

# **Eidos Patch System**

*for Playstation®2 and Microsoft Windows*

## **Client-Side Overview**

*Version 1.2*

*Updated April 6, 2005*

Prepared by Pinniped Software

Pinniped Software



game and internet programming

Copyright © 2005, Eidos, Inc. All rights reserved.



## Table of Contents

Overview.....	1
Documentation Guide.....	1
Asynchronous Operation.....	1
Important Classes.....	1
Patcher.....	1
PatchExtractor.....	2
Version Numbers.....	2
Where to Go from Here.....	2



# Eidos Patch System Client-Side Overview

*Version 1.2*

## Overview

---

This document is intended to provide a high level overview of using the Eidos Patch System in your game. Once you have familiarized yourself with the basic concepts as described here, you should read more specific information for your platform in the user guide for that platform.

## Documentation Guide

---

The documentation set for the Eidos Patch System consists of the following guides.

- **Client-Side Overview** – This guide.
- **Client-Side Programming Guide for Playstation®2** – Specific information for programming automatic patching into your Playstation®2 game.
- **Client-Side Programming Guide for Microsoft Windows** – Specific information for programming automatic patching into your Microsoft Windows game.
- **Client-Side Reference Manual** – Detailed documentation of the classes and methods used when patching a game.
- **Generating and Serving Patches** – Information on how to generate patches for your game and make them available to the public.

## Asynchronous Operation

---

The patch system is designed to work as part of your main game loop, without the need for your application to explicitly create download threads. As such, the more time-consuming operations, such as network reads, are performed asynchronously using internally managed threads. In order for the patch system to be able to manage this thread, a few requirements are levied on your application:

- Each time through the main loop, you need to call the `Patcher::Update` method. This method allows the patch system to check on outstanding i/o calls and update its internal state.
- If you have an outstanding asynchronous call, you can test for completion of the function using the `Patcher::TestCompletion` method.
- Different platforms may have additional requirements, such as assigning a proper priority to the patcher. See the guide for your specific platform for more information.

## Important Classes

---

### **Patcher**

The **Patcher** class is used to test for the availability of a patch and download it to the user's machine. Only one instance of a **Patcher** is allowed, but it can be used to download multiple different patches.

For example, in a Playstation®2 game, you may wish to download one patch for your executables, and a separate patch for your data, so that the executable patch may be removed from memory once patching is complete, and the data patch may be retained in memory for dynamic loading of data.

In addition, each patch can have its own description available which can be downloaded and displayed in your game.

## **PatchExtractor**

The **PatchExtractor** class handles the patching of individual files which are encoded in a single patch (it *extracts* a file from the patch). On the Playstation®2, the **PatchExtractor** class is used directly to request and patch the files in memory. On Windows, the **PatchExtractor** class is rarely used directly. Instead, patches are applied using the external **WinUpdater.exe** program, or (for some assets) by using a subclass of **PatchExtractor** called the **FilePatcher** to modify files directly on disk. However, a **PatchExtractor** could be used directly if desired.

## **Version Numbers**

---

The patch system depends on your maintaining version numbers for your game. (If you break up your patches into multiple downloads, you are free to use multiple version numbers.) The version number is a single 32 bit unsigned integer which can have any value. When your game queries the server to see if a patch is available, your version number is compared to that on the server. If they do not match, your game is told it needs a patch.

Normally you will keep incrementing the version number as you come out with new patches, thus simplifying your own version control information. However, other schemes could also be used, such as using a checksum of some data which changes with each version. Because the version numbers are only checked for a match, they do not need to monotonically increase.

## **Where to Go from Here**

---

More information on using the patch system classes can be found in the *Client-Side Programming Guide for Playstation®2* and the *Client-Side Programming Guide for Windows*. The **Patcher** class, used for patch discovery and download, has the same interface for each platform, and so the code can be used in multi-platform titles without any special consideration. The application of patches (**PatchExtractor**, `LaunchPatcherApp`, or **FilePatcher**), however, is necessarily different between platforms, so you should carefully consider the application flow separately for each platform.