entity B”).
3. **Attach** (p.119) the CTutorial6 behavior to the entities.
4. **View** (p.213) the game on a target.
5. Perform the three test procedures (below), in order.

### Test 1

1. Select Entity A.
2. In the Entity A **Receive Event**, type `msg1` and then press **Enter**.
3. In the Entity A **Transmit Event**, type `msg1` and then press **Enter**.
4. Select Entity B.
5. In the Entity B **Receive Event**, type `msg1` and then press **Enter**.
6. Select Entity A again.
7. Select the Entity A **Send Event** check box.

In the target view, both entities start rotating, as shown in the diagram below.



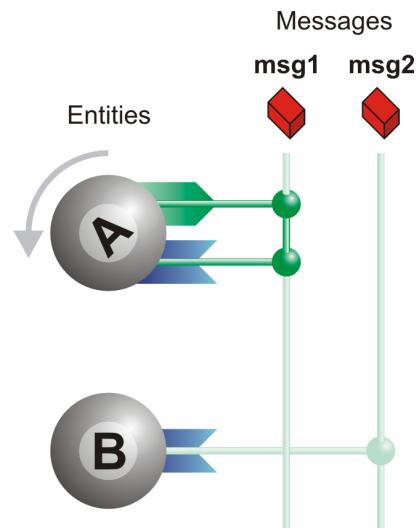
8. Unselect the Entity A **Send Event** check box.

Both entities stop rotating.

## Test 2

1. Select Entity B.
2. In the Entity B **Receive Event**, type `msg2` and then press **Enter**.
3. Select Entity A.
4. Select the Entity A **Send Event** check box.

In the target view, only Entity A starts rotating, as shown in the diagram below.



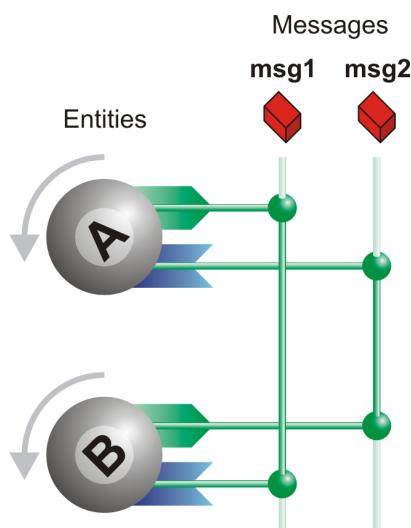
This is because Entity B is now set to receive msg2, not msg1.

5. Unselect the **Send Event** check box for Entity A.

Entity A stops rotating.

## Test 3

1. In the Entity A **Receive Event**, type `msg2` and then press **Enter**.  
Leave the Entity A **Transmit Event** set to `msg1`.
2. Select Entity B.
3. In the Entity B **Receive Event**, type `msg1` and then press **Enter**.
4. In the Entity B **Transmit Event**, type `msg2` and then press **Enter**.
5. Select the Entity B **Send Event** check box.  
Entity A starts rotating.
6. Select Entity A.
7. Select the Entity A **Send Event** check box.  
In the target view, Entity B starts rotating, too.



Entity A is set to transmit `msg1`, and Entity B is set to receive `msg1`. So, when you selected the Entity A **Send Event** check box, Entity B started rotating.

Similarly, Entity B is set to transmit `msg2`, and Entity A is set to receive `msg2`. So, when you selected the Entity B **Send Event** check box, Entity A started rotating.

8. Unselect the Entity A **Send Event** check box.  
Entity B stops rotating.
9. Select Entity B.
10. Unselect the Entity B **Send Event** check box.  
Entity A stops rotating.

This is the end of the lesson.

## Lesson 7. Debug macros

---

### In this lesson

We demonstrate how to use other attribute types in your behaviors (such as color pickers, list boxes and separators), and how to display debug information in your developer environment when the RenderWare Studio Workspace user modifies attribute values.

These are the attribute definitions in the public section of the CTutorial7 class declaration:

```
RWS_BEGIN_COMMANDS
    RWS_SEPARATOR("Attribute List", 0)
    RWS_MESSAGE( CMD_testmessage, "Receive Event", "Type in event name.", ,
RECEIVE, "RwUInt32", 0)

    RWS_ATTRIBUTE(CMD_testboolean, "Test Boolean", "Check box to send
event.", BOOLEAN, RwUInt32, DEFAULT(0))
    RWS_ATTRIBUTE(CMD_testslder, "Test Slider", "Set rotation.", SLIDER,
RwReal, RANGE(-4000, 0, 4000))
    RWS_ATTRIBUTE(CMD_testcolor, "Test Color", "Object color.", COLOR,
RwRGBA, DEFAULT(0))
    RWS_ATTRIBUTE(CMD_testlist, "Test List", "List Selection", LIST,
RwUInt32, LIST("item1|item2|item3"))
RWS_END_COMMANDS;
```

Source: tutorial7.h

The RWS\_SEPARATOR macro specifies parts of the attribute [editing interface](#) (p.270) that you can collapse out of view. This is particularly useful if your behavior has many attributes that you can modify.

**RWS\_SEPARATOR (*title*, *state*)**

***title***

Text string that will be displayed on the separator.

***state***

The initial state of the separator: 0 for closed, 1 for open.

The RWS\_MESSAGE macro specifies a message in RenderWare Studio (for example, if the behavior is for a pickup, you may want to specify within the Workspace that the behavior should respond to a ACTN\_PLAYERTOUCH, which will get triggered if the player collides with the object).

**RWS\_MESSAGE** (*id*, *shortdesc*, *longdesc*, *controltype*, *valuetype*, *defaultmessage*)

*id*

The ID of this message attribute. A constant with this value will be defined for you for use in the class.

*shortdesc*

A short description of the purpose of this attribute. This will appear in the interface label.

*longdesc*

A long description of the purpose of this attribute. This will appear in the context sensitive help.

*controltype*

The kind of attribute:

### RECEIVE

To specify that this is for a receiving message

### TRANSMIT

To specify that this is for a transmitting message

*valuetype*

The type of the attribute. These are specified in RenderWare types, for example:

- RwUInt32
- RwInt32
- RwV3d
- RwReal

*defaultmessage*

This parameter specifies a default message (for example, ACTN\_PLAYERTOUCH)

The RWS\_ATTRIBUTE macro lets you specify attribute controls like list views, check boxes, sliders, and color pickers. For a description of each RWS\_ATTRIBUTE parameter, see [lesson 3](#) (p.346).

Before looking at how to read values back from these attributes controls, we will first look at how to use the debugging macros in the RenderWare Studio Workspace to print debug messages in your development environment.

To use these debug macros, you need to include at the start of your source code file:

1. A define that enables function tracing, allowing you to see how control passes between functions:

```
#define RWS_CALLSTACKENABLE
```

2. A define that enables you to print your own debug trace messages:

```
#define RWS_TRACEENABLE
```

3. After the defines, an include for the debug macros:

```
#include "framework/core/macros/debugmacros.h"
```

If you enable function tracing, then you can call RWS\_FUNCTION and

RWS\_RETURNVOID within your functions, to show debug messages indicating when these functions are called.

```
CTutorial7::CTutorial7(const CAttributePacket& attr)
    : InitCEventHandler(&m_pAtomic)
{
    RWS_FUNCTION("RWS::Tutorial::CTutorial7");

    m_pAtomic = CreateAtomicInWorldFromResource(attr, this);

    RWS_ASSERT(m_pAtomic, "Failed to create atomic");

    RWS_RETURNVOID();
}
```

Source: tutorial7.cpp

The example above prints the function name when it is called; this is useful when tracing the flow of the behavior when it is running, especially as it can keep a count of the call depth of the behavior, and indents the debug message in the Message Log window. The RWS\_RETURNVOID does not print anything to the Message Log window, but it decreases the call depth counter, so you have to put it in.

To print your own debug trace message, call RWS\_TRACE. If you have function tracing enabled, as in the example below, then the call to RWS\_TRACE must be between the RWS\_FUNCTION and RWS\_RETURNVOID calls:

```
CTutorial7::CTutorial7(const CAttributePacket& attr) :
InitCEventHandler(&m_pAtomic)
{
    RWS_FUNCTION("CTutorial::CTutorial7");
    RWS_TRACE("This is my message");
    RWS_RETURNVOID();
}
```

Now that we have seen how to print a debug trace message, let's look at printing a debug trace message when the Workspace user changes a behavior attribute value.

The HandleAttributes function contains the code that retrieves behavior attribute values:

```
void CTutorial7::HandleAttributes(const CAttributePacket& attr)
{
    RWS_FUNCTION("RWS::Tutorial::CTutorial7::HandleAttributes");

    CAttributeHandler::HandleAttributes(attr);

    // Handle system commands, e.g. if object moved from within
    RenderWare studio, then
    // move them on console.
    m_pAtomic.HandleSystemCommands(attr);

    CAttributeCommandIterator attrIt(attr,
    RWS_CLASSID_OF(CTutorial7));

    while (!attrIt.IsFinished())
    {
        switch (attrIt->GetCommandId())
        {
            case CMD_testmessage:
                RWS_TRACE(attrIt->GetAs_RwChar());
                break;
        }
    }
}
```

Source: tutorial7.cpp

The first attribute value to get is the message trigger box. This is straightforward; the attribute packet has accessor member functions that return the required data. In this case, we are returning a pointer to a `char`, which we use as a parameter in the `RWS_TRACE` function.

```
case CMD_testboolean:
    RWS_TRACE(attrIt->GetAs_RwUInt32( ));
    break;
```

Source: tutorial7.cpp

The next attribute value to get is the check box; for this we are going to use the `attrIt->GetAs_RwUInt32()`. This will return either a 1 or a 0, depending on whether the box is checked or not.

```
case CMD_testslder:
    RWS_TRACE(attrIt->GetAs_RwReal( ));
    break;
```

Source: tutorial7.cpp

The slider attribute value is just as easy to get data from; use the `attrIt->GetAs_RwReal()`. The value returned will be within the bounds described in the header; in this case, -4000.0 to 4000.0.

```
case CMD_testcolor:
    m_Color = attrIt->GetAs_RwRGBA( );

    RWS_TRACE(reinterpret_cast<RwUInt32*>(m_Color.alpha) << " - 
ALPHA");
    RWS_TRACE(reinterpret_cast<RwUInt32*>(m_Color.red) << " - 
RED");
    RWS_TRACE(reinterpret_cast<RwUInt32*>(m_Color.green) << " - 
GREEN");
    RWS_TRACE(reinterpret_cast<RwUInt32*>(m_Color.blue) << " - 
BLUE");
```

Source: tutorial7.cpp

For the next attribute value, we want to get data from the color picker control. First, you use the `GetCommandData` function to reinterpret the cast to the type of the input parameter.

```
const RwChar* pvar1;
const RwUInt32* pvar2;
const RwV3d* pvar3;

cmd.GetCommandData(&pvar1);
cmd.GetCommandData(&pvar2);
cmd.GetCommandData(&pvar3);
```

The above example shows this in use: for the first example, `RwChar*` `pvar1` casts the data in the attribute packet to a `RwChar*`. So in the case of our color picker, the data from the attribute packet is being cast into a `RwUInt32`. Then some shift operations are performed to extract the red, green, blue, and alpha values into a separate `RwRGBA` data structure.

```
RWS_TRACE(reinterpret_cast<RwUInt32*>(m_Color.alpha) << " - 
ALPHA");
RWS_TRACE(reinterpret_cast<RwUInt32*>(m_Color.red) << " - RED");
RWS_TRACE(reinterpret_cast<RwUInt32*>(m_Color.green) << " - 
GREEN");
RWS_TRACE(reinterpret_cast<RwUInt32*>(m_Color.blue) << " - BLUE");
break;
```

Source: tutorial7.cpp

The last property is a list box, as you can see from the attribute definition the list box has been set up with three entries: "item1", "item2", and "item3".

```
RWS_ATTRIBUTE(CMD_testlist, "Test List", "List Selection", LIST,
RwUInt32, LIST("item1|item2|item3"))
```

What we want to do is return back the number of the list element selected, so if in the RenderWare Studio Workspace the first item is selected, we will get the number 0 returned. To get the list element number, we will use the GetCommandData function again to fill the data into a `RwUInt32`.

```
RwUInt32 type;
attrIt->GetCommandData(type);
```

The number returned will simply be the element number, so we can simply use a switch to display a debug message stating which element has been selected.

```
case CMD_testlist:
    RwUInt32 type;

    attrIt->GetCommandData(type);

    switch (type)
    {
        case 0:
            RWS_TRACE("Selected List Element 1")
            break;

        case 1:
            RWS_TRACE("Selected List Element 2")
            break;

        case 2:
            RWS_TRACE("Selected List Element 3")
            break;
    }

break;
```

Source: tutorial7.cpp

To test this behavior, create an entity in the Workspace and attach the behavior CTutorial7. Then view the game on the target.. When the data has been transferred to the target, make sure your entity with the behavior has been selected. Then, in the [Attributes](#) (p.270) window, change the attribute controls.

In the developer environment debug window you will see all the trace messages displayed as you alter the controls.

# Lesson 8. Inheritance

---

## In this lesson

We inherit the capabilities of the CTutorial2 behavior into CTutorial8. (CTutorial2 rotates its associated entity around the y axis.) In CTutorial8, the rotation will be controlled by a counter, and CTutorial8 will also grow and shrink the entity.

Here is an extract of the header for CTutorial8:

```
{
    namespace Tutorial
    {
        class CTutorial8 : public CTutorial2
        {
            public:
                RWS_MAKENEWCLASS(CTutorial8);
                RWS_DECLARE_CLASSID(CTutorial8);
                CTutorial8(const CAttributePacket& attr);
                ~CTutorial8(void);

                virtual void HandleEvents(CMsg &pMsg);
                virtual void HandleAttributes(const CAttributePacket&
attr);

            protected:
                void Scale_Object(void);
            };
        } //namespace Tutorial
    } //namespace RWS
}
```

Source: tutorial8.h

As you can see, CTutorial8 inherits from CTutorial2, which in turn inherits from CSystemCommands, CAttributeHandler and CEventHandler, so you will need to override HandleEvents and HandleAttributes.

CTutorial8 will not directly need to know about the atomic that gets rotated in CTutorial2, so you will not need to call CreateAtomicFromFirstResource in the constructor and call HandleSystemCommands in Handle attributes. All you will need to do is call the CTutorial2 HandleAttributes and HandleEvents functions, as shown below:

```
void CTutorial8::HandleEvents(CMsg &pMsg)
{
    RWS_FUNCTION( "CTutorial8::HandleEvents" );
    :

    CTutorial2::HandleEvents(pMsg);
    :

    RWS_RETURNVOID();
}

void CTutorial8::HandleAttributes(const CAttributePacket& attr)
{
    RWS_FUNCTION( "CTutorial8::HandleAttributes" );
    :
```

```
CTutorial2::HandleAttributes(attr);
:
RWS_RETURNVOID( );
}
```

In addition to the rotation applied by CTutorial2, CTutorial8 includes code to stop and start the rotation (based on a counter) and applies scaling to grow and shrink the entity.

The scaling uses the following code to get a pointer to the entity frame:

```
RwFrame* pFrame = RpAtomicGetFrame(m_pAtomic.ptr());
```

The scaling is done using the following RenderWare code:

```
RwMatrixScale(&scale_Mat, &scale_vec, wCOMBINEPRECONCAT);
```

To use this behavior, create an entity in your world, and then add the CTutorial8 behavior to it. Then, when you have connected to the console, you will see the object rotating on its y axis and scaling in x, y and z.

# Lesson 9. Creating a behavior without using the CAtomicPtr object

---

## In this lesson

We create a new CTutorial9 behavior from scratch, without relying on the CAtomicPtr object used in previous lessons. This involves reproducing the functionality provided by CAtomicPtr within the new behavior class, including cleaning up and safely destroying the atomic, and handling system attribute commands.

In the previous lessons, the example behaviors used the CAtomicPtr class to contain the atomic. This class is a smart pointer that also provides some extra functionality; as well as automatically cleaning up the atomic when it goes out of scope, it also provides a method that handles the system attribute commands.

In this behavior, CTutorial9, we will implement the same skeleton behavior as CTutorial1; the main difference is that the atomic is contained directly by the behavior class, and not a CAtomicPtr. The behavior therefore must implement the code that the CAtomicPtr would have provided. CTutorial9 is declared like this:

```
class CTutorial9 : public CSystemCommands, public
CAttributeHandler, public CEventHandler, public
LinearAllocationPolicy
#ifndef RWS_EVENTVISUALIZATION
, public CEventVisualization
#endif
{
public:
    RWS_MAKENEWCLASS(CTutorial9);
    RWS_DECLARE_CLASSID(CTutorial9);
    RWS_CATEGORY("Tutorial");
    RWS_DESCRIPTION("Tutorial9", "Tutorial 9");

    CTutorial9(const CAttributePacket& attr);
    ~CTutorial9(void);

    virtual void HandleEvents(CMsg &pMsg);
    virtual void HandleAttributes(const CAttributePacket& attr);

#ifndef RWS_EVENTVISUALIZATION
    virtual RwV3d *GetWorldPos(void);
#endif

protected:
    RpAtomic *m_pAtomic;
};

Source: tutorial9.h
```

You can see that `m_pAtomic` is declared as a pointer to an `RpAtomic`, and not a `CAtomicPtr` as in the previous tutorials. Since the destruction of the atomic is not longer automatically handled by a smart pointer we have to do this manually in the behavior's destructor function. The destructor looks like this:

```
CTutorial9::~CTutorial9(void)
```

```

{
    RWS_FUNCTION( "RWS::Tutorial::CTutorial9::~CTutorial9" );

    // Safely destroy the Atomic
    if (m_pAtomic)
    {
        RwFrame* pFrame = RpAtomicGetFrame(m_pAtomic);
        if (pFrame)
        {
            // Clean up any frame hierarchy attached to the atomic (the
            assumption is
                // that any child frames are managed by other objects, so
            it is sufficient
                // for us to simply detach them and let them 'float off').
            //
            RwFrameForAllChildren(pFrame,
FrameHelper::RemoveChildFrame, 0);

            // If this frame is itself attached to another, detach it.
            //
            if (RwFrameGetParent(pFrame)) RwFrameRemoveChild(pFrame);

            RpAtomicSetFrame(m_pAtomic, 0);
            FrameHelper::FrameDestroy(pFrame);
        }

        // Remove atomic from world.
        RpWorld* pWorld = RpAtomicGetWorld (m_pAtomic);
        if (pWorld) RpWorldRemoveAtomic(pWorld, m_pAtomic);

        // RenderWare destroy atomic.
        RpAtomicDestroy(m_pAtomic);
    }

    RWS_RETURNVOID();
}

```

Source: tutorial9.cpp

Here, the destructor is cleaning up any frame hierarchy before destroying the frame attached to the atomic. It does this by removing any child frames and also removing itself from any parent frame. This is important since the atomic may have been attached to another frame, or had other frames attached during the course of the application (for example, attaching a player to a moving platform). Once the frame is destroyed the atomic is then removed from the world using `RpWorldRemoveAtomic()` and also destroyed by a call to `RpAtomicDestroy()`.

The `CAtomicPtr` class handles system attribute commands in the method `HandleSystemCommands()`. These system attributes will now have to be handled explicitly within the `HandleAttributes` method of our behavior:

```

void CTutorial9::HandleAttributes(const CAttributePacket& attr)
{
    RWS_FUNCTION( "RWS::Tutorial::CTutorial9::HandleAttributes" );

    CAttributeHandler::HandleAttributes(attr);

    // handle system commands, e.g. if object moved from within
    RenderWare studio, then
    // move them on console.
    CAttributeCommandIterator attrIt(attr,
RWS_CLASSID_OF(CSystemCommands));
    while (!attrIt.IsFinished())
    {

```

```

switch (attrIt->GetCommandId())
{
    case CSysystemCommands::CMD_LoadMatrix:
    {
        RwFrame* pFrame = RpAtomicGetFrame(m_pAtomic);
        if (pFrame) CSysystemCommands::UpdateFrame(*pFrame,
*attrIt);
    }
    break;
    case CSysystemCommands::CMD_SetSolidFlag:
    {
        RwUInt32 flag;
        attrIt->GetCommandData(flag);
        RpAtomicCollisionProperties::SetIsSolid( *m_pAtomic,
flag?true:false );
    }
    break;
    case CSysystemCommands::CMD_SetInvisibleFlag:
    {
        RwUInt32 flag;
        attrIt->GetCommandData(flag);
        AtomicHelper::SetIsVisible( *m_pAtomic,
flag?false:true );
    }
    break;
    case CSysystemCommands::CMD_SetCollisionFlag:
    {
        RwUInt32 flag;
        attrIt->GetCommandData(flag);
        AtomicHelper::SetCanCollide( *m_pAtomic,
flag?true:false );
    }
    break;
}
++attrIt;
}

RWS_RETURNVOID();
}

```

Source: tutorial9.cpp

Our HandleAttributes function is now handling the system attribute commands to update the atomic's matrix and the various flags controlling visibility, solid, and collisions. We now have an "empty" behavior class just like in [Tutorial 1](#) (p.341).

## RenderWare Studio Attribute Editor Overview

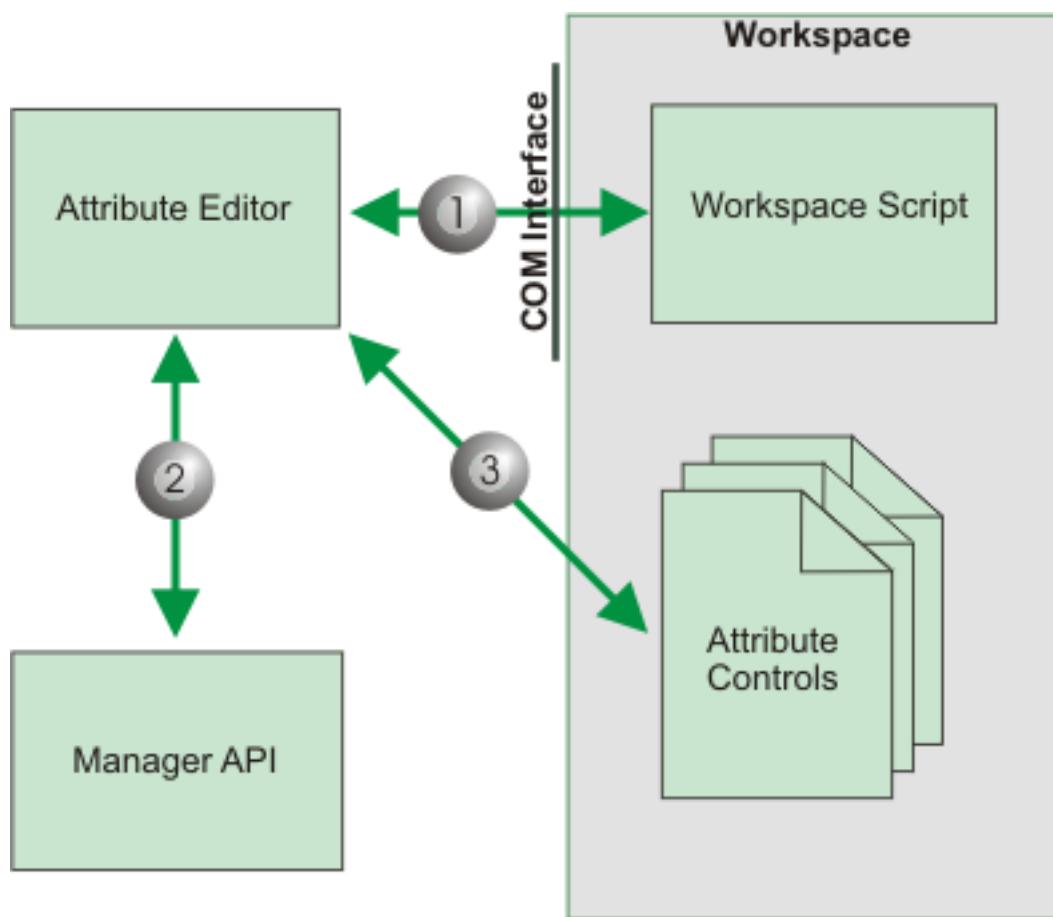
---

This section provides an overview of the architecture behind the RenderWare Studio Attribute Editor, how the controls integrate, and how the editor communicates with the RenderWare Studio workspace. For a practical example of using the attribute editor and its wizard, take a look at the [tutorials](#) (p.382).

# Attribute Editor Architecture

---

The diagram below illustrates the links between the RenderWare Studio Workspace script and the attribute editor and its controls.



In order to display a custom control the following steps have to take place:

1. The Attribute editor control exposes a COM Automation interface that the script calls when entities are selected or de-selected.
2. The Attribute editor then queries the RenderWare Studio Manager API data for the selected entity's attributes
3. ...and finally displays the relevant controls.

# Attribute Editor Script Integration

---

The following attribute editor interface is available to the script:

```
interface IAttrCtl : IDispatch
{
    [id(1), helpstring("method AddSelection")]
    HRESULT AddSelection([in] long EntityID);

    [id(2), helpstring("method RemoveSelection")]
    HRESULT RemoveSelection([in] long EntityID);

    [id(3), helpstring("method AttachToDatabase")]
    HRESULT AttachToDatabase([in] long DatabaseID);

    [id(4), helpstring("method ClearSelection")]
    HRESULT ClearSelection();
};
```

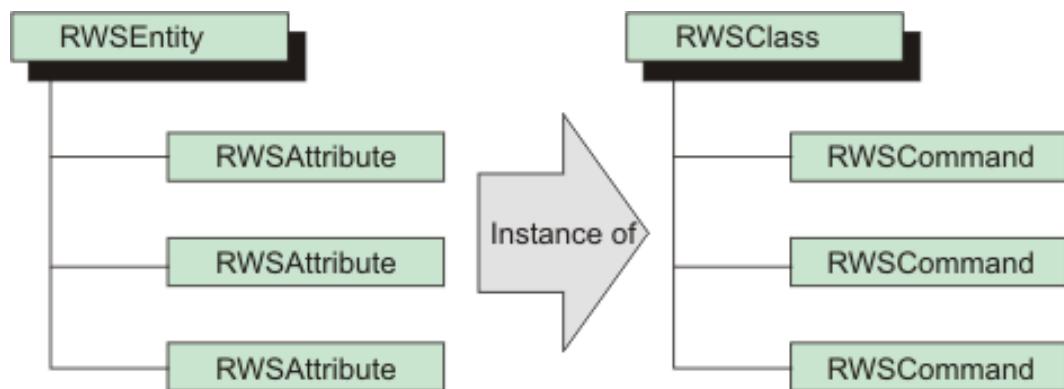
The attribute editor maintains a set of entities that are being edited. The first two methods (`AddSelection` and `RemoveSelection`) are used to add and remove entities from the selected set. The third method (`AttachToDatabase`) is used to tell the attribute editor to attach to a new RenderWare Studio Manager database (or the current database should be detached from by using an ID of zero). The fourth method (`ClearSelection`) is used to clear the current set of selections.

# Classes, Commands, Entities and Attributes

---

The RenderWare Studio API stores nodes in its hierarchy to represent classes and commands, and their instanced equivalents, entities and attributes. Classes and commands are created by the source parser toolkit when it parses the game source header files; whereas entities and attributes are saved in the XML.

Each command structure in the API represents an RWS\_ATTRIBUTE macro between an RWS\_BEGIN\_COMMANDS and RWS\_END\_COMMANDS block in the header file. This macro contains a number of user-defined parameters to allow attributes to be created with default data.



Each entity is an instance of a class, and each attribute is an instance of a command .

## Attribute Editor Controls

---

To create and initialize the correct controls, the attribute editor queries the selected RWSEntity for its behavior class name, then locates the RWSClass object whose UID corresponds to the class name. If an RWSClass object is located, its child RWSCommands are queried, and used to create the correct type of control.

Each RWSCommand structure contains a string member (ParamList) that stores the parameters provided to the RWS\_ATTRIBUTE macro.

For example a game-source header file might contain:

```
RWS_ATTRIBUTE( CMD_Movie,
                "MediaPlayer Test",
                "Specify a movie file to play",
                RWSMediaPlayer.RWSMediaPlayerCtrl,
                RwChar,
                DEFAULT( "C:\mssdk\samples\Multimedia\Media\Butterfly.mpg" ) )
```

The above attribute declaration will generate an RWSCommand structure whose name is RWS\_ATTRIBUTE, with a ParamList string that looks like this:

```
"CMD_Movie\0\"MediaPlayer Test\"\0J\"Specify a movie file to
play\"\0RWSMediaPlayer.RWSMediaPlayerCtrl\0RwChar\0
DEFAULT(\"C:\\\\mssdk\\\\samples\\\\Multimedia\\\\Media\\\\Butterfly.mpg\"\0"
```

A null terminator ("\\0") separates each parameter in the list, and the RWSCommand's ParamLength member specifies the total number of characters in the string.

To create the correct ActiveX attribute control, the fourth parameter (RWSMediaPlayer.RWSMediaPlayerCtrl) is extracted from the ParamList member, and used as the control's ID. Each control must implement an interface to allow the editor to communicate with it. The control is initialized via this interface.

# Control Interfaces

---

For a control to be contained within the attribute editor, it must implement the following COM interfaces (which are generated automatically by the RenderWare Studio Attribute Editor wizard):

```
interface __declspec(uuid("3E73B80D-D779-486d-889C-13F2CCFA6D33"))
IRWSAttribEditEvents : public IUnknown
{
public:
    STDMETHOD (CreateUndo) (RWSID AttributeID) = 0;
};
```

This interface provides a callback mechanism to allow each control to create an attribute editing operation that can be undone in the workspace.

```
interface __declspec(uuid("60CDE305-09A3-4966-84DB-255085359B75"))
IRWSAttribEdit : public IUnknown
{
public:
    STDMETHOD (Advise)(IRWSAttribEditEvents
__RPC_FAR *pSource) = 0;
    STDMETHOD (UnAdvise)() = 0;
    STDMETHOD (CreateData)(RWSID AttributeID) = 0;
    STDMETHOD (SetCommandID)(RWSID CommandID) = 0;
    STDMETHOD (AddEntity)(RWSID EntityID, RWSID
AttributeID) = 0;
    STDMETHOD (RemoveEntity)(RWSID EntityID) = 0;
    STDMETHOD (OnAttribChanged)(RWSID AttributeID)
= 0;
    STDMETHOD (OnAttribAdded)(RWSID EntityID, RWSID
AttributeID) = 0;
    STDMETHOD (OnAttribRemoved)(RWSID EntityID,
RWSID AttributeID) = 0;
    STDMETHOD (OnAttribDeleted)(RWSID AttributeID)
= 0;
    STDMETHOD (GetHeightExtra)(/*[out]*/RWSUInt32
__RPC_FAR *Height) = 0;
    STDMETHOD (ExpandUI)(RWSBool bExpand) = 0;
};
```

All of these methods are implemented by a helper class (defined in `RWSAttribEdit.h`), inherited by all controls. This helper class simplifies implementation of each control significantly, and makes editing multiple attributes with different data much easier. The methods have the following functionality:

#### **Advise/Unadvise**

The first two methods in the above interface are used to set up the “connection point” interface that provides the callback mechanism.

#### **CreateData**

This method allows the caller to create the default data for an attribute.

#### **SetCommandID**

This method is used to initialize the control with the command ID that corresponds to the attribute to be created or edited.

#### **AddEntity**

This method is called whenever a new entity is selected in the workspace. If an attribute exists (whose Attribute ID corresponds to the command ID) for

the control, it will be passed as a non-zero second parameter. Otherwise, the control is expected to display the default attribute data in the control.

**RemoveEntity**

This method is used whenever an entity is removed from the currently selected entities in the RenderWare Studio workspace. A zero-parameter passed to this method indicates that all selections are to be removed.

**OnAttribChanged/Added/Removed/Deleted**

These four methods are called when attributes are changed, added, removed or deleted from the RenderWare Studio

**GetHeightExtra**

This method is used to query the height of the expanded part of the user interface.

**ExpandUI**

This final method, is called to allow the control to update its user interface when the “expand” button is clicked in the workspace.

# Attribute Editor Control Wizard

---

The control wizard generates an ActiveX dialog-based control, along with a helper class to simplify implementation of the required interface methods.

Although the helper class alleviates much of the tedium of writing an attribute editor control, some work is still required. The following functions (called by the helper class) must be implemented by your control:

```
RWSUInt32 GetNumDefaultTypeChars (const RWSChar * const szParamList, RWSUInt32 nParamLength);
```

This function must return the number of characters in the string that represents the name of the attribute data type. The return value is used to allocate space for the *Type* member in the RWSAttribute Manager API structure.

For example, if an attribute will store a user-defined structure called "MyQuaternion", your function must return the string length of "MyQuaternion".

If the command your control will edit contains the "DEFAULT" parameter (5th parameter), and you selected the "Read default data type from command" option in the wizard, the following function will automatically be implemented for you by the wizard-generated code.

```
void GetDefaultType (RWSChar * const szType, const RWSChar * const szParamList, RWSUInt32 nParamLength);
```

The *GetDefaultType* function must copy the default data type string into the buffer provided (*szType*). This buffer is guaranteed to be one character longer than the value returned by the *GetNumDefaultTypeChars* function.

```
RWSUInt32 GetDefaultDataSize (const RWSChar * const szParamList, RWSUInt32 nParamLength);
```

*GetDefaultDataSize* must return the size in bytes of the default data that will be stored with this attribute. The return value of this function is used to allocate storage for the *pData* member of the RWSAttribute API structure. For more information on the format of the attribute data, see [Attribute Editor Data](#) (p.378).

```
void GetDefaultData (RWSByte * const pData, const RWSChar * const szParamList, RWSUInt32 nParamLength);
```

The *GetDefaultData* function must copy the default data into the buffer provided (*pData*). This buffer is guaranteed to be the same size as the number of bytes returned by the *GetDefaultDataSize* function (above).

```
void CreateUI ();
```

This function is called to allow the control to create or initialize any user interface items in the control.

```
void UpdateControls (const RWSByte * const pData);
```

The *UpdateControls* function must use the provided data buffer (*pData*) to update the state of the control's user interface. For more information on the

format of the attribute data, see [Attribute Editor Data](#) (p.378).

# Attribute Editor Data

---

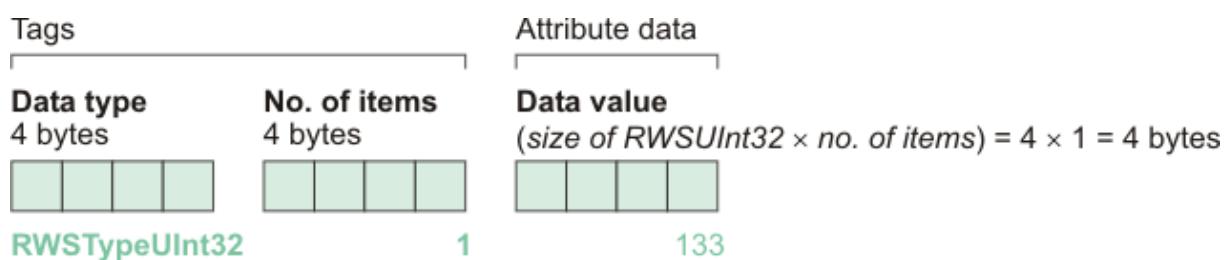
To allow users to customize the data stored with their attributes, the RenderWare Studio Manager API stores data as a stream of bytes, along with a size parameter specifying the data length. This enables complex structures to be stored within the API attribute data structure, and serialized correctly.

To enable the RenderWare Studio Comms network toolkit to send the data to targets in the correct endian-order (and to narrow any Unicode strings) the attribute data must contain special “tags” in the data. These tags contain information about the type and size of the data. Each tag consists of two 4-byte values. The first value defines the type of the data items that follow, whilst the second defines the number of data items.

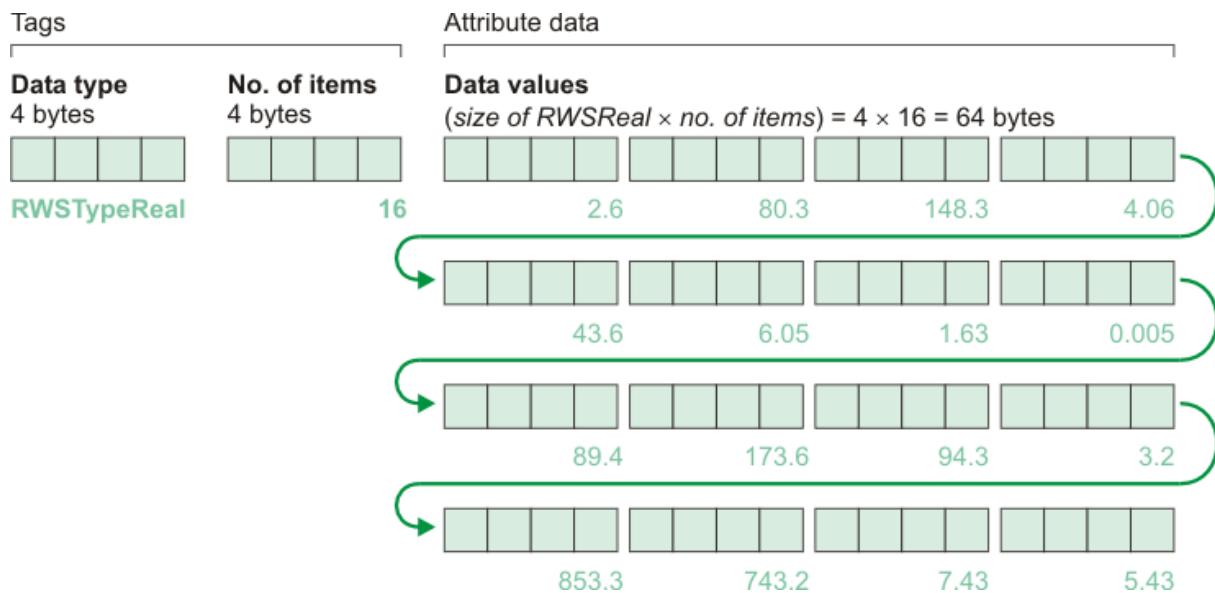
The data type is specified as an enumeration (`RWSDataType`) of the following basic types:

Data Type	Number of bytes required
<code>RWSTypeBool</code>	4
<code>RWSTypeInt32</code>	4
<code>RWSTypeUInt32</code>	4
<code>RWSTypeReal</code>	4
<code>RWSTypeDouble</code>	8
<code>RWSTypeByte</code>	1
<code>RWSTypeChar</code>	2

The data to represent a single data value (such as an `RWSInt32`) would be laid out like so:



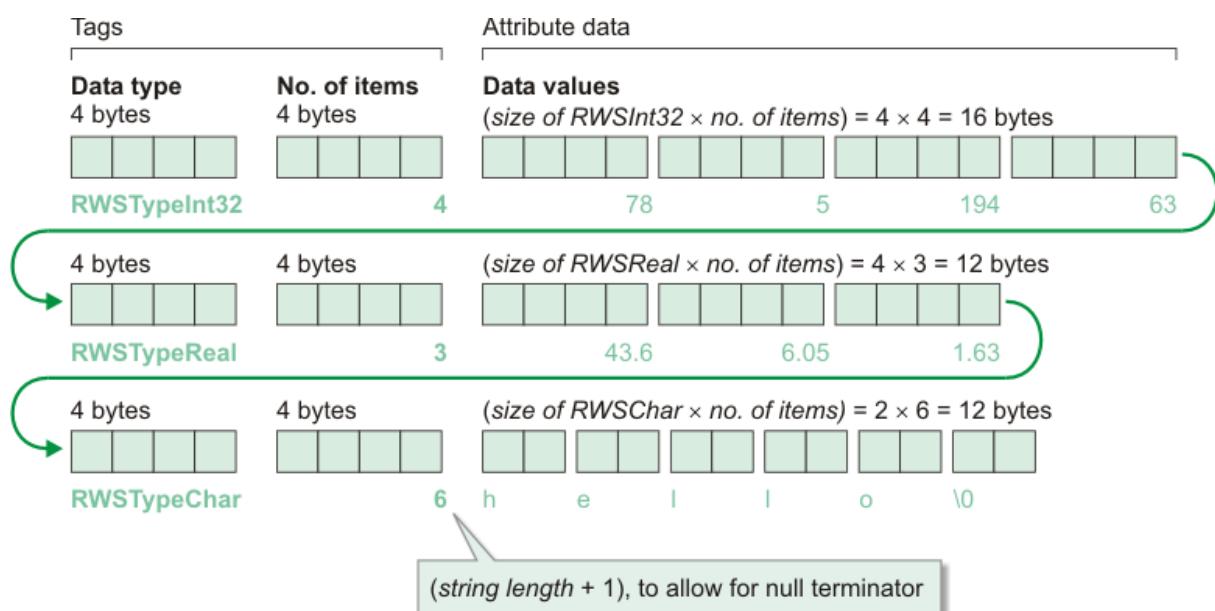
A  $4 \times 4$  matrix might be represented as:



More complex structures can be stored in attribute data. For example, this structure:

```
typedef struct
{
    Rwi32 i[4];
    RWSChar *s; // if "hello\0", then
    char pad[2]; // 2-bytes of padding
    Rv3d v;
} RWSGroup;
```

Would be represented as:



The first five data types listed above need to be located in memory so that they start on a four-byte multiple boundary.

Therefore in this example, if you had the string "hello" plus one byte for the null terminator, you would have the following amount of data needing space in memory:

Data Type	Number of bytes required
RwInt32	16
RWSChar *s;	6
<i>Total bytes required</i>	22

Because the total number of bytes required is not a multiple of four there would need to be a further two bytes of padding to ensure that the next data type (in this example a vector incorporating three RWSReal types) starts at the next four-byte multiple boundary.

## Saving attribute editor data

---

A helper function is provided (in `RWSAttribEdit.h`) to assist with saving data back to attributes:

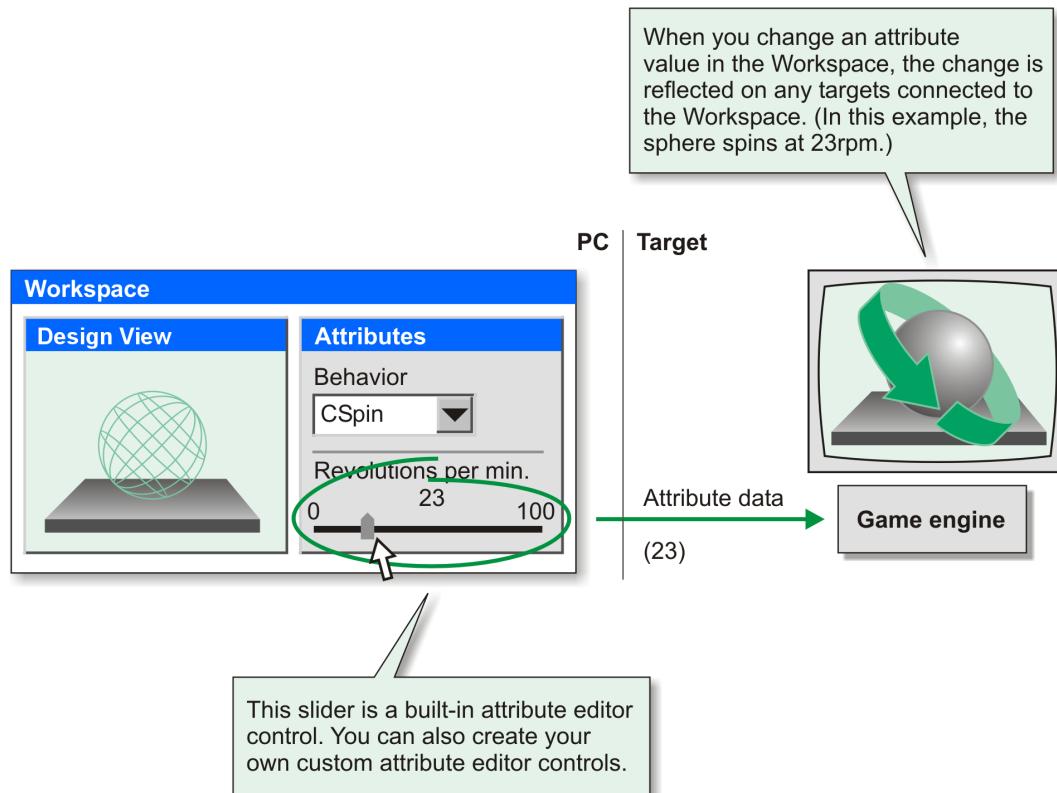
```
void SaveDataToAttributes (const RWSByte * const pData,
                           const RWSUInt32 nDatasize,
                           RWSBool bCreateUndo = RWSTrue)
```

Simply call this function from within your control class (often in response to a windows message such as `WM_CHAR`) to save the current state of the attribute data. The first parameter is a pointer to the attribute data buffer (see [Attribute Editor Data](#) (p.378)), and the second specifies the size in bytes of the data in the buffer. The `bCreateUndo` parameter determines whether the helper class creates an undo operation before saving the data.

For a practical example of an attribute control created with the wizard, see the [tutorials](#) (p.382).

# Creating custom attribute editor controls

The RenderWare Studio Workspace [Attributes window](#) (p.270) allows you to change attribute values, and immediately see the effect of your changes on any targets connected to the Workspace:

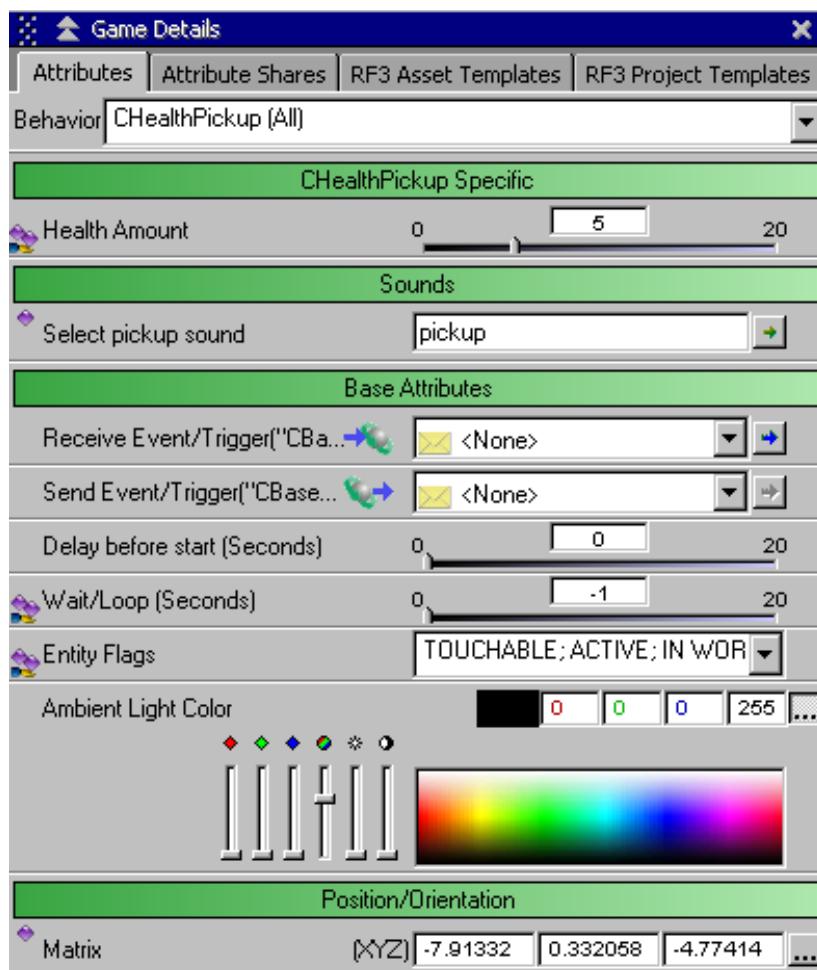


You define attributes, and choose the controls for editing them, in `RWS_ATTRIBUTE` macros in the header (.h) file of a [behavior](#) (p.337).

There are two types of attribute editor control:

- The ActiveX controls or built-in Windows user interface controls installed on your system that are created in C++ —the tutorials introduced below take you through creating these.
- The [HTML attribute editor control](#) (p.405) that displays an HTML file in the Workspace interface.

RenderWare Studio supplies several built-in types of attribute editor control, including check boxes, sliders, text boxes, and drop-down lists.



Creating a custom attribute editor control involves the following steps:

1. Creating a new project in Microsoft Visual C++ 7.1, using the RenderWare Studio Attribute Editor Control wizard
 

**Note:** This wizard works only with Microsoft Visual C++ 7.1, not with CodeWarrior or other development tools.
2. Building and registering a .dll for the new control
3. Creating or modifying a behavior with an attribute that uses the control
4. Implementing the attribute's data helper functions
5. Adding user interface elements to the control
6. Initializing the control
7. Adding the required Windows message handlers

The two lessons presented here lead you through these steps. [Lesson 1](#) (p.385) demonstrates a basic attribute editor control that does not participate in any other RenderWare Studio processing. [Lesson 2](#) (p.396) demonstrates an attribute editor control that communicates with RenderWare Studio via the Manager API.

## Before you begin

- The RenderWare Studio Attribute Editor Control wizard requires an environment variable called RENDERWARESTUDIO that points to the root folder where RenderWare Studio is installed (for example, C:\RW\Studio).

This environment variable is created when you install RenderWare Studio, so you should not have to do anything; but it's worth mentioning, in case you have subsequently edited or deleted this environment variable.

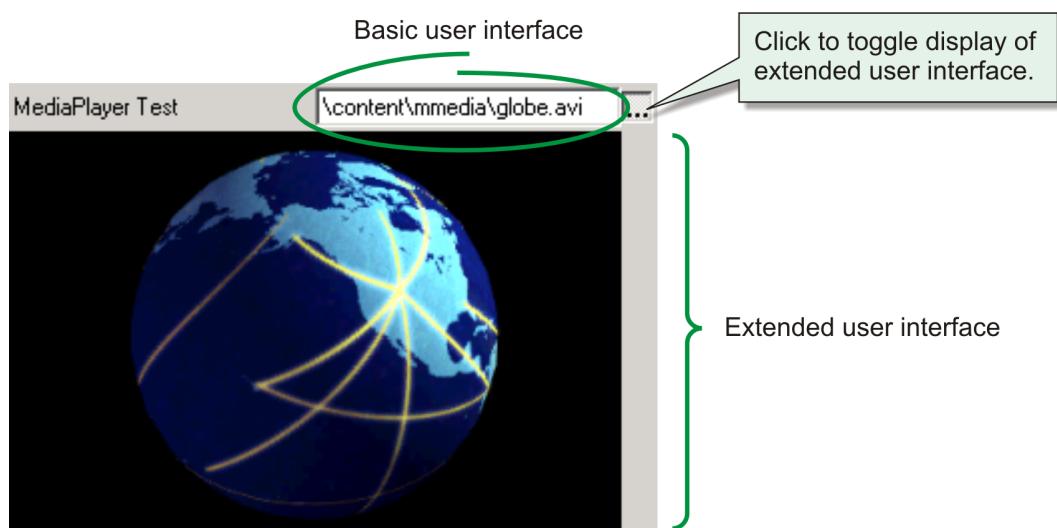
- You should understand the principles of creating user interface controls using Microsoft Visual C++ 7.1.
- Before attempting this tutorial, it is recommended that you are familiar with the [architecture of the Workspace attribute editor](#) (p.369).

# Lesson 1. Creating a control with basic and extended user interfaces

---

## In this lesson

We create an attribute editor control with a basic user interface *and* an extended user interface. The basic user interface is a edit box where you enter the value of the attribute: in this case, the path of a movie file (such as an .avi). The extended user interface uses the Windows Media Player to play the movie.



**Note:** It is possible at most stages throughout the lesson to build and test the control.

## Creating a new project in Microsoft Visual C++ 7.1

1. Start Visual Studio .NET 2003.
2. Select **File ▶ New**.  
The New Project dialog box appears.
3. In the **Project Types** list, select **Visual C++ Projects**; in the **Templates** list, select **RenderWare Studio Attribute Editor Control**.
4. In the **Name** box, type **RWSMediaPlayer**. Leave all other settings at their default.
5. Click **OK**.  
The RenderWare Studio Attribute Editor Control wizard appears.
6. Select the **Host ActiveX controls** check box.
7. Select the **Support extended user interface** check box.
8. In the **Extended user interface height** box, type **200**.
9. Click **Finish** to create the project.

10. Set the build configuration to *Release*:
  - In the *Standard* toolbar, select **Release** from the **Solution Configurations** drop-down.
11. Build `RWSMediaPlayer.dll`. The control is registered automatically at the end of the build process.

## Adding an attribute to test the new control

12. Add the following attribute to an existing behavior class definition (such as `ctutorial8.h`), between the `RWS_BEGIN_COMMANDS` and `RWS_END_COMMANDS` macros:

```
RWS_ATTRIBUTE(CMD_Movie,
              "MediaPlayer Test",
              "Specify a movie file to play",
              RWSMediaPlayer.RWSMediaPlayerCtrl,
              RwChar,
              DEFAULT( "C:\mssdk\samples\Multimedia\Media\Butterfly.mpg" ) )
```

Change the default parameter string to the path of a movie file on your system.

## Testing the control

13. Start RenderWare Studio.
14. Open an existing project that uses the behavior you have just edited, or create a new project that uses the behavior.
15. Select an entity that uses the behavior.

If the control works correctly, then a **MediaPlayer Test** attribute should appear in the Attributes window, along with a ... button to expand the user interface.

**Note:** Before rebuilding the Visual C++ project again, you need to quit RenderWare Studio. If you try to build the project while you are using the control in RenderWare Studio, then the build will fail: Visual C++ cannot write over the DLL while it is in use.

## Implementing the attribute's data helper functions

In the `RWSMediaPlayerCtrl.cpp` file:

16. Add the following line in the unnamed namespace, near the top of the file:

```
const RWSChar g_szDataType[] = _T("RwChar");
```

This tells the helper class (`IRWSAttribEditImpl`) the number of characters in the attribute data type text.

17. In the `GetNumDefaultTypeChars()` function, replace the existing return statement with:

```
return _tcslen(g_szDataType);
```

18. In the `GetDefaultType()` function, append the following code:

```
ATLASSERT(szType);
_tcscpy(szType, g_szDataType);
```

This tells the helper class (`IRWSAttribEditImpl`) that the attribute's default data type is `RwChar`.

19. Replace the contents of the `GetDefaultDataSize()` function with the following code:

```

// 2 unsigned int header
RWSUInt32 nDataSize = 2 * sizeof (RWSUInt32);

if (const RWSChar *szDefault = RWS::GetNthParam
(szParamList, nParamLength,
RWS::g_nDataTypeParam))
{
    // now look for the default string
    if (const RWSChar *cszDefault = RWS::FindParamString (
szParamList, nParamLength,
RWS::g_szDefaultValue))
    {
        cszDefault += _tcslen (RWS::g_szDefaultValue);

        // if first character is a ", return length between
        quotes
        if (_T(',') == cszDefault[0])
        {
            cszDefault++;
            while (*cszDefault && _T(',') != cszDefault[0])
            {
                nDataSize += sizeof (RWSChar);
                cszDefault++;
            }
            nDataSize += sizeof (RWSChar); // allow space for
            null terminator
        }
        else
        {
            // return length before closing )
            while (*cszDefault && _T(')') != cszDefault[0])
            {
                nDataSize += sizeof (RWSChar);
                cszDefault++;
            }
            nDataSize += sizeof (RWSChar); // allow space for
            null terminator
        }
    }
    else
        nDataSize += sizeof (RWSChar); // null terminator
}

return nDataSize;

```

The code above calculates the [data size](#) (p.378) required for the attribute data header, and (unquoted) default string contents.

20. Replace the contents of the `GetDefaultData()` function with the following code:

```

ATLASSERT (szParamList);
ATLASSERT (pData);

// set up the data header
RWSUInt32 *pHeaderData = reinterpret_cast<RWSUInt32
*>(pData);
pHeaderData[0] = RWSTypeChar;
pHeaderData[1] = 0; // number of items

```

```

        // now look for the default data
        if (const RWSChar *cszDefault = RWS::FindParamString (
                szParamList, nParamLength,
RWS::g_szDefaultValue))
        {
            const RWSChar *cszDef = &cszDefault[_tcslen
(RWS::g_szDefaultValue)];

            // if first character is a ", return characters between
quotes
            RWSUInt32 nChars = 0;
            if (_T('"') == cszDef[0])
            {
                const RWSChar *pCh = ++cszDef;
                while (*pCh && _T('"') != pCh[0])
                {
                    nChars++;
                    pCh++;
                }
            }
            else
            {
                // return string before closing )
                const RWSChar *pCh = cszDef;
                while (*pCh && _T('}') != pCh[0])
                {
                    nChars++;
                    pCh++;
                }
            }
        }

        // copy the characters, and null terminate
        memcpy (reinterpret_cast<RWSChar *>(pHeaderData + 2),
cszDef,
                nChars * sizeof (RWSChar));

        *(reinterpret_cast<RWSChar *>(pHeaderData + 2) + nChars)
= 0;

        pHeaderData[1] = nChars + 1;
    }
    else
    {
        // default to empty string
        pHeaderData[1] = 1;
        *(reinterpret_cast<RWSChar *>(pHeaderData + 2)) = 0;
    }
}

```

The above code copies the attribute data header, and default string characters into the buffer provided (via the `pData` function parameter).

## Adding user interface elements to the control

21. In Visual C++, click the Resource View tab.
22. Expand the **RWSMediaPlayer** ▶ **RWSMediaPlayer.rc** ▶ **Dialog** node.
23. Double-click **IDD\_RWSMEDIAPLAYERCTRLDLG**.

This displays a blank dialog, on which we will draw the user interface for the new attribute editor control. The user interface will consist of two controls:

- An edit box, where the user can type the pathname of a movie file.
- A Windows Media Player control, to play the movie.

First, let's add the edit box.

24. Add an edit box control in the top right-hand corner of the dialog pane.

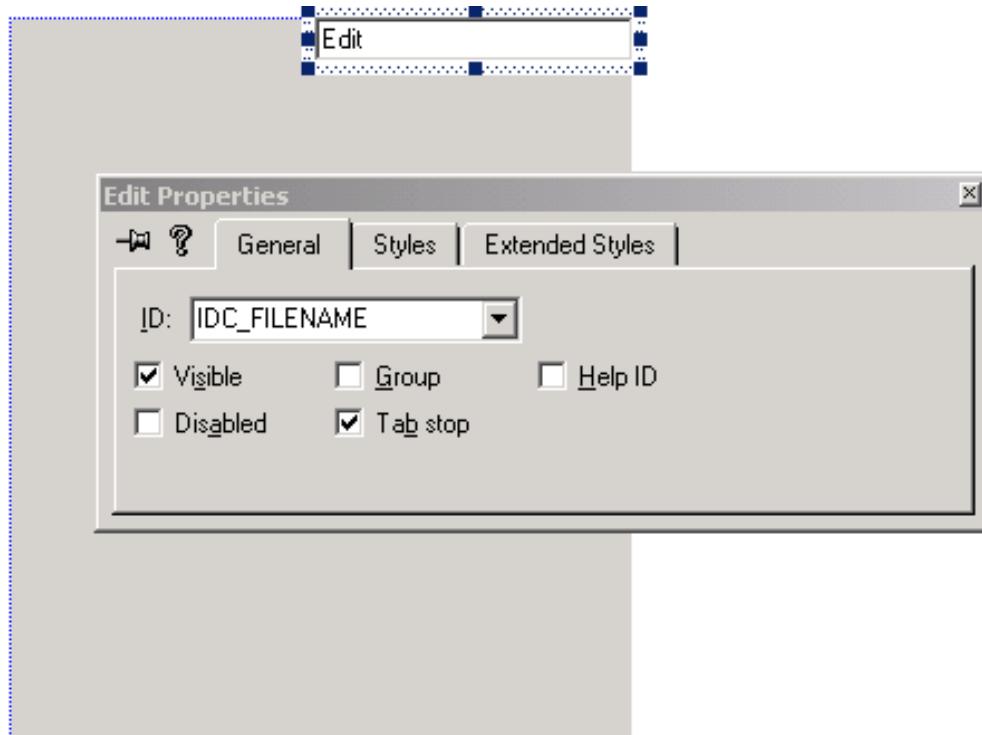
**Note:** You need to place the edit box control up against the top corner. This is because of the way the Workspace uses this dialog to display the attribute editor control user interface.

#### Basic versus extended user interface

By default, Workspace displays only the first few rows of pixels (the height of a single-line edit box) from the dialog as the “basic” user interface for the attribute editor control. Anything below this is considered an “extended” user interface, and is only displayed when you click the ... in Workspace.

In this example, Workspace displays only the edit box, until you click the ... button to display the Windows Media Player (we will add this control to the dialog soon).

25. Right-click the edit box control, and set its ID to IDC\_FILENAME.



26. In `RWSMediaPlayerCtrl.cpp`, replace the contents of the `UpdateControls()` function with the following code:

```

ATLASSERT ( IsWindow () );
CWindow WndEdit = GetDlgItem ( IDC_FILENAME );
ATLASSERT ( WndEdit.IsWindow () );

if (!pData) // indeterminate UI state (multiple different
items)
{
    // clear the edit box
    WndEdit.SetWindowText ( _T( " " ) );
}
else
{

```

```

// read the data, and update the edit box
const RWSUInt32 *pHeaderData = reinterpret_cast<const
RWSUInt32 *>(pData);
ATLASSERT (RWSTypeChar == pHeaderData[0]);

if (pHeaderData[1]) // some characters to copy
{
    WndEdit.SetWindowText (reinterpret_cast<const RWSChar
*>(pHeaderData + 2));
}
else
    WndEdit.SetWindowText (_T( " " ));
}

```

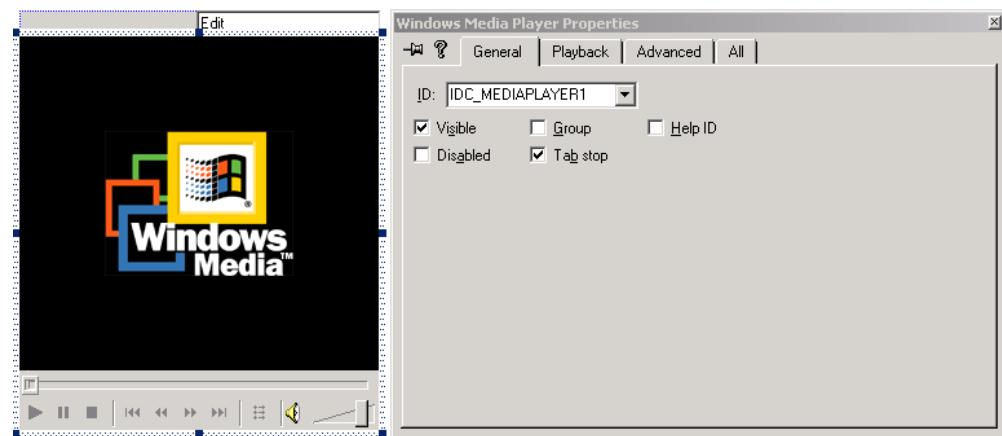
The above code reads the provided pointer to the attribute data (`pData`), and updates the filename edit box contents.

If multiple entities (with the same behavior) are being edited, and the attribute data for each is not the same, the pointer passed in will be null to indicate this. This is known as an indeterminate UI state.

## Creating the extended user interface

Now we create the extended user interface, consisting of the Windows Media Player.

27. Open the dialog again in the resource editor.
28. Right-click a blank area of the dialog pane, and then click **Insert ActiveX Control...**
29. Select the **Windows Media Player** control from the list.
30. Position the control below the edit box (you may want to shrink the dialog pane to fit around the controls):



Leave the control with its default ID, `IDC_MEDIAPLAYER1`.

## Initializing the control

31. In `RWSMediaPlayerCtrl.cpp`, replace the contents of the `CreateUI()` function with the following code:

```

ATLASSERT (IsWindow ());
CAxWindow WndMPlayer = GetDlgItem (IDC_MEDIAPLAYER1);
ATLASSERT (WndMPlayer.IsWindow ());

```

```

// query dispatch interface
IDispatch * pdispMediaPlayer = 0;
if (S_OK == WndMPlayer.QueryControl (
    IID_IDispatch, reinterpret_cast<void
*>(&pdispMediaPlayer)))
{
    // hide media player's UI controls
    OLECHAR *szMember = _T("ShowControls");
    DISPID dispid;
    if (S_OK == pdispMediaPlayer->GetIDsOfNames (IID_NULL,
&szMember, 1, LOCALE_USER_DEFAULT, &dispid))
    {
        DISPPARAMS dispparams;
        VARIANTARG vararg[1];
        dispparams.rgvarg = vararg;
        VariantInit (&dispparams.rgvarg[0]);
        dispparams.rgvarg[0].vt = VT_BOOL;
        dispparams.rgvarg[0].boolVal = FALSE;
        DISPID mydispid[1] = { DISPID_PROPERTYPUT };
        dispparams.rgdispidNamedArgs = mydispid;
        dispparams.cArgs = 1;
        dispparams.cNamedArgs = 1;

        pdispMediaPlayer->Invoke (
            dispid, IID_NULL,
            LOCALE_USER_DEFAULT,
            DISPATCH_PROPERTYPUT, &dispparams,
            0, 0, 0);
    }
}

```

The above code queries the dispatch interface for the control and, if this succeeds, the `ShowControls` method is called to hide the playback controls.

32. In `RWSMediaPlayerCtrl.h`, add the following member function declaration to the `CRWSMediaPlayerCtrl` class:

```

private:
    void SetMplayerFile ();

```

33. In `RWSMediaPlayerCtrl.cpp`, add the following member function definition:

```

void CRWSMediaPlayerCtrl::SetMplayerFile ()
{
    ATLASSERT (IsWindow ());

    CWindow WndEdit = GetDlgItem (IDC_FILENAME);
    ATLASSERT (WndEdit.IsWindow ());

    int nLen = WndEdit.GetWindowTextLength ();
    if (nLen > 0)
    {
        // save edit box text to attribute
        // allocate 2 uint header, and string data
        RWSInt32 nDataSize = 2 * sizeof (RWSUInt32) +
            (nLen + 1) * sizeof (RWSChar);
        RWSUInt32 *pHeader = reinterpret_cast<RWSUInt32
*>(_alloca (nDataSize));

        pHeader[0] = RWSTypeChar;
        pHeader[1] = nLen + 1; // number of characters

```

```

        RWSChar *szText = reinterpret_cast<RWSChar *>(pHeader +
2);
        WndEdit.GetWindowText (szText, nLen + 1);

        SaveDataToAttributes (reinterpret_cast<RWSByte
*>(pHeader), nDataSize);

        CAxWindow WndMPlayer = GetDlgItem (IDC_MEDIAPLAYER1);
        ATLASSERT (WndMPlayer.IsWindow ());

        // query dispatch interface
        IDispatch * pdispMediaPlayer = 0;
        if (S_OK == WndMPlayer.QueryControl (
            IID_IDispatch, reinterpret_cast<void
**>(&pdispMediaPlayer)))
        {
            // set filename
            OLECHAR *szMember = _T("FileName");

            DISPID dispid;
            HRESULT hr = pdispMediaPlayer->GetIDsOfNames (
                IID_NULL, &szMember, 1,
                LOCALE_USER_DEFAULT, &dispid);
            if (S_OK == hr)
            {
                DISPPARAMS dispparams;
                VARIANTARG vararg[1];
                dispparams.rgvarg = vararg;
                VariantInit (&dispparams.rgvarg[0]);
                dispparams.rgvarg[0].vt = VT_BSTR;
                dispparams.rgvarg[0].bstrVal = T2OLE (szText);
                DISPID mydispid[1] = { DISPID_PROPERTYPUT };
                dispparams.rgdispidNamedArgs = mydispid;
                dispparams.cArgs = 1;
                dispparams.cNamedArgs = 1;

                pdispMediaPlayer->Invoke (
                    dispid, IID_NULL,
                    LOCALE_USER_DEFAULT,
                    DISPATCH_PROPERTYPUT,
                    &dispparams, 0, 0, 0);
            }
        }
    }
}

```

The function above saves the attribute data, checks that the file is readable, queries the control's dispatch interface, and calls its *Filename* method.

34. Add the [following two calls](#) to the `ExpandUI()` member function:

```

if (bExpand)
{
    Invalidate ();UpdateWindow ();
}

```

These calls ensure that the media player redraws correctly when you click the ... button to display the extended user interface.

## Add some Windows message handlers

To ensure that the edited file name is updated, and resize works correctly, we need to respond to the `EN_CHANGE` edit box notification, and the `WM_SIZE` Windows message.

To add these message handlers, locate the message section in `RWSMediaPlayerCtrl.h`, inside the class definition, starting with the lines:

```
BEGIN_MSG_MAP(CRWSMediaPlayerCtrl)
    CHAIN_MSG_MAP(CComDlgControl)
```

35. Add the following lines in the message map before the `END_MSG_MAP` macro:

```
MESSAGE_HANDLER(WM_SIZE, OnSize)
COMMAND_CODE_HANDLER(EN_CHANGE, OnEditChanged)
```

These message map macros call member functions that we need to declare and implement.

36. Add the following protected member declarations to the class definition:

```
LRESULT OnSize (UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled);
LRESULT OnEditChanged (WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL& bHandled);
```

37. In `RWSMediaPlayerCtrl.cpp`, add the following code to implement these functions:

```
/*
-----
*/
LRESULT CRWSMediaPlayerCtrl::OnSize (UINT, WPARAM, LPARAM lParam, BOOL&)
{
    // set edit box size based on half total width
    CWindow WndEdit = GetDlgItem (IDC_FILENAME);
    ATLASSERT (WndEdit.IsWindow ());

    RECT rc;
    GetClientRect (&rc);
    int x = LOWORD (lParam) / 2;
    rc.left = x;
    rc.right = rc.left + x;
    rc.bottom = g_nHeightContracted - 2;
    rc.top = 2;

    WndEdit.MoveWindow (&rc);

    // set media player control size
    CAxWindow WndMPlayer = GetDlgItem (IDC_MEDIAPLAYER1);
    ATLASSERT (WndMPlayer.IsWindow ());

    GetClientRect (&rc);
    rc.top = g_nHeightContracted;

    WndMPlayer.MoveWindow (&rc);

    return 0;
}

/*
-----
*/
LRESULT CRWSMediaPlayerCtrl::OnEditChanged (WORD, WORD, HWND, BOOL&)
{
    SetMPlayerFile ();
    return 0;
}
```

```

}
/*
-----
*/

```

## Avoiding an infinite loop

This lesson is almost complete; however, one problem remains. When our user interface is updated via a call to the `UpdateControls()` member function, the edit box contents will be updated. This results in a call to `OnEditChanged`, which in turn saves the data to the attribute, and calls `UpdateControls()` again. At this point a loop can occur: we need to guard against this. A simple way to fix this problem is to add a boolean member variable which is set true for the duration of the `UpdateControls` function. In the `SetMPlayerFile` function, we don't save the attribute data if the member variable is set to true.

Let's implement this fix now:

38. In `RWSMediaPlayerCtrl.h`, add the following protected member variable to the `CRWSMediaPlayerCtrl` class:

```
RWSBool m_bUpdating;
```

39. Add the following initializer (shown **shaded** below) to the `CRWSMediaPlayerCtrl` constructor's initializer list:

```
CRWSMediaPlayerCtrl () : m_bExpandedUI (RWSFalse), m_bUpdating (RWSFalse) { m_bWindowOnly = TRUE; }
```

40. In `RWSMediaPlayerCtrl.cpp`, add the following code to the start of the `UpdateControls()` member function:

```
m_bUpdating = RWSTrue;
```

and the following code to the end:

```
m_bUpdating = RWSFalse;
```

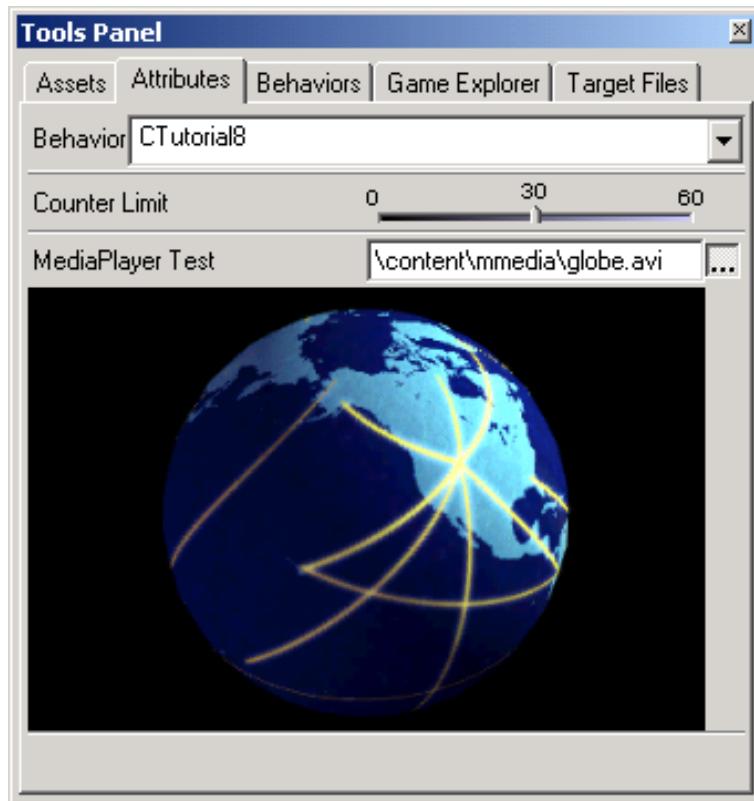
41. In the `SetMPlayerFile()` member function, change the line:

```
SaveDataToAttributes (reinterpret_cast<RWSByte *>(pHeader),  
nDataSize);
```

to:

```
if (!m_bUpdating)  
    SaveDataToAttributes (reinterpret_cast<RWSByte  
*>(pHeader), nDataSize);
```

Your attribute editor control should now be able to accept the path of a movie file, and then play the movie:



**Note:** To view the movie file again (after the first time), click the media player control.

This is the end of the lesson.

## Additional exercises

Some additional exercises that you might wish to try:

- Add a browse button to the user interface that pops up a file dialog asking for the movie file path.
- Insert a different ActiveX control, or your favourite web page, instead of the Windows Media Player control.
- Rather than saving the filename in an attribute, open the file and embed it into the attribute data as a byte stream (type `RWSTypeByte` instead of `RWSTypeChar`). The embedded data will be sent to a connected target console across the network, rather than sending the file name.
- Use the RenderWare Studio Manager API from within your control to display all the assets of a certain type that are children of the currently selected entities. The API is fully accessible from the control code, and the particular entities, attributes, commands, and classes are available as members of your control class via the following (inherited) class members:

```
RWSID m_CommandID;
RWSID m_ClassID;
EntityToAttribMap m_EntityToAttribMap;
```

## Lesson 2. Creating a new attribute editor control (advanced)

---

### In this tutorial

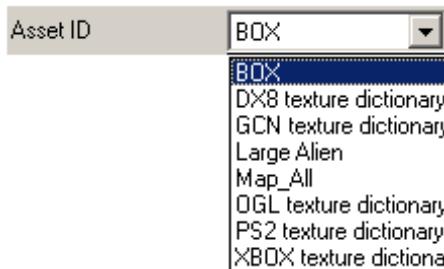
You use a Visual C++ wizard (supplied with RenderWare Studio) to create a new type of control for editing attributes.

You will need:

- Microsoft Windows Template Library 7.0 toolkit, available free from the [Microsoft download center](#) ([www.microsoft.com/downloads/search.aspx?displaylang=en](http://www.microsoft.com/downloads/search.aspx?displaylang=en)).

After downloading the toolkit, open the self-extracting zip file and select a suitable location in your computer's folder hierarchy for the contents of the file. After extracting the files, note down the path of the folder named `Include` and add this path to your C++ compiler's "include" search path.

The new control is a drop-down list box that lists all assets in the game database:



When you select the asset name from the list, the control sends the GUID of the asset to any connected game targets.

When you add a new asset to the game, an entry corresponding to that asset is registered in the [game database](#) (p.76) in the form of an XML file. Each XML file that references an asset is named using its own global unique identifier (GUID).

**Note:** The sample code provided here is intended to demonstrate how to build your own attribute controls in RenderWare Studio. It is not a set of code that you would use in practice as the list of assets generated would be too long and would require further filtering.

To generate the control, you use the [RWS Attribute Editor Control wizard](#) (p.376). (To generate more general ActiveX controls in RenderWare Studio you would use the RWS ActiveX Control wizard).

### Creating a new project in Microsoft Visual C++ 7.1

1. Start Visual Studio .NET 2003.
2. Select **File ▶ New**.

The New Project dialog box appears.

3. In the **Project Types** list, select **Visual C++ Projects**; in the **Templates** list, select **RenderWare Studio Attribute Editor Control**.
  4. In the **Name** box, type `RWSAssetList`. Leave all other settings at their default.
  5. Click **OK**.
- The RenderWare Studio Attribute Editor Control wizard appears.
6. Leave all boxes in the dialog unchecked, and then click **Finish** to generate the project.
  7. Set the build configuration to *Release*:
    - In the *Standard* toolbar, select **Release** from the **Solution Configurations** drop-down.

**Note:** To use the *Debug* configuration:

    - a. Select the appropriate item in the **Solution Configurations** drop-down.
    - b. Select **Project ▶ Properties...**
    - c. In the dialog that appears, choose the **Configuration Properties ▶ Linker ▶ Input** item.
    - d. In the **Additional Dependencies** box, change the file name from `RWSud.lib` to `RWSu.lib`.  8. Build the control, which is registered automatically at the end of the process.

## Modifying a behavior by adding an attribute to test the control

To test the new control:

9. Select one of the behavior class definitions supplied with RenderWare Studio, for example “CTutorial3”.
10. In the corresponding header file, add the following attribute between the `RWS_BEGIN_COMMANDS` and `RWS_END_COMMANDS` macros.

```
RWS_ATTRIBUTE(CMD_AssetId,
              "Asset ID",
              "Specify an asset to stream from the
workspace",
              RWSAssetList.RWSAssetListCtrl,
              RwChar, 0)
```

**Tip:** If you specified a different project name from that suggested in the tutorial and need to find the name of the identifier used to create and register the control:

- Open your project's `.rgs` file.
- Search for the string `"VersionIndependentProgID = s"`
- The project's identifier will immediately follow the string above.

## Implementing the attribute's data helper functions

In this tutorial, you need to edit two files: `RWSAssetListCtrl.cpp` and `RWSAssetListCtrl.h` (assuming that you've named your project “RWSAssetList”). These files contain the functions that define how your user

interface works and the data you wish to set within your control's attribute.

These functions are called as soon as you need to display your control and the default value for its attributes in the Renderware Studio workspace, but before any actual attribute values have been assigned. If a console is connected to the workspace while the attribute is in this state, these functions are called to send the [data structure](#) (p.378) for the default value to the connected targets.

A helper class is provided to simplify implementing the necessary interfaces that make your control an attribute editor control. The interface used by the Workspace to communicate with each attribute editor control is defined in `RWSAttribEditIF.h`. The helper class that implements this interface is defined in the file `RWSAttribEdit.h`.

**Note:** It is not normally necessary to modify these two files, as they perform tasks required by all attribute editor controls, for example:

- Monitoring database changes in attributes, classes and commands.
- Calling several helper functions that you must implement.

11. The first function to implement is `GetNumDefaultTypeChars()`. This is one of four functions responsible for generating default attribute data that is then sent to the game. These functions build up the RWS attribute data structure that represents the **default value** of your attribute. In this example the default value is an empty string since the list box control is empty when first displayed.

To implement this function: In the file `RWAssetListCtrl.cpp`, return the number of characters in the string that represents your default data type as follows:

```
RWSUInt32 CRWSAssetListCtrl::GetNumDefaultTypeChars (const
RWSChar * const,
RWSUInt32)
{
    return wcslen (L"RwChar");
}
```

Source: `RWAssetListCtrl.cpp`

The return value of this function is used by the helper class file (`RWSAttribEdit.h`) to allocate space in memory for the `Type` member of the RWS Attribute structure.

12. The next function is `GetDefaultType()`. This is called by the helper class to fill in the `TypeGetNthParam` field of the RWS Attribute structure with your default data string as defined by the value returned above by `GetNumDefaultTypeChars`. Implement this as follows:

```
void CRWSAssetListCtrl::GetDefaultType (RWSChar * const
szType,
                                         const RWSChar * const,
                                         RWSUInt32)
{
    wcscpy (szType, L"RwChar");
}
```

Source: `RWAssetListCtrl.cpp`

As this is a descriptive string that is only used for storing in the XML database, it could equally well be any of the other possible Attribute Editor Data Types or some other user-defined data type such as, for example

MyAIdata.

13. Next, implement the `GetDefaultDataSize()` function as follows:

```
RWSUInt32 CRWSAssetListCtrl::GetDefaultDataSize (const RWSChar
* const,
                                                 RWSUInt32)
{
    // 2 unsigned int header
    RWSUInt32 nDataSize = 2 * sizeof (RWSUInt32);

    // + 1 Char for an empty string
    nDataSize += sizeof (RWSChar);

    return nDataSize;
}
```

Source: `RWSAssetListCtrl.cpp`

The attribute is not displayed in the Game Explorer panel of the Workspace until the default value has been changed. At the stage where it is not yet displayed, this function is called to allocate a buffer in memory to store the value of a binary representation of the default value.

**Tip:** To query the default data type from the `RWS_ATTRIBUTE` macro, use the functions provided in the files `utility.h` and `utility.cpp`. For example, the function `GetNthParam` returns a string containing a parameter from the source-code macro defining the attribute in your game header file. This would be useful where you needed to change just one parameter in an attribute containing many different parameters.

14. Replace the `GetDefaultData()` function with the following code:

```
void CRWSAssetListCtrl::GetDefaultData (RWSByte * const pData,
                                         const RWSChar * const,
                                         RWSUInt32)
{
    // fill in first data tag header here
    RWSUInt32 *pDataHeader = reinterpret_cast<RWSUInt32
*>(pData);
    // specify data type
    pDataHeader[0] = RWSTypeChar;
    // specify the number of data items
    pDataHeader[1] = 1; // just 1 item

    // fill in the default data size - just an empty string
    RWSChar *szDefaultType = reinterpret_cast<RWSChar
*>(pDataHeader + 2);
    *szDefaultType = 0; // null terminator
}
```

Source: `RWSAssetListCtrl.cpp`

This function returns the value of the empty string to populate the `RWSUInt32` section of the data structure. In this example the number of bytes required in the buffer would be allocated as laid out in the [data structure](#) (p.378) for the `RWSUInt32`.

## Adding user interface elements to the control

The next step is to add a combo box for the drop-down list of assets:

15. Open the file `RWSAssetListCtrl.h`
16. Add a `CComboBox` member variable:

17. At the top of the file, insert the following lines:

```
#include <atlapp.h>
#include <atlctrls.h>
```

18. Add the following member variable in the class declaration:

```
protected:
    // data
    CComboBox m_WndCombo;
```

19. Re-open the file RWSAssetListCtrl.cpp.

20. To allow the control to create or initialize any user interface items in the control, implement the CreateUI() member function as follows:

```
void CRWSAssetListCtrl::CreateUI ()
{
    ATLASSERT ( IsWindow () );

    if ( !m_WndCombo.IsWindow () )
    {
        // create the combo box control
        RECT rc;
        GetClientRect ( &rc );
        rc.top += 2;
        rc.bottom = 200; // specify combo's initial drop-height
        rc.left = rc.right / 2;
        m_WndCombo.Create ( m_hWnd, rc, 0, WS_VISIBLE |
WS_CHILD | CBS_DROPDOWNLIST | CBS_SORT | CBS_HASSTRINGS | WS_VSCROLL );
        ATLASSERT ( m_WndCombo.IsWindow () );

        // set a nicer font
        m_WndCombo.SetFont ( HFONT ( GetStockObject
(ANSI_VAR_FONT) ) );

        PopulateCombo ( m_WndCombo );
    }
}
```

Source: RWSAssetListCtrl.cpp

Place the PopulateCombo() function in an anonymous namespace near the top of the file as follows:

```
void PopulateCombo ( CComboBox &WndCombo )
{
    WndCombo.ResetContent ();

    // Query all assets in database
    // (Attribute editor controls are already attached to the
database)
    RWSID AssetID = RWSGetFirst ( RWSAssetID );
    while ( AssetID )
    {
        RWSAsset AssetData = { 0 };
        RWSAssetGet ( AssetID, &AssetData );

        // Add asset's name to combo box
        if ( AssetData.Name )
        {
            int nSel = WndCombo.AddString ( AssetData.Name );
            if ( nSel != CB_ERR )
                WndCombo.SetItemData ( nSel, AssetID );
        }
    }
}
```

```

    }
    RWSAssetFreeData (&AssetData);
    AssetID = RWSGetNext (AssetID);
}
}

```

Source: RWSAssetListCtrl.cpp

**Tip:** For faster lookup, each list box item has some data associated with it corresponding to the asset's run time-ID (RWSID).

## Initializing the control

- To fulfil the requirements of an attribute editor control, you implement the following function, `UpdateControls()`. This function is called once to initialize your user interface and once each time the data changes for the attribute that you are displaying. Implement this function as follows:

```

void CRWSAssetListCtrl::UpdateControls (const RWSByte * const
pData)
{
    // update controls from the given data
    int nSelection = -1; // default to nothing selected

    // null pData represents multiple attributes edited
    // simultaneously with no like-data.
    if (const RWSUInt32 *pDataHeader = reinterpret_cast<const
RWSUInt32 *>(pData))
    {
        // validate data header
        ATLASSERT (RWSTypeChar == pDataHeader[0]);

        // locate the UID in the database
        if (pDataHeader[1] > 1) // string has some characters
        {
            const RWSChar *szAssetID = reinterpret_cast<const
RWSChar *>(pDataHeader + 2);

            if (RWSID AssetID = RWSGetID (RWSAssetID, szAssetID))
            {
                RWSAsset AssetData = {0};
                RWSAssetGet (AssetID, &AssetData);

                // find the string in the combo box
                if (AssetData.Name)
                    nSelection = m_WndCombo.FindStringExact (-1,
AssetData.Name);

                RWSAssetFreeData (&AssetData);
            }
        }
    }

    // update the UI
    m_WndCombo.SetCurSel (nSelection);
}

```

Source: RWSAssetListCtrl.cpp

## Adding the required Windows message handlers

- Next, we need to add a handler for `WM_SIZE` messages sent to our control by its parent window so that the control can be resized.

To do this, open the file `RWSAssetListCtrl.h` and add the following line

inside the control class's message map, below the WM\_MOUSELEAVE handler:

```
MESSAGE_HANDLER(WM_SIZE, OnSize)
```

23. Add the Onsize() function to the class:

```
LRESULT OnSize (UINT, WPARAM, LPARAM, BOOL &);
```

24. Re-open the file RWSAssetListCtrl.cpp.
25. To ensure that the combo box always occupies the right-hand half of the space available, implement the sizing code as follows:

```
LRESULT CRWSAssetListCtrl::OnSize (UINT, WPARAM, LPARAM
lParam, BOOL &)
{
    // set position of combo box
    if (m_WndCombo.IsWindow ())
    {
        RECT rcCombo;
        m_WndCombo.GetClientRect (&rcCombo);

        int x = LOWORD (lParam) / 2;
        m_WndCombo.SetWindowPos (0, x, 2, x, rcCombo.bottom,
SWP_NOZORDER);
    }
    return 0;
}
```

Source: RWSAssetListCtrl.cpp

**Note:** The left hand side of the combo box contains the attribute name that has been drawn by the helper class.

26. The remaining functionality to add to the control is the ability to save the new data to the attribute or attributes that you are editing. To do this, add a message handler to the class to respond to the combo box's selection changing. Adding the following inside the class's message map in the file RWSAssetListCtrl.h does this:

```
COMMAND_CODE_HANDLER(CBN_SELENDOK, OnSelEndOK)
```

**Note:** For other types of control you would need to add different Windows message-handlers.

Complete adding this functionality with a declaration of the class member function in the file RWSAssetListCtrl.cpp to handle the message with the following:

```
LRESULT CRWSAssetListCtrl::OnSelEndOK (WORD wNotifyCode, WORD
wID,
                                         HWND hWndCtl, BOOL&
bHandled)
```

27. Next, add the OnSelEndOK implementation to the class definition:

```
LRESULT CRWSAssetListCtrl::OnSelEndOK (WORD wNotifyCode, WORD
wID,
                                         HWND hWndCtl, BOOL&
bHandled)
{
    // called when combo box selection has changed
    // get current selection
    int nSel = m_WndCombo.GetCurSel ();
    if (nSel != CB_ERR)
```

```

{
    // get the item data (RWSID of asset)
    RWSID AssetID = m_WndCombo.GetItemData (nSel);
    ATLASSERT (AssetID);

    // inquire the UID of the asset object
    if (RWSUInt32 nChars = RWSGetUID (AssetID, RWSAssetID,
0, 0))
    {
        // allocate some stack memory large enough for
header, and UID
        int nDataSize = 2 * sizeof (RWSUInt32);
        nDataSize += nChars * sizeof (RWSChar);
        RWSByte *pData = reinterpret_cast<RWSByte *>(_alloca
(nDataSize));

        // set up the header
        RWSUInt32 *DataHeader = reinterpret_cast<RWSUInt32
*>(pData);
        DataHeader[0] = RWSTypeChar; // Data type
        DataHeader[1] = nChars; // Number of data items

        // copy the UID
        RWSChar *szAssetUID = reinterpret_cast<RWSChar
*>(DataHeader + 2);
        *szAssetUID = 0;
        RWSGetUID (AssetID, RWSAssetID, szAssetUID, nChars);

        // now save the data
        if (!SaveDataToAttributes (pData, nDataSize)) // save
with undo
        {
            // This call can fail if the AlienBrain check-out
for the entity was
            // denied.
            //
            // To fix this, the last selection should be saved
as a member variable
            // and set if successful, or restored if failed.
            // Code elided for clarity ;
        }
    }
    return 0;
}

```

Source: RWSAssetListCtrl.cpp

This function builds up the data required to save the attribute. The save is performed by the helper class function SaveDataToAttributes.

This function in turn calls RWSAttributeSet with the attribute data provided.

A separate ActiveX control, known as Targets, monitors the database, and receives a Manager API callback when the attribute data changes. The callback responds by sending the attribute across the network to any games that are connected.

To complete the functionality for any behavior class using this control, you need to update the behavior class's HandleAttributes function, to receive a string containing the asset's UID using the following sample code:

```

case CMD_AssetId:
    if (const RwChar *szAssetUID = attrIt->GetAs_char_ptr
())

```

```
{  
    OutputDebugString (szAssetUID);  
}  
  
        break;  
}  
++attrIt;
```

# Creating an HTML attribute editor control

---

You can define a custom attribute editor control that displays an HTML file. In your behavior source code, you refer to the URL of the HTML file:

```
RWS_BEGIN_COMMANDS
...
RWS_ATTRIBUTE(CMD_8, "behavior name", "tooltip", HTML,
STRING, URL("file:///c:/test.html"))
...
RWS_END_COMMANDS;
```

The HTML file can contain any code that can be displayed by Internet Explorer. For example, it can contain simple HTML (pictures or documentation for behaviors), dynamic HTML with script that performs user interface functions such as buttons, list boxes, check boxes, etc., or a mixture of both.

## Setting default data

To create default data for your attributes, add a script method called `OnGetDefaultData` to the HTML page. This will be invoked whenever the default data needs to be generated for this attribute (for example, during a build):

```
Sub OnGetDefaultData (oAttributeData, strType)
...
End Sub
```

The `oAttributeData` parameter is an interface to an `IRWSData` interface. The `IRWSData` object is to be populated with data that represents the default state of the attribute.

The `strType` parameter is a string that is to be filled in with the default type for your attribute. This string can be obtained from the command that this attribute represents (see below).

## Updating the UI when the data changes

If you provide a script method called `OnRefresh` in your HTML page, this will be invoked whenever the attribute's data is changed, and also after the page has loaded (in order for you to update your user interface).

```
Sub OnRefresh (oAttributeData)
...
End Sub
```

Here the `oAttributeData` object represents an `IRWSData` interface that can be read to populate your user interface.

## Communicating with the host control

You can access the interface for the hosting control through an object which is added to the page's namespace. This object is accessed through the `RWSHTMLControl` object. For example, when your page loads, you can set the height of the user interface through the `height` method; for example, `RWSHTMLControl.Height = 400`.

## RWSHTMLControl COM interface

The RWSHTMLControl COM interface consists of just three properties.

### **AttributeData property**

Gets or sets the data for the attribute(s) being edited. If the value retrieved is null, this indicates that multiple attributes with different data are being displayed.

*RWSHTMLControl.AttributeData [= RWSData]*

You can access the command ID of your control (which gives you access to the parameters in your attribute macro in behaviour source), get and set the attribute's data, and set the height of the user interface.

### **CommandID property**

Read-only. Gets the command ID of the attribute(s) being edited.

*Long = RWSHTMLControl.CommandID*

### **Height property**

Sets the height of the control's window.

*IRWSHTMLInterface.Height [= Long]*

## Access to IRWSScript

You can access the Manager API's scripting interface (IRWSScript), using an object that's automatically added to the page's namespace, called RWSScript. For example, in your page's script you can do:

```
Dim oCommand
Set oCommand = RWSScript.Command (RWSHTMLControl.CommandID)
If Not oCommand Is Nothing Then
    MsgBox oCommand.Name

    ' You can also extract parameters from the attribute macro
    ' using the oCommand.ParamList property
    ' (see the IRWSCommand documentation in the IRWSScript
    documentation)
End If
```

You can also handle events through the script interface, for example:

```
Sub RWSScript_OnChange (oAPIObject)
    ' This will be called for *any* Manager API object that
    changes, unless you set the RWSScript.EventMask property!
    MsgBox "OnChange!"
End Sub
```

## Adapting the Game Framework

---

The Game Framework is a set of services, implemented in C++, that you use to develop a game with RenderWare Studio.

The Game Framework is supplied with RenderWare Studio as source code. It is designed to be incorporated into your own game code from the earliest “proof of concept” through to the final commercial game.

# Building the Game Framework

---

## Source files

The Game Framework source files are supplied in:

C:\RW\Studio\console\game\_framework\source

**Note:** C:\RW\Studio is the default location for RenderWare Studio. You might have installed RenderWare Studio in a different folder.

## Project files

To build a target executable from the Game Framework source, use the project files supplied in:

C:\RW\Studio\console\game\_framework\your *platform*\game\_framework  
where *your platform* depends on your compiler and your target console:

### **mw\_sky**

Contains CodeWarrior project files for building a PlayStation 2 executable.

### **mw\_gcn**

Contains CodeWarrior project files for building a GameCube executable.

### **sn\_gcn**

Contains Visual C++ 7.1 project files for using ProDG to build a GameCube executable.

### **sn\_sky**

Contains Visual C++ 7.1 project files for using ProDG to build a PlayStation 2 executable.

**Caution:** Before using ProDG to compile any PlayStation 2 build, you must add the PlayStation 2 include paths to your list of include directories in Visual Studio (**Tools ▶ Options ▶ Projects ▶ VC++ Directories**). For example:

```
c:\usr\local\sce\common\include
c:\usr\local\sce\ee\include
```

### **win32**

Contains Visual C++ 7.1 project files to build a 32-bit Windows executable.

### **xbox**

Contains Visual C++ 7.1 project files to build an Xbox executable.

### Game Framework versus empty framework

The project in the . . . \game\_framework folders include the entire Game Framework. If you would prefer to start with the *empty framework* (a much smaller code base, with just enough functionality to boot up the consoles and connect to the Workspace), use the projects in:

```
C:\RW\Studio\console\empty_framework\your
platform\empty_framework
```

## Subprojects

The Game Framework (and empty framework) project includes two subprojects:

### **Core** (also known as `gfCore`)

Stored alongside the Game Framework project files in:

`C:\RW\Studio\console\game_framework\your_platform\gfcore`

(The empty framework project also refers to this path.)

### **Loading screen**

Stored in a separate folder structure:

`C:\RW\Studio\console\plugins\loading_screen\your_platform`

The loading screen project creates the screen that the Game Framework shows when it is awaiting connection to the Workspace:



## Build configurations

Each supplied project defines several build configurations. These configurations are divided into two types:

### **Design**

Includes the code that allows the Game Framework to receive game data from the Workspace. Design builds of the Game Framework expect to receive a data stream from the Workspace.

### **Production**

Excludes the code that allows the Game Framework to receive game data from the Workspace. Production builds read a stream file, so they do not require the Workspace. For this reason, production builds are sometimes referred to as “stand-alone” builds.

For either configuration type, you can select between:

### **Debug**

Includes debugging code in the target executable.

### **Metrics**

Generates metrics data, useful for tuning the application during development.

### **Release**

Excludes both debugging and metrics code from the target executable.

If the build configuration name in the project does not include the word “Design”, then it is a production build. For example, “Release” identifies a production build that does not include any debugging or metrics code; “Design Debug” identifies a design build that includes debugging code. (Typically, you use the “Release”

build to build the final, stand-alone target executable for distribution.)

## PlayStation 2 build configurations

The PlayStation 2 project files also include the following CD-ROM build configurations:

### Design Release CDROM

Allows you to test your game on a Debugging ("Test") Station, while continuing to use Workspace to change the game data.

(Unlike the more expensive Development Tool, the Debugging Station can read an executable from CD only.)

### Release CDROM

The "master" version for testing and final distribution.

## GameCube build configurations

The GameCube projects also include "Design ... Broadband" build configurations, for connecting the RenderWare Studio Workspace to a GDEV or DDH via a [broadband adapter](#) (p.215) (across a LAN).

**Note:** These are only the most commonly used build configurations. The supplied projects also include other build configurations for specific purposes, depending on the platform; the names of these additional build configurations should be self-explanatory.

## #define directives that define the build configurations

Each build configuration is defined by a particular combination of #define directives:

#define	Design builds			Production builds	
	Design Debug	Design Metrics	Design Release	Debug	Metrics
_DEBUG	●	○	○	●	○
NDEBUG	○	●	●	○	●
RWDEBUG	●	○	○	●	○
RWMETRICS	○	●	○	○	●
RWS_BOOTUP_FILE "path"	○	○	○	●	●
RWS_DEBUGSWITCHES	●	●	○	○	○
RWS_DEBUGTOOLS	●	●	○	○	○
RWS_DESIGN	●	●	●	○	○
RWS_DISABLE_MEMORY_CHECKING	●	●	○	○	○
RWS_EVENTVISUALIZATION	●	●	○	○	○

In addition to the above directives:

- The PlayStation 2 CD-ROM build configurations specify #define CDROM.
- The GameCube "Design ... Broadband" build configurations specify

```
#define RWS_BROADBAND.

_DEBUG
    Includes debugging code.

CDROM
    For PlayStation 2 CD-ROM builds, as described above.

NDEBUG
    Excludes debugging code.

RWDEBUG
    Includes RenderWare Graphics debugging code and links with the
    RenderWare Graphics debug libraries. For details, see the RenderWare
    Graphics documentation.

RWMETRICS
    Includes RenderWare Graphics metrics code and links with the RenderWare
    Graphics metrics libraries. For details, see the RenderWare Graphics
    documentation.

RWS_BOOTUP_FILE "path"
    Specifies the path of the boot-up stream file (p.233) containing the game
    data for the default level (the level that you want to appear when the game
    starts).

RWS_BROADBAND
    For GameCube design builds that connect to the Workspace via a
    broadband adapter.

RWS_DEBUGSWITCHES
    Enables various true/false flags useful for debugging. You can control these
    switches via the CDebugTools (p.219) behavior.

RWS_DEBUGTOOLS
    Enables debugging tools; these include various immediate-mode drawing
    primitives such as lines, ellipses, and triangles. Requires DEBUGSWITCHES.

RWS_DESIGN
    Enables the Game Framework to connect to and receive game data from
    the Renderware Studio Workspace. Requires DEBUGSWITCHES.

RWS_DISABLE_MEMORY_CHECKING
    Disables memory debugging tools.

RWS_EVENTVISUALIZATION
    Enables the ability to overlay (p.219) the game display with the events firing
    between 3D objects (atomics, clumps, or cameras) in the scene. Requires
    DEBUGSWITCHES and DEBUGTOOLS.

Tip: If you override the CeventVisualisation::Get3Dpos  

(CeventVisualisation.cpp) virtual function, then you can visualize the  

events firing for any object that is capable of returning a world space 3D  

position.
```

## Other build-related #define directives

The following #define directive is not necessarily specified by the supplied projects; depending on your target, you might need to specify this directive in your code.

**WITH\_AUDIO**

Includes support for RenderWare Audio.

## PlayStation 2-specific directives

**DVDROM**

For DVD-ROM builds.

**IOP\_MODULEPATH "path"**

Specifies the path to the Sony IOP modules (typically  
/usr/local/sce/iop/modules/, defined in  
core\startup\SkyIOP.h).

**IOP\_RWSCOMMSPATH "path"**

Specifies the path to the RenderWare Studio IOP modules (typically  
/rwstudio/console/game\_framework/bin/modules/, defined in  
core\startup\SkyIOP.h).

**RWS\_AUDIO\_MODULEPATH "path"**

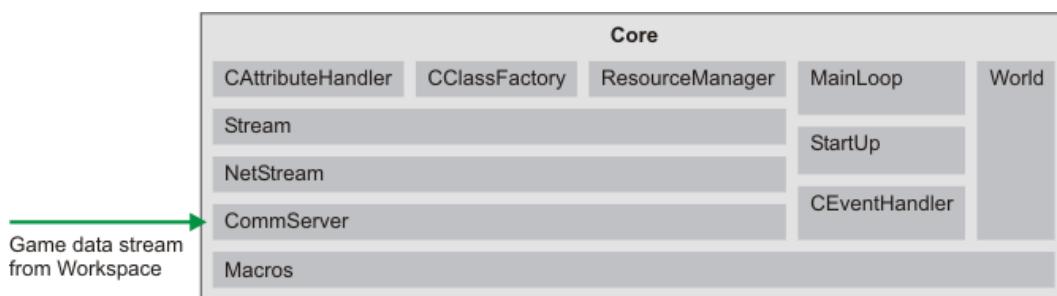
Specifies the path to the RenderWare Audio IOP modules (typically  
/rw/audio/lib/ps2/[debug/metrics/release], defined in  
core\startup\SkyIOP.h).

# Game Framework core architecture

---

After working on several games, and discussing games programming with many other developers, you start to see that the majority of games are built in a very similar way. This suggests that the code and data that form the building blocks of these games is likely to be quite similar.

The RenderWare Studio Game Framework core provides the common building blocks for developing a game of any genre; it also defines a flexible, extensible structure for developing code that is specific to your game.



By providing this common functionality and structure, the Game Framework core dramatically increases the potential for code reuse, saving developers time (and therefore money).

Just as artists and audio designers create artwork and audio “assets” with which to build the final game, one task of games programmers is to build blocks of game code that can be associated with these assets. In RenderWare Studio, these blocks of code are called *behaviors*. Behaviors share several common traits:

- They may be instanced many times within the game.
- They have parameters or attributes that are used to initialize them.
- They often communicate with each other to create complex game play.
- They are typically associated with other data such as graphics and audio, known in RenderWare Studio as assets (or, sometimes, resources).

Behaviors are typically written as a single C++ class, and always derived at some level from the abstract base class `CAttributeHandler` (see “Attribute handlers”, below). When behaviors are associated with geometry in the Workspace then, on the target console, an instance of the class derived from `CAttributeHandler` will be created, and a pointer to the `CAttributeHandler` is maintained by the core. When an attribute of a behavior is modified in the Workspace then, on the target, this pointer is used to call the `CAttributeHandler`s virtual function `HandleAttributes`.

The key components of the Game Framework core are described in more detail below.

## Network communication layer

The Game Framework provides a data-driven solution to game development. A network communication layer transports this data between the Workspace and

the target console. Typically, this layer sends data to the framework as binary streams. These binary streams are the same format as the data used in the final commercial game. This means that the code that the games programmer writes to initialize the behavior classes during development is the same code that is used in the final commercial game. The advantage of this is there is very little additional code that needs to be written in order to take advantage of the real-time tuning of RenderWare Studio.

The network communication layer is implemented by overloading the RenderWare Graphics stream object; these streams are typically used for loading data from files stored on a mass storage device such as a DVD. By implementing the network layer in this way, existing code that makes use of RenderWare streams becomes network enabled. The advantage of this is that the same code can be used during game design and final production. Using the same high-level method to load streams (that typically contain the game assets) ensures a similar memory footprint for design and production versions of the game.

## The class factory

The class factory provides a data-driven solution to instancing C++ classes on the target. Classes are identified by name and mapped to a factory method that returns an instance of the class; this is similar to the Factory Method creational pattern [Gamma, E et al. (2000)].

The class factory is closely linked to the `CAttributeHandler` class. The class factory can only create instances of classes derived from `CAttributeHandler` (described in “Attribute handlers”, below). Using the name of the class allows classes to be registered with the class factory at run time, minimizing the number of files that need to be modified or recompiled as new classes are added. This is achieved by implementing the registration method as a statically instanced class. Registering a behavior associates a `MakeNew` method for the class with the class factory. The standard implementation of the `MakeNew` function can be overridden by the class to handle special cases such as restricting how the class is allocated on the target.

## Attribute handlers

To remotely modify an instance of a class on the target, we needed to solve a few problems. First, we needed to identify the instance of the class on the target. Second, we needed to create a standard interface for communicating with the class. The abstract base class `CAttributeHandler` provides this interface, and maintains a map of IDs to instances of `CAttributeHandler`. These IDs uniquely identify each instance of the class. The Workspace generates this unique ID for each instance of a class derived from `CAttributeHandler`; the target maps each ID to a pointer to the appropriate instance of `CAttributeHandler`.

Data is passed to the class in a binary format and is accessed via the `CAttributePacket` class; the data is generated by the RenderWare Studio Workspace by the RenderWare Studio Manager API in a format suitable for the target platform (that is, issues such as endianess are resolved before the data is sent to the target). A `CAttributePacket` provides a method of serializing the data required to initialize the class. The format of the serialization also enables individual attributes to be modified without recreating the class; this is a necessary feature during real-time editing, as it enables attributes to be modified

without affecting the overall state of the behavior. The `CAttributePacket` also identifies attributes associated with different classes within a class hierarchy. This enables base classes to be specialized by derivation without any modification regarding the attributes.

Associated with the `CAttributeHandler` class is the code decoration that enables programmers to specify the attributes that need to be serialized and the user interface controls that should be used to generate this data. This is added as part of the game code to the class declaration and removed during compilation. The immediate effect of automating the generation of the user interface associated with the behavior is to reduce the need to develop and maintain custom tools; the vast majority of the user interface controls that the programmer requires in order to parameterize the behavior are provided by the RenderWare Studio Workspace. Any components that are not available as standard can be added to the Workspace as a standard ActiveX control. A more subtle effect is to tie the definition of the user interface of the game component or behavior to the implementation of the behavior; this means that if the game code is shared between teams, the user interface goes with it. The code decoration is not compiled to generate the user interface; rather, the RenderWare Studio Workspace generates the user interface by parsing the code decoration in the game code. If the code decoration is modified, the Workspace can reparse and regenerate the user interface dynamically, without needing to be shut down and restarted.

To summarize: the `CAttributeHandler` class, in conjunction with the `CClassFactory` and network layer, enables developers to create game entities with attributes that can be modified in real time from the RenderWare Studio Workspace. The `CAttributeHandler` class does this by providing:

- A method of transferring data to the target while it is running
- Registration of game entities that can be instanced by name on the target
- A method of initializing these game entities during their construction
- A mapping of unique IDs that are used to identify an instance of a class, in order to pass on modifications to their attributes
- An interface that derived classes can implement to receive these modifications

Here is an example of decorated behavior code, and the resulting attribute editor user interface (compare the `RWS_ATTRIBUTE` macros in the code with the controls in the user interface):

```
RWS_BEGIN_COMMANDS

RWS_ATTRIBUTE(CMD_Set_t_Light_Type, "Type", "Set light
type, see RpLightCreate",
    LIST, RwUInt32,
    LIST("rpNALIGHTTYPE|rpLIGHTDIRECTIONAL|rpLIGHTAMBIENT|rpLIGHTPOINT|rpLIGHTSPOT")

RWS_ATTRIBUTE(CMD_Set_t_LightConeAngle, "Cone angle", "Set light
cone angle, see RpLightSetConeAngle",
    SLIDER, RwReal, RANGE(0,45,180))

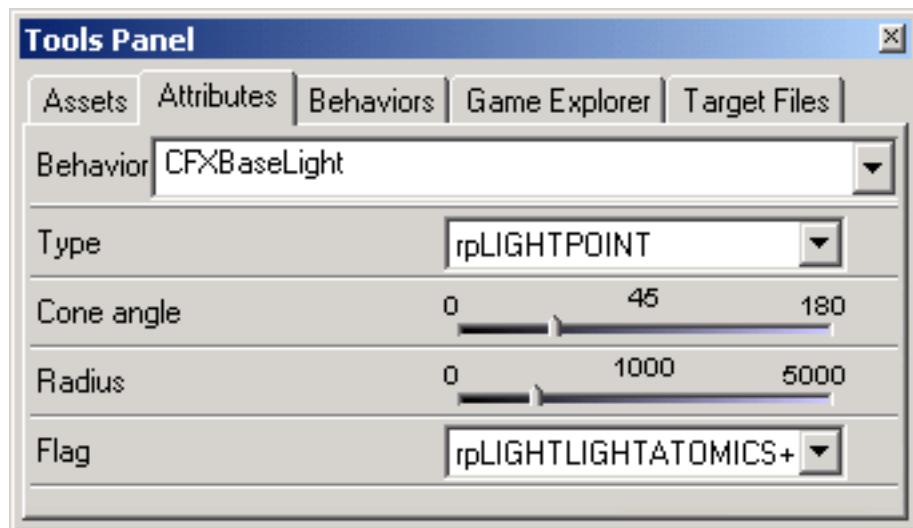
RWS_ATTRIBUTE(CMD_Set_t_radius, "Radius", "Set light
radius, see RpLightSetRadius",
    SLIDER, RwReal, RANGE(0,1000,5000))

RWS_ATTRIBUTE(CMD_Set_t_flag, "Flag", "Set light
flags, see RpLightSetFlags",
```

```

LIST,      RwUInt32,
LIST( "rpLIGHTLIGHTATOMICS+rpLIGHTLIGHTWORLD | rpLIGHTLIGHTATOMICS | rpLIGHTLIGHTWORLD"
RWS_END_COMMANDS;

```



## Events and event handlers

The Game Framework provides an event-based, data-driven methodology for allowing behaviors to communicate with each other. This formalized approach greatly reduces the interdependencies between behaviors and increases the ease of code reuse. You can think of events as a way of providing run-time linking between game entities. The implementation of the event system is based on the “Chain of Responsibility” pattern [Gamma, E et al. (2000)]. A similar event system, coupled with a finite state machine, is described in *Designing a General Robust AI Engine* [Deloura, M. (2000)].

The event system provides the following features:

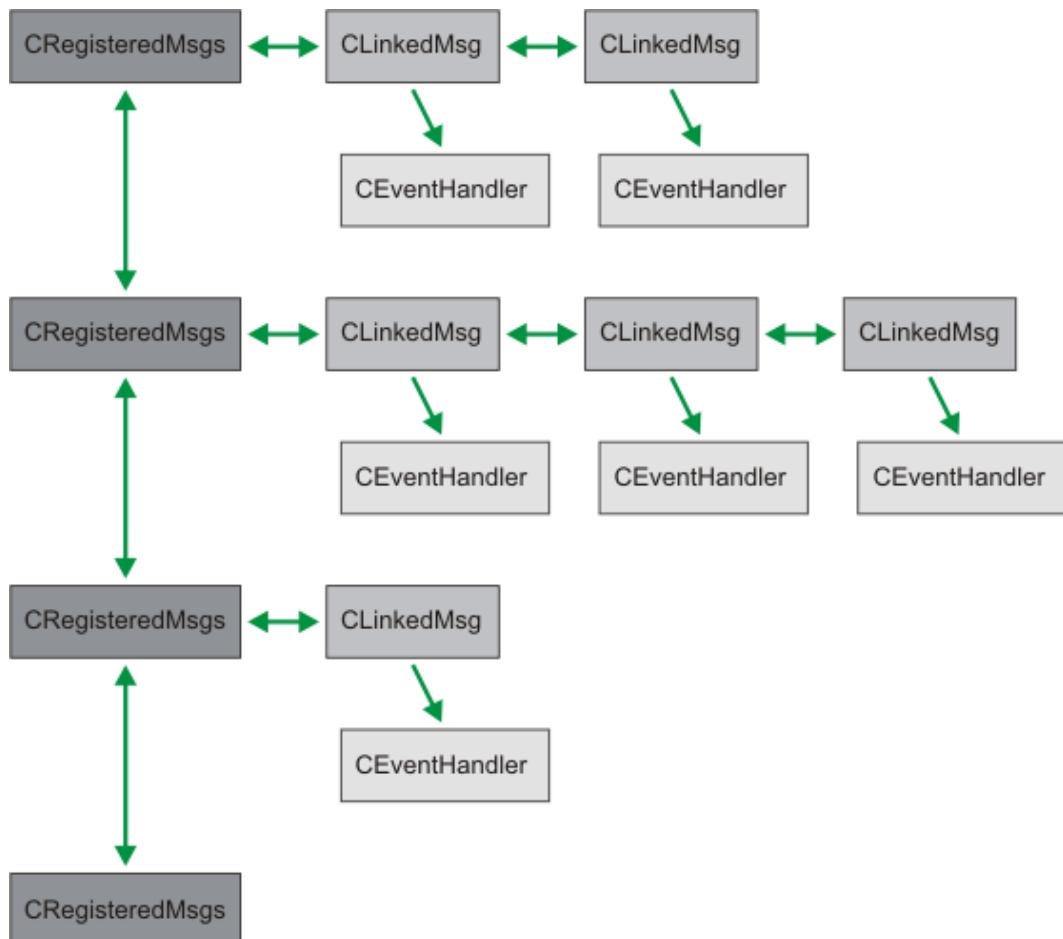
- Events can be dynamically created and destroyed
- Each event can have many senders
- Each event can have many recipients
- Events are processed by the recipients before execution returns to the sender
- Events can carry arbitrary data with them
- Events can be debugged by reviewing the call stack

The implementation of the event system is a generalization of a common technique; for example, most game developers will be familiar with the technique where a list of active objects is maintained. Each object is updated every frame; a pointer to the function to be called is maintained by each element in the linked list. Traversing the list and calling the function updates each of the game objects.

The Game Framework event system differs from this in that these lists can be created dynamically as well as allowing game objects to attach and detach themselves from any number of lists; it is a list of registered events each with a list of interested game objects.

The recipients of events are known as event handlers. Event handlers must be

derived from the abstract base class `CEventHandler`, and must implement a `HandleEvents` function. The event system maintains a list of registered events; each registered event maintains a list of event handlers that have linked to that event, as shown in the diagram below.



Registered events are maintained by a list of `CRegisteredMsg` objects; in turn, the `CRegisteredMsg` object maintains a list of `CLinkedMsg` objects that maintain the pointer to the instance of the `CEventHandler`. Sending an event or message starts at the `CRegisterMsg` object and traverses the list of `CLinkedMsg` objects, calling the `HandleEvents` method for each `CEventHandler`. The implementation of the event system differs from many message-based systems in that events or message are sent immediately with the calling function, only resuming execution once the event has been processed by all interested parties.

Events are referenced by the `CEventId` object, which is a smart pointer to the instance of the `CRegisteredMsg` associated with the event; this eliminates the need to search the registered messages when events are sent. Only the event handlers that have linked to an event will be called when the event is sent.

Properly maintaining the lifetime of both the `CLinkedMsg` and `CRegisteredMsg` objects is the responsibility of the client; this is done as a performance consideration, to reduce the need to search the linked lists during destruction of game objects. (However, the majority of coding errors are trapped by the event system during a debug build of the Game Framework, and reported to the programmer.)

By using the “Chain of Responsibility” pattern [Gamma, E et al. (2000)], we

enable the event handlers to exhibit polymorphism; event handlers can be specializations either extending or replacing the functionality of the base class, as shown in the diagram below.

```
class Base: public CEventHandler
{
public:
    virtual void HandleEvents(CMsg &pMsg);
}

class Specialization: public Base
{
public:
    virtual void HandleEvents(CMsg &pMsg);
}

void Base::HandleEvents(CMsg &pMsg)
{
    if(pMsg.Id == EventA)
    {
        // Process EventA
    }

    if(pMsg.Id == EventB)
    {
        // Process EventB
    }

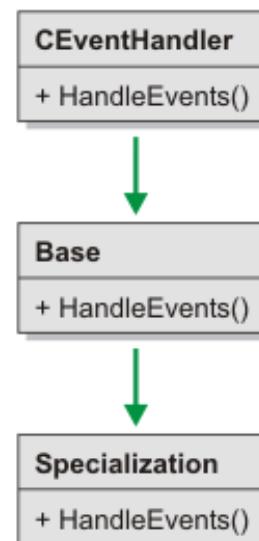
    if(pMsg.Id == EventC)
    {
        // Process EventC
    }
}

void Specialization::HandleEvents(CMsg &pMsg)
{
    if(pMsg.Id == EventA)
    {
        // Replaces Base Class Processing of Event A
    }

    if(pMsg.Id == EventB)
    {
        // Add Processing to Event B
        Base::HandleEvents(pMsg);
    }

    if(pMsg.Id == EventC)
    {
        // Relinquish processing of EventC to base class
        Base::HandleEvents(pMsg);
    }

    if(pMsg.Id == EventD)
    {
        // Add processing of event D
    }
}
```



Traversing the list of linked event handlers enables events to be broadcast: this is useful where the recipients of an event are unknown to the sender. If the recipient

is known to the sender, then events can also be sent directly to a specific event handler. In this case, the `HandleEvents` method of the receiver is called directly without traversing any of the lists. (This means that event handlers can receive events that they have not linked to. They need to handle such events appropriately: typically, this involves ignoring the event.)

## Resources

The Game Framework provides a methodology for managing assets (also known as resources). Resources are typically identified by a unique ID; access to these resources can be obtained by reference to this ID. Accessing resources is handled by the resource manager, implemented by the `CResourceManager` class. Individual resource types are handled by specializations of the `CResourceHandler` class.

Developers can easily extend the types of resources that the resource manager can handle, without modifying the resource manager, by deriving from `CResourceHandler`, and implementing the interface it defines.

The loading, unloading and referencing of resources is all data-driven, often eliminating the need for programmer intervention when new assets are added to the game. By empowering artists and designers, allowing them to add new assets to the game, we remove the typical programmer bottleneck. This allows new assets to be reviewed on the target console much quicker.

## Game Framework behaviors

Behaviors extend the framework to create the developer's game engine. Behaviors in RenderWare Studio typically come in one of several flavors:

- A behavior that needs to be instanced on the target platform for a particular section of the game. In this case, the behavior would be derived from `CAttributeHandler`, so that it can be instanced by the `ClassFactory`.
- A behavior that also has attributes that can be modified in real time during development. Again, the behavior is derived from `CAttributeHandler`, but this time code decoration is used to generate a user interface in the Workspace.
- A behavior that has assets associated with it. In this case, the behavior is derived from both `CAttributeHandler` and `CSystemCommands`. `CSystemCommands` is a class that exposes special attributes that the Workspace understands, such as the position and orientation of a 3D object within the scene.
- A behavior that interacts with other game objects. The behavior is derived from `CEventHandler` and uses events to communicate with the other objects.

## Startup, shutdown, and the main loop

The Game Framework provides code to startup and shutdown the game on each target platform. This code is split into three layers:

- Hardware specific
- Low-level services such as RenderWare Graphics and Audio
- Higher level services such as resource management and event handling

The main game loop drives the event system, synchronizing the update of game logic and rendering. By using the event system, we uncouple the main game loop from the rest of the game code. To illustrate this, consider a game with several levels, where each level requires different rendering techniques. One approach would be to switch on the current level and call the relevant rendering code. By doing this, the implementation of the rendering and the main loop are closely linked. Further, the main loop is now game-specific making it harder to reuse. The main loop will also need to be maintained as game levels are changed or added which requires a programmer. By using events, RenderWare Studio allows different rendering techniques to be attached by the game data for each level; it becomes a data-driven process, reducing the need to modify code.

The main loop is responsible for polling a number of services and firing a number of events, as listed below:

1. Poll network layer for data packets from the Workspace, data packets are used to load assets, create or delete entities, modify attributes and other tasks relating to editing.
2. If the game logic state is running:
  - a. Send event **Pre Running Tick Event**
  - b. Send event **Running Tick Event**
  - c. Send event **Post Running Tick Event**
 otherwise:
  - a. Send event **Pre Paused Tick**
  - b. Send event **Paused Tick**
  - c. Send event **Post Paused Tick**
3. Depending on whether directors camera is enabled or disabled, either:
  - Send event **Render Director's Camera**  
or
  - Send event **Render Camera**
4. Show rendered image.

## References

- Gamma, E and Helm, R and Johnson, R and Vlissides, J. (2000), *Design Patterns: Elements of Reusable Object-Oriented Software* ([www.aw-bc.com/catalog/academic/product/0,4096,0201633612,00.html](http://www.aw-bc.com/catalog/academic/product/0,4096,0201633612,00.html)). Addison-Wesley.
- Deloura, M. (2000), *Game Programming Gems* ([www.charlesriver.com/titles/gamegems.html](http://www.charlesriver.com/titles/gamegems.html)). Charles River Media.

# Game Framework tips

---

## Class factory

For RenderWare Studio to create new behaviors and entities, we need a way of identifying the class to create. `CClassfactory` maintains a mapping of name/function pointer pairs that allow named classes to be created. To register a class in the class factory, you need to insert the following two macros in the header file of a behavior:

```
RWS_MAKENEWCLASS(classname)
RWS_DECLARE_CLASSID(classname)
```

These macros declare the functions required to allow the class factory to construct a new class object, and the member variables required to identify the object.

## Resource handlers

Resource handlers for common RenderWare types such as worlds, atomics, clumps, and texture dictionaries are provided as toolkits as part of the Game Framework. RenderWare Studio loads resources from streams. Streams can be loaded either from memory, from a mass storage device such as a CD-ROM, or from across the network.

You can add your own resource handler by inheriting from `CResourceHandler` and overriding the load functions.

When you derive from `CResourceHandler`, you have to override functions to load, unload, and check whether the resource type is handled.

**Note:** By default, attribute values are incorporated into the stream, but the `RWS_ATTRIBUTE` macros that generate the user interface are not retained. XML is not incorporated into the game stream.

You can add your own resource handler by inheriting from `CResourceHandler` and overriding the load functions.

When you derive from `CResourceHandler`, you have to override functions to load, unload and check whether the resource type is handled.

## Attribute handler overview

See the [behavior tutorial](#) (p.337) for an example of using the attribute handler.

For more information on the coding principles see the Game Framework reference.

The attribute handler provides a formalized method of initializing and modifying parameters of individual or global game data. The attribute handler does not determine the source of the parameter data. It could be:

- Parameter data
- Game data file
- Memory card

- Network source

The RenderWare Studio Workspace uses unique IDs to identify instances of behaviors that have been instantiated on the console. On the target a global map is maintained relating the unique IDs to `CAttributeHandle` pointers. The `CAttributeHandler` class provides methods for maintaining this mapping. `CAttributeHandler` also provides a pure virtual method, `HandleAttributes`, which receives a list of attributes that should be updated on the target.

Override `HandleAttributes` in your behaviors to get access to the attribute data in RenderWare Studio.

The `CSystemCommands` class provides a basic set of commands that RenderWare Studio requires in order to manipulate the entity in the game.

Macros are used in the header for your behavior to build up the user interface and allow you to manipulate the corresponding attributes in the Workspace.

# Customizing Workspace

---

RenderWare Studio—and Workspace in particular—is highly customizable on many levels. In large part, this is due to the fact that Workspace is built around ActiveX technology, which brings a number of advantages:

- ActiveX is a well-documented architecture with a great many resources to support it.
- Components can be developed and tested in a standalone way, reducing interdependencies.
- It's easy to expose a scriptable interface, so the control can expose methods to script code, and fire events to enable integration with other components.
- ActiveX controls written in C++ can access the RenderWare Studio Manager API using native code.

## Enterprise Author

RenderWare Studio comes supplied with an application called Enterprise Author that allows you to customize Workspace to the individual requirements of a game development team. Among other things, Enterprise Author allows you to:

- Change how Workspace's windows are configured by default, to suit the needs of your organization.
- Create definitions for different interface layouts that Workspace can switch between on request.
- Make new windows, menus, and context menus, and write code to integrate them with the rest of the application.
- Introduce custom ActiveX controls to Workspace, and have them interact with the default set of controls.
- Edit the script code that handles the interactions between the ActiveX controls in Workspace's interface and stores some configuration information.

## Other customization tools

Although its remit is broad, Enterprise Author is not the only way of customizing Workspace's appearance and functionality. For example, there's a [Visual C++ Wizard](#) (p.477) for creating ActiveX controls that contain boilerplate code for interacting with some important RenderWare Studio mechanisms. Also, Workspace has XML configuration files that can be modified with nothing more complicated than a text editor.

This part of the documentation describes how to use Enterprise Author, and then explores RenderWare Studio Workspace from a developer's (rather than a user's) perspective. After that, it provides tutorials and practical advice on customizing Workspace's appearance, functionality, and integration with the game database.

# Understanding Workspace

---

The foundations of Workspace's user interface are the same as Enterprise Author's: it has splitters, stackers, tabbers, and so on. *All* applications that are created and edited using Enterprise Author share their creator's functionality, because they have the same basic structure.

## Fundamental operations

Workspace shares its core functionality with Enterprise Author because both applications are based on an executable called `EnterpriseHost.exe`. This executable lies at the heart of Enterprise Author and any application it's used to create—including, of course, Workspace.

- It provides control hosting and visual manipulation, through features like splitters, tabbers, and stackers.
- It processes the other files that Enterprise Author creates, turning them into the application that users see.
- It initializes the controls in the user interface when an application is first run, and provides the conduit for them to communicate as the application proceeds.
- It provides Automation interfaces that make some general functionality available to every control in an application.

## Application-specific functionality

`EnterpriseHost.exe` provides the frame on which an application can be hung, but it doesn't *define* an application. That job is done by the "other files" that Enterprise Author creates, which describe the controls that make up the user interface, the appearance of those controls, and the details of their interactions.

## Distribution choices

When you distribute an Enterprise Author-generated application, you can choose to compile together `EnterpriseHost.exe` and the files that define the application, thereby creating a single executable file. Alternatively, you can opt to keep the two parts separate, which means that you'll have several files to distribute to your users. Workspace takes the second option, but another application that you know well—Enterprise Author—takes the first. There are good reasons for preferring one approach or the other in different situations.

## Enterprise Author

In Enterprise Author, the files that *define* the application are compiled into the same executable as the code that *controls* the application.

`EnterpriseHost.exe` is subsumed into `EnterpriseAuthor.exe`. Putting the definition files inside the executable prevents users from performing large-scale customization of the application, but also prevents them from breaking it! However, users can still:

- Show and hide windows

- Move windows around in the user interface
- Have these changes saved for the next time they run the application

**Note:** You can create an application as a stand-alone executable in Enterprise Author by selecting **File ▶ Make Executable....**

## RenderWare Studio Workspace

Workspace is *not* a stand-alone executable. The command that launches Workspace from the shortcut on your desktop or the **Start** menu looks like this:

```
C:\RW\Studio\Programs\EnterpriseHost.exe /host  
RenderWareStudio.ren
```

Here, `EnterpriseHost.exe` is invoked with an option that causes it to look for an application project file called `RenderWareStudio.ren`. The project file marks the location of the definition files, which in this configuration are available for modification. This means that in addition to the simple customizations above, users can:

- [Create new windows for the application](#) (p.477)
- [Edit menus and toolbars](#) (p.462)
- [Change the interactions between different parts of the application](#) (p.445)

It's these application definition files that Enterprise Author allows you to edit in order to customize the appearance and behavior of RenderWare Studio Workspace.

# Application source files

---

The files that define Workspace's specific appearance and functionality (rather than the more general "plumbing" that `EnterpriseHost.exe` provides) come in four types, and fulfil particular roles in the running Workspace application.

## `ApplicationName.ren`

In a customizable Enterprise Author-generated application, the `.ren` file is passed to `EnterpriseHost.exe` to identify the other files that define the application's behavior. In this, it acts rather like one of Visual Studio's project files, or a RenderWare Studio `.rwstudio` file. The `.ren` file doesn't contain much information itself. Instead, it acts as a pointer to the important data.

For example, a RenderWare Studio game project is represented by a file called `GameName.rwstudio`, with the data itself stored in a directory called `GameName` at the same level. Similarly, when you customize Workspace with Enterprise Author, you modify the files in a directory called `RenderWareStudio`, which shares a directory with a file called `RenderWareStudio.ren`.

`EnterpriseHost.exe` can read files at any level in the directory beneath the `.ren` file, so you're free to organise the files into subdirectories according to their purpose without fear of "breaking" the application. Shortly, we'll consider the subdirectory structure employed by Workspace, but in general you're free to use one that best suits your needs.

## `LayoutName.Layout`

`.Layout` files (p.430) are XML files that store information about the positions of objects in the different layouts you've defined for your application. There's one file per layout, detailing the positions of stackers and tabbers in the user interface, the objects contained in each stacker or tabber (and those that should float above the application window), and the order of objects within each containing structure.

## `ObjectName.RENObj`

`.RENObj` files are XML files that describe Enterprise Author objects. The file for a menu bar, for example, specifies things like the name of each item, any shortcut keys to be used, whether an icon should be displayed, and the name of the event to be fired when an item is selected. The file for a page object specifies a set of default property values for the window, and contains binary data for any images it requires. Here's a snippet from such a file:

```
...
<param name="Caption" value="Search in Game" />
<param name="Group" value="grpDialogs" />
<param name="Top" value="0" />
<param name="Left" value="0" />
<param name="Width" value="366" />
<param name="Height" value="144" />
<param name="Visible" value="1" />
<param name="Border" value="1" />
<param name="ShowCaptionBar" value="1" />
<param name="WindowSizeable" value="1" />
<param name="ShowInClientArea" value="0" />
<param name="Maximized" value="0" />
<param name="WindowControls" value="1" />
<param name="HideWindowOnClose" value="0" />
```

```
<param name="OverrideWithNodeProperties" value="0" />
<param name="ShowFocusDisplay" value="0" />
<param name="FocusDisplaySize" value="0" />
<param name="FocusDisplayStyle" value="0" />
<param name="FocusDisplayColor" value="-2147483627" />
<stream
encode="BASE64">NAAAAAYAAAAnAAAAe0Q3MDUzMjQwLUNFNjktMTF...
...
```

### *ObjectName.vbs*

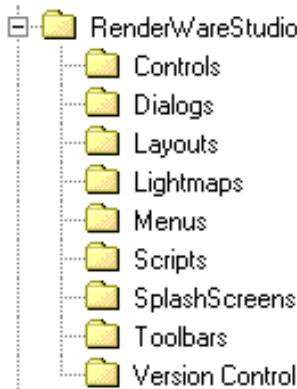
.vbs files are VBScript files containing, at a minimum, code that handles interactions between the user and the object (button clicks, menu item selection, and so on). There may be further code for initializing the object when it first appears, keeping track of changes in other objects, interacting with the file system, etc.

Usually (and especially for ActiveX controls, pages, and menu bars), .RENOBJ and .vbs files come in pairs. However, this is not a hard-and-fast rule. Some objects (a log window, for example) may not need to communicate with other objects. (They might just display what they're told, and leave it at that.) An object like this will have a .RENOBJ file, but no .vbs file. On the other hand, a script module has no associated graphical representation, so in that case we have a .vbs file with no related .RENOBJ file.

# The RenderWareStudio directory

---

The EnterpriseHost.exe application can work with user interface definition files that appear *at any level* beneath the top-level directory. This means that you can divide functionality into further subdirectories for ease of organization. The RenderWareStudio directory, for example, looks like this:



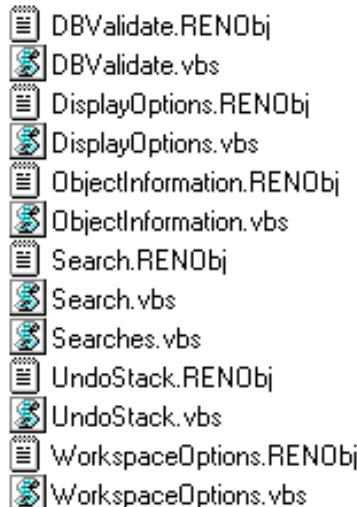
## Controls

Contains files that define the appearance and operation (and sometimes the position and size) of objects such as the Templates and Targets windows that form part of the main user interface.

Files in this directory correspond to ActiveX control objects in Enterprise Author's Objects window.

## Dialogs

Contains files that define the appearance, operation, and state of Workspace dialogs such as Workspace Options and Search in Game.



Files in this directory correspond to page objects in Enterprise Author's Objects window.

## Layouts

Contains files that describe how (and whether) entities in the other directories should appear in the user interface at run time. See the [separate](#)

[section on .Layout files](#) (p.430) for more information.

### **Lightmaps**

Contains .RENOBJ and .VBS files specifically related to Workspace's lightmap editing features. These files could equally be placed in the generic **Dialogs**, **Menus**, and **Toolbars** directories; the split is for reasons of organization alone.

### **Menus**

Contains files that store definitions for the menus and context menus in the application. Files in this directory correspond to menu bar objects in Enterprise Author's Objects window.

### **Scripts**

Contains files storing global script code that isn't associated with any particular object in the Workspace user interface. Procedures that are called from several different places in the UI, for example, might be placed in files here.

Files in this directory correspond to script module objects in Enterprise Author's Objects window.

### **SplashScreens**

Contains .RENOBJ files storing information about and data for the splash screens used in Workspace. Because splash screens typically display quite big images, these files tend to be rather large.

### **Toolbars**

Contains files that store definitions for the toolbars in the application. While menu files just contain item names, toolbar files also include the bitmap resources used for displaying icons.

Files in this directory correspond to menu bar objects in Enterprise Author's Objects window.

### **Version Control**

Contains files that define the pages through which a Workspace user can interact with alienbrain. Again, the use of a separate folder for these files, rather than simply placing them in **Dialogs**, is an organizational choice.

# Understanding .Layout files

---

At any point in time, the structural features of an Enterprise Author-generated application, as displayed on your monitor, are dictated by just two files:

- The `.RENOBJ` file for the application window
- The `.Layout` file for the layout in current use

It's possible to switch between different layouts (and therefore different `.Layout` files) while an application is running. It's also possible to create, edit, and delete layouts, from both Enterprise Author *and* a running Enterprise Author-generated application. To enable this functionality, `.Layout` files are not quite as straightforward as they first appear.

## `.Layout` and `.LayoutDefault`

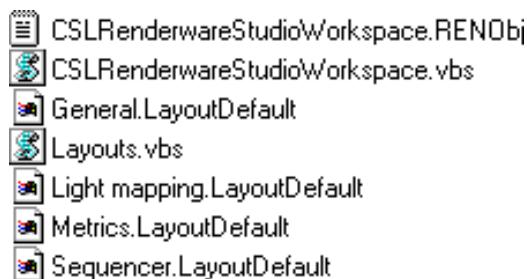
When you create a layout, at design time or run time, you generate a `.Layout` file. When you distribute one of your layouts, you should change the extension to `.LayoutDefault`. At application startup, `EnterpriseHost.exe` processes `.LayoutDefault` files in exactly the same way as it processes `.Layout` files. The behavior changes, however, when you (or a user) edit and save that layout.

Editing a layout in an application that supports doing so results in a `.Layout` file being saved. If you start with a `.Layout`, the old file gets overwritten, and the original version is lost. If you start with a `.LayoutDefault`, on the other hand, the result is two files with the same name but different extensions.

`EnterpriseHost.exe` always loads “Default” files first, but ultimately loads *all* `.Layout` and `.LayoutDefault` files in the directory beneath the `.ren` file. If two layouts have the same name, the one that was loaded most recently takes precedence—so new layouts always outrank default ones. The “Default” file is a fallback mechanism that protects against injudicious layout editing. Delete the `.Layout` file, and the layout in question returns to its default appearance by virtue of the `.LayoutDefault`.

## Workspace layout files

Workspace itself provides an example of this system in action. As shipped, the contents of the `RenderWareStudio/Layouts` folder look like this:



When you change any of these layouts, in Enterprise Author or Workspace, versions of the files called simply `.Layout` are created. `EnterpriseHost.exe` then gives the new layouts precedence next time the application is launched.

**Note:** CSLRenderwareStudioWorkspace.RENOBJ contains property settings for the Workspace application window, as mentioned above. The Layouts.vbs file contains script code that handles the items for creating, renaming, deleting and restoring layouts in Workspace's **View** menu.

### .LayoutDeleted and .LayoutDefaultDisabled

An application like Workspace, where users are able to rename and restore layouts as well create and edit them, needs more flexibility than two file types can provide. Workspace uses layout files with the extensions .LayoutDeleted and .LayoutDefaultDisabled to support its features for restoring and renaming layouts. Consider the following situations:

- The user creates their own layout to complement the default set (so that the Layouts folder contains several .LayoutDefault files and one .Layout file). Then, they select **View ▶ Restore to Default Layouts**. The new layout is renamed from .Layout to .LayoutDeleted, so that it doesn't appear when Workspace is next run, but may be retrieved later through manual renaming.
- The user renames or deletes one of the default layouts. The layout should not appear when Workspace is next run, but it needs to be retained in case of a future "restore" operation. To achieve this, its file is renamed from .LayoutDefault to .LayoutDefaultDisabled. This change is easily reversible, should the need arise.

## What about stand-alone executables?

Although Enterprise Author-generated applications that are compiled into a single executable are much less customizable than those with separate user interface files, it's still possible to edit the supplied layouts at run time. When this happens, the new .Layout files are placed in a directory at the same level as the executable. EnterpriseAuthor.exe's layout files (for example) may be found in the EnterpriseAuthor directory.

# What Enterprise Host does

---

`EnterpriseHost.exe` is fundamental to Workspace, as it is to any application created with Enterprise Author. Its “passive” action is to manage the ongoing appearance of an application as the user resizes and moves objects around the application window, but it can also take a more active role:

- When the application is launched, it parses [the RenderWareStudio directory](#) (p.428), loading all the `.RENOBJ`, `.Layout`, and `.vbs` files at any level in the hierarchy beneath it.
- After loading the files, it draws the objects on the screen, with sizes and locations as defined by the default `.Layout` file
- During execution, it provides services (such as [the Broadcast object](#) (p.450)) that can be called upon by any object in the application. More basically, it contains the script engine that allows the script code associated with each object to run.
- At particular times, it fires events to all the objects it hosts, so that the objects can coordinate their initialization and shutdown processing, for example.

## Starting up

When `EnterpriseHost.exe` is invoked in the context of starting Workspace, its immediate action is to load the files stored in the RenderWareStudio directory. The first ones it loads are those connected with splash window objects, so that the application can display a splash screen while its other components are loading. When it has loaded and displayed the splash screen, it loads the `.RENOBJ` and `.vbs` files for the other objects.

## Loading object files

Apart from splash screens, the order in which `EnterpriseHost.exe` loads object files is unpredictable, and you should avoid writing script code that relies upon any one object being loaded before any other.

This rule is potentially quite difficult to follow. When `EnterpriseHost.exe` loads a script file, any code at global scope in that file (that is, code outside `Sub` blocks) is executed *immediately*.

You *must not* refer to other objects in code at this level, as they may not exist when you try to use them! Instead, you should place any such code inside a procedure called [OnLoad\(\)](#) (p.435). `EnterpriseHost.exe` invokes this method on every object it's hosting, after *all* objects in the current application have been loaded.

## Service provision

Any piece of script code running inside `EnterpriseHost.exe`'s script engine has access to any other piece of script code in the application being edited. But `EnterpriseHost.exe` also provides two general-purpose objects that may be used at any time by all hosted code. These are:

**RENManager**

Defines events that fire when the application starts up and shuts down, and when a different layout is selected from a menu. Contains methods that perform some file-handling duties and user-interface adjustment; and properties for retrieving the parent window's handle, the object with focus, and the [Broadcast object](#) (p.450).

**RENHost**

Contains methods to connect objects to the application's scripting architecture, so that other script code can handle their events and use their methods and properties.

## Firing events

When all the objects in the application have loaded, `EnterpriseHost.exe` fires `RENManager.OnLoad`, which controls can handle to perform initialization.

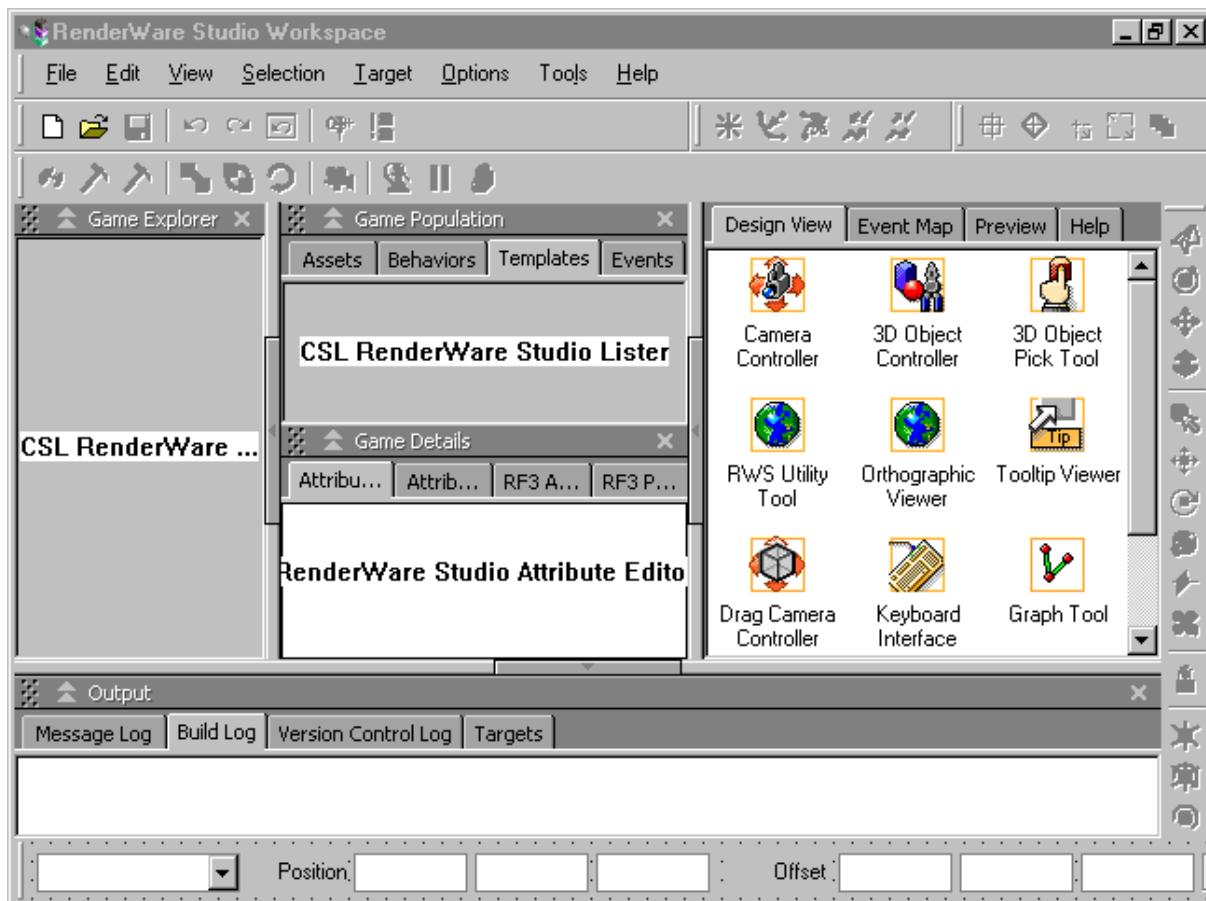
Similarly, they can handle `RENManager.OnTerminate` to perform any necessary shutdown processing. Other events are fired just before and just after the application's layout is changed from one preset to another, and when an non-existent method or property is called on any object.

# How Workspace works

---

To understand the part of Workspace that its users see (rather than the functionality that it inherits from `EnterpriseHost.exe`), you need to open it for editing in Enterprise Author:

To see the Workspace user interface in the Design window, you can either open the RENFrame object called `CSLRenderwareStudioWorkspace` for editing, or go to the **Select** menu and choose one of the four options there.



The screenshot above shows Workspace's General layout. This is the first item under the **Select** menu, and the default layout loaded by the RENFrame object, as dictated by its `DefaultDockState` property.

**Tip:** Since we'll be experimenting with the user interface in the topics ahead, this is a good time to save a copy of the Workspace project files:

- Select **File > Save As...** and save the unchanged project under a new name. This will create a new `.ren` file, and a new subdirectory.

# Startup processing

---

In discussing [EnterpriseHost.exe](#) (p.432), we touched upon what happens when an Enterprise Author-generated application is first opened. Now we can take a look at how that works out in practice, in Workspace.

With your copy of `RenderWareStudio.ren` open in Enterprise Author, select **Run ▶ Start**. In response, Workspace starts just as if you had clicked its link in the Windows **Start** menu. Our interest here, however, is exactly what's going on during this process.

At startup, the objects in Workspace get two chances to perform some initialization.

## Global script module code

First, the script module associated with every object is processed, and any VBScript statements that are not contained within procedures are executed. You can guarantee that this will occur for every object before any further processing takes place, but the order in which script modules are processed is undefined.

Because of this restriction, this phase is not used to set up relationships between user interface objects—an object being referred to in code may not have been created yet! Instead, it's typically used to define any constants that an object requires, and to instantiate (or create relationships with) external (COM) objects.

A lot of this work happens in the *GlobalScript* script module, in code like this:

```
...
Set RWSUtils          = CreateObject( "CSL.RWSUtils.Utility" )
Set RWSScript         = CreateObject( "CSL.RWSScript.RWSScript" )
Set RWSComs           = CreateObject( "CSL.RWSTarget.RWSComs" )
...
' Allow the other scripts to receive the COM objects' events:
RENHost.ConnectObject RWSUtils,           "RWSUtils"
RENHost.ConnectObject RWSScript,          "RWSScript"
RENHost.ConnectObject RWSComs,            "RWSComs"
...
```

Objects like [RENManager](#) and [RENHost](#) (p.432) are provided by `EnterpriseHost.exe` and allow control over the layout of the RenderWare Studio user interface. The objects being created in this code, on the other hand, allow interaction with RenderWare Studio at a lower level; `RWSScript`, for example, is the RenderWare Studio Manager object, which allows you to manage the game database programmatically.

Outside *GlobalScript*, another common use of initialization at this stage is to set up and install the entries for any list or combo boxes in the user interface of the object in question. There's a good example of this in the script module for the *DisplayOptions* object:

```
...
strDispMode0 = "Textured wireframe"
strDispMode1 = "Solid"
strDispMode2 = "Wireframe"
```

```

strDispMode3 = "Untextured solid"
...
PerspectiveDisplayMode.AddItem strDispMode1
PerspectiveDisplayMode.AddItem strDispMode2
PerspectiveDisplayMode.AddItem strDispMode0
PerspectiveDisplayMode.AddItem strDispMode3
...

```

## The OnLoad( ) method

Second, the `OnLoad()` method is called in every script module that contains one. Once again, you can guarantee that this will occur for every object before anything else happens, but not the order in which it will occur.

When `OnLoad()` is called in a script module, you know that all the other objects in the application have been created and are ready for use if you need them. The *GlobalScript* object is again the most active here, performing two especially important operations:

- It loads and processes the [RwStudio.settings file](#) (p.443). This is an XML file that contains settings data for the Workspace application and many of its windows. Because of this file, when Workspace starts up, it looks identical to how it looked when it was last shut down.  
Any Workspace object can arrange to have its settings stored in `RwStudio.settings` through [a standard mechanism for doing so](#) (p.512).
- It loads the game project file that was open when Workspace was last shut down.

Away from *GlobalScript*, there are two particularly common reasons for implementing an `OnLoad()` method in an object's script module:

- Menu bars, and especially toolbars, can contain items that make or reflect changes in some other Workspace object. They can also offer some feedback to the user, by showing some of their items as being selected, or activated, or unavailable at different times. These settings can be made at startup, but doing so requires interrogation of the “target” object, which *must* be loaded when this happens. `OnLoad()` is therefore the right place to perform such processing.
- Workspace includes a mechanism that keeps track of the currently selected entity in the Design View window, and sends events to its user interface objects when the selected object changes. Subscribing to this mechanism requires a value that's defined in *GlobalScript*, and any objects wishing to use it must be sure that it exists before they make the attempt. Subscriptions to the mechanism typically take place in `OnLoad()`.

### ActiveX control initialization

The information above is about the initialization that takes place for objects that have been loaded into Workspace as a result of appearing in a `.ren` file set. For ActiveX control objects, an earlier initialization step is possible in the `OnCreate()` method of the C++ code that executes when the ActiveX control is instantiated.

# Normal operation

---

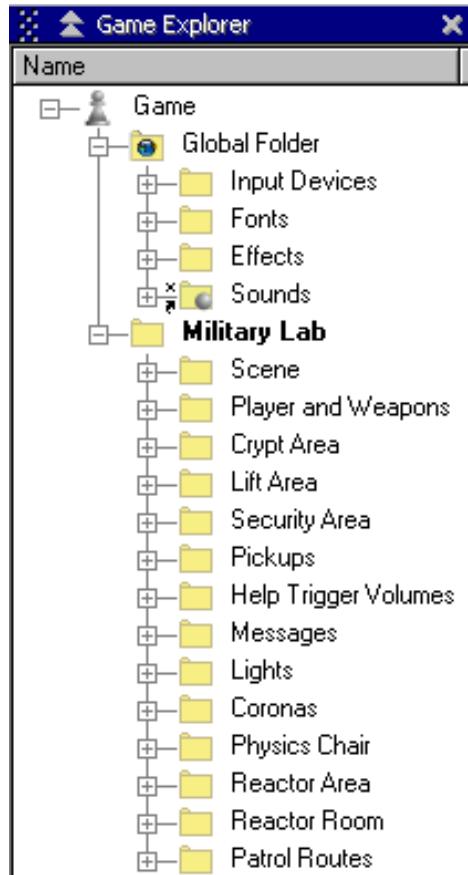
After both stages of `EnterpriseHost.exe`-driven [startup processing](#) (p.435) are complete, a game project may have been loaded into Workspace, but what happens next is entirely down to the user. We *can* say, however, that what they do will be detected by an ActiveX control. How Workspace reacts will depend on the kind of object that the ActiveX control is a part of.

## A cooperative approach

Workspace, like other Enterprise Author-generated applications, relies on the cooperation of its constituent objects in order to work correctly. There is no defined “entry point” (there is no compulsion to have an object named *GlobalScript*, for example), so the application programmer has the freedom and responsibility to ensure that execution proceeds sensibly.

## Control host objects

In a control host object, a single ActiveX control is responsible for everything the user sees. Changes to its appearance and reactions to button presses, for example, are handled within the control. The control communicates with its associated script module by firing events for the latter to handle. Methods on other objects are then called from the event-handling code.



When the user right-clicks on the Game Explorer window, for example, the

*GameDatabase* object handles the click, does some processing, and then fires an event called `OnContextMenu`. The code in the script module associated with the object handles this event and causes the appropriate context menu to be displayed:

```
Sub GameDatabase_OnContextMenu(Context, nX, nY, hWnd)
    ContextMenu.DisplayListerMenu Context, nX, nY, hWnd
End Sub
```

## Page objects

For an ActiveX control that's one of several on a page object (like the Validate Database dialog), then the process (in terms of script code, at least) can be more involved. In this situation, the handling code may be called upon to manage the interactions between the controls on the page, as well as to call methods on objects elsewhere in the application.



The code in the script module for the `DBValidate` object has to deal with the actual act of a user clicking the **Validate** button, rather than the event generated by a more complex control after some initial processing. Before it can call `RWSUtils.OptimizeDatabase()`, the VBScript code has to check the status of all the check boxes for itself:

```
Sub DBValid_OKButton_Click()
    nFlags = 0

    If DBValid_BehvLessEnts.Value Then
        nFlags = nFlags + 1
    End If
```

```
If DBValid_AttribDataTypes.Value Then
    nFlags = nFlags + 2
End If
If DBValid_SharedPrivateAttrs.Value Then
    nFlags = nFlags + 4
End If
...
If RWSUtils.OptimizeDatabase(nFlags) Then
    ...
End Sub
```

## Menu bar objects

In a menu bar object, the events fired by menu items are handled in the manner discussed in the section on creating and editing menus and toolbars. The event-handling procedures can call methods on other objects in the interface, or instantiate and display new objects, to carry out appropriate operations.

# Mapping Studio windows to Author objects

When you open `RenderWareStudio.ren` for editing in Enterprise Author, 70+ objects appear in the Objects window. The `CSLRenderwareStudioWorkspace` object is responsible for Workspace's application window, and there are around 20 script modules that have no graphical representation. However, that still leaves more than 50 objects!

## Missing windows?

There are some user interface elements (such as the Properties dialogs) that can appear in Workspace but do not appear in Enterprise Author's Objects window.

These are ActiveX controls that are instantiated from VBScript code in the application, but are not permanent members of the user interface, and were not created in Enterprise Author.

In their first columns, the following tables list the names of the graphical objects in the default Workspace user interface (and therefore the files in the `.ren` file set) that Enterprise Author manages. In their second columns, they list the names or locations of those objects in Workspace.

## Control host objects

Control hosts contain a single ActiveX control. How a control host looks in Enterprise Author depends entirely on the design-time appearance that's been defined for the control by its programmer.

Object name	Name in the Workspace user interface
<code>AssetLister</code>	<a href="#">Assets</a> (p.264)
<code>AttributeShareLister</code>	<a href="#">Attribute Shares</a> (p.269)
<code>BehaviorLister</code>	<a href="#">Behaviors</a> (p.273)
<code>BuildLog</code>	<a href="#">Build Log</a> (p.297)
<code>EntityAttributes</code>	<a href="#">Attributes</a> (p.270)
<code>EventVisualizer</code>	<a href="#">Event Map</a> (p.284)
<code>GameDatabase</code>	<a href="#">Game Explorer</a> (p.288)
<code>HelpWindow</code>	<a href="#">Help</a> (p.291)
<code>LightmapAssetEditor</code>	<a href="#">Asset light map</a> (p.267)
<code>LightmapSceneEditor</code>	<a href="#">Scene light map</a> (p.313)
<code>MemoryCtrl</code>	<a href="#">Memory Stats</a> (p.305)
<code>MessageLog</code>	<a href="#">Message Log</a> (p.297)
<code>MetricsCtrl</code>	<a href="#">Metrics</a> (p.305)
<code>RF3AssetTemplateEditor</code>	<a href="#">RF3 Asset Templates</a> (p.309)
<code>RF3ProjectTemplateEditor</code>	<a href="#">RF3 Project Templates</a> (p.311)

<i>RWSEventList</i>	<a href="#">Events</a> (p.286)
<i>RWSFileSaver</i>	<a href="#">Target Files</a> (p.329)
<i>RWSPreview</i>	<a href="#">Preview</a> (p.300)
<i>RWSProfileViewerCtrl</i>	<a href="#">Profiler</a> (p.305)
<i>SearchResults</i>	<a href="#">Search Results</a> (p.315)
<i>Sequencer</i>	<a href="#">Sequencer</a> (p.317)
<i>StreamPropertyList</i>	<a href="#">Stream Properties</a> (p.325)
<i>StreamViewer</i>	<a href="#">Stream View</a> (p.325)
<i>TargetLink</i>	<a href="#">Targets</a> (p.330)
<i>TemplateLister</i>	<a href="#">Templates</a> (p.334)
<i>VersionControlLog</i>	<a href="#">Version Control Log</a> (p.297)

## Enterprise control object

The Enterprise control object hosts the Enterprise control, which displays perspective and orthographic views of your game world in its four resizeable panes.

Object name	Name in the Workspace user interface
<i>DesignView</i>	<a href="#">Design View</a> (p.276)

## Menu bar objects

Menu bar objects can represent menu bars, toolbars, and context menus in the Workspace user interface. As such, they can either appear in the user interface at design time, or be made to appear through code that executes at run time.

Object name	Name or location in the Workspace user interface
<i>ContextMenu</i>	A holder for most of Workspace's context menus, <i>except</i> those for the Targets and Design View windows
<i>CurrentTargetToolbar</i>	<a href="#">Active Target</a> (p.251)
<i>DesignViewsContextMenu</i>	The context menu for the Design View window
<i>LightmapToolBar</i>	<a href="#">Light Map</a> (p.251)
<i>MainMenu</i>	<a href="#">Menu Bar</a> (p.244)
<i>MainToolbar</i>	<a href="#">Standard</a> (p.251)
<i>SelectionToolbar</i>	<a href="#">Selection</a> (p.251)
<i>SplineToolbar</i>	<a href="#">Spline</a> (p.251)
<i>TargetContextMenu</i>	The context menu for the Targets window
<i>VolumeToolbar</i>	<a href="#">Volume</a> (p.251)

## Page objects

Page objects are typically “forms”, populated with several smaller ActiveX controls. As such, they are often used as pop-up dialogs that request information from the user.

Object name	How to display it in Workspace
<i>About</i>	<b>Help ▶ About Workspace</b>
<i>DBValidate</i>	<b>Tools ▶ Validate Database...</b>
<i>DisplayOptions</i>	<b>Options ▶ Display</b>
<i>DlgImport</i>	<b>File ▶ Import... ▶ Open</b>
<i>FlightOptions</i>	<b>Options ▶ Flight</b>
<i>GetLatest</i>	<b>File ▶ NXN alienbrain ▶ Get Latest</b>
<i>ObjectInformation</i>	Appears as a toolbar at the bottom of the Workspace application window
<i>Search</i>	<b>Edit ▶ Search in Game...</b>
<i>SplineOptions</i>	<b>Options ▶ Spline</b>
<i>SubmitChanges</i>	<b>File ▶ NXN alienbrain ▶ Submit Default Change Set</b>
<i>UndoStack</i>	<b>Edit ▶ Show Undo Stack</b>
<i>WorkspaceOptions</i>	<b>Options ▶ Workspace</b>

## Splash window objects

Splash window objects appear together in a vertical column at application startup. They contain an image and dynamically modifiable text.

Object name	Location in the Workspace user interface
<i>SplashScreen1RWStudio</i>	The splash screen for RenderWare Studio Workspace
<i>SplashScreen2GenrePack</i>	The splash screen for Genre Pack 1

# Application settings file

---

Workspace's `RwStudio.settings` file

(`C:\RW\Studio\Programs\RwStudio.settings`) is an XML file that contains settings data for the application itself, and many of its windows. It's loaded and processed at startup by the `OnLoad()` method (p.435) in the *GlobalScript* module. Exactly what information is stored in the file is determined on an object-by-object basis by the objects concerned, but typical examples include:

- The values of the properties of an ActiveX control
- The names and widths of the columns in a list view
- The most recent settings of edit controls, check boxes, etc.

The common theme is that these are data about Workspace that you'd usually want to retain when you close and reopen the application. In other words, `RwStudio.settings` provides a way of making sure that Workspace looks the same when you open it as it did when you last closed it.

## Default file structure

The root element of `RwStudio.settings` is `<OBJECT>`. In the version of the file that ships with RenderWare Studio, `<OBJECT>` has two child element types:

`<OBJECT>`

Child `<OBJECT>` elements represent the “property bags” of ActiveX controls in the Workspace application. The contents of these elements (which can be `<PARAM>` elements, or further `<OBJECT>` elements) are written by the compiled control code. They include things as diverse as the list of recently open files:

```
<OBJECT NAME="RecentFileList">
    <PARAM NAME="C:\RW\Studio\Examples\GP1 Tutorial.rwstudio"
    VALUE=" " />
</OBJECT>
```

and the class IDs of the plugins that are used to edit different attribute types in Workspace's Attributes window:

```
<OBJECT NAME="AttributeEditorControls">
    <PARAM NAME="BITFIELD" VALUE="CSL.RWSControls.RWSBitfield"
    />
    <PARAM NAME="BOOLEAN" VALUE="CSL.RWSControls.RWSBoolean" />
    ...
    <PARAM NAME="VECTOR" VALUE="CSL.RWSControls.RWSVector" />
</OBJECT>
```

`<RWSScriptedSettings>`

The `<RWSScriptedSettings>` element acts as a container for settings that are written to `RwStudio.settings` by script code. The children of this element are named by the routines that create them. Data stored inside the `<RWSScriptedSettings>` element includes the default view modes of the various list controls:

```
<ListerTools>
```

```

<AssetLister ViewMode="Hierarchy">
    <Column id="Name" width="200" />
</AssetLister>
...
<TemplateLister ViewMode="Hierarchy">
    <Column id="Name" width="200" />
</TemplateLister>
</ListerTools>

```

and the layout that was last selected from the **View ▶ Layouts** menu item:

```

<Layouts>
    <LastLayout>General</LastLayout>
</Layouts>

```

If you decide to create custom Workspace objects, and use script code to save information about those objects to the `RwStudio.settings` file, then it will be stored by default in a child element of `<RWSScriptedSettings>`. It's possible, however, to create a new element that's a child of the root `<OBJECT>` element if that's what you need.

## Loading and saving the file

`GlobalScript.OnLoad()` tells the other script modules to load data from `RwStudio.settings` through a broadcast method call:

```
Broadcast.LoadApplicationSettings RWSRootNode
```

Any script module that wants to load settings from the file must implement `Broadcast_LoadApplicationSettings()`. The “root node” argument passed into this method contains the `OBJECT.RWSScriptedSettings` element, so the script can easily find the information it needs.

Similarly, the instruction that script modules should save data to `RwStudio.settings` comes from a broadcast method call that's made by `GlobalScript.RENManager_OnTerminate()`:

```
Broadcast.SaveApplicationSettings XMLDoc, RWSRootNode
```

The additional `XMLDoc` argument here allows a receiving script's `Broadcast_SaveApplicationSettings()` method to create new elements as children of the document root, rather than accepting the default location. You may decide to place all your custom settings data inside a new element, keeping it completely separate from Workspace's standard information.

# Programming Workspace objects

---

Workspace's default objects contain a wealth of functionality, and they have a high degree of interdependence. The C++ and VBScript code that defines an object's behavior typically relies on the presence of code in other objects in the application. It's important to understand these relationships before you consider changing or adding to them through customization.

What any particular object "sees" of the rest of Workspace can vary enormously depending on the type and role of the object. You can [write script code](#) (p.437) that deals with user interaction and calls methods on other objects, but there are other, less visible mechanisms at work too. In particular, there are conventions and facilities within Workspace that provide help when dealing with some common tasks.

## The Layout Manager

As an application based on [EnterpriseHost.exe](#) (p.432), Workspace has access to the general-purpose objects made available by that executable.

## [Additional initialization requirements](#) (p.446)

After `OnLoad()`, there's no built-in mechanism for performing further initialization in the objects that make up Workspace, but there are conventions for doing so.

## [Keeping track of the currently selected entity](#) (p.447)

Workspace's selection mechanism provides objects with a way of detecting when something happens to the currently selected entity, and a way of changing that entity themselves.

## [Calling a method on every Workspace object](#) (p.450)

Workspace has a broadcast mechanism that allows an object to call a method on every other object with a single command.

## [The GlobalScript script module](#) (p.453)

The *GlobalScript* script module sets up a number of objects and helper functions that can be called from anywhere within Workspace.

## Additional initialization

---

When they're loaded into Workspace, objects are given [three opportunities](#) (p.435) to initialize themselves before the user begins to interact with the program:

1. The C++ `OnCreate()` method that's called when an ActiveX control is instantiated (control host objects only)
2. Any code outside procedure definitions in an object's script module, which is executed when Workspace first loads it, on startup
3. The `OnLoad()` method in an object's script module, which is executed after all the objects in Workspace have loaded

If any further initialization is required (remember, the order in which objects' `OnLoad()` methods are called is undefined, and `OnLoad()` will be called only once for each object), the script code that does it is usually placed in a separate method that must be called first by any other object that needs to use it. In Workspace, the script modules of objects like these often contain a method called `Initialize()`.

For example, the handler for Workspace's **Edit ▶ Search in Game...** menu item displays the *Search Page* object, but only after it has called a method named `Initialize()` that pre-loads the dialog with settings from the previous search operation.

(From the script module for the *MainMenu* object:)

```
Sub OnMenuSearchGame
    Search.Initialize
    RENManager.ShowModal "Search", "CSLRenderwareStudioWorkspace"
End Sub
```

(From the script module for the *Search* object:)

```
Sub Initialize
    SearchText.Text = LastText
    WholeWordOnly.Value = LastWholeWordOnly
    MatchCase.Value = LastMatchCase
End Sub
```

Similar calling arrangements exist for the *About*, *DisplayOptions*, *DlgImport*, *GetLatest*, *UndoStack*, and *WorkspaceOptions* page objects. In each case, the `Initialize()` method performs object-specific processing of some kind.

# Selecting game entities

---

One of Workspace's most important features is the way it allows users to interact with game entities by selecting them in the Design View or Game Explorer windows, and then modifying their properties in one of the other windows. It's vital that every object that needs to is able to change or keep track of the currently selected entity.

You can see the selection mechanism in action by watching the Attributes window while you select and deselect entities in Design View. The contents change to show the settings of the most recently selected entity, and are cleared when all selections are cleared. Similarly, the **Selection ▶ Locate** menu item discovers the currently selected entity and centers it in Design View.

## The Selection object

In RenderWare Studio, a `Selection` object is used to manage all aspects of the entity selection mechanism. It fires events when the current selection changes, and provides a set of methods for discovering, changing, and iterating through the set of selected entities.

Different Workspace objects use `Selection` objects in different ways. ActiveX controls created using the RenderWare Studio ActiveX Control wizard automatically contain C++ code that instantiates and connects to (that is, opts to receive any events fired by) a `Selection` object. Page objects, on the other hand, have to do that work for themselves, although they can omit the second step (connecting) if they don't care about the events. (The **Selection** menu, for example, doesn't need to know when the selection changes.)

## Using a Selection object

Even though the exact procedure can vary, there are at most five actions that must take place over the course of the interaction between a Workspace object and a `Selection` object:

### 1. Create a selection object

In the global code at the start of a page object's script module, you can use the VBScript `CreateObject()` function to create a `Selection` object:

```
Set ObjectName = CreateObject("CSL.RWSSelection.Selection")
```

Here, `ObjectName` is the name to be used when calling the `Selection` object's methods.

This step is not necessary for wizard-generated ActiveX controls, but is replaced by the need to attach such objects to the game database. You can see this taking place for many of Workspace's control host objects in the `GlobalScript` script module:

```
Sub RWSUtils_OnDatabaseAttached(DatabaseID)
    If Not EntityAttributes.AttachToDatabase(DatabaseID) Then
        DatabaseAttachedError "Entity Attributes"
    End If
    If Not EventVisualizer.AttachToDatabase(DatabaseID) Then
        DatabaseAttachedError "Event Visualizer"
    End If
```

```

    ...
End Sub

```

The objects are detached from the game database in a similar fashion during shutdown processing:

```

Sub RWSUtils_OnDatabaseDetached()
    EntityAttributes.AttachToDatabase(0)
    EventVisualizer.AttachToDatabase(0)
    ...

End Sub

```

2. **Connect to the Selection object, so that the Workspace object can receive its events**

If a page object needs to be able to handle the events fired by a Selection object, then immediately after creating that object, it should *connect* to it:

```
RENHost.ConnectObject ObjectName, "ScriptName"
```

Here, *ObjectName* is the same as in Step 1 above, while *ScriptName* is a string to be used as the prefix for the names of event handlers.

This step isn't necessary for a wizard-generated ActiveX control, because the connection is set up in advance by the C++ code.

3. **Specify what kinds of selections the Workspace object is interested in processing**

In the Workspace object's script module, the *OnLoad()* method is implemented to specify what selections the object wants to deal with:

```

Sub OnLoad()
    ObjectName.SelectionIdentifier =
    GlobalScript.g_strGlobalSelection
End Sub

```

**Note:** This code is necessary in page objects *and* ActiveX controls, but there's a subtle difference between the two. For a page object, *ObjectName* is the same as we specified in Steps 1 and 2, but for an ActiveX control it's the name of the parent object itself.

The constant used here, *g\_strGlobalSelection*, specifies that we want this object to receive *all* selection events. It's defined in the *GlobalScript* script module and would not necessarily have been available during the first round of initialization. This is a very common situation in which to use the *OnLoad()* method.

4. **Call methods and manipulate properties to discover and change which entities are selected**

With the Selection object set up, Workspace objects can safely use its methods and properties. Here, for example, is the handler for the **Selection ▶ Locate** menu item:

```

Sub OnMenuLocateEntity
    If GlobalSelection.Count > 0 Then
        DesignView.RWSUtility.LocateEntity
        GlobalSelection.Item(GlobalSelection.Count)
    End If
End Sub

```

The Selection object's Count and Item properties are used to discover and then view the most recently selected entity.

## 5. Handle the events fired as a result of selections taking place

When it comes to responding to a change in the current selection, ActiveX controls and page objects diverge again. For the former, the wizard provides skeleton C++ code for handling the Selection object's three events. The latter require additional script code in their script modules:

```
Sub ScriptName_OnAddSelection(id)
End Sub

Sub ScriptName_OnRemoveSelection(id)
End Sub

Sub ScriptName_OnClearSelection()
End Sub
```

## Different selection types

The “global” selection mechanism described in Step 3 above is used by *most* of the objects in Workspace. Selecting an entity in Design View, for example, causes the same entity to be selected in the Game Explorer window, and its attributes to appear in the Attributes window.

There are a handful of windows that don't subscribe to the global mechanism. Selections in the Assets, Attribute Shares, and Behaviors windows do not cause selections to be made elsewhere, and vice versa. It's possible, however, that one of your custom controls could need to affect or react to selections in one of these windows. For such occasions, there are different strings that you can assign to an object's SelectionIdentifier property:

- AssetSelections, to link up with selections in the Assets window.
- AttributeShareSelections, to link up with selections in the Attribute Shares window.
- BehaviorSelections, to link up with selections in the Behaviors window.

Apart from specifying a different value for SelectionIdentifier, interacting with the selection mechanism for these windows is identical to doing so with the “global” mechanism.

# Broadcasting method calls

---

RenderWare Studio's `Broadcast` object makes it possible for code in the script module of any Workspace object to call a method in *every other* script module.

In its default configuration, Workspace uses `Broadcast` to invoke "standard" methods at various intervals during execution. For example, a call to `Broadcast_WorkspaceStart()` is broadcast after all `OnLoad()` calls are complete, while `Broadcast_PreNewProject()` is broadcast before Workspace begins the process of starting a new game project.

Any object that needs to do work on these occasions (such as loading or saving some custom settings) can just implement the methods in question and be sure of receiving the right call at the appropriate time.

## Calling standard broadcast methods

For convenience, calls to Workspace's standard broadcast methods are made from wrapper functions stored in the *Broadcast* script module. This means that a broadcast method call made from elsewhere in Workspace always has the following form:

```
Broadcast.MethodName argOne argTwo
```

Inside *Broadcast*, this call is converted into a broadcast, like so:

```
Sub MethodName(paramOne, paramTwo)
    RENManager.Broadcast.Broadcast_MethodName paramOne paramTwo
End Sub
```

As you can see, this looks just like calling an ordinary method of the `Broadcast` object. As a result of this call, however, any script module in Workspace containing a method called `Broadcast_MethodName()` that takes exactly two parameters will have that method invoked.:

```
Sub Broadcast_MethodName paramOne paramTwo
    ' Do work in response to the broadcast method call
End Sub
```

If no such method exists in any module, there is no error report.

**Note:** There is no compulsion for broadcast methods to go through the *Broadcast* script module; `RENManager.Broadcast` is available at all times in all script modules. Workspace does things this way for clarity of organization.

## Fire and forget

The broadcast mechanism makes it easy to add objects to Workspace without having to edit the script modules of existing objects. The broadcast method calls *always* go out, and it's up to individual objects whether they decide to handle them. This also means that objects (and their associated script modules) should work with minimal modifications in future versions of Workspace. As long as those broadcast calls are being made, objects are insulated from changes taking place elsewhere in the application.

## When are broadcast methods called?

The standard distribution of Workspace includes calls to broadcast methods that surround nine important events:

- Starting up and shutting down the application
- Before displaying application context menus
- Before and after projects are created, loaded, and saved
- Loading and saving the application's settings file
- Loading and saving the game's settings file
- Before and after "clean" or "build" operations
- Connecting to and disconnecting from a target console
- Changing the active folder, and viewing and editing assets
- Before and after parsing project source code

In the descriptions below, the names are given of the methods that are actually called through the broadcast mechanism. The wrappers in *Broadcast* have the same names and parameter lists, without the *Broadcast\_* prefix.

## Startup and shutdown

Objects that need to perform post-OnLoad() processing at startup, or some tidying up before application shutdown, should implement one or both of these methods:

```
Sub Broadcast_WorkspaceStart()
Sub Broadcast_WorkspaceEnd()
```

## Displaying context menus

The items in context menus can change, become enabled, etc. depending on the state of the application when they are requested. This work can be done in implementations of the following methods:

```
Sub Broadcast_PreShowObjectContextMenu(oContextMenu, nObjectType)
Sub Broadcast_PreShowListerContextMenu(oContextMenu, nListerType)
```

## Creating, loading, and saving projects

Creating a project implies closing one project and starting a fresh one. Some objects will need to load or save data during that process. They can do so in the implementations of these methods:

```
Sub Broadcast_PreNewProject()
Sub Broadcast_PostNewProject()

Sub Broadcast_PreLoadProject(strFilename)
Sub Broadcast_PostLoadProject(strFilename)

Sub Broadcast_PreSaveProject(strFilename)
Sub Broadcast_PostSaveProject(strFilename)
```

## Loading and saving application settings

The *GlobalScript* and *Persist* script modules both contain calls to broadcast methods in which objects should load and retrieve data from the *RwStudio.settings* file:

```
Sub Broadcast_UpdateApplicationSettingsVersion(XMLDoc,
XMLRootNode, nFileVersion)
Sub Broadcast_LoadApplicationSettings(XMLRootNode)
Sub Broadcast_SaveApplicationSettings(XMLDoc, XMLRootNode)
```

## Loading and saving game settings

As well as being able to load and save settings at the application level through the methods in the previous category, objects can choose to do the same thing on a game-by-game basis. The *Persist* script module broadcasts two method calls to enable this:

```
Sub Broadcast_LoadGameSettings(XMLRootNode)
Sub Broadcast_SaveGameSettings(XMLDoc, XMLRootNode)
```

## Before and after “cleans” and “builds”

Workspace objects are able to perform processing on either side of a “clean” or “build” operation by providing implementations for the following methods called from the *BroadcastEvents* script module:

```
Sub Broadcast_PreCleanProject(strConnectionID, strConnectionName)
Sub Broadcast_PostCleanProject(strConnectionID, strConnectionName)

Sub Broadcast_PreBuildProject(strConnectionID, strConnectionName)
Sub Broadcast_PostBuildProject(strConnectionID, strConnectionName)
```

## Target console connections

These two method calls are broadcast after a target console is connected to, or disconnected from. Workspace objects might choose to modify their appearance, for example, in response to such events:

```
Sub Broadcast_ConnectToTarget(nConnectionID)
Sub Broadcast_DisconnectFromTarget(nConnectionID)
```

## Active folders and assets

There are two methods that are called on either side of the active folder being changed and the new folder's assets being imported, and two more that are broadcast when an asset's type is requested, and when a user asks to edit an asset:

```
Sub Broadcast_ActiveFolderChanged(NewRootFolder)
Sub Broadcast_PostImportAllAssets()

Sub Broadcast_GetAssetResourceType(AssetID, strResourceFilename,
strAssetType, bHandled)
Sub Broadcast_EditAsset(Asset, bHandled)
```

## Parsing project source code

Finally, by implementing methods that are called just before and just after the game's source code is parsed, a Workspace object could (say) clear and then update itself based on the new source:

```
Sub Broadcast_PrepParseSource()
Sub Broadcast_PostParseSource()
```

# Global methods and constants

---

The *GlobalScript* module defines a range of methods and global constants that are used by many of the other Workspace objects over the duration of the application. It also instantiates a group of RenderWare Studio objects (making their properties, methods, and events available to script code elsewhere in Workspace), and implements handlers for several of these objects' events.

## Objects instantiated by *GlobalScript*

### **RWSUtils**

As its name suggests, the *RWSUtils* object provides a set of utility methods for Workspace. Many of these are to do with implementing “standard” functionality for RenderWare Studio projects; they include *NewGame()*, *ISDirty()*, and *SaveGame()*. The object also fires events on occasions such as the active folder being changed, or the game database being attached or detached as a project is opened or closed.

### **Undo**

The *Undo* object provides control over Workspace's undo stack, including stepwise undo and redo operations, and the ability to determine the number of items on the stack.

### **StreamEditControl**

The methods of the *StreamEditControl* object provide management of game data streams, so that their contents are synchronised with changes that take place in Workspace. *StreamEditControl* provides services to the *StreamViewer* control host object.

### **WorkspaceSettingsObj**

Methods of *WorkspaceSettingsObj* are called at startup and shutdown so that the current settings of Workspace itself are loaded from and saved to the *RwStudio.settings* file. (Workspace object settings are dealt with through the *Load* and *SaveApplicationSettings()* methods that are called through the *Broadcast* object.)

### **RWSScript**

The *RWSScript* object represents the RenderWare Studio Manager object, which supports the *IRWSScript* COM interface.

### **RWSComms**

The *RWSComms* object provides methods for managing the connections to consoles that appear in Workspace's Targets window.

### **RWSMakeProxy**

The *RWSMakeProxy* object provides two methods that allow objects in Workspace to handle any events fired during the game building process.

# Making changes

---

The topics in this part of the documentation expand on the information presented earlier about Enterprise Author and [Workspace](#) (p.424). They provide demonstrations of using Enterprise Author and Microsoft Visual C++ 7.1 to make changes to the user's experience of Workspace, through a set of tutorial projects and worked examples:

- [Using Enterprise Author to create a new layout for RenderWare Studio Workspace](#) (p.455)
- [Building a new menu bar and a new toolbar for Workspace with Enterprise Author](#) (p.462)
- [Using a custom Visual C++ 7.1 wizard to create a Workspace-compatible ActiveX control](#) (p.481)
- [Adding a new ActiveX control to a Workspace layout, and sending its events to other objects in the interface](#) (p.490)
- [Writing code in a Workspace-hosted ActiveX control that allows it to interact with the game database](#) (p.500)
- [Arranging for an object in Workspace to store and retrieve data from the RenderWare Studio setting file](#) (p.512)

# A new layout for Workspace

---

In this tutorial, you'll use what you've learned about the Enterprise Author user interface, and creating and editing alternative layouts, to make a new layout for Workspace. This layout is a simplified version of the built-in *General* layout that demonstrates how the latter was created, and provides a starting point for future customization.

## Who is it for?

This tutorial is for users of Enterprise Author and Workspace who seek further practice at manipulating windows in those applications. It's also for those looking for a practical example of layout creation in Enterprise Author. Following these instructions requires just a little knowledge of the Enterprise Author user interface; no coding is necessary.

**Note:** See [the RenderWare Studio documentation](#) (p.335) for information on how to create, edit, and delete layouts directly from Workspace.

## Tutorial structure

The single lesson in this tutorial takes you right through the process of creating a new layout for Workspace.

### [Creating the \*Simple\* layout](#) (p.456)

You use Enterprise Author to create, design, and test a new layout for Workspace. The new layout is similar in design to *General*, but features fewer objects by default.

# Lesson 1. Creating the “*Simple*” layout

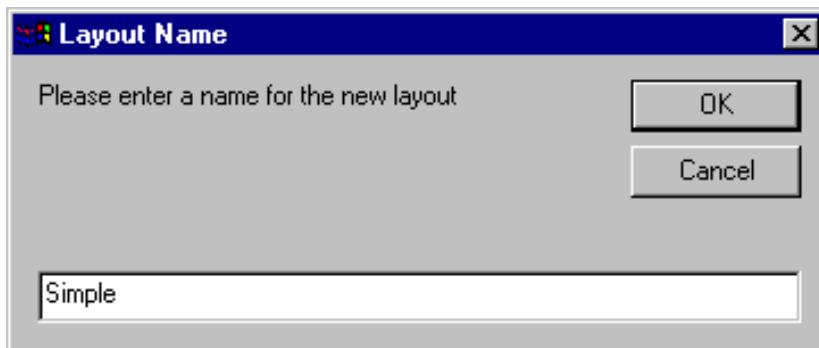
---

## In this lesson

You use Enterprise Author to create, design, and test a new layout for Workspace. The new layout is similar in design to *General*, but features fewer objects by default.

## Creating an empty layout

1. In Enterprise Author, select **File ▶ Open** and choose to open RenderWareStudio.ren for editing.
2. Choose **Select ▶ General** to display the default RenderWare Studio Workspace user interface in the Design window.
3. Select **Layouts ▶ Create** to begin the process of making a new layout. In the dialog that pops up, call your new layout *Simple*.

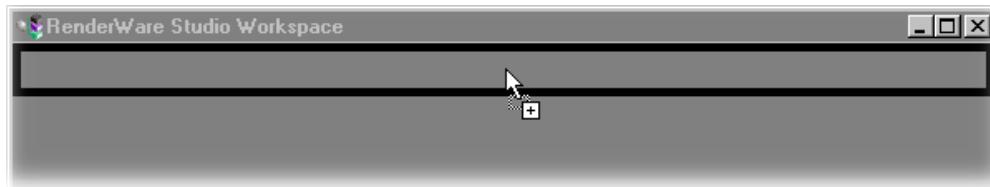


The view in Design changes to show an empty Workspace window, ready for you to add objects from the Objects window.

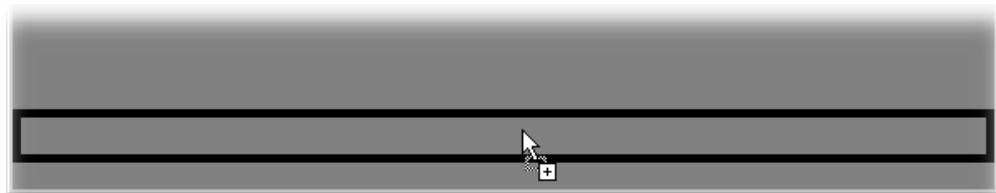
## Adding horizontal bars to the layout

Having created the new layout, we can start the process of populating it by adding the menu bar at the top of the window, and the Object Information bar at the bottom.

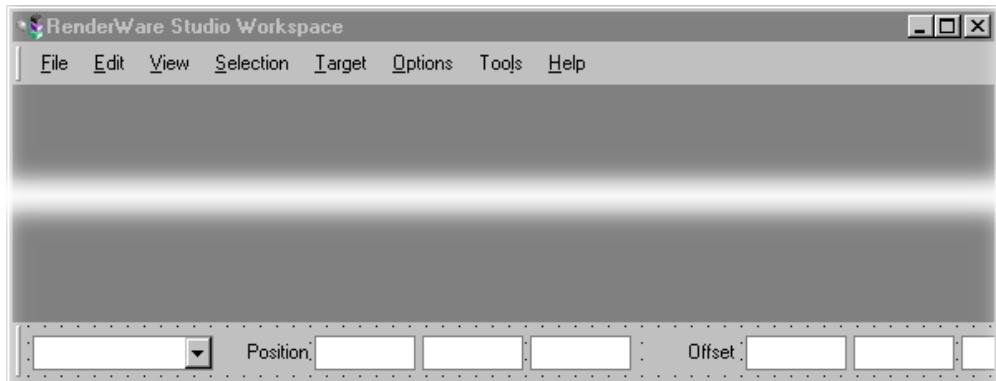
4. Locate *MainMenu* in the Objects window, and drag it to the top of the empty Workspace window. Drop the menu bar when you see the *thicker cue* rectangle:



5. Repeat the process above for *ObjectInformation*, this time dropping the object toward the bottom of the Workspace window:



With these two objects in position, the new layout has menu bar-like windows at its top and bottom edges, and you can move on.



## Adding a vertical toolbar to the layout

Docking the selection toolbar on the right of the application window, so that its buttons appear in a column, is easier than creating a horizontal toolbar. “Window-like” behavior is not possible for a vertical toolbar, so there's no danger of creating the wrong kind of object.

6. Drag *SelectionToolbar* from Objects to Design. When the cue rectangle appears over the right-hand edge of the Workspace window, drop it to make the toolbar:



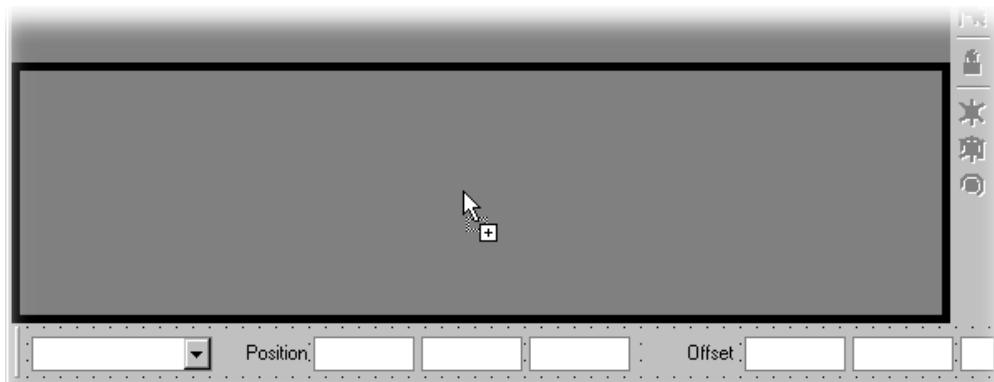
If you wish, menu bars and toolbars (and any other objects that have been installed with “toolbar-like” behavior) can be stacked several rows or columns deep at the vertical and horizontal extremes of Workspace's application window.

## Placing “horizontal” stackers in the layout

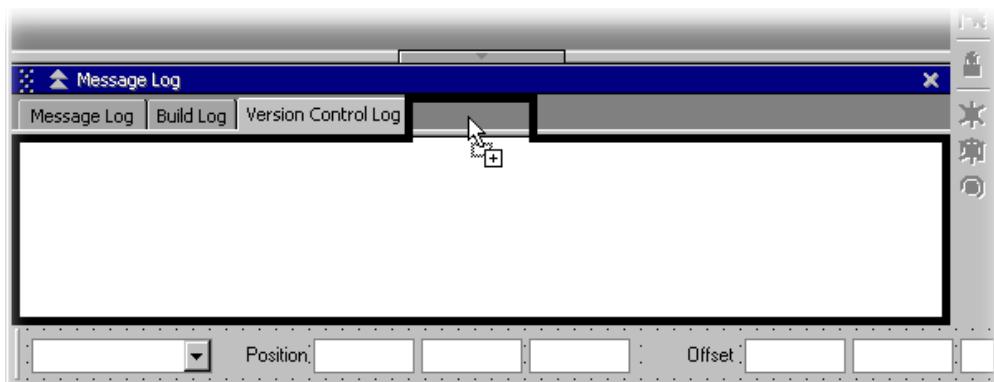
The next step is to begin populating the Workspace window with Workspace objects, starting at the bottom with the Output stacker.

**Note:** To save time, we won't actually rename the stackers in this tutorial. "Output" refers to the name of the stacker in the default *General* layout.

7. Drag *MessageLog* from the Objects window to the Design window, dropping it above the Object Information bar:



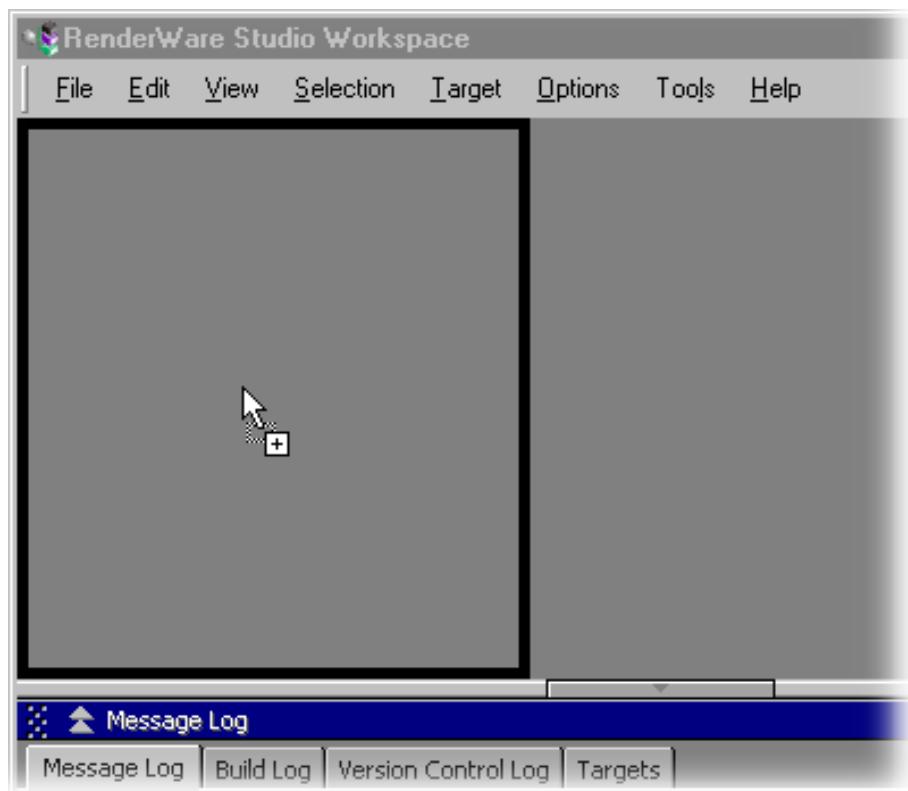
8. Right-click the new window's title bar, and select **Convert to Tabs**.
9. Drag-and-drop the *BuildLog*, *VersionControlLog*, and *TargetLink* objects into position alongside the **Message Log** tab control to finish this stacker:



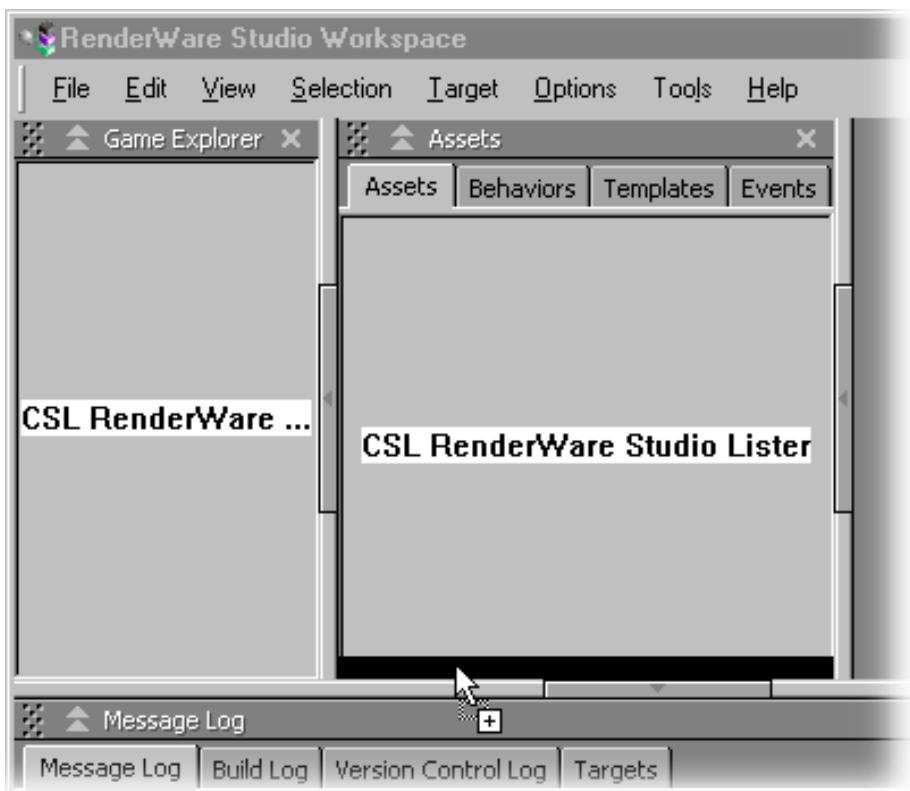
## Placing “vertical” stackers in the layout

“Vertical” stackers differ from “horizontal” ones only in that they *must* share a row with the work area. The simplest one in this new layout is Game Explorer, so we'll deal with it first.

10. Drag the *GameDatabase* object from Objects into Design, dropping it when you see the cue rectangle appear docked on the left-hand side of the Workspace window:



11. Repeat the previous step for the *AssetLister* object, docking it just to the right of *GameDatabase*. When it's in place, convert its contents to tabs, as you did earlier.
12. Add the *BehaviorLister*, *TemplateLister*, and *RWSEventList* objects as tabs alongside the **Assets** tab.
13. Drag the *EntityAttributes* object into the Design window to create a second stacker element beneath Assets:

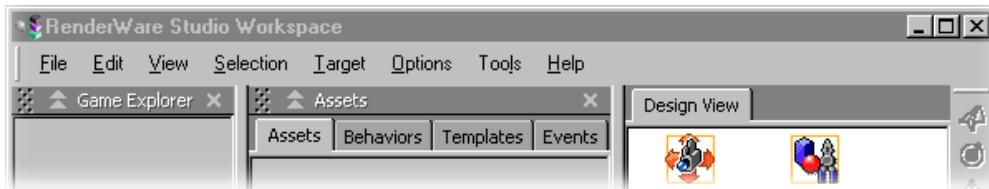


14. Drag the new title bar so that the stackers are of roughly equal size, convert the contents of the new one to tabs, and drag in the *AttributeShareLister*, *RF3AssetTemplateEditor*, and *RF3ProjectTemplateEditor* objects to complete it.

## Filling the work area

In Enterprise Author, it's not possible to create a set of tabs in the work area directly. Instead, you have to create a stacker as usual, and then perform the appropriate conversions.

15. Drag the *DesignView* object into the “work area” of the Workspace window. On this occasion, it doesn't matter which side you choose to dock it.
16. Right-click the title bar and select **Convert to Tabs**, and then right-click again and select **Set as Client**. The Design View window is now a tab in the work area:



17. To complete the layout, add the *EventVisualizer*, *RWSPreview*, and *HelpWindow* objects alongside the **Design View** tab in the work area.

## Testing the new layout

18. Press **Ctrl+S** to save your new layout into the RenderWareStudio.ren

project, and then **F5** to launch Workspace from within Enterprise Author. The application appears with its default *General* layout on view, but with a new item in its **View ▶ Layouts** menu.

19. Select **View ▶ Layouts ▶ Simple** and watch as the default layout is replaced by your new one.

This is the end of the tutorial.

# Making a new menu bar and a new toolbar

---

This tutorial builds on our theoretical discussion by providing a worked example of creating a new menu bar and a new toolbar for Workspace. The two new objects, which will offer equivalent functionality, will provide a quick way to clear the contents of the Build Log and Message Log windows.

**Note:** In general, it is not necessary to create an equivalent toolbar item for every menu item, so this tutorial will equip you to create one or the other, or both.

## Who is it for?

This tutorial is aimed at designers who want to customize Workspace with new menu bar and toolbar items that suit their way of working. It's also intended for developers who want to create new menu bars and toolbars for their custom ActiveX controls. It assumes familiarity with Workspace's user interface, and some elementary VBScript knowledge.

## Tutorial structure

The first three lessons in this tutorial explain how to create, customize, and then install a menu bar in Workspace. The fourth lesson then repeats the procedure with the changes necessary to create a toolbar, before the fifth describes how to insert the items from one menu into another dynamically. More specifically, the lessons proceed like this:

### [Creating a new menu bar object](#) (p.463)

You use Enterprise Author to create a menu bar object in the `RenderWareStudio.ren` file and provide sensible settings for some of its most basic properties.

### [Adding items to a menu bar object](#) (p.464)

You add menus and menu items to the menu bar you created in the previous lesson, and set up the names of the events that will be fired when users select menu items.

### [Installing a new menu bar in Workspace](#) (p.467)

You install the new menu bar in Workspace's user interface, and write the VBScript code that performs actions in response to menu items being selected.

### [Complementing a menu bar with a toolbar](#) (p.469)

You create, configure, and install a second menu bar object containing a toolbar with buttons that perform the same actions as the menu items of the previous lessons.

### [Merging menus at run time](#) (p.472)

You take the items from your custom menu bar and merge them dynamically into one of Workspace's standard menus at run time, without the need to modify the default application source files.

# Lesson 1. Creating a new menu bar object

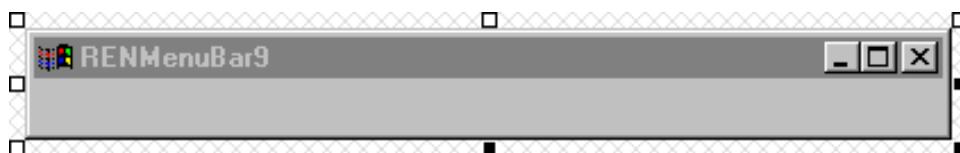
---

## In this lesson

You use Enterprise Author to create a menu bar object in the RenderWareStudio.ren file and provide sensible settings for some of its most basic properties.

## Creating a new menu bar

1. In Enterprise Author, select **File ▶ Open** and choose to open RenderWareStudio.ren for editing.  
The Objects window fills up with the collection of objects that comprise Workspace's user interface.
2. Select **Create ▶ Menu/Toolbar** to add a new menu bar to the project.
3. Right-click the new object, choose **Rename**, and then give your custom menu bar the new name *LogMenu*.
4. Double-click the new object to open it for editing in the Design window:



## Editing the menu bar's properties

Before you start adding menu items to your new menu bar, you need to make a couple of modifications to the properties of the object itself.

5. In the Property List window, change the *Caption* property to *Log Menu*.  
Workspace uses menu bar objects' *Caption* properties to populate its **View ▶ Toolbars menu** (p.244). Menu bars and toolbars can be displayed or hidden by selecting their names in this menu.
6. Use the center-right anchor point to reduce the width of the menu bar to its minimum extent. Enterprise Author will let you take this size down to around 100 pixels in this way.

You should now have a menu bar object that looks something like this one:



## Lesson 2. Adding menu items to the LogMenu menu bar

---

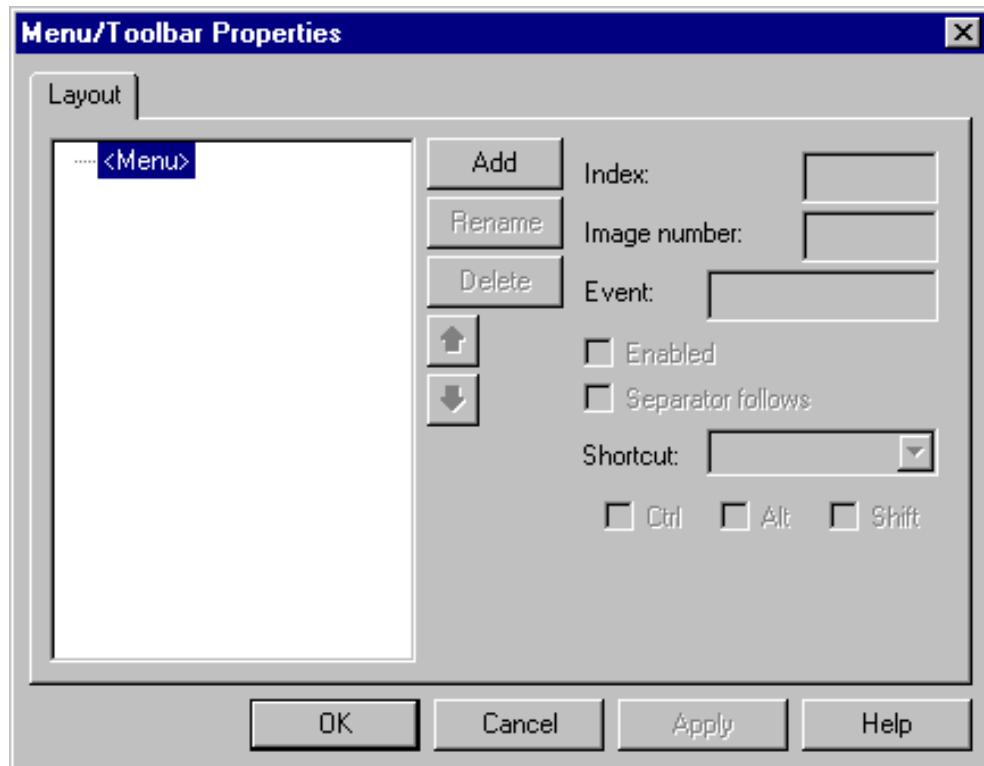
### In this lesson

You add menus and menu items to the menu bar you created in the previous lesson, and set up the names of the events that will be fired when users select menu items.

### Adding items to the LogMenu menu bar

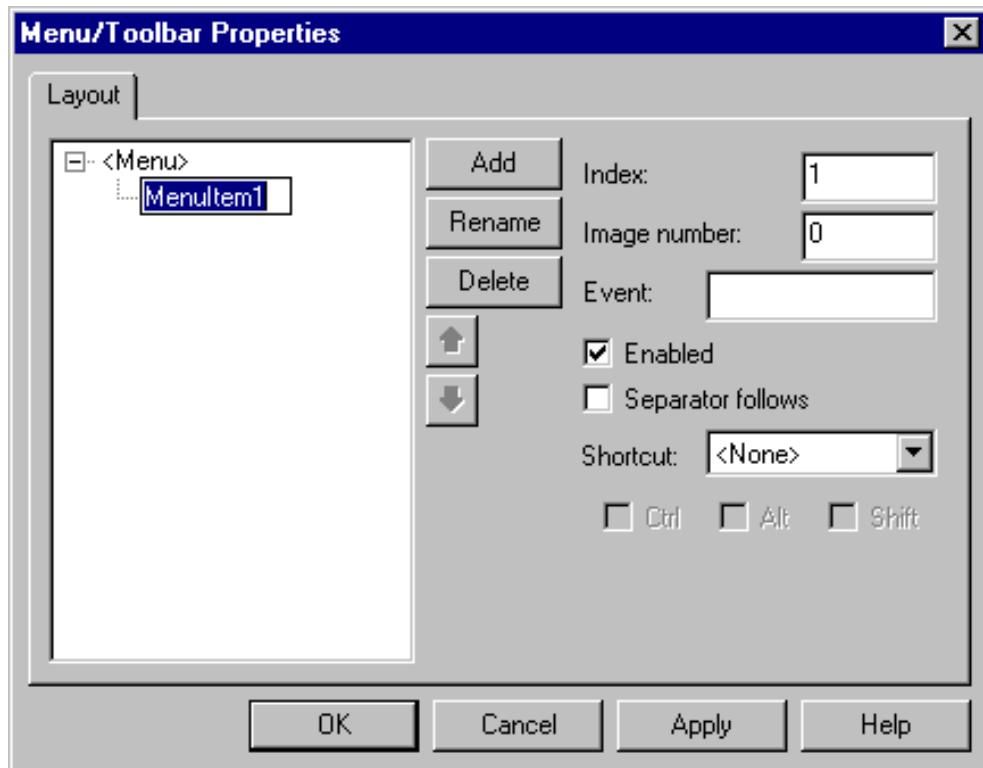
- With *LogMenu* still open for editing in the Design window, go to Property List and click the ... button in the space to the right of the (**Layout**) item.

The Menu/Toolbar Properties dialog appears:



A menu bar can contain several menus, and each menu can contain several items. In this dialog, all menus and items in the current menu bar are displayed in tree format, beneath the **<Menu>** item that represents the menu bar itself.

- Click **Add**, and the dialog changes to the following:



A new item has been created underneath **<Menu>**, and many of the other controls on the dialog have been enabled.

3. Change the name of the new item to **Logs**, and then click **Apply**.

Child items of the top-level **<Menu>** item become menus in your menu bar, and you'll see that **Logs** has appeared in Enterprise Author's representation of *LogMenu*.

4. Select **Logs** in the tree view, and then click **Add** to add an item beneath the **Logs** menu. Then repeat the process to give **Logs** a second item.
5. Rename the new items **Clear Build Log** and **Clear Message Log**, click **Apply** again, and (temporarily) dismiss the dialog.

If you examine *LogMenu* in Design, you'll find that your menu now has two items: **Logs ▶ Clear Build Log** and **Logs ▶ Clear Message Log**:



## Preparing LogMenu for installation in Workspace

As well its visual appearance, there are some further things that we need to specify in order for our new menu bar to work in Workspace. Specifically, we need to:

- Set up the **Logs** menu so that it opens in response to an **Alt+key**

combination.

- Provide names for the events that our menu items will fire when selected.
6. In Enterprise Author, reopen the Menu/Toolbar Properties dialog for the *LogMenu* menu bar, select **Logs** in the tree view, and then click **Rename**.
  7. Insert an ampersand (**&**) between the “o” and the “g” of “Logs”, and click **Apply**.

The visible effect of this change is that the name of the menu gains an underline (**Logs**); the practical effect is that when the menu bar has been installed in Workspace, **Alt+G** will open the **Logs** menu.

8. Select the **Clear Build Log** sub-item, and type `OnClearBuildLog` into the **Event** box. This is the name of the event that will be fired when a user selects this menu item.
9. Select **Clear Message Log**, and this time set the event name to `OnClearMessageLog`. Click **OK** to dismiss the dialog, and save the project at this stage.

# Lesson 3. Installing a new menu bar in Workspace

---

## In this lesson

You install the new menu bar in Workspace's user interface, and write the VBScript code that performs actions in response to menu items being selected.

## Positioning the new menu bar in the user interface

1. In Enterprise Author's Objects window, double-click *CSLRenderwareStudioWorkspace* to see Workspace's user interface in the Design window.
2. Drag and drop *LogMenu* from the Objects window to the menu bar area at the top of the Workspace user interface:



3. Drag the *LogMenu* menu bar from the position it took up by default (probably in a new line, by itself) to one next to the existing *MainMenu* menu bar:



## Writing code to handle menu item events

The final piece of the jigsaw is to use Enterprise Author's Script Editor to write code that handles the events fired by the new menu items. The most natural place to put this is the script module that's attached to the *LogMenu* object.

4. Double-click *LogMenu* to open it for editing once again, and click on Enterprise Author's Script Editor tab.
5. In the blank window that appears, enter these two procedures:

```
Sub OnClearBuildLog()
    BuildLog.Clear
End Sub

Sub OnClearMessageLog()
```

```
    MessageLog.Clear  
End Sub
```

The first of these procedures responds to the selection of **Logs ▶ Clear Build Log** by calling the `Clear()` method of the `BuildLog` object. The second responds to **Logs ▶ Clear Message Log** by calling the `MessageLog` object's `Clear()` method.

**Note:** No matter how complex the code you run as a result of user interaction with your menu items—in general, it will tend to involve more than a simple method call—it will always follow the same general structure. Take a look at the script module of the `MainMenu` menu bar for some more complex examples.

6. Save the project again, and select **Run ▶ Start**. Workspace will run as normal, and a click on **Logs ▶ Clear Message Log** will remove the startup messages from the Message Log window.

# Lesson 4. Complementing a menu bar with a toolbar

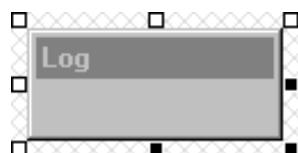
---

## In this lesson

You create, configure, and install a second menu bar object containing a toolbar with buttons that perform the same actions as the menu items of the previous lessons.

## Creating a toolbar

1. If it's open, close Workspace. Then, select **Create ▶ Objects ▶ Menu/Toolbar** in Enterprise Author to create another menu bar object in the RenderWareStudio.ren project.
2. Change the *Name* of the new object to *LogToolbar*.
3. In the Design window, reduce the width of the toolbar to its smallest setting, and use the Property List window to change its *Caption* property to *Log*.



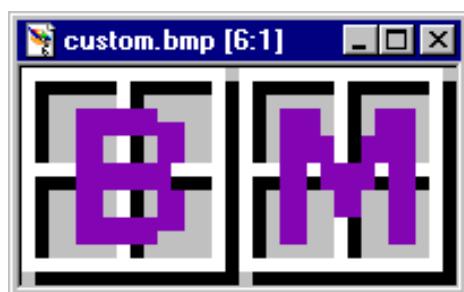
So far, this is no different from the creation of a menu bar, but now we need to follow a different path. For a start, we need to tell Enterprise Author that we're creating a toolbar rather than a menu bar.

## Configuring the toolbar

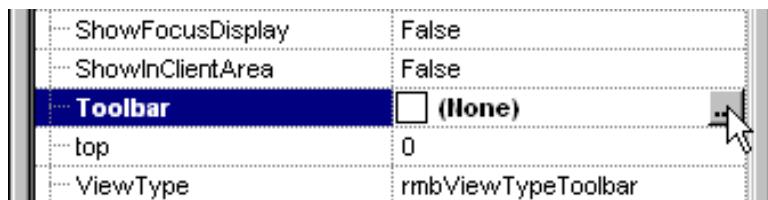
4. Return to the Property List window and change the value of the *ViewType* property from *rmbViewTypeMenu* to *rmbViewTypeToolbar*.

Our aim here is to create a toolbar button for each of our two new menu items, so the next things we need are images for two new buttons.

5. Ask your graphic designers to create a 32 x 16 pixel bitmap file called *custom.bmp* that contains two 16 x 16 button images, next to each other. The images should iconify the concepts of clearing the Build Log and Message Log windows respectively, and will surely look better than these:



- In the Property List window, highlight the *Toolbar* property, and click the ... button that appears next to its current setting (“(None)”).

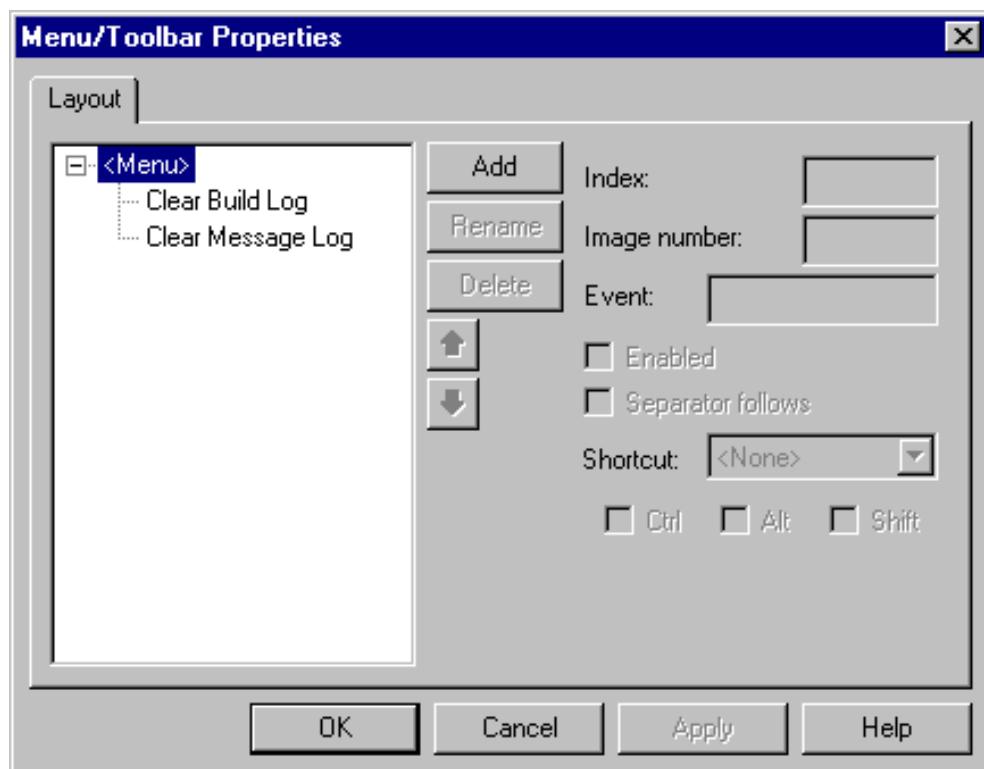


- Browse to your `custom.bmp` file in the dialog that appears, select it, and click **Open** to bring it into Enterprise Author.
- Still in Property List, select (**Layout**), and then click the ... button that appears opposite it to display the Menu/Toolbar Properties dialog for our *toolbar* object.

Although the toolbar's user interface is graphical, the names you provide here will appear in tooltips when the user hovers over the buttons, so it's important to make them friendly.

Also, you need to specify names for the events that will be fired when the buttons are clicked.

- Add items named **Clear Build Log** and **Clear Message Log** directly beneath **<Menu>** in the tree view:



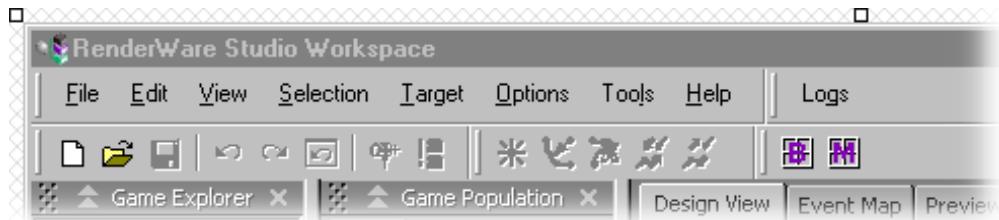
Remember that unlike menus, toolbars are not typically hierarchical. It is therefore unusual to have more than one level of nesting underneath the **<Menu>** item in Enterprise Author's menu bar objects.

- Set an **Image number** of 0 and an **Event** of `OnToolbarClearBuildLog` for the first sub-item, and an **Image number** of 1 and an **Event** of `OnToolbarClearMessageLog` for the second.

The images in the `.bmp` file are numbered from zero, so this associates the “B” image with the “Build” button, and the “M” image with the “Message” button.

## Installing the toolbar in Workspace

- Double-click `CSLRenderwareStudioWorkspace` in the Objects window, and add `LogToolbar` to the user interface, just as you did for `LogMenu`.



With the toolbar in place, the last step is to write the VBScript code that makes the buttons perform the same operations as the menu bar items they represent. In a simple case like this, we could reproduce the calls to the windows' `Clear()` methods, but in general it's more efficient to make the button event handlers call the menu event handlers.

- In the `LogToolbar` object's script module, add the following code:

```
Sub OnToolbarClearBuildLog()
    LogMenu.OnClearBuildLog
End Sub

Sub OnToolbarClearMessageLog()
    LogMenu.OnClearMessageLog
End Sub
```

These procedures hand responsibility for event processing straight to their companions in the `LogMenu` object, ensuring that selections from the menu and clicks on the toolbar result in exactly the same action.

- Save the project, and select **Run ▶ Start** for the final time. The Workspace user interface will appear with both of your new objects in it. The menu bar and the toolbar provide your users with a choice of ways to perform the operations you've assigned to them.

# Lesson 5. Merging menus at run time

---

## In this lesson

You take the items from your custom menu bar and merge them dynamically into one of Workspace's standard menus at run time, without the need to modify the default application source files.

## Dynamic menu merging

Enterprise Author allows the design-time editing of every menu bar, toolbar, and context menu in Workspace. Rather than creating a whole new menu for the two menu items in this tutorial, you could just as easily have added the items to one of Workspace's existing menus.

Hard-wiring customizations into Workspace's default menu set is not ideal, however. If Workspace's default menus change, so that you need to use a new set of application source files, you lose the edits you've made to the old set. To avoid this problem, you can use Enterprise Author's dynamic menu merging feature.

1. With RenderWareStudio.ren still open in Enterprise Author, double-click *CSLRenderWareStudioWorkspace* in the Objects window to display it in the Design window.
2. Right-click the *LogMenu* menu bar's grabber control and select **Close**. This removes the menu from the user interface, but not from the project. You're now going to add code to the *LogMenu* object's script module so that when Workspace is executed, the menu's items are inserted into the **View** menu.
3. In the Objects window, double-click *LogMenu*, and then select the Script Editor tab.
4. Add the following method definition beneath the two you wrote in Lesson 3:

```
Sub OnLoad()
    MainMenu.Merge LogMenu,
        LogMenu.MenuItems("Lo&gs"),
        MainMenu.MenuItems("&View").MenuItems("&Refresh
Event View")
End Sub
```

During startup, the *OnLoad()* method is called on every Workspace object that implements it. Here, it's being used to specify that the items in the **Logs** menu of the *LogMenu* menu bar should Merge with the **View** menu of the *MainMenu* menu bar. (Just like menu item creation, the ampersands here reflect underscores in the menu items' on-screen appearance.)

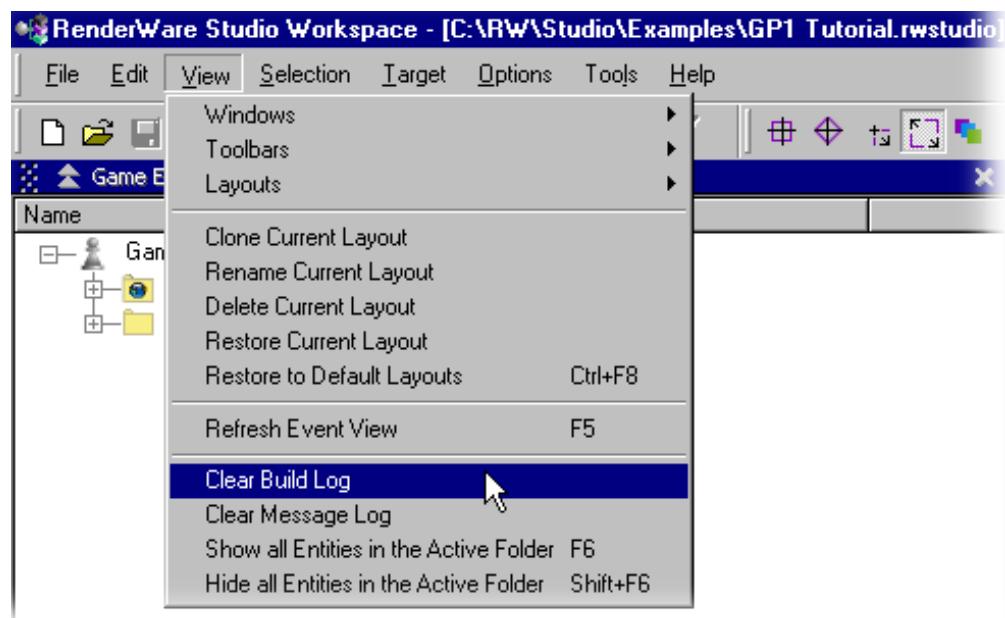
The third argument to *MainMenu*'s *Merge()* method dictates that the items in question should be placed immediately after **View ▶ Refresh Event View**.

**Caution:** There's a potential problem with menu merging in RenderWare Studio 2.0. The **Index** values of any items that you merge into a menu *must*

be different from those of the items already in the menu. To make this example work, change the **Index** values of **Clear Build Log** and **Clear Message Log** from 2 and 3 to (say) 102 and 103 respectively.

## Testing the merged menu

5. Save the project, and then select **Run ▶ Start** to lauch Workspace from Enterprise Author.
6. Select the **View** menu, and you'll see this:



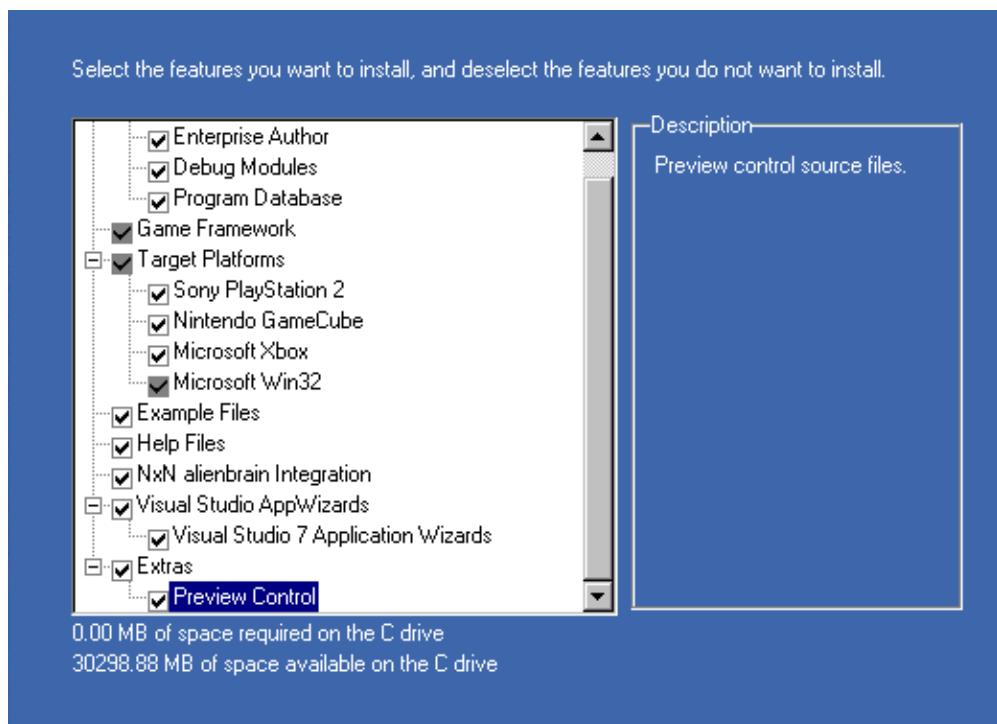
The items from the old **Logs** menu now appear in the **View** menu, as you specified. Furthermore, the **Clear Build Log** and **Clear Message Log** items still perform the tasks they performed when they occupied their own menu bar in their own menu bar.

This is the end of the tutorial.

# Customizing the Preview window

RenderWare Studio ships with the C++ source code for Workspace's [Preview window](#) (p.300), giving users the ability to change it to work with the custom asset types they use. The project files that enable this facility are not installed by default, but doing so is straightforward.

1. Select **Start ▶ Programs ▶ RenderWare Studio ▶ Change or Remove RenderWare Studio**, and click **Next >** in the dialog that appears. (The **Modify** option is selected by default.)
2. Select the **Preview Control** feature in the **Extras** section at the foot of the list on the next page.

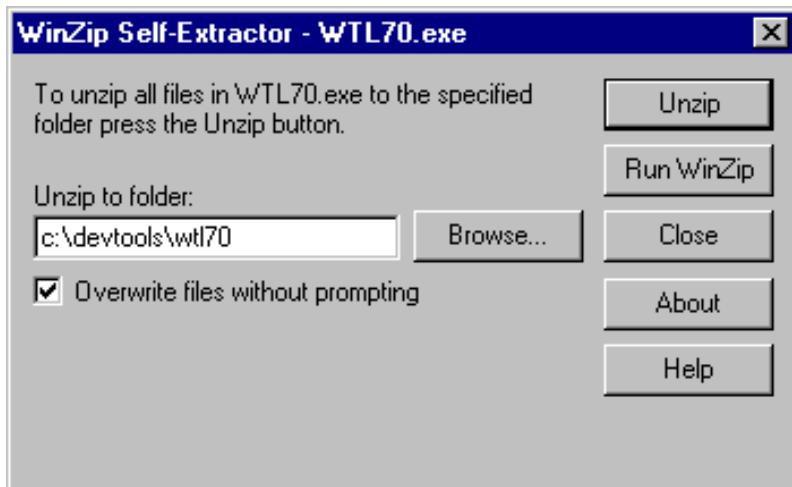


Completing the remaining steps of the setup tool as usual then installs the source code files for the *RWSPreview* ActiveX control on your machine, in a new directory called `C:\RW\Studio\Components`.

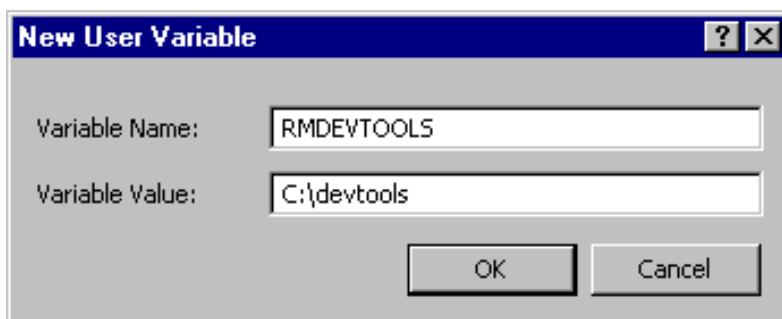
Before you can build the ActiveX control, you need to install Microsoft's *Windows Template Library* (which the control uses) on your computer. You also need to create an environment variable that points to the location of the new installation.

3. Browse to [Microsoft's download page](#) ([www.microsoft.com/downloads](http://www.microsoft.com/downloads)) and search for "windows template library" by typing that phrase into the **Keywords** box.
4. Select the link to download **Windows Template Library (WTL) 7.0**. The Preview control was developed using this version of WTL, not the newer 7.1 library.
5. Double-click `WTL70.exe` from its downloaded location, and extract its

contents to a new directory on your hard drive. C:\devtools\wtl70 is a good place for it.



6. Right-click My Computer and select **Properties | Advanced**. Click **Environment Variables...** to display the dialog of the same name, and then click **New...** in the **User variables** group box.



7. Create a new environment variable called *RMDEVTOOLS*, and set its value to the path of your WTL installation's *parent* directory.

If you followed the advice above, this will be C:\devtools.

Now that WTL is installed correctly, you can load and build the Preview control project in Visual Studio .NET 2003 and create RWSPreview.exe.

8. In Visual Studio .NET 2003, open the project file located at C:\RW\Studio\Components\ActiveX Controls\RWSPreview\RWSPreview.vcproj into an empty solution.
9. Build the solution by pressing F7. If all has gone well, Visual Studio will place the intermediate files at ... \Components\ActiveX Controls\RWSPreview\DebugU (or ReleaseU), and RWSPreview.exe itself at ... \Components\debugu or ... \Components\releaseu (depending on the active configuration).

Finally, you need to register this new version of the Preview control, so that any changes you make to the control will be reflected in the Workspace user interface.

10. Open a command prompt, browse to the directory containing RWSPreview.exe, and execute the following command:

> **rwspreview.exe /regserver**

This replaces the information in the Registry with details of the location of the custom-built Preview control. If you ever need to switch back, just execute the same command against the original version of the file, which is in C:\RW\Studio\Programs.

# Creating ActiveX controls for Workspace

---

When you install RenderWare Studio, and provided that Microsoft Visual C++ 7.1 is already installed on the same machine, you're given the option to install the **Visual Studio .NET Application Wizards**. Selecting this option adds two new entries in the list of Visual C++ project types that can be created in Visual Studio .NET 2003:

- RenderWare Studio ActiveX Control
- RenderWare Studio Attribute Editor Control

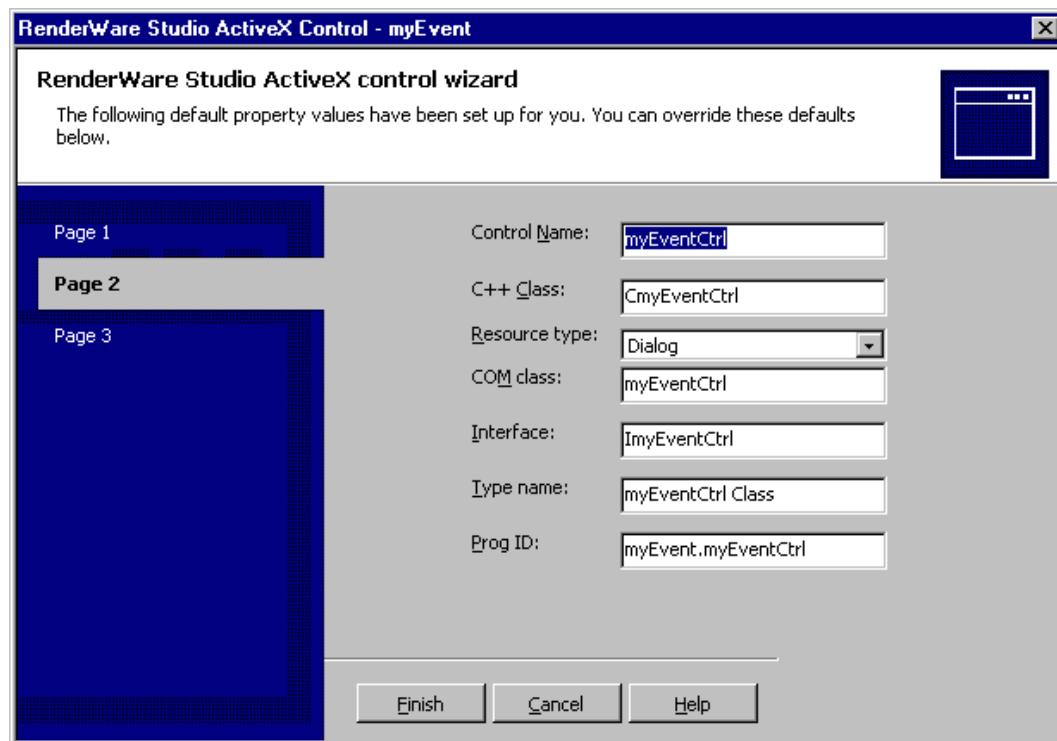
Creating controls based on the second of these project types is discussed in the section on [developing behaviors](#) (p.382). In this section, we'll examine the Renderware Studio ActiveX control wizard.

## The RenderWare Studio ActiveX control wizard

After choosing the project type and giving it a name in the usual way, Visual C++ 7.1 pitches you into the first page of a three-page wizard:



The text here explains what the wizard's about and what it will create, but the only thing to do is click **Page 2** and move to the next page:



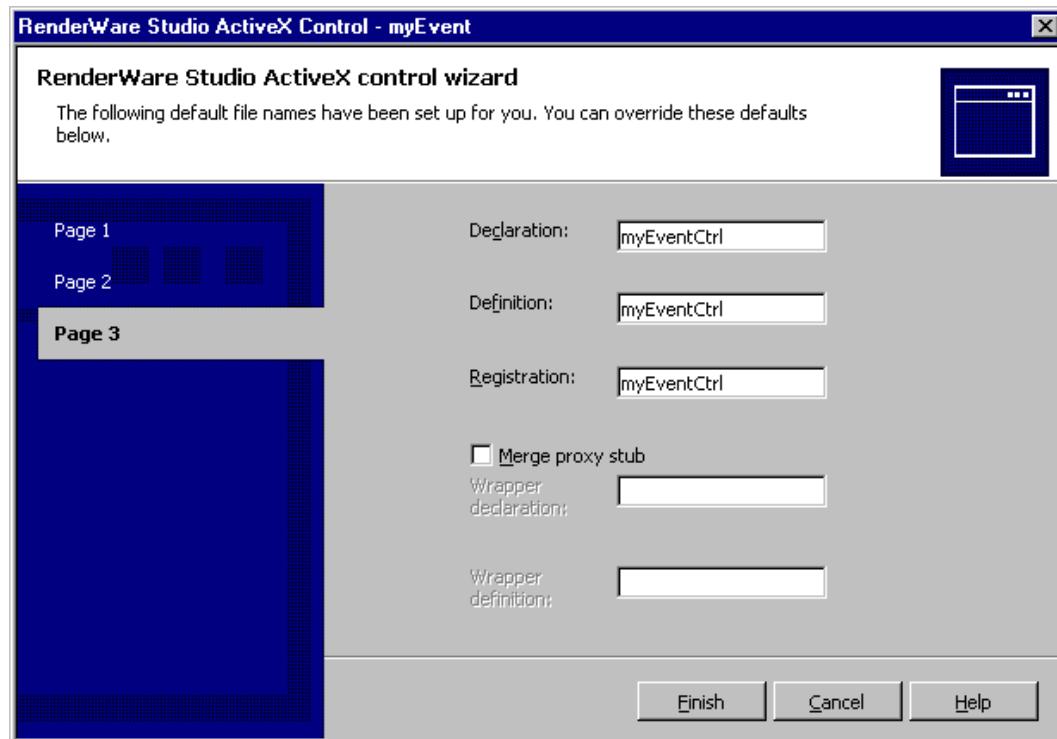
Like most wizards, this one allows you to change the default features of the project. Page 2 gives you control over the COM aspects of your control. The one you're most likely to want to alter is the **Resource type** drop-down, which dictates the built-in functionality of the control you're creating. The settings are:

**Dialog:** A standard dialog control that can host ordinary Windows controls—text boxes, list boxes, radio buttons, and so on

**ActiveX Dialog:** As above, but with the ability also to host ActiveX controls

**Window:** A control that's just a plain window, with no additional features

Page 3, on the other hand, lets you set the names of the .cpp, .h, and .rgs files that will be created:



## Generated code

When you click **Finish**, the wizard goes away and generates the source code for an “empty” RenderWare Studio-compatible ActiveX control. From the developer’s point of view, the two most important files are *ProjectNameCtrl.h* and *ProjectNameCtrl.cpp*. They contain the definition and implementation of the ActiveX control, and come complete with some built-in functionality.

In total, *ProjectNameCtrl.cpp* has implementations for ten methods. Between them, these deal with three important RenderWare Studio mechanisms:

- Calling `RWSInitialize()` and `RWSShutdown()`, so that code elsewhere in the control can use the features of the RenderWare Studio Manager API.
- Providing a means for script code to call `RWSDatabaseAttach()` and `RWSDatabaseDetach()`. All controls must “attach” to the game database before they can perform operations on it.
- Setting up [the Selection object](#) (p.447), so that the control can be configured to receive events when game entities are selected in Workspace windows.

In (roughly) the order in which they’re called during the lifetime of a Workspace object, the ten methods implemented by the RenderWare Studio ActiveX Control wizard are:

Name	Description
<code>FinalConstruct()</code>	Called when the control is created. Creates a <a href="#">Selection object</a> , which allows the control to keep track of selected entities.

<code>OnCreate()</code>	Called when the control is instantiated. Calls <code>RWSInitialize()</code> to hook the control up with the RenderWare Studio Manager API. Also sets up the control as the default handler for events fired by the <code>Selection</code> object.
<code>AttachToDatabase()</code>	Called from script code—usually, from <code>RWSUtils_OnDatabaseAttached()</code> in the <i>GlobalScript</i> script module. Calls <code>RWSDatabaseAttach()</code> to link the control to the game database.
<code>put_SelectionIdentifier()</code> <code>get_SelectionIdentifier()</code>	Implement the <code>SelectionIdentifier</code> property, which is used from script code to specify what kinds of selections the control should keep track of (that is, in what Workspace window they appear).
<code>OnAddSelection()</code> <code>OnRemoveSelection()</code> <code>OnClearSelection()</code>	Handle the events fired by the <code>Selection</code> object, allowing the control to respond to users' selection changes. You'll typically re-implement these methods when you write your own controls.
<code>OnDestroy()</code>	Called when an instance of the control is destroyed. Calls <code>RWSDatabaseDetach()</code> (indirectly) to break the link to the game database; calls <code>RWSShutdown()</code> to unhook from the RenderWare Studio Manager API; arranges for the control no longer to receive events from the <code>Selection</code> object.
<code>FinalRelease()</code>	Called when the last instance of the control is destroyed. Calls <code>Release()</code> on the <code>Selection</code> object.

## Customizing the control

These (relatively small) additions aside, the RenderWare Studio ActiveX control wizard just creates ordinary ActiveX control projects. When creating your own controls, you have free rein to use any of the functionality you'd normally use in an ActiveX control, plus the features afforded by the wizard: the Manager API, the game database, and the selection mechanism.

For examples of using all three of these mechanisms from C++ code, see the tutorials on [creating](#) (p.481), [installing](#) (p.490), [enhancing](#) (p.500) a RenderWare Studio ActiveX control.

# Creating a custom ActiveX control for Workspace

---

This tutorial uses Microsoft Visual C++ 7.1 and the RenderWare Studio ActiveX Control Wizard to create a simple ActiveX control that can be embedded into Workspace to customize its user interface.

## Who is it for?

This tutorial is aimed at developers who seek to add custom functionality to Workspace. It assumes that you are familiar with using Microsoft Visual C++, and it would help if you also have some experience of creating ActiveX controls.

## Tutorial structure

For the purposes of this example, we're going to keep things simple. The control will have a single button that, when clicked, will send messages for other controls in Workspace to receive. In building the control, we'll look at the following topics:

### [Using the Wizard to create a custom ActiveX control project](#) (p.482)

You step through the RenderWare Studio ActiveX control wizard and create a skeleton project.

### [Building and debugging the custom ActiveX control](#) (p.483)

You execute the custom ActiveX control in the Test Container, using breakpoints to understand the actions that take place.

### [Adding a command button to the control](#) (p.486)

You use Visual C++ to add a button to the control's user interface.

### [Handling the button click event within the control](#) (p.488)

You write code that performs an action in response to the button being clicked.

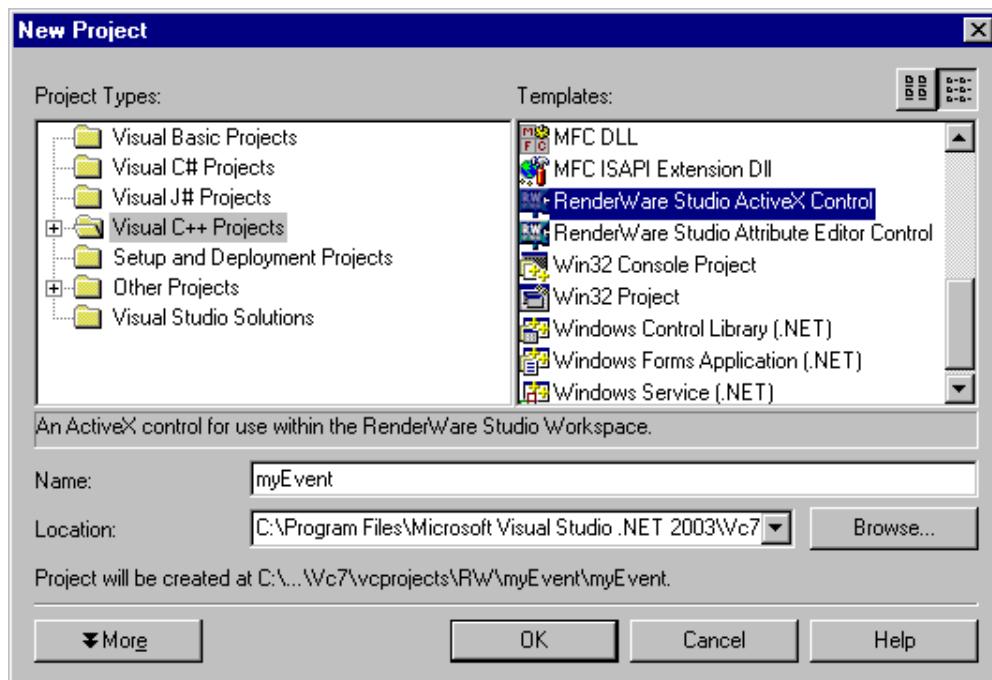
# Lesson 1. Using the RenderWare Studio ActiveX Control Wizard

---

## In this lesson

You step through the RenderWare Studio ActiveX control wizard and create a skeleton project.

1. Start up Visual Studio .NET 2003 and use **File ▶ New ▶ Project** to bring up the New Project window. Since installing RenderWare Studio, you'll find that some new project types have been added to the list. For our purposes, the important one is **RenderWare Studio ActiveX Control**.



2. Select the wizard and provide a name (we've used `myEvent` here) and a location for your project. When you click **OK**, the first page of a three-step wizard appears.

**Note:** For more information about this wizard and the code it generates, take a look at the topic on [the RenderWare Studio ActiveX control wizard](#) (p.477).

3. On this occasion, leave the default settings alone and click **Finish**. Visual Studio .NET 2003 will then create the project.

## Lesson 2. Building and debugging the control

### In this lesson

You execute the custom ActiveX control in the Test Container, using breakpoints to understand the actions that take place.

To confirm that our new project creates an ActiveX control, we can try hosting it in the ActiveX Control Test Container.

1. Set the build configuration to *Release*:

- In the *Standard* toolbar, select **Release** from the **Solution Configurations** drop-down.

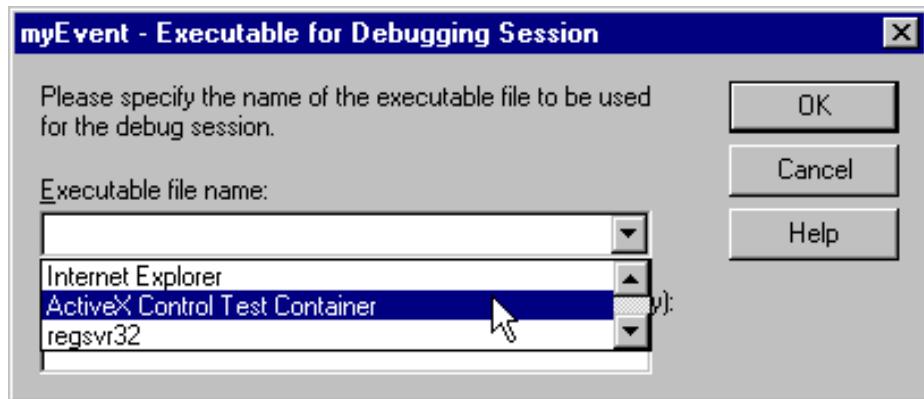
**Note:** To use the *Debug* configuration:

- a. Select the appropriate item in the **Solution Configurations** drop-down.
- b. Select **Project ▶ Properties...**
- c. In the dialog that appears, choose the **Configuration Properties ▶ Linker ▶ Input** item.
- d. In the **Additional Dependencies** box, change the file name from `RWSud.lib` to `RWSu.lib`.

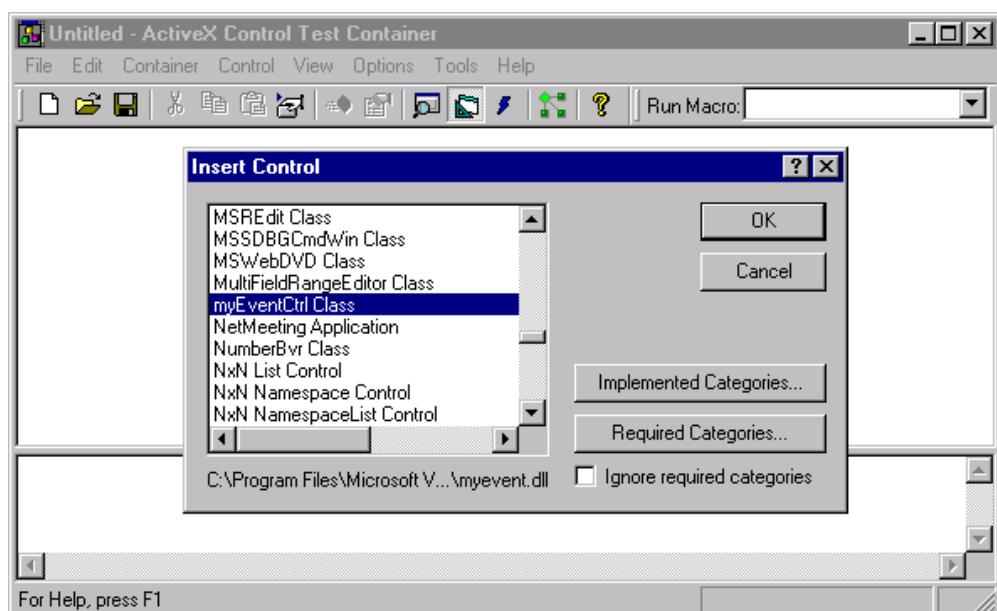
2. Press **F7** to build the project and create the `myEvent.dll` file that contains our skeleton ActiveX control. Visual Studio .NET 2003 also registers our new ActiveX control at this stage. Now we can make sure that everything is working correctly.

**Note:** If Visual Studio .NET reports that it is unable to register the control, you need to do it by hand:

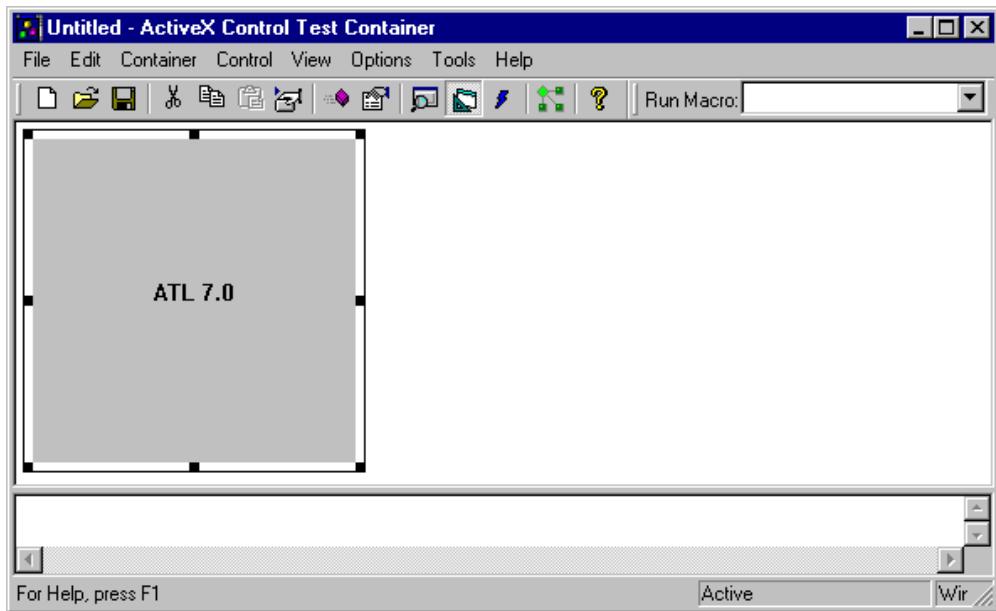
- a. Browse to the DLL in Windows Explorer, right-click the file, and then select **Choose Program**.
  - b. In the Open With dialog, click **Other...**, and then navigate to `C:\WINNT\system32\regsvr32.exe`.
  - c. Double-click the executable, and then click **OK** in the Open With dialog.
3. Press **F5** to start the debugging process, and agree to build any out-of-date files if you're prompted. Because control projects need a host application to run them, Visual Studio will now ask you to choose an appropriate executable to perform this function. Later, this will be Workspace, but for now you can choose the ActiveX Control Test Container:



4. To add our control to the test container, go to **Edit > Insert New Control** and select **myEventCtrl Class** as the control to insert:



5. The grey rectangle of our control appears in the window. We've yet to add any custom functionality, but we can at least confirm that it works as expected:



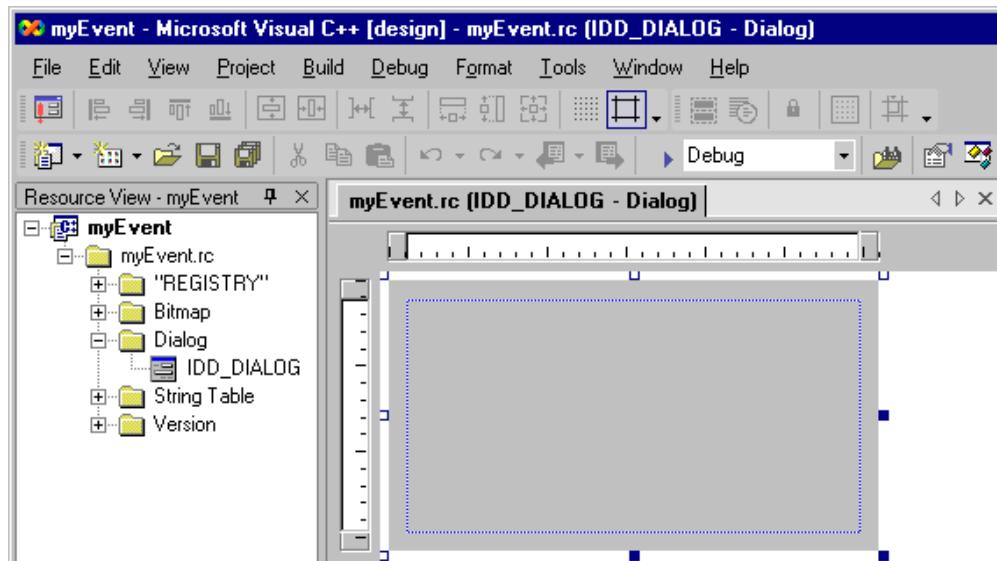
6. Click the icon (or press **Shift+F5**) to stop the debugging process (and thereby close the test container). Our task now is to make the control *do* something.

## Lesson 3. Adding a command button

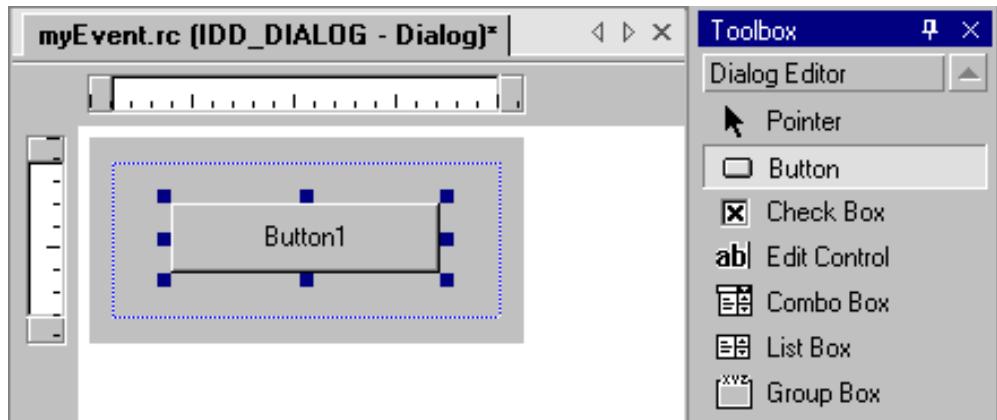
### In this lesson

You use Visual C++ to add a button to the control's user interface.

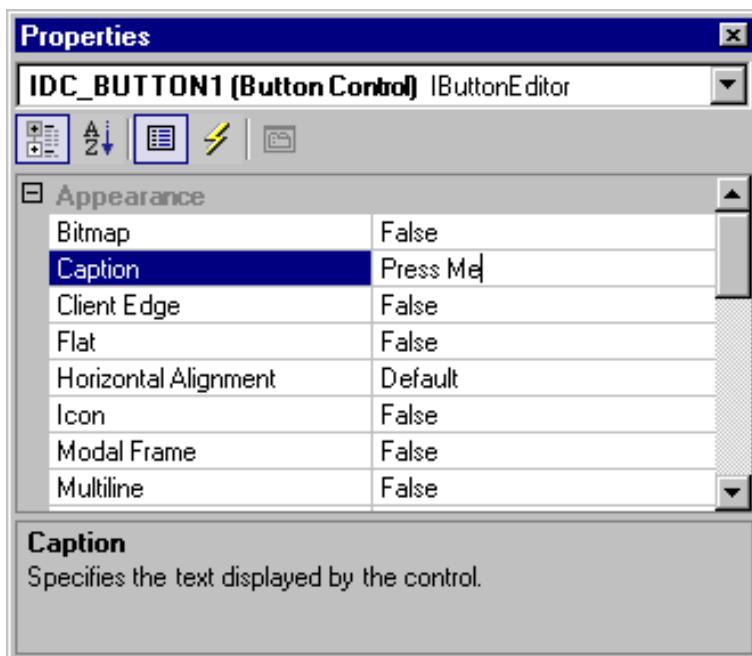
1. In Visual Studio, switch to Resource View and select the dialog called IDD\_DIALOG:



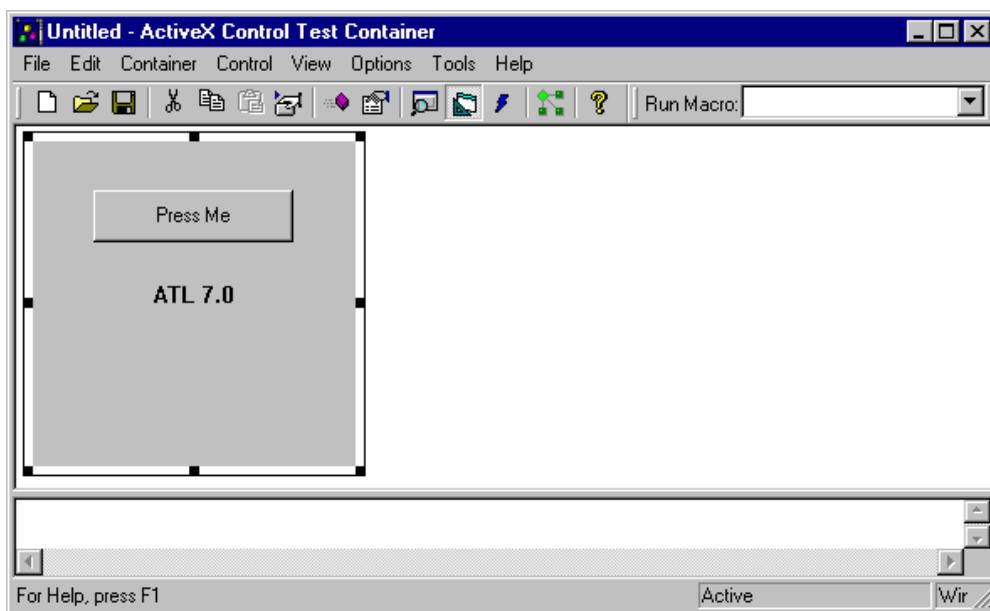
2. Choose **Button** from the Dialog Editor section of the Toolbox. Draw a button on the dialog:



3. Right-click the button, choose **Properties**, and change the caption to **Press Me**:



4. Build the project again and debug it in the test container. This time, you should see the button we've added to our control.



#### Get rid of the “ATL 7.0” text

The **ATL 7.0** label appears on all controls created using the RenderWare Studio ActiveX control wizard as a result of default code for the `OnDraw()` method located in `atlctl.h`. To get rid of it from your control, add a trivial `OnDraw()` implementation of your own to `myEventCtrl.h`:

```
protected:
    LRESULT OnCreate (UINT, WPARAM, LPARAM, BOOL& bHandled);
    LRESULT OnDestroy (UINT, WPARAM, LPARAM, BOOL& bHandled);
    virtual HRESULT OnDraw(ATL_DRAWINFO& di) { return S_OK; }
```

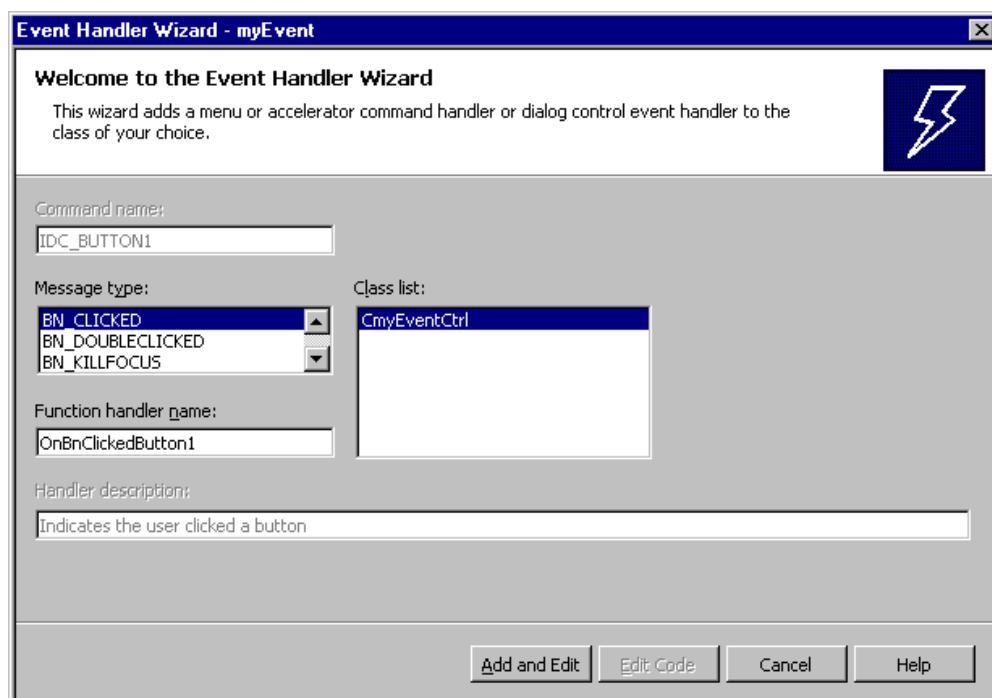
## Lesson 4. Handling the button click event

---

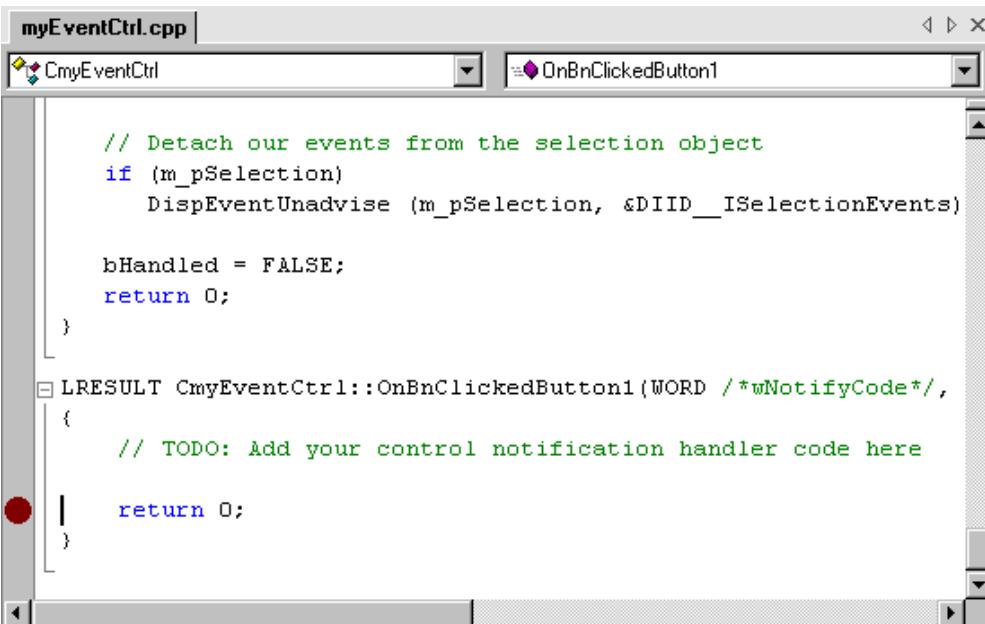
### In this lesson

You write code that performs an action in response to the button being clicked.

1. Lastly, we'll add some code that allows our new button to interact with its surroundings. Right-click the button again, this time choosing **Add Event Handler....**
2. In the resulting dialog, choose the `BN_CLICKED` message type and click **Add and Edit**.



3. When you dismiss this dialog, Visual C++ will take you to the new code it has generated in `myEventCtrl.cpp`. Our `CmyEventCtrl` class has gained a method that will be called whenever the button is clicked.



The screenshot shows a debugger window titled "myEventCtrl.cpp". The code editor displays the "CmyEventCtrl" class definition. A red dot, indicating a breakpoint, is placed on the line "return 0;" within the "OnBnClickedButton1" method. The code itself includes comments about detaching events and TODO notes for adding control notification handler code.

```
// Detach our events from the selection object
if (m_pSelection)
    DispEventUnadvise (m_pSelection, &DIID__ISelectionEvents)

bHandled = FALSE;
return 0;
}

HRESULT CmyEventCtrl::OnBnClickedButton1(WORD /*wNotifyCode*/,
{
    // TODO: Add your control notification handler code here
    |
    return 0;
}
```

4. To test this, place a breakpoint on the `return` statement in this method, and debug the application as before. Click the button and confirm that the breakpoint is hit.

In this tutorial, we've created a simple ActiveX control that's designed to work with Workspace. To move forward, we need to add our new control to the Workspace user interface, and that's the subject of the next section.

# Adding an ActiveX control to Workspace

---

In this tutorial, we'll take the simple ActiveX control that we developed in the last section and add it to the Workspace user interface.

## Who is it for?

This tutorial is aimed at developers seeking to understand how the Workspace user interface works, and at designers wanting to see how to use Enterprise Author to customize Workspace.

## Tutorial structure

Like the last tutorial, this one has four sections. Together, they explore how Workspace and Enterprise Author work, and then explain how to code the interaction between your new control and the Workspace user interface.

### [Installing a control in the Workspace user interface](#) (p.491)

You use Enterprise Author to add a custom ActiveX control to the Workspace user interface.

### [Debugging custom controls using Workspace](#) (p.493)

You set up Workspace as the application in which to debug your new ActiveX control.

### [Firing events from an ActiveX control](#) (p.494)

You create an event that your control will fire, to be handled by Workspace.

### [Handling ActiveX control events in Workspace](#) (p.497)

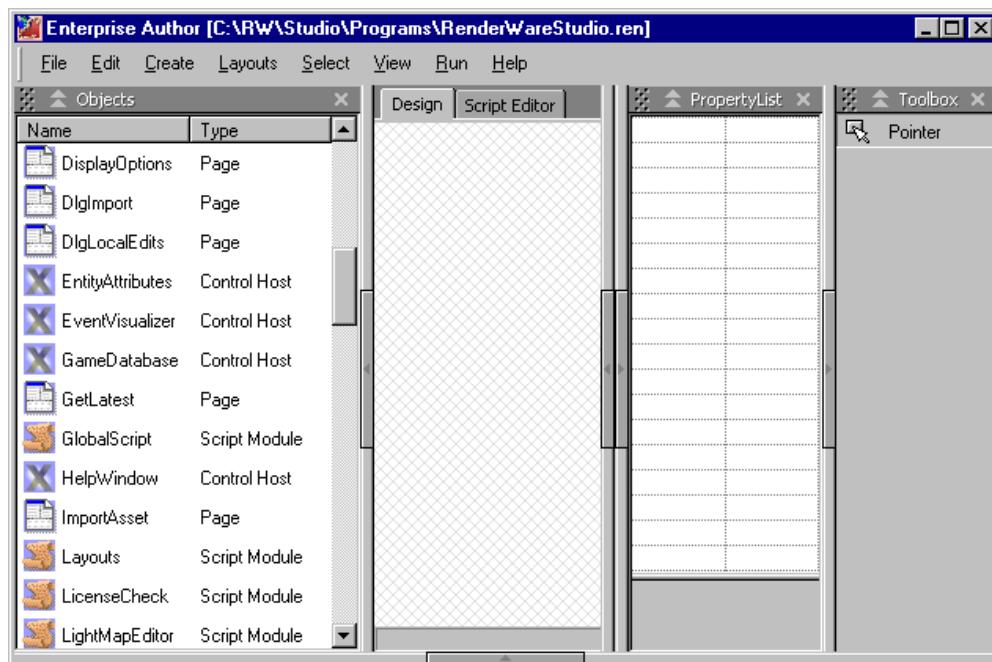
You add script code to Workspace that handles the event you fired in the previous lesson.

# Lesson 1. Installing a control in the Workspace user interface

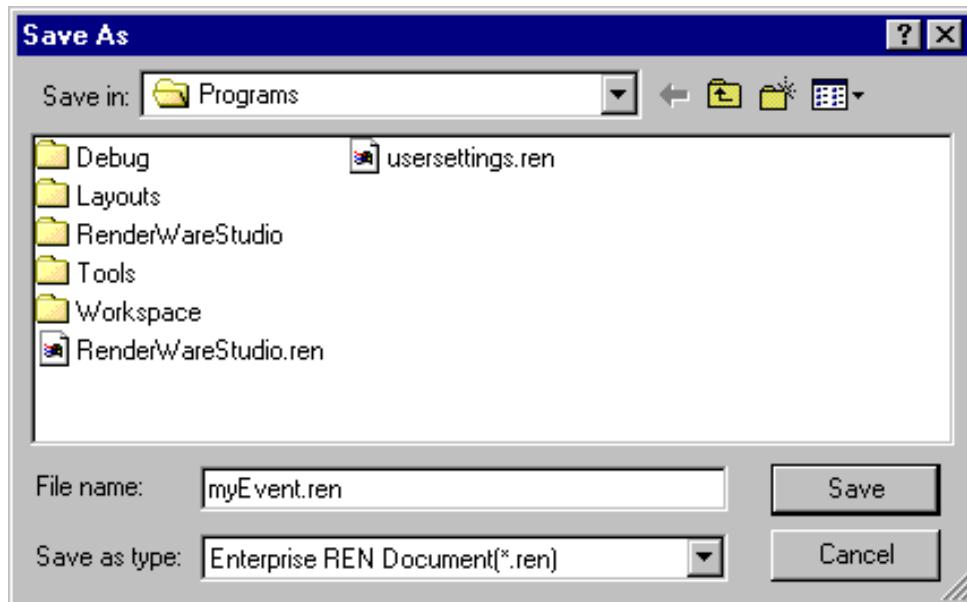
## In this lesson

You use Enterprise Author to add a custom ActiveX control to the Workspace user interface.

1. Start up Enterprise Author, and open the RenderwareStudio.ren file. For a default installation, you'll find it in C:\RW\Studio\Programs.



2. Double-click *CSLRenderwareStudioWorkspace* in the Objects window to open it for editing. A schematic representation of the Workspace window appears in the middle of the Enterprise Author window.
3. Select **File ▶ Save As...** and save a copy of the project as `myEvent.ren`:

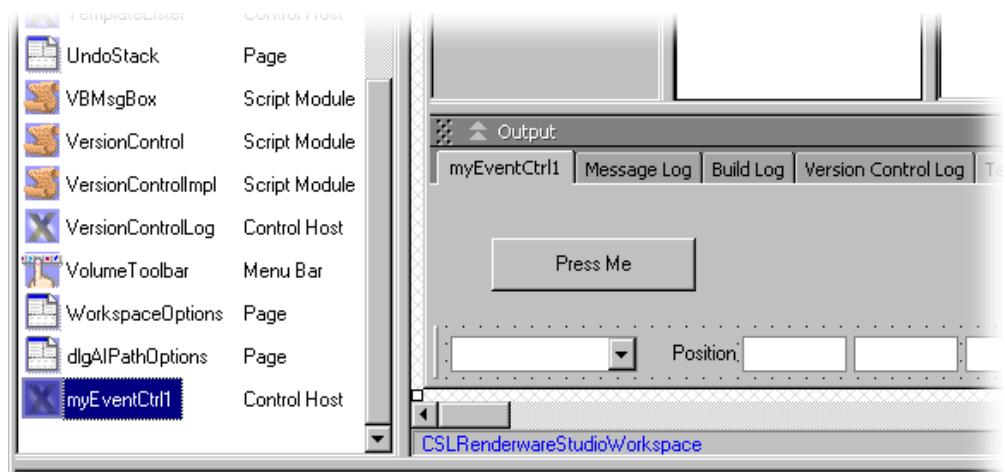


This action creates a project file that we can customize without the worry of damaging the Workspace application. We'll start that process now by adding our *myEvent* ActiveX control to the Objects window.

4. Select **Create ▶ Control Window...** to display the same dialog that we used in the [ActiveX Control Test Container](#) (p.483), and again choose **myEventCtrl Class**.

Our control appears at the bottom of the list in the Objects window. (In fact, this window shows *instances* of controls, so the new addition is called *myEventCtrl1*.)

5. Drag *myEventCtrl1* from the Objects window to your choice of location in the schematic layout of the Workspace window:



6. Save the project, and then run the application using **Run ▶ Start**.

Workspace will start up with the new control wherever you chose to place it. Once you've confirmed that this is the case, we can proceed to the next stage.

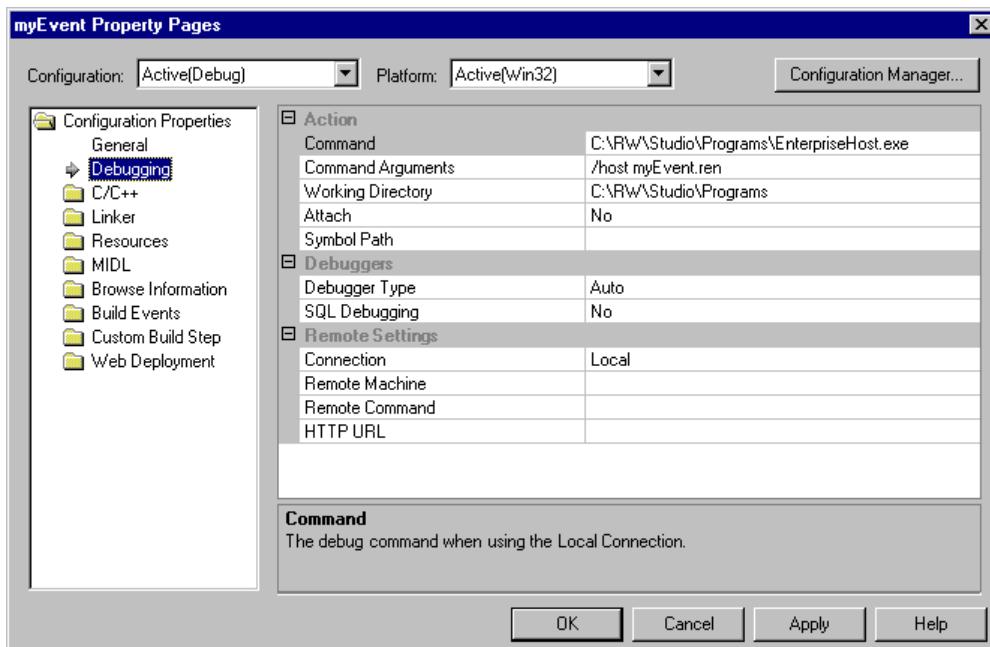
## Lesson 2. Debugging controls in Workspace

### In this lesson

You set up Workspace as the application in which to debug your new ActiveX control.

When you were [testing the RenderWare Studio ActiveX control wizard](#) (p.483), we said that it was possible to use Workspace as the host application for debugging purposes. You're going to set that up now.

1. In Visual C++, and with your myEvent project open, select **Project ▶ myEvent Properties...** and change the Debugging settings for the Active(Debug) configuration:



For the **Command**, specify the full path to [EnterpriseHost.exe](#) (p.432) in C:\RW\Studio\Programs.

Set the **Working Directory** to C:\RW\Studio\Programs.

Set the **Command Arguments** to be /host myEvent.ren. This tells the executable to read the .ren file you created a few moments ago.

2. Select **Debug ▶ Start** (or just press **F5**), and after a short delay you should see Workspace start up using your newly created layout.
3. Click **Press Me**.

If you still have the debug breakpoint set up in your Visual C++ project, execution should halt at the same point as before.

4. Click **Stop Debugging** or press **Shift+F5** to return to Visual C++.

You're ready to begin your final task. You want your ActiveX control to communicate with others in the Workspace interface.

# Lesson 3. Firing events from an ActiveX control

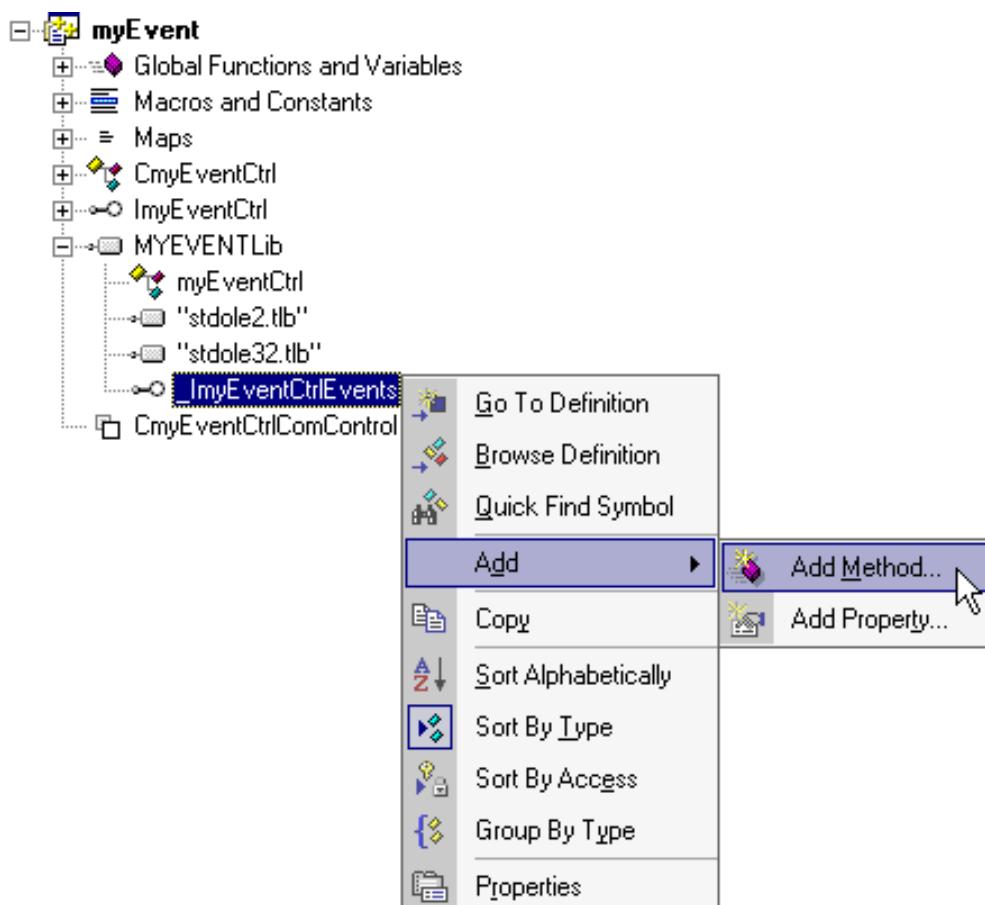
## In this lesson

You create an event that your control will fire, to be handled by Workspace.

ActiveX controls in Workspace are normally concerned with representing some values, or providing a GUI with which a designer can interact. Though possible, it's unlikely that the controls you develop will communicate with other controls directly. More commonly, communication between controls is handled using script code.

In addition to the methods and properties that they expose, ActiveX controls can also fire events. The model in Workspace is that these events are intercepted by VBScript code, which then changes properties or calls methods on other controls. Your next task is to make our `myEventCtrl` control fire such events.

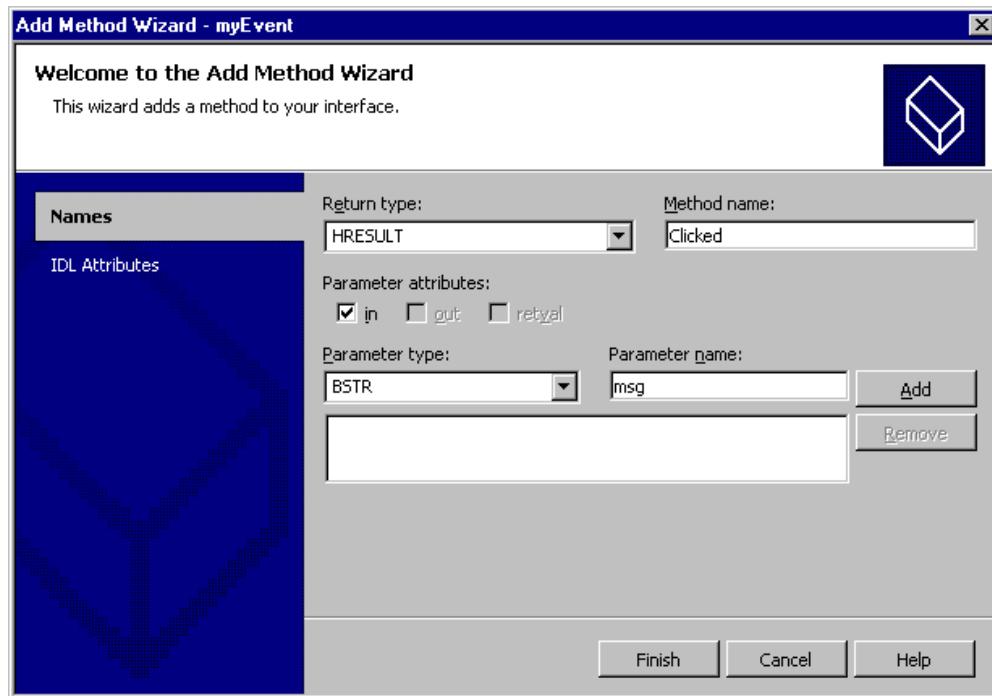
1. In Visual C++, go to Class View, right-click the `_ImyEventCtrlEvents` interface, and select **Add ▶ Add Method...**:



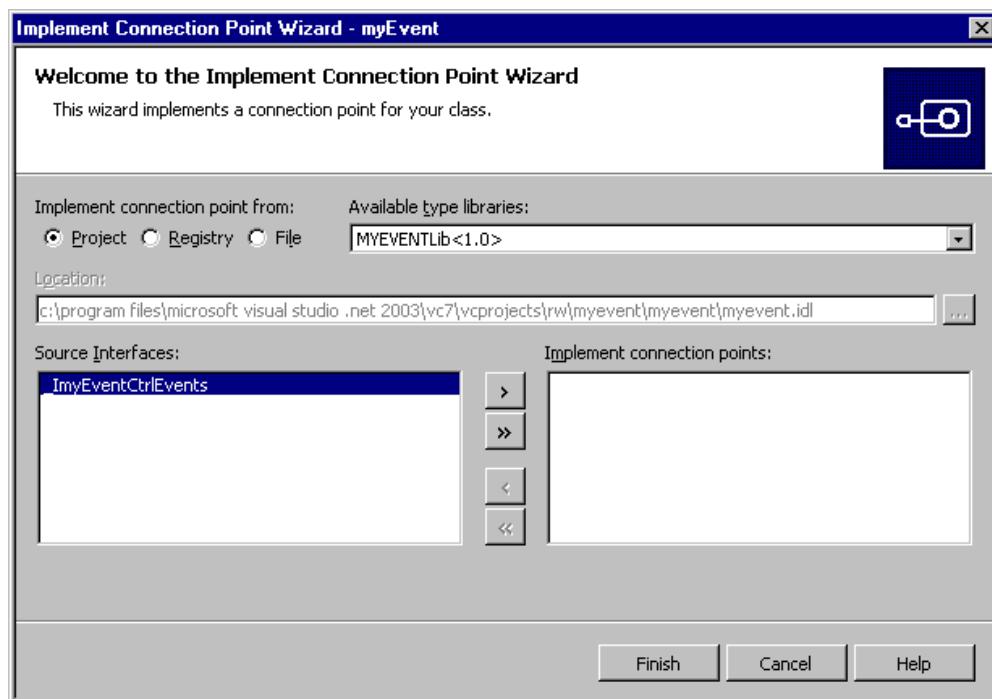
This COM interface contains definitions of the events that your custom ActiveX control will fire. You're going to create an event that's fired when **Press Me** is clicked. The event will contain a string for any control that handles your event to display.

2. In the **Add Method Wizard** dialog, enter `Clicked` as the **Method name**, `BSTR` as the **Parameter type**, and `msg` as the **Parameter name**.

Check the `in` box under **Parameter attributes**, then click **Add**, and then **Finish** to have the new method's definition added to the `.idl` file for the control.



3. Switch to Solution Explorer and double-click `myEvent.idl`. Towards the bottom of this file, you should see the new event listed under the `dispinterface` section.
4. Right-click `myEvent.idl` in Solution Explorer, and select **Compile**. This will generate a type library file.
5. Return to Class View, right-click `CmyEventCtrl`, and select **Add ▶ Add Connection Point...** This displays the following dialog:



6. Press the **>** button to add `_ImyEventCtrlEvents` to the **Implement connection points** box, and then click **Finish**.

You've created a file called `_ImyEventCtrlEvents_CP.H` that contains the definition of a new class, `CProxy_ImyEventCtrlEvents`. This class implements a method named `Fire_Clicked()` that can be called to fire your new event.

7. Check that the project still builds successfully.

**Note:** As the project stands, this step will fail due to a problem with the wizard. In `myEventCtrl.h`, there's a faulty `#include` directive for a file named `$(InputName)_h.h`. Remove that line, and the project will compile correctly.

8. In `myEventCtrl.cpp`, add a line to the handler that's invoked when the button is clicked, causing your ActiveX control to fire its `Clicked` event:

```
LRESULT CmyEventCtrl::OnBnClickedButton1(WORD, WORD, HWND,
BOOL&)
{
    Fire_Clicked(_T("Hello World"));
    return 0;
}
```

This causes the `Clicked` event to be fired when **Press Me** is clicked, which is exactly what you want to happen.

9. Compile the project one last time, and close Visual C++.

# Lesson 4. Handling ActiveX control events in Workspace

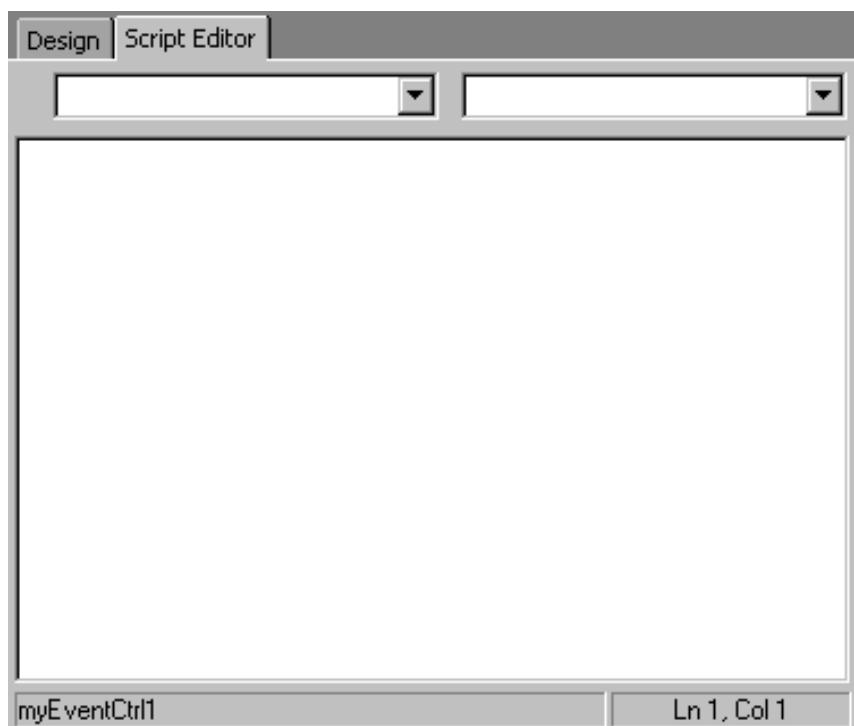
---

## In this lesson

You add script code to Workspace that handles the event you fired in the previous lesson.

As described in previous lessons, we need to add some script code to the application definition files in Enterprise Author to handle the event that our control generates.

1. Start Enterprise Author, load `myEvent.ren`, and double-click on `myEventCtrl1` in the Objects window. The control will appear in the Design window.
2. Select the Script Editor tab to display a text area where we can write some VBScript code that will be associated with our control.



The two drop-down lists at the top of the text area contain, respectively, the objects and events that `myEventCtrl1` exposes.

3. In the left-hand box, select **myEventCtrl1**. In the right-hand box, select **Clicked**.

When you complete the second operation, Enterprise Author will generate a subroutine to handle the `Clicked` event. Make it look like the following:

```
Sub myEventCtrl1_Clicked(msg)
    MsgBox msg
End Sub
```

- Save and then run the project. This time, when you click the button, you should see a message box appear. The text in this message box is the string we wrote in Visual C++.

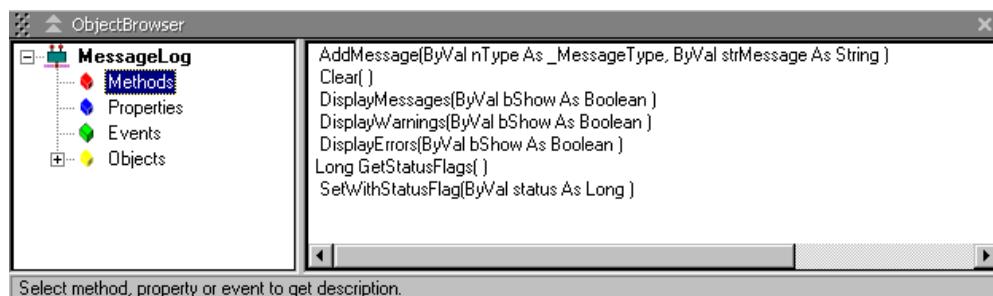


You've now seen how to get an ActiveX control to interact with Workspace. The next step is to use VBScript to talk to *other* controls. In the next example, we'll pass the string we receive from our control to Workspace's Message Log window, which is implemented by the *MessageLog* object.

- Select *MessageLog* in Enterprise Author's Objects window, and then look at the Object Browser window.

You'll see a tree view with branches for the methods, properties, events, and objects exposed by *MessageLog*.

- Select **Methods**, and a list of all the methods that *MessageLog* exposes appears in the right-hand pane. The first of these is *AddMessage()*, which tells us that we can insert our own messages into the log.



- Go back to the Script Editor window, select *myEventCtrl1* in the Objects window, and add the following:

```
Sub myEventCtrl1_Clicked(msg)
    MsgBox msg
    MessageLog.AddMessage 0, "myEventCtrl1 said " & msg
End Sub
```

This new line of VBScript code uses the *AddMessage()* method to append a string to the Message Log window.

- Run the application. When you click the button on your control, you should find that after you close the message box that appears, the Message Log window will report our message too.



You can now exit from Workspace; this tutorial is complete.

# Closer interaction with an ActiveX control

---

This tutorial builds on the ones about creating custom ActiveX controls to work with Workspace [earlier in this section](#) (p.481). In it, you improve the ActiveX control produced earlier, enabling it to get data from and send data to RenderWare Studio Workspace and the game database:

- The control will display the name of any entity selected in Workspace's Design View window.
- The user will be able to select an entity in Design View by entering its name into the control.

## Who is it for?

This tutorial is aimed at developers who need to understand more about RenderWare Studio's mechanisms for processing game data. It demonstrates how to implement handlers for the events that Workspace fires to its hosted windows.

Before reading this tutorial, you should have completed the two preceding tutorials in this part of the documentation.

## Tutorial structure

There are just two lessons in this tutorial. In the first, you add an edit control to the custom ActiveX control that can display the name of the entity selected in Workspace's Design View window. In the second, you hook up the edit control to Workspace's user interface, and add some more code that makes the interaction a two-way process.

### [Creating and coding the edit control](#) (p.501)

You write code in an ActiveX control that Workspace calls as part of its automatic processing, to be executed when entities are selected and deselected.

### [Deeper integration with RenderWare Studio](#) (p.504)

You attach VBScript code to an ActiveX control that hooks it up to Workspace's messaging architecture, and then write additional C++ code that interacts with RenderWare Studio's game database.

# Lesson 1. Creating and coding the edit control

---

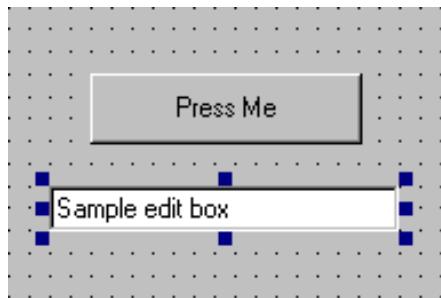
## In this lesson

You write code in an ActiveX control that Workspace calls as part of its automatic processing, to be executed when entities are selected and deselected.

## Adding an edit box

The first thing you're going to add to your control is an edit box for entering and displaying entity names.

1. With your ActiveX control project open in Visual Studio .NET 2003, go to the Resource View and double-click the `IDD_DIALOG` dialog resource to open it for editing. You'll see the dialog for the ActiveX control, containing the button created in the previous tutorial.
2. Use the Toolbox to add an **Edit Control** to the dialog, like this:



3. Use the Properties window to change the ID of the new edit control from its default value to `IDC_ENTITY_EDIT`.

You'll use this more memorable identifier to refer to the edit control in the code you write to implement your ActiveX control.

If you test the control in Workspace, you'll see that it now contains the new edit control, though it doesn't yet do anything of use.

## Writing code to integrate with Workspace

Having your control react to entities being selected in Design View means writing implementations for three of the skeleton methods created by the wizard:

- When an entity is selected (alone, or into a group), `OnAddSelection()` is called
- When an entity is deselected (alone, or from a group), `OnRemoveSelection()` is called
- When all selected entities are deselected (using **Esc**, for example), `OnClearSelection()` is called

You need to arrange for the name of the current entity to be written into the

edit control in `OnAddSelection()`, and for the edit control to be cleared when either of the other two methods is called. All three methods are defined in `myEventCtrl.cpp`.

4. In Solution Explorer, open `myEventCtrl.cpp` and locate `OnAddSelection()`. You're going to add code to this method that looks up the given entity ID in the database, gets its name, and then enters that name into your edit control. Here's the code:

```
STDMETHODIMP CmyEventCtrl::OnAddSelection(long ID)
{
    if (m_DatabaseID && ID)
    {
        // Get the RWSEntity structure of the selected entity
        RWSID EntityID = ID;
        RWSEntity EntityData = {0};
        RWSEntityGet (EntityID, &EntityData);

        // We need to ensure that data was returned correctly
        if (RWSER.NoError == RWSER.ErrorGetLast())
        {
            CWindow EntityEdit(GetDlgItem(IDC_ENTITY_EDIT));
            _TCHAR* szText = EntityData.Name ? EntityData.Name : _T("Unnamed");
            EntityEdit.SetWindowText(szText);

            // We are responsible for freeing data returned by
            the RWS Manager
            RWSEntityFreeData(&EntityData);
        }
    }
    return S_OK;
}
```

This code starts by checking that the control has been properly initialized into Workspace, and that the entity ID passed as a parameter is non-zero. Then, as the comments explain, it calls some RenderWare Studio library functions to discover the name of the entity with the given ID, and then displays it in the edit control.

5. Replace the implementation of the `OnRemoveSelection()` method with:

```
STDMETHODIMP CmyEventCtrl::OnRemoveSelection(long ID)
{
    // Clear the contents of the edit window
    CWindow EntityEdit(GetDlgItem(IDC_ENTITY_EDIT));
    EntityEdit.SetWindowText(_T(" "));
    return S_OK;
}
```

This method is much simpler. It just responds to the entity being deselected by clearing the edit control. We have the option of using the `ID` parameter to do something more specific with the particular control being deselected, but that's not our intention here.

6. Replace the implementation of the `OnClearSelection()` method with:

```
STDMETHODIMP CmyEventCtrl::OnClearSelection()
{
    // Clear the contents of the edit window
    CWindow EntityEdit(GetDlgItem(IDC_ENTITY_EDIT));
    EntityEdit.SetWindowText(_T(" "));
    return S_OK;
}
```

Because of the simple requirements, this implementation is the same as the one for `OnRemoveSelection()`. Deselecting one entity and clearing all selections are handled in identical fashion.

Your ActiveX control is now set up to handle entity selections, so it can be built and re-registered. Before it will function, however, you must install it fully into Workspace.

# Lesson 2. Deeper integration with RenderWare Studio

---

## In this lesson

You attach VBScript code to an ActiveX control that hooks it up to Workspace's messaging architecture, and then write additional C++ code that interacts with RenderWare Studio's game database.

Getting Workspace to recognize and communicate with your new ActiveX control requires you to follow some standard procedures. You need to add code to the control's script module that:

- Attaches the control to the game database.
- Allows the control to respond to entities being selected and deselected.

## Attaching the ActiveX control to the game database

In the previous tutorial you added the control to Workspace, but now you must also attach it to the game database. This is necessary if you wish to access the database in any way in your control.

`RWSEntityGet()` is called from the `OnAddSelection()` function, which requires access to the game database. The control has been provided with a function, `AttachToDatabase()`, to connect it to the database. This function must be called at the same time as Workspace's default objects perform this operation.

1. Start Enterprise Author. Select **File ▶ Open** and open the `myEvent.ren` file.
- Workspace's default objects connect to the game database through script in [the GlobalScript module](#) (p.453), in a handler for the `RWSUtils` object's `OnDatabaseAttached` event. You can handle the same event in your control's script module.
2. In the Objects window, locate and double-click the `myEventCtrl1` script module.
  3. Switch to the application's Script Editor tab, and add the following code:

```
Sub RWSUtils_OnDatabaseAttached(DatabaseID)
    If Not myEventCtrl1.AttachToDatabase(DatabaseID) Then
        GlobalScript.DatabaseAttachedError "myEvent"
    End If
End Sub
```

The call to your control's `AttachToDatabase()` method here attaches it to RenderWare Studio's game database. Should the call fail, the global `DatabaseAttachedError()` method is called.

Before you can continue, you must also arrange to disconnect your control from the database at the appropriate time. The event to listen out for is `OnDatabaseDetached`.

4. Add this handler to your *myEventCtrl1* object's script module, immediately after the last one:

```
Sub RWSUtils_OnDatabaseDetached()
    myEventCtrl1.AttachToDatabase(0)
End Sub
```

Passing a zero to the control's `AttachToDatabase()` method makes it detach itself from the game database.

## Hooking up with the selection mechanism

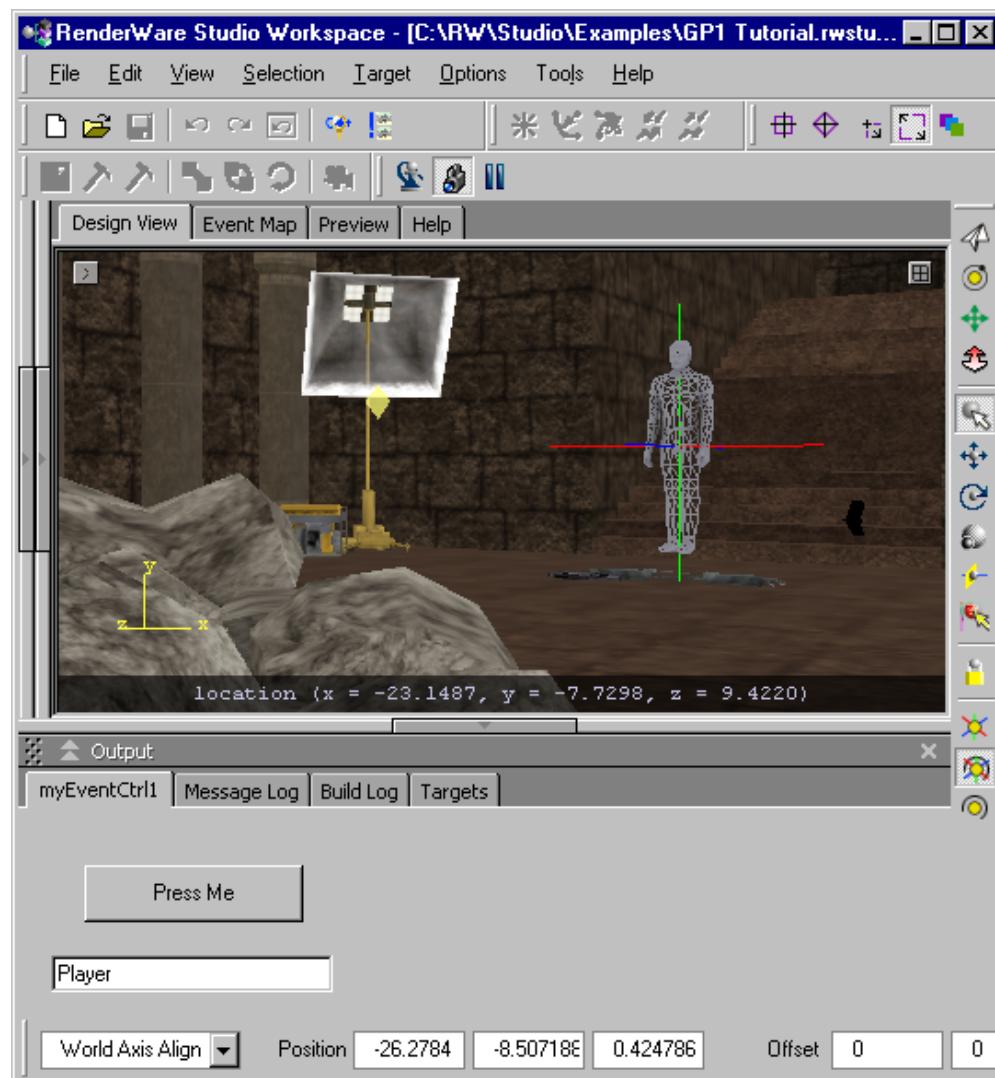
You also need to hook up your control with Workspace's [selection mechanism](#) (p.447), so that when entities are selected and deselected in Design View, the code you've added to your ActiveX control is called. This involves providing a special value to the control's `SelectionIdentifier` property.

5. Add the following procedure to the script module:

```
Sub OnLoad()
    myEventCtrl1.SelectionIdentifier =
    GlobalScript.g_strGlobalSelection
End Sub
```

The `SelectionIdentifier` property was added to your control by the wizard. Assigning the global value `g_strGlobalSelection` to it turns the control into a sink for events from RenderWare Studio's `ISelectionEvents` interface. It's this connection that causes the `OnAddSelection()`, `OnRemoveSelection()`, and `OnClearSelection()` methods to be called appropriately.

6. Save your changes to the `.ren` file, and start Workspace using the custom interface. If you now select an entity in Design View, you should see its name displayed in the new edit control.



## Selecting entities in Workspace

You're now going to extend the ActiveX control so that the user can enter the name of an entity, and have that entity selected when the button is clicked. The control is already set up to connect to the game database and hook into the selection mechanism, so all that's necessary is some modification of the `OnBnClickedButton1()` method.

7. Replace the current implementation of `OnBnClickedButton1()` with the following:

```
LRESULT CmyEventCtrl::OnBnClickedButton1(WORD, WORD, HWND,
BOOL&)
{
    // Get handle to edit control
    CWindow EntityEdit = GetDlgItem(IDC_ENTITY_EDIT);

    // String to hold edit control text
    CComBSTR EntityName;

    // Put window text in string
    EntityEdit.GetWindowText(&EntityName);

    // Ensure that we're connected to the database
}
```

```

if(m_DatabaseID)
{
    // Structure to hold entity data
    RWSEntity EntityData = {0};

    // Get the ID of the first entity in the database
    RWSID EntityID = RWSGetFirst(RWSEntityID);

    while(EntityID)
    {
        // Get the RWSEntity structure of the selected entity
        RWSEntityGet(EntityID, &EntityData);

        // We need to ensure that data was returned correctly
        if (RWSENoError == RWSErrorGetLast())
        {
            // Compare the entity's name with the name entered
            if(EntityName == EntityData.Name)
            {
                // Select the entity in Workspace
                m_pSelection->Add(EntityID);
            }

            // Get the next entity in the database
            EntityID = RWSGetNext(EntityID);

            // We are responsible for freeing data returned by
            // the RWS Manager
            RWSEntityFreeData(&EntityData);
        }
    }

    return 0;
}

```

The comments explain the code in detail, but here's a simplified summary:

- Get the text from the edit control
  - Loop through the entities in the game database, comparing their names with the entered text
  - If a match is found, select that entity in Workspace
8. Rebuild the control and start Workspace using the custom interface. Choose the name of an entity in the Game Explorer window, and then type that name into the edit control.
  9. Click **Press Me**. Your chosen entity should now be selected.

This is the end of the tutorial.

# Opening arbitrary asset types for editing

---

When you right-click an asset in Workspace's [Assets window](#) (p.264), the context menu contains an **Edit** item that opens the asset for editing in a suitable application. By default, Windows makes the decision about which application should be used, but you can modify Workspace's source files with Enterprise Author to change this behavior and specify the application of your choice.

## Default edit handling

The **Edit** menu item for assets is handled by a method called `OnAssetEdit()` in the *ContextMenu* menu bar object's script module (`ContextMenu.vbs`). When **Edit** is selected, the method checks the type of the asset against a handful of types that require special processing. If it doesn't match any of these, default processing takes over:

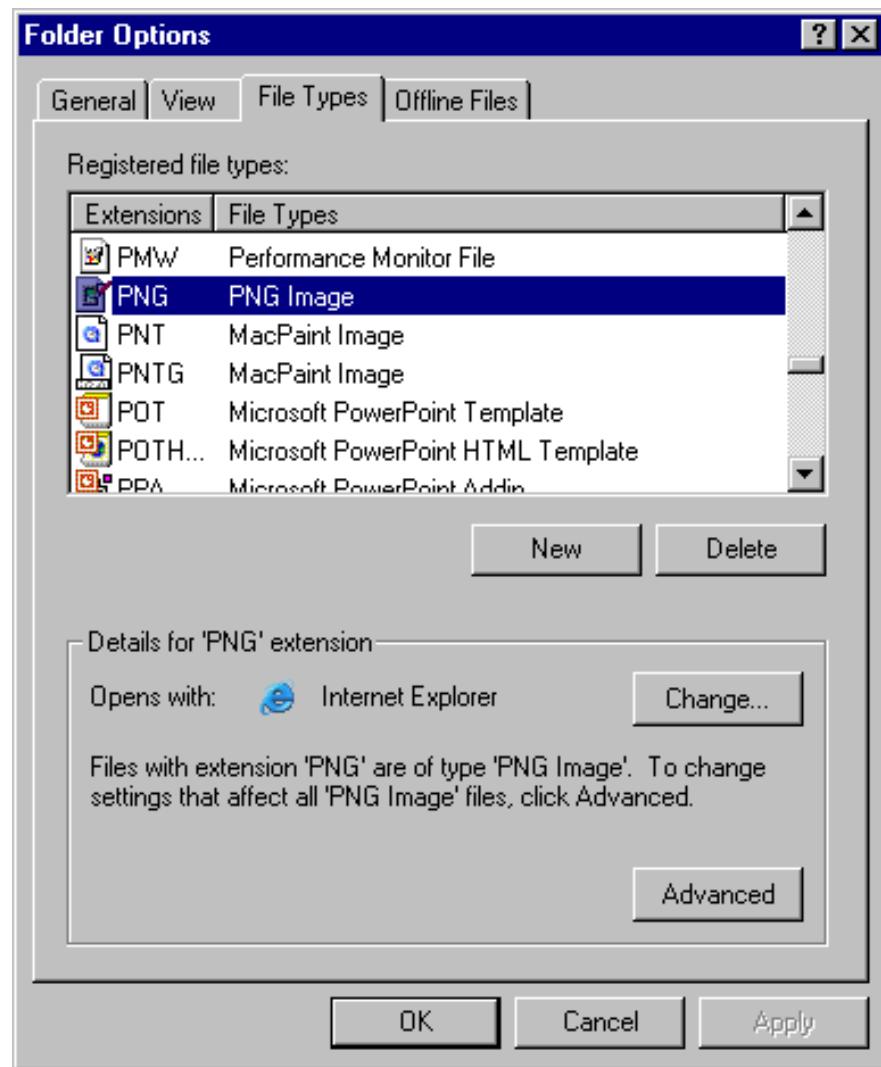
```
Sub OnAssetEdit
    ...
    Dim bHandled
    bHandled = False
    Broadcast.EditAsset Asset, bHandled
    If bHandled = False Then
        ' Use editor defined by Windows file extension association
        Dim Shell
        Set Shell = CreateObject("Shell.Application")
        Shell.ShellExecute strFileName, , , "edit"
    End If
    ...
End Sub
```

First, `OnAssetEdit()` [broadcasts](#) (p.450) a call to `EditAsset()`, giving the other script modules in Workspace the chance to edit the asset. (If you were to write an asset-editing [ActiveX control](#) (p.477) for Workspace, you could implement `EditAsset()` yourself to handle this call.) If the broadcast receives no replies, processing falls back to the `ShellExecute()` method.

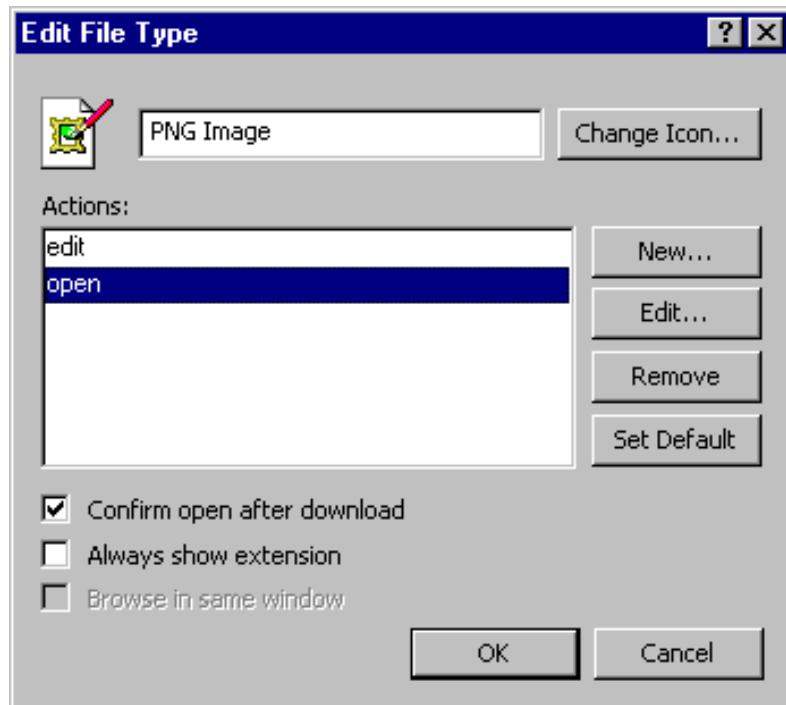
## Configuring `ShellExecute`

Windows associates file types (as indicated by their extensions) with the application that should be used to deal with them. `ShellExecute()` takes advantage of this association: when you call it, passing only the name of the file you're interested in, that file opens in its associated application. Calling "`ShellExecute new_image.png`", for example, would open the file `new_image.png` in the application configured for doing so—quite possibly, Internet Explorer.

You can set up the associations between file types and applications through Windows Explorer. Select **Tools** ▶ **Folder Options** ▶ **File Types**, and you'll see this dialog:



By default, files are associated with a single application, but you can specify that different applications should be used in different circumstances. Click the **Advanced** button above, and a smaller dialog is displayed:



The **Actions** list names different operations that may be performed on files of a particular type. For many files, only *open* is specified, but the call to `ShellExecute()` from `OnEditAsset()` relies on the existence of another operation, *edit*:

```
Shell.ShellExecute strFileName, , , "edit"
```

If the file type of the asset in question has an *edit* entry in its **Actions** list, the application referenced by that entry is launched when you select the **Edit** menu item. If there's no such entry, you'll see a dialog like this:



Workspace's use of this standard Windows mechanism is especially convenient when different Studio users need to use different applications. Where an artist might want to launch Adobe Photoshop to edit a .bmp file, a designer might choose simply to inspect the file with a more lightweight application. You just need to set up the *edit* action appropriately, on a per-user basis.

## Launching a particular application for editing

If you don't want to rely on Windows handling, the alternative is to add your file type to the list in `OnAssetEdit()` that gets processed before default handling takes over. To cause .bmp files always to be edited in Microsoft Paint, for example, you could write VBScript code like this:

```
Sub OnAssetEdit
  ...
  If bCanEdit Then
```

```
If Asset.Type = "rwID_SPLINE" Then
    If RWSSpline.Edit(Asset) Then
        ...
    End If

    ElseIf SequencerTools.IsSequence(Asset) Then
        ...
    ElseIf Asset.Type = "rwaiID_SCRIPT" Then
        ...
    ElseIf Asset.Type = "BMP" Then

        ' Set up shell object
        Dim wshShell
        Set wshShell = CreateObject("WScript.Shell")

        ' Send file name as argument to specific program
        wshShell.Run "mspaint.exe " & strFileName
    Else
        ...
    End If
End If
...
End Sub
```

The new `ElseIf` clause uses the Windows Script Host Shell object to launch Microsoft Paint, passing the name of the selected file as an additional argument.

# Adding to the RenderWare Studio settings file

---

To add or read items in the RenderWare Studio settings file, `RwStudio.settings`, objects' script modules need to implement two methods. At appropriate moments (application startup and shutdown, for example), these methods are invoked by calls that are broadcast to every script from the `BroadcastEvents` script module. Their names are:

```
Broadcast_SaveApplicationSettings()
Broadcast_LoadApplicationSettings()
```

The act of loading or saving data from or to the settings file involves reading or writing data in XML elements within `RwStudio.settings`. The example below demonstrates the kinds of operation that are generally involved.

## Worked example

These sample implementations of `Broadcast_SaveApplicationSettings()` and `Broadcast_LoadApplicationSettings()` demonstrate how to record the current date to the settings file, and how to retrieve it. (This information could represent the date and time of the last build, for example.)

```
<OBJECT>
  ...
<RWSScriptedSettings>
  ...
<Custom>
  <Test>
    <Date>Date</Date>
  </Test>
</Custom>
</RWSScriptedSettings>
</OBJECT>
```

### `Broadcast_SaveApplicationSettings()`

`Broadcast_SaveApplicationSettings()` is passed an XML document object representing the `RwStudio.settings` file, and an XML node object representing the `<RWSScriptedSettings>` element. To insert `<Custom>` and its children into the document, the method needs to do the following:

1. Check whether the `<Custom>` element exists; create and add it if it doesn't.
2. Check whether the `<Test>` element exists; create and add it if it doesn't.
3. Remove any child elements from the `<Test>` element.
4. Create and add the `<Date>` element and the data it contains as a child of the `<Test>` element.

```
Sub Broadcast_SaveApplicationSettings(XMLDoc, XMLRootNode)
  Dim CustomElement, TestElement, DateElement, ChildElement

  ' Create the root custom element if it isn't already in the
  ' file
```

```

Set CustomElement = XMLRootNode.SelectSingleNode("./Custom")
If CustomElement Is Nothing Then
    Set CustomElement = XMLDoc.CreateElement("Custom")
    XMLRootNode.AppendChild CustomElement
End If

' Check we have a Test element to add our settings to
Set TestElement = CustomElement.SelectSingleNode("./Test")
If TestElement Is Nothing Then
    Set TestElement = XMLDoc.CreateElement("Test")
    CustomElement.AppendChild TestElement
End If

' Remove all child nodes
For Each ChildElement In TestElement.ChildNodes
    TestElement.RemoveChild ChildElement
Next

' Add in new child nodes
Set DateElement = XMLDoc.CreateElement("Date")
DateElement.Text = Date
TestElement.AppendChild DateElement
End Sub

```

## **Broadcast\_LoadApplicationSettings()**

`Broadcast_LoadApplicationSettings()` is passed an object representing the `<RWSScriptedSettings>` element. This implementation simply verifies that the `<Date>` element exists, and then sends its contents to a message box.

```

' Handle broadcast message to load settings
Sub Broadcast_LoadApplicationSettings(XMLRootNode)
    Dim DateElement, strSettingsFileDialog

    Set DateElement =
XMLRootNode.SelectSingleNode("./Custom/Test/Date")
    If Not DateElement Is Nothing Then
        strSettingsFileDialog = DateElement.text
        msgbox "The date in the settings file is: " &
strSettingsFileDialog
    End If
End Sub

```

## **The RenderWare Studio settings file in practice**

In general, script modules store their settings in similarly named child elements of `<RWSScriptedSettings>`. (`Layouts.vbs`, for example, stores information in a `<Layouts>` element.) This is more than just convenience: script modules have unique names within an application, so using these names guarantees safety from naming collisions.

When you add custom data to the `RwStudio.settings` file, you don't have to put it beneath `<RWSScriptedSettings>`. If you wish, you can leave the structures used by RenderWare Studio alone, and use XML DOM manipulation to create and use a new element that's a child of the file's root `<OBJECT>` element instead.

## Examples

---

RenderWare Studio supplies several example projects that demonstrate how to use different behaviors and other features.

These projects are stored in the `Examples` folder. If you installed RenderWare Studio in the default location, then the path of the `Examples` folder is:

`C:\RW\Studio\Examples`

To view an example, open its project file (`.rwstudio`) in the Workspace. For information on each example, browse to the corresponding example topic by expanding the “Examples” heading highlighted in the “Contents” pane opposite.

# Alpha sort

---

This example shows how FPSRender and CRpLevel have been extended to support alpha sorting of atomics.

The example game contains three levels:

## **Alpha sort using a single world**

Shows a single FPSRender rendering an RpWorld containing both opaque entities and translucent entities. The FPSRender behavior has World Render Back To Front and Depth Sorting of atomics enabled; this means that the world sectors are rendered from furthest to nearest and that all of the atomics attached to the world are depth-sorted.

## **Alpha sort using opaque and translucent worlds**

Shows a scene consisting of two worlds. Due to the order of the objects and worlds in the hierarchy view in the game database, some of the atomics are added to the first world, and the rest are added to the second world.

When connecting to the target, the Opaque World is created first and sets the Global Opaque World pointer. Then the objects in the Opaque entity folder are created and added to the Global Opaque World. This first world is drawn front to back with no depth sorting and contains the opaque objects.

Next, the Translucent (or Alpha) world is created which sets the Global Translucent World pointer (this is now the active world). Any entities following this are added to the translucent world; as in the first game level folder, this world is depth-sorted.

## **Alpha sort using opaque and translucent worlds (atomics auto-select which world, using CSetCRpLevelHint)**

In this level, a behaviour is used to attach a hint to the assets (see the entity folder Set CRpLevel Hints). Opaque assets have the hint set to HINT\_OPAQUE\_WORLD; translucent assets have the hint set to HINT\_TRANSLUCENT\_WORLD.

When connecting to the console, the hints are run first, then the two worlds are created. Because the hints have been set, any behaviors using the assets that have a hint set know which world to add the asset to, to render it (that is, either the opaque world or the translucent world).

# Animation

---

This example demonstrates how to use the CMultiAnimate behavior to add animations to an entity.

In this example, we have attached the CMultiAnimate behavior to the Alien entity.

To add an animation to an entity:

1. Drag the CMultiAnimate behavior to the entity.
2. Add the animation to the entity:
  - a. Change from the Game Explorer window to the Assets window.
  - b. Right-click the folder named “Alien Animation” and select **New ▶ Asset as child**.
  - c. Right-click the new asset and select **Properties**.
  - d. In the **File** box, type the path of the animation (.anm) file.

(If you have specified a valid animation file, then the Workspace automatically sets the asset **Type** to rwID\_HANIMANIMATION.)

As shown in this example, if you want to add several animations to an entity, then, rather than adding the animation assets one at a time, it is a good idea to place them in a new asset folder, and then add the asset folder (in this example, Alien Animation) to the entity.

In this example, the Time Trigger entity (attached to the FPSTimer behavior) sends an event to the CMultiAnimate behavior. When this event is triggered, the CMultiAnimate behavior plays the next animation in the asset folder.

# Area triggers

---

RenderWare Studio supplies three simple area trigger behaviors that send an event when an entity enters the behavior area:

## **ATBox**

Defines a box-shaped area.

## **ATCylinder**

Defines a cylindrical area.

## **ATSphere**

Defines a spherical area.

## Area triggers in the example project

The example project contains two area trigger entities:

### **Warning Light Trigger** (surrounding the alien monster model)

Uses ATCylinder to trigger a pulsing “warning light” when the player enters the area.

### **Firework Trigger** (at the far end of the scene)

Uses ATBox to trigger a particle generator when the player enters the area.

## Related events and messages

The area triggers interact with entities via the Game Framework event system.

The events used directly by the area triggers in this example are:

### **iMsgRunningTick**

Each time FPSPlayer receives this message, it sends an area trigger query message to test for collisions with area triggers.

### **trigger\_test**

This is the area trigger query message sent out by the FPSPlayer. Both area triggers also link to this, to respond to the query.

### **warning\_light\_on**

Message sent by the Warning Light Trigger entity when the player enters the ATCylinder area. This turns on the pulsing warning light.

### **warning\_light\_off**

Message sent by the Warning Light Trigger entity when the player leaves the ATCylinder area. This turns off the pulsing warning light.

### **firework**

Sent by the Firework Trigger entity once a frame for each frame that the player is within the ATBox area. Each time the message is sent, the firework particle generator generates a particle.

## Using the Workspace to define an area trigger

The supplied area trigger behaviors use simple equations to define their shapes, so they do not need separate geometry assets (such as RpClump or RpAtomic).

However, to help you define area triggers in the Workspace, RenderWare Studio supplies “dummy” primary asset files that match these area shapes:

`ATrig_UnitBox.dff`, `ATrig_UnitCyl.dff`, and `ATrig_UnitSph.dff`.

To use the Workspace to define an area trigger:

1. Create an entity using one of the dummy primary asset files (depending on the shape of the area you want).

**Note:** To avoid sending this asset to the Game Framework, uncheck all of its platform flags (right-click the asset, and then select **Properties...**).

2. Attach the area trigger behavior for that shape.
3. Move and resize the entity according to the trigger area you want.

**Tip:** In design builds of the Game Framework, you can display a wireframe version of the trigger area: in the Entity Attributes window for an area trigger entity, select the ATBase parent class; under Debug Tools, select the **Render the area** check box.

# Audio and audio groups

---

These two examples demonstrate the RenderWare Audio 2.0 integration with the RenderWare Studio Game Framework.

To run the example:

1. Start RenderWare Studio and load the **Examples Audio** or **Examples Audio Group** project.
2. Launch the Game Framework and connect to it.
3. When the Workspace has connected to the framework you should hear some ambient droning sounds.

In the audio example, somewhere in the level there is a tiger sound, which you should be able to find by just listening to the sound orientation on your speakers or headphones.

In the audio group example, on the console screen you will see the channel strip info for each sound playing in the level. Every few seconds a timer event triggers the fading down of one of the sound groups. This is useful because you can define different group within your game and fade between them easily.

The audio behaviors used for these examples are:

## **AudioGlobalMixer**

Controls the parameters for the output mixer, and controls the position of the listener.

## **AudioSound3D**

Responsible for assigning a sound to a virtual voice, and handling whether it is playing or not.

## **AudioReverb**

Controls the reverberation in a level. You can select from a list of prebuilt reverb types or create your own. You can use this behavior to enable and disable the reverbs, making it possible to have multiple reverbs available. For example, if you are creating a driving game and want the car to go through a tunnel, you can change the reverb accordingly.

## **AudioGroup** (group example only)

Defines groups of sounds which can be enabled/disabled as well as faded. To define a group, you attach the AudioGroup behavior to an object and then add wave dictionaries as assets to the group. When your AudioGroup entity receives a fade message it will only fade the sounds in its group.

For an explanation of how the audio behaviors and toolkit works, see the Game Framework Help.

# Camera look-at-point

---

This example contains one look-at-point camera, a directors camera, and a FPS player. The purpose of this example is to show how to link a look-at-point camera to another entity and make it follow that entity. It also demonstrates how to connect up the camera system, so that both the directors camera and the look-at-point camera can co-exist happily.

To see the example working, disable the directors camera.

## How the first person camera connects

Within the Input Devices folder there are a series of pad and controller behaviors. All of these send control messages to the Player view ► player FPSPlayer entity. Which one is used depends on the platform the code is running. The FPSPlayer behavior uses these control inputs to move itself around the map. More information on this behavior can be found in other examples.

Everything within the player FPSPlayer entity can be ignored, except for the messages section. This is relevant to this example. Here, the behavior is sending out a message (player\_pos) with its position (as an RwFrame). The message name is set in the Send Player Position attribute. Note also, that the Render msg (input) and Render msg (output) attributes are *not* set. The FPSPlayer behavior will render a first-person view, but here we want a look- at-point or third-person view. Therefore, the FPSPlayer behavior does not need to send any render messages, as it is not being used for rendering.

Instead, the Player view ► 3rd person camera CcameraLookAtPoint entity is being used to do this. If you open the Event Map, you can see that the 3rd Person Camara is receiving a total of three incoming messages. One is sent from the player and the other two are system messages. If you examine the 3rd Person Camera in the Attributes window, then you will see how these messages are received by the behavior.

All the attributes are in the Messages section. The player\_pos message from the player FPSPlayer entity is received via the position update attribute. This causes the location, which the 3rd person camera looks at, to be changed. In other words, it makes the camera look at the player and keeps it updated as to any changes. The system iMsgStartSystem is sent after all initialisation is complete and is used here to enable camera. Finally, the system iMsgDoRender is sent when the directors camera is off. This is attached to the Render msg (input) attribute. It causes the camera to set up the view matrix and send it out via the message in the Render msg (output) attribute. Here, the 3rd\_person\_cam message.

This message is picked up by the debug FPSRender and World CDebugTools entities, causing each to render and hence displays the game's frame and debug overlays (see below).

The DirectorsCamera CDirectorsCamera entity also sends this 3rd Person Cam message. This comes into effect if the directors camera *is* enabled. In this case, the iMsgDoRender system message is *not* sent; instead, the CDirectorsCamera behavior is activated, causing CDirectorsCamera to send the 3rd Person Cam message instead. In this case, the view matrix matches the view in the

Workspace and *not* the one from the 3rd Person Camera CCameraLookAtPoint entity.

## How do the attributes in Camera Look-at-point work?

The 3rd person camera (CCameraLookAtPoint) behavior has 4 sets of attributes to control how it operates. The basic idea is that it looks at an object that is emitting messages (giving its position) as it moves about the map. The attributes control the way in which the behavior follows and views the object.

Distance Attributes control how far back from the object the camera is. The camera will stay between the min distance and max distance values. It will aim towards being at the preferred distance. The distance smoothing controls how fast it heads towards this value. A value of 1 means always stay exactly at the preferred distance. A value of 0 means do not bother to stay at preferred distance. Values in between are faster or slower catch up when the object moves. No matter what this value is, the camera will not go outside the min/max distance values.

Elevation Attributes work in the same way; they just control the distance above the object being followed instead.

Angle Attributes, again, work the same way. This time, controlling the angle around the object being followed. There is a slight complication here, in that the ranges for min and max angle appear a little odd. The preferred value is as normal, -180 degrees to +180 degrees. That is from front via one side, to back, to front via the other side. The min and max values go beyond this range to allow for up to 180 degrees of variance at all the preferred angles.

As an example, if a preferred angle of 0 degrees was desired, with + or - 90 degrees of freedom, you would set preferred = 0, min = -90 and max = +90. However, if you want the camera at the side, you would need, preferred = 90, min = 0, max = 180. If the camera was to be in front, you would need preferred = 180, min = 90 and max = 270. The latter indicates the need for the larger ranges. If you wanted + or - 180 degrees of freedom, then value of preferred = 180, min = 0, max = 360, would be needed. The behavior guarantees that if the maximum angle is beyond 180 degrees that the camera will not flip from 180 to minus 179 degrees as it goes over the 180/-180 degree threshold. That is the camera will wind in one direction and unwind in the other.

## What are the text, blue bubble and colored triangles for?

These are debug displays. The blue bubble shows the collision object used by the object being followed. The triangles show the faces being collided with. Green for floors and red for walls.

The text is showing messages (events) being sent by one object to another. Dashed lines indicate system messages.

You can disable these by choosing the Environment ► debug entity in the game database and changing its attributes. Un-checking Visualize Events and Visualize Collisions will remove both debug displays.

# Dynamic light

---

This example demonstrates the use of the following light behaviors in the FX module:

CFXColorLight  
CFXLight\_2Stage  
CFXLight\_Spline  
CFXSpinLight  
CFXWaveLight

To run the example:

1. Open the Example Dynamic Lights project.
2. Launch and connect to a console target.
3. Either move around the world using the Workspace controls, or disable the directors camera and use the platform-specific controls to move the player around the map.

There are five lights in the example:

**CFXColorLight**

Mid-grey light, of type rpAMBIENTLIGHT.

**CFXLight\_2Stage**

Pulsing yellow light, of type rpPOINTLIGHT.

**CFXLight\_Spline**

Pulses between white / green / blue slowly, of type rpDIRECTIONALLIGHT

**CFXSpinLight**

Bright white, rotating around like a lighthouse light, of type rpSPOTLIGHT

**CFXWaveLight**

Blue, flickering rapidly, of type rpPOINTLIGHT.

In the Game Explorer window, the Templates folder contains some more pre-made light entities that you can add to the scene to try out more effects if needed.

**Note:** Each light can be of any of the RenderWare RpLight types; the types used in this example were chosen to make it easier to pick out each individual light entity.

# Environment map

---

This example demonstrates how to apply the environment map behavior to an atomic or clump.

To run the example:

1. Open the Example Env Map project.
2. Launch and connect to a target.
3. Ensure the camera is looking at the group of entities containing a teapot, a capsule, and a torus (placed near the middle of the level). These entities should all be environment mapped. The torus and the Capsule will contain an animated environment map while the teapot will contain a static environment map.

## Overview of the CFXEnvironmentMap behavior

This behavior provides a simple and easy way to control the look of RenderWare materials that are environment mapped. To use the environment map behavior on an atomic or clump, the materials used by the clump or atomic must contain an environment map (for details on setting up a material to contain an environment map, see the *RenderWare Artist Guide*).

The orientation of the environment map can be controlled in four different ways. The behavior has a drop down list of modes, which can be used to change the way source matrix for the environment map is generated:

### **SOURCE\_NONE**

The environment map is static.

### **SOURCE\_CAMERA**

The environment maps orientation is based on the direction of the camera. A good place to use this mode would be chrome on a car window.

### **SOURCE\_ANIMATED**

The orientation of the environment map is updated each frame by applying the transformation specified by X, Y, Z rotation attributes in the animated separator. This could be used on pickups; for example, a nice effect can be created by the environment map rotating over the pickup.

### **SOURCE\_USER**

The orientation of the environment map is set by the rotations specified by the attributes in the user separator. For example, this could be used to ensure a bright spot on the environment map points to a light.

## First-person player (FPS)

---

This example shows the operation of the FPS player and associated FPS behaviors.

After connection to target, switch off the director's camera. You will not have control of the player object in a first-person view. Movement is via the main and secondary controls on the joy-pad or keyboard. It is possible to move forward, back, turn, strafe and jump. (By default, the PC uses the cursor keys.)

The blobs on the ground represent teleporter behaviors. The yellow is the teleporter itself (FPSTeleporter) and the green its target location (FPSTeleportDestination). The bobbing health pickups disappear when picked up (this is achieved by them sending a message to the FPSPlayer object, which then deletes them). These use the FPSSpinningPickup and FPBSBouncyPickup behaviors.

Behind the player's start location and next to a pillar are two further set-ups. The pair of green boxes are used to display the auto-climb or stair climbing behavior of FPSPlayer. The lift demonstrates the FPSPlatform behavior and will lift the player when you travel onto it. This leads up to the FSPTTrain example, which will follow a path when the FPSPlayer behavior is moved (jumped) onto it. The followed path is built up from FSPPathNode behaviors.

# FX motion blur (Xbox, GameCube, and PlayStation 2 only)

---

This example demonstrates how to use the motion blur behavior.

To run the example:

1. Open the Example FX Motion Blur project.
2. Launch and connect to an Xbox, GameCube or PS2 target.
3. In the Game Explorer window, selected one of the game level folders.

## Overview of the CFXMotionBlur behavior

In simple terms, the motion blur behavior works by blending the last frame rendered with the current frame being rendered. The blend is done by rendering a polygon that is textured with the output from the previous frame with the current frame. By changing the amount the previous frame is blended in with the new frame, varying degrees of motion can be achieved.

Normally, the polygon used to blend the previous frame would exactly cover the screen; however, the motion blur behavior in RenderWare Studio allows you to apply a transformation to the blend polygon, providing a number of interesting effects. Each of these effects is demonstrated in the following game level folders in the example game:

### **Motion Blur**

Standard motion blur effect

### **Rotating Motion Blur**

The blend polygon is rotated slightly causing the screen to spin.

### **Scaling Motion Blur**

The blend polygon is shrunk slightly causing the screen to shrink inwards towards the center of the screen.

### **Scrolling Motion Blur**

The blend polygon is offset slightly causing the screen to move sideways.

### **Combined Motion Blur**

A combination of all of the above.

## CFXMotionBlur attributes

### **Start Render Event**

The event used to trigger the motion blur blend.

### **Render Priority**

The priority of the render event. It is important that the motion blur is performed last, however interesting effects can be achieved by performing the blur mid render.

### **Offset X value**

Amount to offset the blend poly in X.

### **Offset Y value**

Amount to offset the blend poly in Y.

**Offset Z value**

Amount to offset the blend poly in Z.

**Scale X axis value**

Amount to scale the blend poly in X.

**Scale Y axis value**

Amount to scale the blend poly in Y

**Scale value**

Overall scale value

**Angle value**

Angle of the blend polygon

# FX particle spray

---

This topic describes the particle behavior example project. This project is split into game level folders to make the different examples easier to follow, and to allow the Event Map to show a clearer, simpler view in each case.

Some examples use the CFXPartSpray behavior, others use the CFXReplicatorGenerator behavior. The simpler behaviors have been made into templates for easier placement within the world.

To see the example entities, you need to either:

- Turn the camera to the left.
- or
- Switch off the director's camera. (This activates an FPSPlayer object which can be guided around the levels. On the PC, use the arrow keys.)

The maximum number of particles can be changed on each emitter. This is the size of the particle tank (that is, the maximum number of particles that can be displayed in any one system). Editing this value can cause memory fragmentation on the console and may ultimately result in the console running out of memory. This will not affect the Workspace, though (and restarting the console will solve the problem). Using “just enough” particle will help memory usage in the final game.

Programmers should note that in a release (non-design) build, the CFXPartSpray behavior allocates the particle tank once only, and that it will be set to the value chosen by the editor of the behavior. This ensures that the memory usage is controllable and does not cause fragmentation.

Artists/editors should note that odd color, size, etc. effects may occur during changes. This is particularly true if lifetimes are extended. These effects are due to the way in which the animation system works. The problems will disappear after the newly emitted particles feed through the system. If the problem is minor, then simply click the test fire button next to the Enable message (Message section) twice. This disables the particle system and then re-enables it, restarting all particles. If this fails, or the problem is more extreme (console slowdown), then set the Max particles (Emitter section) value to zero and then return it to its previous value. This forcibly kills all particles and causes a restart (causing memory fragmentation in the design build).

## Simple examples

These are available as templates for easier placement. They have been placed in a child folder named “Simple Particles” in each game level folder.

### Fountain (level 2)

Reproduces a fountain of water spurting out of the floor.

The emitter is angled by rotating the asset to give an angled spray effect. The spread is changed to remove bands (y axis spread) and extend the area over which the particles are emitted (x and z axis spreads) - ie not emitted from a point. The emitter spreads are in local axis, so they are rotated with the asset.

Angular spreads are adjusted to give a flat fan effect. Primary state velocity and gravity adjusted to give water a curving spray.

Particle color and size is animated over three stages, but velocity and gravity stay at the primary start values. This is achieved by selectively enabling the size and color adjustments in the primary end and secondary end states.

The initial values must be set in the primary start state for velocity, color, size and gravity. These will stay set throughout the lifetime of the particle unless they are changed in any other state. The secondary states are ignored unless the secondary state is enabled. Any changes between the primary start and end are animated over the lifetime up to the secondary "lifetime change" or the end of the lifetime, if the secondary stage is not enabled.

In this example, there are no changes in the secondary start state (these would be snap changes) but the color and size are changed in the secondary end state. Because there are no changes in the secondary start state, the values in the primary end are carried over for used for the start of the animation that ends at the secondary end state. This animation takes place over the period of time from the secondary state lifetime changeover to the lifetime end (set in the emitter section).

Note that the alpha value of the color is also reduced over the two animation periods, becoming almost transparent by the secondary end state. Also, the color in the primary end and secondary end states has been randomised in the green and blue channels. Size has also been randomized in some states.

## **Fire (level 1)**

Reproduces a patch of fire on the floor.

The emitter is extended in x and z to produce an area of fire. The angular spread is set wide to make the fire expand as it rises. The system is in two parts; the first simulates the flames and the second the smoke produced.

The primary start to end states is over a very short lifetime (the secondary state change over is after only 0.2 seconds). The velocity is kept low so the fire only rises a short distance. To stop the velocity being nullified, the drag is kept low and the gravity is removed. The particle's size and color change over this primary state, becoming darker and larger towards the end. There is a large amount of randomness in the size at the end of this stage.

The secondary stage simulates the fire turning into smoke. When the secondary state changeover occurs, the system emits extra particles and deflects the particles direction of movement. For each particle coming into the secondary stage, two extra ones are added. Each of these particles has its direction of movement deflected by between 0 and roughly 60 degrees. This makes the smoke appear to billow out from the end of the fire particle.

Additionally, these particles are given a one time only boost of velocity, have their gravity, drag, color and size changed. These changes make the smoke appear more full. Since these occur in the secondary start state, they are instant (ie not animated) changes. The color is given a degree of randomness. The gravity is adjusted to a negative value to make the smoke particles float upwards.

The color and size are animated over the secondary lifetime by turning these attributes 'on' and adjusting the values in the secondary end state. The particles fade due to the alpha value in the end color. The end size is also largely random.

The particles are also continuously accelerated on each frame between the

secondary start and end state. This is set up in the secondary end state, under 'Acceleration'. This is applied along the current movement vector of the particles. This helps to keep the particles moving upwards at a realistic speed.

## Fireworks (level 2)

Reproduces an exploding rocket firework.

Again, the particle system is in two parts. The first blasts the particles into the air, the second makes them explode into a shower of colored sparks. The particles are emitted in a random number per frame, sometimes none, sometimes 1 or two.

The emitter is set to zero size in X, Y and Z. So the particles come from a single point. The rate is set to zero, with a randomness of two. This causes the particles to not be emitted on every frame. The angular spread is quite wide to allow the rockets to fire off in a multitude of angles and the lifetime is quite short, but has a degree of randomness. The second stage will always cut in at a set lifetime, but the randomness of the lifetime is applied at the start of the particle (during emission the age is not set to zero, but to a range between zero and the lifetime randomness). Because of this the randomness effects when the particle enters the second stage and so the particles explode at different heights, but still complete their explosions.

In the primary stage start, the particles are given a reasonable velocity, but they have zero drag and no gravity, so the particles continue upwards without being slowed. The color is randomly adjusted from a light yellow and the size is kept quite small. None of these values are adjusted in the primary end state, so they are kept constant until the secondary stage is entered.

This occurs after approximately 1 second and when it does each 1 particle in results in an extra 50 particles out (secondary change over -> secondary emitter attribute). This forms the cloud of colored particles. These particles are deflected into a sphere by virtue of the 0 degrees (min) to 180 degrees (max) deflection angle.

As the particles enter the secondary start state, they are given a large velocity boost. This give the effect of an explosion. Remember that the secondary start state values are immediately applied; there is no animation from the primary end state values. The high drag value causes the particles to slow quickly and the gravity causes them to fall earthward. The red color is applied (with some randomness) and the size is considerable increased (with a large degree of randomness).

The secondary end state causes the particles to reduce in size over the remaining lifetime. A small amount of continuous acceleration is applied to counter act the long-term effects of the drag. If the acceleration weren't applied, the particles would stop and not look quite so "explosion-like". Try clicking the **Acceleration on** check box on and off, to see the difference. You will need to give the particle a short time to change over. The change of the check box is for particles being emitted and these new particles need time to progress through the system.

## Complex, grouped, examples

The emitting message and location of the particle systems can be modified. By default the location is fixed and the system emits each system tick.

However, the particles can be emitted from multiple locations per frame and be tied to another behavior via the event system's messages. The following two examples are used to demonstrate these features. However, this example does not explain the use of the CFXReplicatorGenerator behavior. For information on this see the [Replicator FX example](#) (p.539).

## **Box fountain group (level 4)**

Simulates a fountain of gas-emitting, floating boxes!

The CFXReplicatorGenerator behavior attached to the Boxes entity has been set up to launch a spinning box into the air every 10 system ticks. See other examples for how to set up this part of the behavior.

The important section of the set up is under the Particle Messages attributes section. Here the replicator, for each particle, on each tick, is sending a message. This is the Particle active message; its data is an RwFrame structure. This contains the position and orientation of each replicator particle, on each frame.

The Trails entity has been set up to receive this message. Notice that the particles are not being emitted from the asset to which the CFXPartSpray behavior is attached. This is achieved by changing the messages setting in the CFXPartSpray particle system. Instead of leaving the position message as <None>, it has been set to respond to the RwFrame being sent by the replicator (partlink message). Each time this message is received the particle system will move its XYZ position, up, right and look-at vectors to those held within the RwFrame. That is to say, the orientation of the particle system emitter will match that of the RwFrame passed in. However, this incoming data can be modified. Frequently you will need to adjust the location and orientation of the particle emitter relative to the object it is attached to.

When an entity has a particle system attached to it, the default is to use the entity's origin and its local axis for the orientation of the emitter. However, say you are trying to simulate the exhaust trail of a rocket (as the Rockets Group in level 3 attempts to do). The rocket may have been modelled with the origin at the sharp- end, say for collision detection reasons. Let's also say that the positive direction of the local axis points forward, in the direction of movement of the rocket, to make processing of its movement easier. If you attached a particle system to this, then the particles would be emitted from the sharp-end and outwards in the direction of movement. But what we want is for the particles to come out of the motor-end and in the opposite direction to the movement (for every reaction there is an opposite and equal reaction). To do this, the Display properties of the particle system have to be modified.

In the case of this example, the origin is on the edge of the cube, so there is no need to move the positional offset. What this does however, is change where the particle system is positioned relative to the RwFrame message it is receiving (the origin). In the case of the rocket, you would move the particle system's emitter from the rocket's sharp-end to the motor-end using this control. The other change needed is to the vector the emitter uses. In this case (and the rocket example) the particles are emitted in the wrong direction. They need to be emitted in the opposite direction. So, the positional deflection' value is changed from 0 to -180 (+180 would also have worked). This control bends the particles backwards by a controlled amount away from the original vector. The second positional heading control rotates the particle emitters heading around the original vector. At 0 and 180 (-180) this has no effect. If the deflection were 90 degrees, then this would

spin the emitter around in a circle. A value of 45 or 135 degrees would spin in a cone, etc.

## Messages

The next problem is that of actual emitting particles. By default, the emit on message is set to the iMsgRunningTick, which means that on each tick a burst of particles will be emitted. But the replicator will send out several partlink messages to each iMsgRunningTick message. If the emit on message is not changed, then particle will only be emitted at the position received by the last partlink message before the iMsgRunningTick message is received. That is, only one of the boxes in the replicator would have a trail. To get each box to have a trail, the particles need to be emitted when the partlink message is received to reposition the emitter. The emit on message handler does not expect any data, so any message link can be used. Therefore, the partlink message is also used here.

Note that the update on message is not changed. This is because we still want the particles that have been emitted to update once a frame and not several times a frame. If the message was changed to partlink, then the particle system would run too fast and the speed would be the frame rate multiplied by the number of replicator objects the particle system was attached to (that is, be variable). This is far from ideal and not what the effect needs.

It is important to realize that the particle system is being used as a shared resource here. That is to say, there are in effect several trails running, one per replicator object, but only one particle system. This uses less memory than one particle system per replicator object would. This would be especially true if the attached object was coded so that it only emitted particles if it were visible. In this way, several box fountains could be in the level (but only one particle system for the trails). Only a sub-set would be visible and as such, fewer particles would be needed in the one particle system that they all used. It would be necessary to ensure the particle system had time to start up before it became visible again though, otherwise a gap would be visible. This is the same effect as when the example first starts or is reset.

The other parts of the particle system set up are straightforward. Only the primary stage is used with color, gravity, drag and size being animated over it. The idea is to simulate a gas stream coming from the boxes.

To do this, the particles start out with a very small velocity, quite high drag and a small downward gravity, they are also very small (ie compressed droplets). In this way the particles will slowly float downwards. By the end of the primary stage, the size has been increased (randomly) and the color has changed (vaporised into a gas). In addition the drag is total, ie the particles stop, except for the effect of gravity, which is now negative and making the particles float upwards slightly.

## Rockets Group (level 3)

Simulates rockets firing out of the wall and exploding.

The Rockets replicator is linked to the Rocket Trails particle system in the same way as the Box Fountain Group described previously (see the description of level 4 for information on how this has been achieved). The only difference is the link message, which here is rocket\_run.

The Rocket Trails particle system is a very simple single stage (primary only) set up. The linkage to the Rockets replicator requires that the particle system's emitter be reoriented. The origin is the same, but a deflection of minus 90

degrees in needed to get the trails moving in the correct direction.

The emitter spread values make the trials appear over a small area (Y and Z) instead of a point and the X value removes banding in the particle spray. There is initially no velocity, gravity and little drag on the particles. These are not needed, as the emitter is moving with the rocket. As the particles come to the end of their lifetime (primary end state), the color darkens, the gravity becomes negative (they float upwards), size increases and so does the drag; thereby slowing the particles to a stop.

The main difference between this example and the Box Fountain Group is the second linked particle system, Bang. The replicator (Rockets) has been set to output the RwFrame of its objects when they die. This is performed via the Particle expire message in the Rockets ► Particle Messages attribute. So each time a replicator object dies, an RwFrame is sent to the Bang particle system (via the rocket\_bang message). This has the same linkage as the Rocket Trails (and the Trails in the Box Fountain Group; see the notes above on that group).

When the bang particle system receives the message, it moves and orientates itself from the data in the attached RwFrame and emits a block of particles. The particle system itself is quite simple. It requires reorientation, in the Display attributes. The system emits a large quantity of particles over a small emitter spread (so as to simulate the cube being exploded). The particles have a large primary start state velocity, so as to carry on moving forwards. The drag is quite high, so they slow quickly and a substantial gravity pulls them towards the floor. In the primary end state, the color is changed and the size increased. There is also an acceleration applied (per tick) to counteract the effects of the drag and keep the particles moving forwards slowly.

The particles also have a secondary stage, so the Secondary lifetime change and Secondary on attributes (in the Secondary change over section) have been changed from their defaults. The secondary start state changes no attributes; instead just the color is changed in the secondary end state. This has the effect of allowing a three state color change. Because the color is not changed in the secondary start state (which would cause a snap color change) the color at the primary end state is carried over and this is animated over the secondary stage lifetime (the change over lifetime ► total lifetime, as set in the emitter). The final color is total transparent, make the particles fade as the complete there processing.

## FX Xbox pixel shader

---

This example demonstrates how to apply a pixel shader to the rendering of atomics and clumps.

To run the example:

1. Open the Example FX Xbox Pixel Shader project.
2. Launch and connect to an Xbox target.
3. Ensure the camera is looking at the tank placed in the middle of the level.

### Overview of the CFXXBoxExamplePS behavior

When the example is running, the tank will be fading between being rendered in full color and being rendered in black and white. The effect could be used to indicate that the tank is about to explode or has been hit by the player in a game.

To adjust the speed of the fade effect, change the speed attribute of the CFXXBoxExamplePS behavior.

To apply the pixel shader behavior to another asset, simply drag the asset from the asset lister onto an instance of the CFXXBoxExamplePS behavior. This will associate the asset with the pixel shader behavior.

**Note:** Currently, this behavior works only with atomics and clumps that have single textured materials.

## FX Xbox vertex shader

---

This example demonstrates the example flag vertex shader for the Xbox.

To run the example:

1. Open the Example FX XBox Vertex Shader project.
2. Launch and connect to an Xbox target.

### Overview of the CFXFlagShader behavior

The CFXFlagShader behavior can be used to animate the vertices in such a way as they appear to wave in a similar way to cloth. The shader makes no pretence to be a true cloth simulation. It simply provides a rough approximation good enough to animate a flag waving in the wind.

Each vertex in the mesh is animated by passing a number of sine waves into each vertex, the frequency and amplitude of each wave can be changed using the behaviors attributes. A weight value for each vertex is taken from the alpha component of the vertex alpha, this means the mesh used for the flag needs to have vertex alpha assigned on each vertex. Alpha values of 0 mean the vertex will not move while alpha values of 255 allow the vertex to move freely. Using 3ds max, you can paint vertex alpha onto the vertices, providing varying degrees of stiffness over the cloth. You can also exploit this feature to build the flagpole and the cloth as part of the same mesh. Setting the flagpoles vertex alpha to zero will ensure the pole does not wave about.

# G3x Pipelines example (PlayStation 2 only)

---

This example demonstrates the G3x material pipeline behaviors, which are specific to the PlayStation 2. The G3x pipelines are rendering pipelines optimized for various lighting conditions. In a nutshell, if the lighting model of the scene can be considered static (for example one ambient and one directional light), then these pipelines can significantly increase performance.

The example is split into 5 levels with each level rendering as much as possible (given the example artwork) at 60 Hz.

## **Level 1: Using Default G3 Pipelines**

Shows 5 warriors (a skinned character with 4 weights per vertex) being rendered with the standard general purpose pipelines. The world contains 1 ambient and 1 directional light. Running this example on the PlayStation 2 with the metrics data enabled shows that we are reaching the limits of the rendering performance.

## **Level 2: Using G3x\_ADL (1 Ambient + 1 Directional)**

Essentially this level is the same as level 1 except that we have applied several G3x\_ADL pipelines to the various geometries in the scene (the ADL pipelines are optimized for 1 ambient and 1 directional light). We are now able to render 10 warriors.

## **Level 3: Using G3x\_APL (1 Ambient + 1 Point)**

As in level 2 but using the APL pipelines.

## **Level 4: Using G3x\_A4D (1 Ambient + 4 Directional)**

As in level 2 but using the A4D pipelines.

## **Level 5: Using G3x\_ADL (1 Ambient + 1 Point)**

As in level 2 but this time the warrior has only 2 weights per vertex.

The material pipelines are attached using the following behaviors:

- CG3x\_APL\_MatPipelineID
- CG3x\_ADL\_MatPipelineID
- CG3x\_A4D\_MatPipelineID
- CG3x\_APLDup\_MatPipelineID
- CG3x\_ADLDup\_MatPipelineID
- CG3x\_A4DDup\_MatPipelineID
- CG3x\_APAGem\_MatPipelineID
- CG3x\_ADLGem\_MatPipelineID
- CG3x\_A4DGem\_MatPipelineID
- CG3x\_APSSkin\_MatPipelineID
- CG3x\_ADLSkin\_MatPipelineID
- CG3x\_A4DSkin\_MatPipelineID
- CG3x\_APSSkinDup\_MatPipelineID
- CG3x\_ADLSkinDup\_MatPipelineID

CG3x\_A4DSkinDup\_MatPipeID  
CG3x\_APSSkinGem\_MatPipeID  
CG3x\_ADLSkinGem\_MatPipeID  
CG3x\_A4DSkinGem\_MatPipeID

These behaviors work by attaching the relevant pipelines to the assets before they are cloned for use by the normal behaviors such as CEntity and CMultiAnimate.

## Maestro

---

This example demonstrates the use of the CMaestro behavior to allow some control over a Maestro file (.anm) from within the RenderWare Studio Workspace.

To run the example:

1. Open the Example Maestro project.
2. Launch and connect to a RWS Console.
3. There should be two Maestro entities:
  - A menu that can be navigated using up/down to highlight options and **Enter / Backspace** (or the usual select / back keys on the consoles) to select an option or go back a level.
  - An animated square that swirls around.
4. The attributes of the Maestro entities allow for positioning within the game, along with other options.

## Multiple levels

---

This example demonstrates the use of multiple folders in a game, where each of the highest level folders corresponds to a level in the game.

In the Game Explorer window, one of the folders is highlighted in **bold**. This is the [active folder](#) (p.166). When you connect to a target console, it is this folder's contents that will get sent.

Each game also has a [global folder](#) (p.168), which is always sent to the target console before the active folder.

You can also [import folders](#) (p.175) from another project.

# Replicator behavior

---

This example contains a total of 4 replicators, which combine to provide one effect sequence. This effect is initiated at several locations within the level.

When the example loads, turn approximately 80 degrees to the right to see the effect (or turn the director's camera off). There are 3 effect "launcher" locations. The leftmost one first fires at a rate of 3, the centre one at 2 and the right most at 1.

The Launcher is provided as a template. To create a new launch location, drag this template into the Design View window.

## How does it work?

**Tip:** While reading this description, have the Event Map flowchart open for reference.

Each launcher (in the Timer folder) is an FPSTimer object and is used here to send messages (called launch) to the replicator (target attribute). The time between each message can be set in the behavior, under the wait attribute. A random, additional time on top of the wait can be specified by the random attribute. When the launch message is sent, the data attached is the location of the launcher itself (as an RwFrame). Note that each launcher is invisible. This is achieved by selecting the asset and, in the attribute editor, selecting CSystemCommands in the behavior drop-down list, and then selecting the Invisible check box.

The message is picked up by the primary replicator object, RocketEffect (in the Effects folder). The trigger particles attribute (in Messages) is set to the launch message from the FPSTimers. Each time the message is sent the replicator emits one object (number of particles to create = 1, messages required to trigger = 1) and there is no upper limit on the number of objects that can be emitted (total number of particles = 0 - special case). Crucially, the object is not emitted from the asset that the replicator is attached to. Instead, the replicator moves itself to the location held in the passed in RwFrame (from the launcher) and then emits. The net effect is that the objects are emitted from three locations on the level and not one. If another launcher were added, then they would emit from four locations, etc.

The particles have a minimum and random additional lifetime, ensuring they will explode (die) at least 72 ticks from start (expire time bias), with a life of up to 72 + 123 (random expire time), ie lifetime of between 72 and 195 ticks.

At launch, each object is given an upward (Y axis) velocity of 8 (initials velocity -> velocity bias). They also have a random sideways (X and Z axis) velocity of between -4 and +4 (Velocity Random). The values read 8, but the actual applied is + or - a half times this value. They are also initially spinning around the X axis (Angular velocity Random) by -4 to +4 (again + or - a half times the value in the attribute).

Each tick of their lifetime the objects have values in the Acceleration/Friction section applied to them. In this case, there is no "Acceleration" applied to the velocity, but there is "Angular acceleration"; this cause the object to spin faster on each axis as it progresses through its lifetime. Note that the Friction and Angular

Friction value are 1. This may seem odd, but in the replicator behavior, the value of 1 means “no friction”. If you change the values to zero, the replicator objects will not move off the floor. Set the Y axis value to 1.1 and they zoom off into the air.

The Scale values are not used here, so you can ignore them. Also ignore the Particle messages section for the moment; we will come back to this.

The color values allow you to change the objects color over its lifetime, time 0.0 is the start of the lifetime, 0.25 is quarter the way through, 0.5 is halfway, 0.75 is three quarters of the way through and 1.0 is at the end of the lifetime. These colors are blended at intermediate time periods. So between the start and a quarter the way through, the color starts as a dark blue and slowly changes to a medium, slightly mauve blue.

The Render state flags allow you to change how the objects are drawn by the graphics system. These can be ignored for the purposes of this example.

## Inter-replicator messages

Returning to the Particle Messages section of the Rocket replicator behavior: here there are two different messages being sent out from this behavior. The Particle expire (Bang) message is sent when the particle dies (reaches the end of its lifetime). The data for this is the last position the object reached, as an RwFrame.

The second attribute, Particle active (Trail), is sent out on each tick, by each active object in the replicator. The data attached is, once again, the current position of the object, as an RwFrame.

Taking the Trail message first: looking in the Event Map shows that the TrailEffect replicator receives this message. The Bang message, however, is connected to three other behaviors.

## Trail effects

The TrailEffect replicator is responsible for the rocket trails. This is set up in very similar way to the Rocket replicator. However, it does not send out any messages, has a much shorter lifetime and uses different colors. More importantly, the Initial velocity ▶ velocity bias is negative, so the trails fall away from the rocket and there is a per tick linear and angular acceleration/friction applied. This just slows the particle over time, except in the negative y-axis and in the y-axis rotation (due to accelerations counteracting the frictions). The scale has been adjusted too, so the trails are smaller than the rocket (except in the X-axis). This make the trails appear as bars and accentuates their rotation.

## Explosions

The other message (Bang) sent from the Rockets replicator, on death, is received by three separate behaviors. Together they provide the explosion effect. There are two replicators, which are set up in basically the same way as the trail effect. The only real differences being that they emit more particles (four or eight) than the single one emitted by the trails. The colors and initial velocities are different too. In the case of the velocities, all axes have the same values. This creates a spherical explosion. One system (bang effect) has smaller value than the other (bang effect B), and therefore has a smaller radius.

The other behavior receiving the bang message is the Flash Light Effect. This is

an instance of the CFXLight\_2Stage behavior. Its linkage to the bang message is in Light Events ► Trigger Event, Enable Light. When the message is received the light moves itself to the position held in the messages data, an RwFrame.

Because the Reset the light effect on trigger check box is set, the light's color stages are reset. These color stages cause the light to start at a very light purple (color 1) and progress rapidly to black (color 2). The speed of the progression is controlled by the 'lerp step' attribute. Since the playback type is "play once", there is no cycling of these colors. Once the light is a black, it stays there. The overall effect is to produce a brief flash of light at the death location of the replicator's objects.

Note that the Flash Light Effect has settings in the sub-behaviors. To see these, select the entity and change the behavior, in the Entity Attribute window, to CFXBaseLight. Change the type to rpLIGHTPOINT, the cone angle to 180 degrees and the radius to 5000 (this is the angle and radius of objects which are affected by the light, from a point source).

## Assets

Note that all the replicators are attached to instances of the same asset, a box. However, the trails, rockets, and bang effects could have been attached to different shaped assets. They would then have 'replicated' that asset for use in the effect. The only restriction is that the asset must be a plain atomic.

# RF3 files

---

This example illustrates how artwork created in a package such as 3ds max can be exported to .rf3 files, and then be converted to asset files to be viewed in the Workspace using the project templates supplied.

## Project Templates

The templates used for this example are stored in the “Export Templates” child folder of the “Examples” folder. The project templates perform two functions:

1. They automatically read and compile the source .rf3 files into asset files when the example is opened, and
2. As they are reading and compiling the .rf3 files, they optimise the output asset files for displaying on the appropriate target platform.

There are two places to set the project templates:

1. In the [Game Properties dialog](#) (p.98)
2. In the [Targets properties dialog](#) (p.199)

## Assets

The root assets in this example show how an asset references .rf3 files. The root assets - “military\_lab”, “Guard”, and the four assets in the “Guard Animation” asset folder - each reference a .rf3 file (stored in the “Models” folder below the “Examples” folder). Below these root assets are the generated assets described in the root asset’s .rf3 file. These assets are auto-generated from the root asset .rf3 files and reference files that will be generated locally when the game is first loaded. These generated assets are created in a folder referenced by the root asset’s GUID in the folder structure “Workspace Output\Asset\rf3s” beneath the game location.

The files created are:

1. A .xml file detailing each of the assets stored in the .rf3 file. Once created, this enables the Workspace to find out which assets were in a .rf3 file without re-parsing the file.
2. A single .rws file that has all assets from the .rf3 file in it.
3. A number of other .rws files, whose filename ends with the index of the asset. This index is the ChunkIndex attribute of each asset in the .xml file. Each of these .rws files is referenced by a child asset of the root RF3 asset.

## Shared attributes

---

This example demonstrates the use of [shared attributes](#) (p.170) within the Workspace. Each of the folders contains a class of object; these objects are normal entities, except that their attributes are shared with others within the same group. For example, the entity folder, Broken Lights, contains a number of flickering fluorescent lights; if you select one of these broken lights and modify its attributes, (try changing Style to normal) you can see that all of the entities that share the attributes with the selected entity are also updated.

Shared attributes can also be used within templates. If you open the Templates folder, you will see that there is a template setup for each class of object. Selecting the template and modifying its attributes has the same effect as selecting an instance of the template and changing its attributes.

## Sky dome

---

This example demonstrates the use of the sky dome and environment mapped objects.

Load the example, connect and enable the director's camera (  ). If you use the FPSPlayer's camera, you will not be able to see the sky dome because of the room's walls. Choose the flight mode (paper plane button) and, using the right mouse button, click and hold just above the box to move out of the top of the room. Then turn 90 degrees to the left (left click and hold just to the left of the box). This should bring a spinning torus into view.

The sky dome works by being rendered first (hence the higher message priority than the FPSRender) and it is responsible for clearing both the color and z-buffers. The FPSRender object attached to the room does not do this operation as that would result in two clears per frame and give undesired side effects.

The texture matrix for the spinning torus is set up within the Setup Environment Map. This uses a CFXEnvironmentMap behavior, allowing the positioning of the environment map and its animation to be specified. It does NOT render the object, it only sets up the mapping for the object it is attached to (alpha\_torus). The rendering of the torus is performed in the SkyDome ► Object as part of SkyDone. This uses a standard FPSSpinningPickup behavior.

The thing to note is that the CFXEnvironmentMap sets up the mapping for all instances of alpha\_torus asset. Therefore they will all have the same mapping on them. If you want a second torus with different mapping, they will have to use two different assets.

The reason the torus is fixed in screen space is because it is rendered as part of the sky dome, which is also fixed relative to the camera. This is achieved because it occurs in the list of the game level folder after the sky dome and before the room. If it was after the room, it would be rendered relative to it (that is, move with the room).

## Spline camera

---

This example shows how to make a camera follow a curved path - defined by the [spline editor](#) (p.34) - around the scene in the Workspace.

It also demonstrates how to connect up the camera system, so that both the director's camera and the camera that is moving along the curve can co-exist.

**Note:** To see the effect of the camera moving along the curve on the target, you need to ensure that you have turned off the  Director's Camera.

The camera is moved using the `CCameraAsSpline` behavior attached. This behavior takes the spline data and positions and orients the camera based on the control points defined along the spline curve.

# Split screen

---

This example demonstrates using CSubRasterCamera to generate split screen effects; sub-rasters can be defined that are sub-sections of a larger raster such as the screen. In the example, a vertical split is shown with a first-person viewpoint rendered on the left and a third-person viewpoint on the right.

## How does it work?

The quickest way to understand the example is to use the Event Map window in the Workspace: open the example Split Screen project, and then select the Event Map window.

**Tip:** To expand the Event Map to occupy the full screen, press **F12**. (To switch back to the normal window, press **F12** again.)

The FPSPlayer behavior is itself a camera; it responds to the iMsgDoRender event which passes in the RwCamera associated with the screen. FPSPlayer points the camera in the correct direction and sends out the event 1st\_person\_camera with the newly orientated camera. The event 1st\_person\_camera is received by the entity LeftSubRaster which generates an RwCamera with the correct sub-raster dimensions for the left side of the screen. The entity LeftSubRaster then sends this RwCamera to the FPSRender behavior via the render event.

Similarly, the entity 3rd person camera (which is an instance of CCameraLookAtPoint) receives the iMsgDoRender event and generates the 3rd\_person\_cam event. This is sent to the entity RightSubRaster which sets up the right-hand side of the screen. Finally, the RightSubRaster entity sends render to the FPSRender behavior to render the right-hand side of the screen.

By modifying the attributes of LeftSubRaster and RightSubRaster, you can achieve different split screen and picture-in-picture effects.

## Split screen to view PVS

---

This example demonstrates how the split screen behavior and rendering behavior can be used to easily create a tool for viewing the visible sectors of a world with PVS data, while still being able to view the scene as it would normally appear.

This is achieved by sending the camera first to two split screen behaviors CSubRasterCamera, one for the left side of the screen and one for the right. Each of these behaviors is then attached to one of two FPSRender behavior the left side is rendered normally and the right side is rendered in a debug mode showing the visible sectors.

# Glossary

---

## A

### **asset**

An object in the game database.

Identifies data that can be used by an entity. Among other properties, an asset contains the relative paths of:

#### **A primary asset file**

Contains the “primary” data for an entity (for example, a .dff or .rf3 file, defining the geometry of an entity).

#### **A dependencies folder**

The folder containing data files required by the primary asset file (for example, texture bitmaps).

Several entities can use the same asset. You can attach assets to an entity either one by one, or you can group assets into asset folders, and then attach a mix of individual assets and asset folders to an entity.

**Note:** In general usage, the term *asset* is sometimes used to refer to a primary resource file. However, in RenderWare Studio, the term *asset* refers to an object in the game database that points to a primary asset file, not to the primary resource file itself. Like any other object in a game database, an asset is stored in an XML file.

### **attribute**

An object in the game database.

Stores the value of a parameter that controls the behavior of an entity. For example, a vehicle entity might have an “exhaust color” attribute that controls the color of the exhaust smoke that its behavior causes the entity to emit. Each entity can either have its own individual attribute values, or use *attribute shares* to share the same attribute values as other entities.

To change attribute values, use the Attributes window.

Behavior commands and entity attributes are closely related. Commands define the attributes that an entity can have. Attributes store the values of the command parameter values for an entity.

## B

### **behavior**

A C++ class that you can attach to an entity. The class defines:

- How and where an entity appears in the scene.
- The actions that an entity can perform.
- The events that an entity can cause.
- The events that an entity can respond to.

When you create an entity from an asset, the new entity is given the default

behavior `CEntity`.

Behaviors are also represented as objects in the game database, but they are not persisted in the game database XML files; entities refer to behaviors by name.

## **box volume**

Box volumes are simple cuboid entities that you can place into your game without the need to import an asset. They can be used as placeholders for entities that will be created in the future, or (more commonly) as the basis for trigger volumes.

## **C**

### **CEntity**

The default behavior that RenderWare Studio attaches to new entities.

Defines simple rendering, physics, and collision-related actions.

## **D**

### **delete**

Erases an object from the game database, and all references to the object.

## **E**

### **entity**

An object in the game database.

Typically, each individual object in your game (such as human figures, vehicles, tools, weapons, doors, buttons, levers) is represented by an entity in the game database. An entity consists of:

- References to zero or more assets
- A reference to one (and only one) behavior
- Values for the attributes of the behavior

For example, a car entity might consist of:

- A reference to an asset that defines the geometry of the car and its texture (such as paint color and exterior detailing)
- A reference to a behavior that defines how the car moves
- Attribute values relating to the behavior (such as the amount of rubber the car's tires leave behind on the road as it accelerates)

## **F**

### **folder**

A container for other objects in the game database. You can use folders to organize a game database into a structured hierarchy. There are various types of folder.

Only the following types of folder are significant in the game database hierarchy:

**Folder** (without any additional qualifying term; if you like, a “normal” folder)

Can contain entities, assets, asset folders, or other folders. Displayed only in the Game Explorer window.

Depending on the genre of your game, the highest-level folders might correspond to the levels in your game.

### **Asset folder**

Can contain assets or other asset folders. Displayed in the Assets and Game Explorer windows.

### **Global folder**

The global folder is a special type of folder containing objects that are persistent throughout a game (not just one level of a game). A game has one, and only one, global folder. The contents of the global folder are sent to the target console before any other folders. A global folder can contain the same types of object as a normal folder (assets, entities, other folders etc.), but the global folder cannot be nested inside another folder.

The other folder types—attribute share folders, behavior folders, and template folders—exist only to organize the display of objects in their associated Workspace windows.

## **G**

### **game**

The root object in the game database hierarchy. A game database can contain only one game object. Currently, RenderWare Studio supports only one game per project (.rwstudio file).

### **global folder**

See *folder*.

### **game world**

The 3D space in which entities are located in the game. This is the static geometry that usually takes the form of a .bsp file.

## **R**

### **remove**

Erases a reference to an object in the game database (not the object itself). Any other references to the object are not affected.

### **resource**

The file that is referred to by an asset. This file contains data that can be used by an entity: for example, geometry (typically stored in a .dff or .rf3 file).

To allow an entity to use a resource:

1. Create an asset that refers to the resource.
2. Add the asset to the entity.

(Also known as a **resource file**.)

### **resource root**

The absolute path of the root folder where your asset data is stored.

Each asset in the game database specifies the relative paths of:

- A primary asset file (such as a .dff or .rf3)  
and, optionally,
- A dependencies folder (for example, where textures for the primary asset file are stored)

The Workspace uses the resource root folder to resolve these relative paths into absolute paths.

The resource root is a property of the game, and is stored in the project [.settings](#) (p.76) file.

## S

### **source root**

The absolute path of the root folder where the Game Framework C++ source files are stored. The Workspace uses this path to locate the source header files that define behaviors.

The Workspace parses the header files to determine:

- The list of behaviors that you can attach to entities.
- The attributes for each behavior, and the controls that the Attributes window displays for setting these attributes.

The source root is a property of the game, and is stored in the project [.settings](#) (p.76) file.

## Trademarks

---

Canon and RenderWare are registered trademarks of Canon Inc.

ActiveX, Microsoft, Visual Basic, Xbox and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Adobe, Acrobat, and the Acrobat logo are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions.

Nintendo is a registered trademark and NINTENDO GAMECUBE a trademark of Nintendo Co., Ltd.

PlayStation is a registered trademark of Sony Computer Entertainment Inc.

NXN and alienbrain are registered trademarks of NXN Software AG.

ProDG is a registered trademark and SN-TDEV is a trademark of SN Systems Limited.

Araxis Merge is a registered trademark of Araxis Ltd.

Other trademarks mentioned herein are the property of their respective owners.

# Contact us

---

## General information

For general information about RenderWare Studio, email:  
[info@csl.com](mailto:info@csl.com)  
or visit the [RenderWare website](http://www.renderware.com/) ([www.renderware.com/](http://www.renderware.com/)).

## Developer support

For support issues, please use our fully managed support services (FMSS) [website](http://support.renderware.com/) ([support.renderware.com/](http://support.renderware.com/)).  
To contact our Developer Relations team directly, email:  
[devrels@csl.com](mailto:devrels@csl.com)

## Sales

For sales information, email:  
[rw-sales@csl.com](mailto:rw-sales@csl.com)