Generic SWGEmu Launcher Design

*The design specification and usage document for an implementation of a generic launcher to install, update and launch a client for a custom SWGEmu based server.*

*Version 0.09.000*

*March 7, 2016*

# Contents

[Contents 3](#_Toc448959292)

[Tables 6](#_Toc448959293)

[1 Background 7](#_Toc448959294)

[2 Development Environment 8](#_Toc448959295)

[2.1 Development Environment Cost 8](#_Toc448959296)

[2.2 Target Environment 8](#_Toc448959297)

[2.3 Language Selection 8](#_Toc448959298)

[2.4 Installer Selection 9](#_Toc448959299)

[2.5 Install and Update Mechanism 10](#_Toc448959300)

[3 Installer design modules 11](#_Toc448959301)

[3.1 Fresh install of the Game clients 11](#_Toc448959302)

[3.2 Verifying the game clients 13](#_Toc448959303)

[3.3 Efficiently installing a test server directory 14](#_Toc448959304)

[3.4 Updating the Launcher code 15](#_Toc448959305)

[3.5 Manifest creation tool 15](#_Toc448959306)

[3.6 Registry Entries 19](#_Toc448959307)

[3.6.1 The location of the commercial install of SWG 19](#_Toc448959308)

[3.6.2 Registry entries used by the Launcher 19](#_Toc448959309)

[4 Development Environment Operations 20](#_Toc448959310)

[4.1 Setting the executable version number 20](#_Toc448959311)

[5 Class documentation 21](#_Toc448959312)

[5.1 WebFileDownloader Class 21](#_Toc448959313)

[5.1.1 DownloadFileWithProgress Function 22](#_Toc448959314)

[5.1.2 FileDownloadSizeObtained Handler 22](#_Toc448959315)

[5.1.3 AmountDownloadedChanged Handler 22](#_Toc448959316)

[5.1.4 FileDownloadComplete Handler 22](#_Toc448959317)

[5.1.5 FileDownloadFailed Handler 22](#_Toc448959318)

[5.2 textUtils class 22](#_Toc448959319)

[5.2.1 clearStatus(ByVal box As TextBox) 23](#_Toc448959320)

[5.2.2 presentStatus(ByVal box As TextBox, ByVal text As String, ByVal delay As Double) 23](#_Toc448959321)

[5.2.3 extendStatus(ByVal box As TextBox, ByVal text As String, ByVal delay As Double) 23](#_Toc448959322)

[5.2.4 presentText(ByVal box As TextBox, ByVal text As String, ByVal delay As Double) 23](#_Toc448959323)

[5.2.5 extendText(ByVal box As TextBox, ByVal text As String, ByVal delay As Double) 23](#_Toc448959324)

[5.2.6 blankLine(ByVal box As TextBox, ByVal delay As Double) 23](#_Toc448959325)

[5.3 configUtils Class 23](#_Toc448959326)

[5.3.1 Commercial SWG Functions 23](#_Toc448959327)

[5.3.1.1 Function getSWGLocation() As String 24](#_Toc448959328)

[5.3.2 Launcher Functions 24](#_Toc448959329)

[5.3.2.1 Function getLauncherVersion() As String 24](#_Toc448959330)

[5.3.2.2 Sub setLauncherVersion(ByVal setValue As String) 24](#_Toc448959331)

[5.3.2.3 Function getLauncherURL() As String 24](#_Toc448959332)

[5.3.2.4 Function getLauncherHasTest() As Boolean 24](#_Toc448959333)

[5.3.2.5 Function getLauncherHasEULA() As Boolean 24](#_Toc448959334)

[5.3.2.6 Function getLauncherReadEULA() As Boolean 24](#_Toc448959335)

[5.3.2.7 Sub setLauncherReadEula(ByVal setValue As String) 25](#_Toc448959336)

[5.3.2.8 Function getLauncherEULA() As String 25](#_Toc448959337)

[5.3.2.9 Function getLauncherHasCalc() As Boolean 25](#_Toc448959338)

[5.3.2.10 Function getLauncherCalc() As String 25](#_Toc448959339)

[5.3.2.11 Function getLauncherHasForum() As Boolean 25](#_Toc448959340)

[5.3.2.12 Function getLauncherForum() As String 25](#_Toc448959341)

[5.3.3 Game Server Functions 25](#_Toc448959342)

[5.3.3.1 Function getGameVersion() As String 25](#_Toc448959343)

[5.3.3.2 Sub setGameVersion(ByVal setValue As String) 25](#_Toc448959344)

[5.3.3.3 Function getGameURL() As String 25](#_Toc448959345)

[5.3.3.4 Function getGameManifest() As String 26](#_Toc448959346)

[5.3.3.5 Function getGamePatchNotes() As String 26](#_Toc448959347)

[5.3.3.6 Function getGameStatus() As String 26](#_Toc448959348)

[5.3.3.7 Function getGamePath() As String 26](#_Toc448959349)

[5.3.3.8 Sub setGamePath(ByVal setValue As String) 26](#_Toc448959350)

[5.3.4 Test Server Functions 26](#_Toc448959351)

[5.3.4.1 Function getTestVersion() As String 26](#_Toc448959352)

[5.3.4.2 Sub setTestVersion(ByVal setValue As String) 26](#_Toc448959353)

[5.3.4.3 Function getTestURL() As String 26](#_Toc448959354)

[5.3.4.4 Function getTestManifest() As String 27](#_Toc448959355)

[5.3.4.5 Function getTestPatchNotes() As String 27](#_Toc448959356)

[5.3.4.6 Function getTestStatus() As String 27](#_Toc448959357)

[5.3.4.7 Function getTestPath() As String 27](#_Toc448959358)

[5.4 installUtils class 27](#_Toc448959359)

[5.4.1 Secondary routines 27](#_Toc448959360)

[5.4.1.1 Function PrintByteArray(ByVal array() As Byte) As String 27](#_Toc448959361)

[5.4.1.2 Function getFileMD5(ByVal file As String) As String 27](#_Toc448959362)

[5.4.1.3 Function isGoodFile(ByVal file As String, ByVal cSum As String) As Boolean 27](#_Toc448959363)

[5.4.2 Primary routines 27](#_Toc448959364)

# Tables

[**Table 1 - Files to be used to validate a commercial install of SWG** 17](#_Toc448959365)

[**Table 2 - Registry entries containing the commercial SWG install location** 19](#_Toc448959366)

[**Table 3 - Registry entries for the Installer configuration** 20](#_Toc448959367)

[**Table 4 - Registry entries for the Play Server client information** 20](#_Toc448959368)

[**Table 5 - Registry entries for the Test Server client information** 20](#_Toc448959369)

Generic SWGEmu Launcher Design

# Background

The SWGEmu team currently does not provide an open source version of their Installer/Launcher. For servers that do not want to add any in game items, the launcher that the SWGEmu team does provide allows the player to add entries for other servers and launch the game to play on them. The SWGEmu team has implied that their launcher will eventually be made open source by the time that the project reaches its Version 1.00 goal.

The problem that the community of SWGEmu bases servers currently has is that they want to keep the current alpha version of the server and expand the content available on each of the servers. In the commercial version of Star Wars Galaxies, Sony Online Entertainment used a launcher to not only complete the game install, but to also check to see if the game client files were up to date. If the client files were not up to date and in sync with the server the client was updated before it would launch the game. Since the SWGEmu team does not need this update functionality even if the launcher source were made open source today it would be lacking this update functionality. So the community has a need for an open source launcher to support their players and to facilitate sharing between server development communities.

When I was working with the SWGChoice development team I did a very quick and dirty launcher because the team knew that they would be adding items regularly during early development and had experienced the problems that the Bloodfin SWGEmu server had when the game client files were out of date. This launcher did work but it lacked the performance and functionality that the team felt was reasonable in a production version of the launcher.

When the server shut down I had the ownership of the code that I had written for the launcher, considerable experience from creating the launcher and a laundry list of desired changes to improve the performance and functionality of the launcher. I eventually planned to rewrite the launcher code to be easy to adapt for use by other servers and post it to the [Mod The Galaxy](http://www.modthegalaxy.com/) website. This was delayed due to some real life issues but I was eventually able to return to the project. To facilitate the work I started with a design document and as work was being done on the rewrite, mostly from scratch, to add to the design document a user manual section.

# Development Environment

## Development Environment Cost

To make it easier to develop server code the SWGEmu team chose a philosophy of creating a development environment that could be very powerful while costing nothing. This philosophy should be retained as part of the scope for the Launcher development environment. The goal will be to try to find tools that can be legally obtained and used at no or low cost.

## Target Environment

While it is possible to run the SWG Client on Linux and Macintosh systems, the distributed binaries are designed to run in a Microsoft Windows environment. Most of the people wanting to play on a SWGEmu server will be trying to do so using a Microsoft Window System. So producing binaries for other operating systems or using other operating systems to develop the launcher will not be a part of the consideration.

While Microsoft is no longer supporting the Microsoft Windows XP operating system, there are plenty of SWG gamers that are still using systems with that operating system to play the game. So the development environment must be able to produce binaries that will work on that version of the Microsoft operating system. The primary concern will be if the compiler produced code that needs libraries that are not part of the default operating system. If the compiler required that the target systems install redistributable libraries, then they must still supply these files for the Microsoft Windows XP operating system.

Ideally the development tools should be part of an integrated development environment with a robust debugging facility.

## Language Selection

The SWGEmu server code is written primarily in C++ and Lua with some scripting tools. So it would have been logical to use C++ to write the launcher in. This is the language that it appears that the SWGEmu team used to create their launcher. One thing that I relearned with the SWGChoice development team was that there are different levels of experience and skill when it comes to enthusiasts that want to work on a SWGEmu based server. For this reason C++ was rejected. While there is lots of training materials out there for this language, those training materials target people with a good general programming foundation. This opens up considering other languages.

Java is a very popular language that like C++ has all of the tools to create a launcher. It also has the ability to produce operating system independent executables. This feature would allow the future modification of the launcher to run native on Linux and the Macintosh operating system and then to invoke an emulator to run the SOE client files under. The primary provider of the Java language also provides a very good integrated development environment with debug and unit test functionality built in at no cost. This is very enticing but like C++ all of the documentation and the language itself targets people who have a good general programming foundation.

Basic was created as a language that would be a good introduction to programming language. Over time companies like Microsoft enhanced the language so that it could be used for some very powerful application development while it retained the ease of learning. Microsoft through Visual Studio 2015 has a very powerful integrated development environment for their Visual Basic. Their .NET framework also provides tools to create and use a very clean and useful GUI based installer. It also includes tools for getting files over the internet. Currently Microsoft makes available Visual Studio 2015 – Community edition at no cost. It does not have built into it tools to facilitate co-operative development on a large scale or an application installer tool like the commercial versions. Lacking these components does not make the free version of Visual Studio 2015 unsuitable for the project.

Upon further investigation the latest .NET framework, version 4.6.1, includes a redistributable install package for Windows XP even though it was release nearly a year after support for Windows XP officially ended. Also starting with Visual Studio 2015 the [GitHub](https://visualstudio.github.com/) web site now provides an extension to Visual Studio that allows a seamless integration with your GitHub account and repositories.

In the end Visual Basic and .NET 4.6.1 was selected for the language and Visual Studio 2015 – Community edition for the development environment was chosen. Using C++ in Visual Studio was an option but the higher skill set to come up to speed with it was taken as a detriment. Also the code base for the launcher is small enough that if anyone wanted to port it to Visual C++ it should not be a difficult task.

## Installer Selection

The number of files for the installer are reasonably small. It would have been possible to simply distribute the files associated with the installer in a zip file and require people to run a .bat file to do the initial setup of the registry entries.

The equivalent 2013 version of Visual Studio 2015 – Community edition did include a free InstallshieldLE 2015 version for integration into Visual Studio. Flexera decided that they would no longer support InstallshieldLE integration into Visual Studio 2015 – Community. Developers had the option to upgrade to a paid version of Visual Studio 2015 which Flexera does support with InstallshieldLE, purchase a full version of Installshield or migrate to another install package.

Given the goal of no or low cost for the environment this automatically rules out Installshield as an option. A simple search of the web produces some alternatives.

The first is that Microsoft included installer code in Visual Studio through Visual Studio 2010. In Visual Studio they switched to an integrated version of InstallshieldLE and no longer shipped their integrated installer. The do still make it available [here](https://visualstudiogallery.msdn.microsoft.com/f1cc3f3e-c300-40a7-8797-c509fb8933b9) as a free plug in. It is not being updated with newer install technologies so while it is an option it is low on the list.

The InstallAware product is another installer that integrates with Visual Studio. A quick check of their web site shows that they have a free version that integrates into Visual Studio and supports the newer installer abilities. A problem quickly arises on their download page. They ask for a valid email address to send the key to unlock the installer package. However, they reject my email address since it is @gmail.com. I sent them a message asking if there was any way around this. So while others may be able to use this it looks like it will not be an option for me to develop with.

**Update: The InstallAware sales team did respond to my message with the password needed to run the installer for their product. While this does help their cause since I can now evaluate their product the final decision will be made based on ease of use. The top two candidates so have the needed functionality so ease of use will most likely break the tie.**

This exhausts the installers that can integrate with Visual Studio. The next step is to look at good standalone products that have a free option.

The first option that popped up was [Advanced Installer](http://www.advancedinstaller.com/). It does not have a truly free version. If you are a certified Microsoft Technician, regularly review and blog about software or are an open source project they can give you a license for free. I guess that they see these people as advertising their product and are willing to give these people free licenses. While this is an open source project and I may be able to get a free license for my use in the project, I doubt that they would give a free license to every SWGEmu based server a free license. So this probably rules out this product.

Another product that jumped out is [Inno Setup from jrsoftware](http://www.jrsoftware.org/isinfo.php). This product is capable of doing everything that would be needed by the installer package and is free. This is what was used for the SWGChoice launcher install. It supports setting up registry keys, software dependencies, digital signing and of course install/uninstall. So this is the installer that will be used for this project.

## Install and Update Mechanism

Updating the client over the internet is a fundamental premise of the project. Transferring files over the internet can be done several ways and using different protocols.

Unless SWGEmu based servers wish to restrict who can install the game for their servers, none of the information being communicated between the launcher and the server contains user identification information. In fact the only unique information that needs to be communicated is the IP address that the launcher is being run from and this information is sent and stored by the SWGEmu game server every time the user logs in. So given this there is no need to use an encrypted internet transport mechanism. This is good since outside of the https transport mechanism encrypted transport mechanisms actually want to authenticate the connection. By definition this authentication is trying to verify that “you are you” and needs some level of information associated with the user to be transported.

From the server administration standpoint there is an additional issue with using most encrypted protocols and that is that implementations allow bidirectional communication and this could be exploited to compromise the sever and this is a possibility to be avoided.

This leaves both anonymous ftp and http/https for the transfer protocol. Anonymous ftp takes some experience and knowledge to set up so it is less desirable than http. To enable http on a system is actually very easy. Simply add the install web server to the clean install package options and use scp to copy the files to the server like you did to copy the .tre files there when initially setting up the SWGEmu server on the system. If the system is running simply install the Apache web services package. Both methods will cause the web server to be initialized and if you point a browser to the server it will be presented with a web page indicating that there is no web content available. In this state it can still be used to transfer files to the client. For the purposes of the installer it will require that the files be located on a server where they can be accessed using the http or https protocol. Since http does not require a validated authentication signature that is what will be recommended.

If you set up apache server under Debian 8 Linux the file accessed with the URL of http://server.com/PatchNotes.rtf the file will be located at /var/www/html/PatchNotes.rtf and using the default installation will only allow files on the server in the /var/www/html directory and its sub-directories to be accessed. Also the server hosting these files does not need to be the same server hosting the SWGEmu game server.

So with the files hosted where they can be accessed with a URL the installer will be able to dynamically update the client files.

# Installer design modules

## Fresh install of the Game clients

For reasons to be explained later the name of the game client and game client configuration executables as defined in launcher.vb as the following:

'The name of the SWG game client executable

Public Const GameClient As String = "SWGEmu.exe"

'The name of the SWG configuration executable

Public Const GameSetup As String = "SWGEmu\_Setup.exe"

These are the names that the SWGEmu Team uses. SOE used SWGClient\_r.exe and SWGClientSetup\_r.exe respectively as the names of these executables. If you use the SOE names, game community tools like Raptr will recognize the SWGEmu client install as a version of SWG and automatically track game time and share when you are playing with others on your friends list. The use of tools like this is optional but it is an option that players on your server may like and you may want to consider changing the names of the executables.

When the launcher initializes it gets from the registry where you told it to install the game. If that registry entry is empty or the file defined as GameClient does not exist the toolbar option will be set to install. If the registry entry and the file exists the toolbar option will be changed to verify. The button is reused just as the code is reused. The only difference between install and verify is that install asks where to install the game to and when told where to install it will verify that the player has a legal copy of Star Wars Galaxies installed on their system.

When install is selected the launcher will do the following steps.

* Check the Microsoft Windows Registry for where the player installed the commercial version of Star Wars Galaxies. Using a system with a clean install of Windows followed by a fresh install of Star Wars Galaxies from an original set of Empire Divided disks it was noted that the registry entry   
  HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\StarWarsGalaxies\Path  
  contains the location where the commercial version of the game was installed. If the directory exists it is checked to see if bottom.tre exists. If it does the launcher concludes that it has found where the commercial version of the game was installed.  
    
  If you really want to verify that a clean install of the commercial version of the game has been done, check for the SwgClientSetup\_r.exe file instead of the bottom.tre file. On a clean install of the commercial product on CD this executable was used to download the game executables and the reminder of the client files. After they have been downloaded this file was deleted in the cleanup. The problem with using this file is that a legal install copy from Steam may not include this file. In addition to the five CD sets that I have from live I also have three legal copies of the game purchased through Steam. Since it is still possible to install from Steam I will do a steam install of SWG and verify what files it puts on the system and update this portion of the design.
* If the Microsoft Windows Registry does not exist or there is no files from the install in that directory the launcher will assume that an error in design occurred. When this happens it will prompt the user to tell it where they installed a commercial copy of the game. After getting this information it will verify that there is a file from the install there.
* Once the launcher believes that it has located a potential install location for the game it stores that information in the registry area used by the installer for future uses.
* The server now downloads the manifest for the play server client files from the update server.
* This manifest will contain at least information on the file names and which files must exist to verify an SOE install directory along with their MD5 checksum. The server will verify that all of the mandatory files match the checksums of the files in the commercial install. If they all match, the routine that verifies a legal copy will return that a legal copy has been found.
* A location of where to install the game will be asked of the user and stored in the launcher registry area.
* The verify function will now be called. The verify function will use information in the manifest to do the following:
  + Registry entries used by the installer will be set to values defined in the manifest.
  + Registry entries previously used by the installer and flagged to be deleted by the manifest will be deleted if they still exist.
  + Directories flagged to be created will be created if they do not exist.
  + Files flagged as mandatory to verify a commercial install or as SOE provided static files will be checked to see if they exist in the play server install location. If they are there the MD5 checksum will be calculated and compared to what is in the manifest. If it does not match the launcher will check to see if the file is in the directory where the commercial install was found. The MD5 checksum will be calculated, if the file exists, and if it matches what the manifest has they will be copied from the commercial directory to the game play server directory. If the MD5 checksum does not match what is in the registry or if the file does not exist, the file is downloaded from the server.
  + Files flagged as being exclusive to the SWGEmu server will be checked to see if they exist in the play server directory. If they exist the MD5 checksum will be calculated and compared to the value in the manifest. If the file does not exist of the checksum does not match the file is downloaded from the server.
  + Files and directories flagged for deletion in the play server directory that still exist will be removed.
  + If the server has an End User License Agreement it will be downloaded to the launcher directory.
  + If there is an End User License Agreement (EULA) it will be presented for the user to read and either accept or decline. If the user declines the EULA the launcher will not launch the game.
  + If the user accepts the EULA and there is no swgemu\_machineoptions.iff file in the game play server directory the executable defined by GameSetup will be launched. This is done because some machines may have problems with the default window size that the launcher uses if that file does not exist. Running the configuration tool and forcing the user to check and set the default values insures that there is little likelihood of their having this problem.
  + The version in the registry will be set to match the version on the server that matches the manifest.

Installing the test server directory, if the SWGEmu server uses one, is similar except that the test server will simply check to see if the program defined by GameClient entry exists in the play server client directory. A separate manifest for the test server client will be downloaded along with a separate copy of the test server version number. The code will also use the play server directory instead of the commercial install directory as a potential source of where to copy the files from locally to reduce load on the server hosting the game files.

## Verifying the game clients

By breaking the install into separate functions to verify a legal copy, set the intended install directory and finally an install and verify function the largest piece of the code, the install and verify function can be used to verify by adding some additional code.

If the install and verify function were to simply install it only would need to do the following for each file in the manifest:

* See if the file exists in a location on the computer
* If the file exists see if the checksum matches the checksum in the manifest
* If the checksum matches copy the file to where it is being installed
* If the checksum does not match download the file from the server

To change the code to do a verify the function would need to do the following for each file in the manifest:

* See if the file exists in the directory where the file is being installed
* If the file exists see if the checksum matches the checksum in the manifest
* If the checksum matches go to the next entry in the manifest. If they do not match or the file does not exist then do the following:
* See if the file exists in a location on the computer
* If the file exists see if the checksum matches the checksum in the manifest
* If the checksum matches copy the file to where it is being installed
* If the checksum does not match download the file from the server

The highlighted code in red is the additional steps that a verify function would do. In a clean install situation there would be no files in the install target directory so the only overhead compared to a pure install function is a check to see if the file exists in the target directory. Given the number of files this overhead is very inconsequential especially when you consider the advantages of only having one function to maintain. It also presents an interesting performance boost situation. If a player has been playing on a SWGEmu server and now wants to play on a different one, they can copy the .tre files from the directory of the old server to the directory where they will be installing the new server. While they may be copying .tre files that they do not need the nearly 3 GB of files that SOE provided for pre-CU would not need to be downloaded from the server. The verify would made sure that they were unaltered and if they were unaltered simply leave them in place and not download them again from the server.

So if both the installer and verify function of the is modularized like this the installer functions that verify a commercial copy and get/save where they want to install the game can be skipped and with that information stored in the registry the verify can jump straight to the common code. So this is how the code will be written.

## Efficiently installing a test server directory

SOE was very thoughtful when they designed their game client executable to support a test server. SOE reserved a subdirectory of the live play server install directory called Testserver for the test server client to be installed.

An empty install of the test server would contain program files, dll’s and configuration files but no .tre files. To facilitate this you could adjust the entries in the live.cfg file to use a relative path. So the swgemu\_live.cfg file in the play directory would have entries like this for the .tre files:

searchTree\_00\_23=patch\_14\_00.tre

Which would tell the client program to look for the .tre file in the same directory as the client was running in. With the test server being installed in a subdirectory of the live server client the swgemu\_live.cfg file would replace that entry with one that looked like this:

searchTree\_00\_23=../patch\_14\_00.tre

This tells the test client to look for that .tre file in its parent directory. Since the .tre files layered upon each other SOE created new .tre files for each update. SWGEmu servers can do this or they can simply use a single .tre file for their own extension and keep updating it. Either system works. In the client directory the swgemu\_live.cfg file would contain entries like this:

searchTree\_00\_24=patch\_15\_00.tre  
searchTree\_00\_23=../patch\_14\_00.tre

The patch\_15\_00.tre file uses the version in the Testserver sub directory and the patch\_14\_00.tre file from the parent directory where the play server files are. There could be a patch\_15\_00.tre in the play server directory that is different from what is in the Testserver directory or no patch\_15\_00.tre in the play server directory. The entry in the example will only look for it in the Testserver directory.

Using this layering system the only .tre files in the Testserver directory will be those that are different or new in the next patch. This system would reduce over 2.75 GB of space on the user’s system when installing a test server client. Also every file that is reused like this from the play server client is a file that does not have an entry in the test server manifest.

All of this makes keeping track of the client side files for a test server much easier and is a good programming practice.

## Updating the Launcher code

Microsoft Windows differs significantly from Unix/Linux based system in that a running program can not have the program deleted from the file system. In a Unix/Linux system if you delete a running program it no longer is shown in a directory listing and when the program ends the disk space is moved to the empty pool. This allows the running program to not have to be loaded entirely into memory and still has access to the program code on the disk. The Microsoft Windows code is simpler but means that a running program cannot update itself. If you have any applications running on Windows that updates itself you will see them tell you that it needs to update itself, that it is downloading the update and finally that it needs to reboot to complete the update.

Since no code is perfect this functionality is something that needs to be a feature of the launcher. There are several ways to perform these steps. The launcher that this code is based on downloaded the updated launcher to a <program>.exe.new file and then launched a .bat file to rename <program>.exe.new to <program>.exe. The problem with this is that not only did it expose some of the update mechanism but Microsoft never implemented the full package of runtime access protections for .bat batch files that they did for executable .exe files. Also with the move to storing launcher configuration information in the Microsoft Windows Registry it required the .bat file to take input from the command line and manipulate the registry entries.

It is far more secure to create a small helper .exe file that backups the old launcher, moves the new launcher over the old one, updates the launcher version in the registry and finally deletes the backup of the old launcher.

This helper file will be written in the same programming language as the launcher itself to maximize code reuse.

## Manifest creation tool

This program will not be distributed with the launcher. It will be provided to assist the creation of the manifest file by the server developers.

The manifest file will be a CSV file with no file extension. The Manifest file will have three fields.

* Field 1 – This field can be one of two things based on the Action flag.  
    
  Most of the time it will be the relative path to the file. In most cases this will simply be the name of the file like default.tre or s205\_r.dll. In some cases there will be a relative path like miles/mssdsp.flt. In both cases it will be where the file is installed on the local system relative to the game client install directory. It is also the path off of the base URL to find where to download the file from.  
    
  When the Action flag is 5 or 6 which denotes a registry action, this field will contain specific registry entry. Off of the base for that manifest. The base for each manifest will be. In each example <SWG Emu> should be set in the launcher code to the name of the SWGEmu based server:
  + HKEY\_CURRENT\_USER\SOFTWARE\<SWG Emu>\Launcher – for the launcher manifest
  + HKEY\_CURRENT\_USER\SOFTWARE\<SWG Emu>\Game – for the play server manifest
  + HKEY\_CURRENT\_USER\SOFTWARE\<SWG Emu>\Test – for the optional test server manifest
* Field 2 – Action flag. The action flag tells the installer what each entry is for. The action flags are:
  + 0 = Delete file or directory if it exists  
    This flag is allocated to allow for cleaning up a directory. For example it was decided to merge two .tre files and this was used to remove the now unneeded file.
  + 1 = Directories to create  
    These are directories to create to speed up the initial launch of the game.
  + 2 = Verify Legal Copy  
    All files with this flag must exist in the SOE install directory and match the provided checksum for the launcher to consider a legal copy to have been installed.  
      
    At install time the files are copied from the SOE directory to getGamePath(). At verify time if the file in getGamePath() does not match the checksum the file is copied from the SOE directory if the SOE install directory has been removed. An additional test will be done at verify time with files copied from the install directory. They will be tested to see if they are undamaged and match the checksum. If the SOE install directory has been removed to save disk space or does not match the checksum it is downloaded from the game server.  
      
    One additional note. The various retail distributions included more and more of the .tre files to speed up the install by downloading fewer files from the server. The original distribution disks included the following .tre files which are checked to verify a legal distribution:

|  |
| --- |
| bottom.tre |
| data\_animation\_00.tre |
| data\_music\_00.tre |
| data\_other\_00.tre |
| data\_sample\_00.tre |
| data\_sample\_01.tre |
| data\_sample\_02.tre |
| data\_sample\_03.tre |
| data\_sample\_04.tre |
| data\_skeletal\_mesh\_00.tre |
| data\_skeletal\_mesh\_01.tre |
| data\_static\_mesh\_00.tre |
| data\_static\_mesh\_01.tre |
| data\_texture\_00.tre |
| data\_texture\_01.tre |
| data\_texture\_02.tre |
| data\_texture\_03.tre |
| data\_texture\_04.tre |
| data\_texture\_05.tre |
| data\_texture\_06.tre |
| data\_texture\_07.tre |
| default\_patch.tre |
| patch\_00.tre |
| patch\_01.tre |

**Table 1 - Files to be used to validate a commercial install of SWG**

If you want you can also add the following client executables to this category, but most were updated for versions of the client that were shipped between the initial disk and Publish 14.1. If you add them to this category they probably need to be category 2 also so that you make sure that the correct version for the client is downloaded with the client.  
  
 The Launcher manifest has no need to use this action.

* + 3 = Verify Legal copy – Do not copy file  
    This file is required to verify a legal copy but not copied as part of the install.
  + 4 = SOE static files  
    This category is all of the client executables, dlls, config files and executables associated with Publish 14.1.  
      
    SOE layered .tre files so that each new publish that had additional items in the .tre files added new .tre files. It is recommended that SWGEmu custom servers continue to use this philosophy.  
      
    Files in this category will be checked to see if they exist in the SOE install directory. If they do and match the checksum they will be copied instead of downloaded to speed things up for people with later versions of the install disks or still have their fully installed directory from when they played the live game.  
      
    The Launcher manifest has no need to use this action.
  + 5 = SOE Static File – Make Backup  
    This is for configuration files that players may modify. Because of this potential modification potential, any file with a different checksum is copied to <filename>.bak before being replaced to make it easy to restore player edits.
  + 6 = Server extension files  
    For servers that are adding items to their server this is the category for lumping the additional .tre files.  
      
    It would also be used for the server specific swgemu.cfg, swgemu\_live.cfg, swgemu\_login.cfg, swgemu\_preload.cfg and user.cfg files. These will exist in a legacy SOE install directory from when the game was still running but will also not necessarily match your SWGEmu server. They may even be different between the play and test versions of the server. All of these are reasons to place them into this category.  
      
    The Launcher manifest would use this for supplemental files, like profession calculator stored in the launcher directory.
  + 7 = EULA.  
    If the EULA has changed the readEULA flag in the registry is set to False and it is presented after the processing is complete. It is always downloaded to the Launcher Directory.
  + 8 = Add new/Update existing registry entry
  + 9 = Delete registry entry
  + 10+ = Set aside for future use
* Field 3 – This has a different meaning based upon the action flag. These meanings are:
  + 0 – Comment
  + 1 – Comment
  + 2 – MD5 Checksum
  + 3 – MD5 Checksum
  + 4 – MD5 Checksum
  + 5 – String value for the registry entry
  + 6 – Comment
  + 7 – MD5 Checksum

This manifest creation tool will take a CSV file to drive it. The driving CSV file must also have three fields and each field will have the same ordering and values as the manifest file. In the case where in the manifest file field 3 is intended to have the MD5 Checksum the field in the driving CSV file is ignored. All other field values will be copied from the driver file to the manifest file.

Where the action field indicates that field 3 will be a MD5 Checksum the tool will find the file and generate the MD5 Checksum and put it in the manifest.

The tool will allow the user to specify the file to be used to drive the tool, the location where the files to create the MD5 checksum from are located and where to write the manifest file.

A second pass on this tools is desirable. The already described functionality would be a batch mode operation. The second pass to the project would be to add an interactive mode where the driver file would be used to fill in each of the fields and stop to allow the user to change all values but the MD5 Checksum which will be generated based on the specified file.

As a potential extension an add function could be added to the interactive mode. In this mode all fields would start blank and new entries could be added to the manifest.

The format of the driver CSV and the manifest is deliberate. The reason is that an existing manifest file could be used to drive the manifest creation tool. If there is time to fully implement the second pass this means that a developer could start with an empty driver CSV file and type in all of the entries. Then this could be used to dive further work to extend or revise the file.

## Registry Entries

The SWGChoice version of the launcher that this design is based on was going to be enhanced to use configuration information stored in an XML based configuration file. At the time it was considered an advantage to be able to have users edit the configuration if there is a problem. Since the information was not intended to be that great and the coding to use the Microsoft Windows Registry is far simpler so initially that will be how the configuration information will be stored. However, since the revised launcher will be heavily structured a common set of functions will be used to access each configuration item. This use of OOP will hide from the bulk of the program the details of how the configuration is stored and retrieved. So individual items or even all could be moved to a locally stored xml based configuration file without having to change any of the rest of the source code.

### The location of the commercial install of SWG

Based upon research using the original SWG install disks on a clean install of Microsoft windows there seems to be two locations that store a pointer to where the commercial SWG install is placed. These are:

|  |
| --- |
| HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\StarWarsGalaxies\Path |
| HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\SWG\_Client\Path |

**Table 2 - Registry entries containing the commercial SWG install location**

Both of these locations contain the same information so initially the entry under StarWarsGalaxies will be used to see where the commercial game files were installed. Further investigation may give more information and cause this section to be revised.

### Registry entries used by the Launcher

The configuration information will be stored in three sections of the registry. In all examples where <SWG Emu> is listed in the path that should be replaced with the unique name of the server.

**Launcher**

|  |  |  |
| --- | --- | --- |
| HKEY\_CURRENT\_USER\SOFTWARE\<SWG Emu>\Launcher | | |
|  | HasCalc | Yes|No – Yes if the server has a profession calculator |
|  | Calc | Full path to the profession calculator executable if there is one |
|  | HasEULA | Yes|No – Yes if the server has an End User License Agreement |
|  | readEULA | Yes|No – Yes if the user has said that they agree to the EULA |
|  | HasForum | Yes|No – Yes if the server operates a forum |
|  | Forum | The URL to the top page of the forum |
|  | HasTest | Yes|No – Yes if the server has a test server that needs to be configured separately from the play server |
|  | URL | The URL where the server version, manifest and update files for the launcher to get updates from. |
|  | Version | The current version of the launcher package. This may be different from the version of the executable |

**Table 3 - Registry entries for the Installer configuration**

**Play Server**

|  |  |  |
| --- | --- | --- |
| HKEY\_CURRENT\_USER\SOFTWARE\<SWG Emu>\Game | | |
|  | Patchnotes | The URL where the Launcher can download the rtf file that contains the play server patch notes. This must include the name of the file with the .rtf extension |
|  | Path | The full path to where the client files are located |
|  | Status | The URL where the play server status information can be downloaded as an rtf file. This includes the full file name including the rtf extension. |
|  | URL | The URL where the server version, manifest and update files for the game server client to get the updates from. |
|  | Version | The current version of the game server client package. |

**Table 4 - Registry entries for the Play Server client information**

**Test Server**

|  |  |  |
| --- | --- | --- |
| HKEY\_CURRENT\_USER\SOFTWARE\<SWG Emu>\Test | | |
|  | Patchnotes | The URL where the Launcher can download the rtf file that contains the test server patch notes. This must include the name of the file with the .rtf extension |
|  | Status | The URL where the test server status information can be downloaded as an rtf file. This includes the full file name including the rtf extension. |
|  | URL | The URL where the server version, manifest and update files for the test server client to get the updates from. |
|  | Version | The current version of the test server client package. |

**Table 5 - Registry entries for the Test Server client information**

# Development Environment Operations

## Setting the executable version number

For those that have not used Visual Studio setting the version number that is embedded in the executable can be a little hard to locate. The Launcher displays the assembly version number. To get to the screen to set this you do the following:

* Select the Project pulldown
* From the Project pulldown select the <Project Name> Properties option. This will bring up a new window.
* In this new window on the left hand side select Application
* On this screen there will be an Assembly Information… button. Use that button
* A new window will pop up. On this window you will be able to set:
  + Title
  + Description
  + Company
  + Product
  + Copyright
  + Trademark
  + Assembly version
    - Major
    - Minor
    - Build
    - Revision
  + File version
    - Major
    - Minor
    - Build
    - Revision
  + GUID
  + Neutral Language
  + Make assembly COM-Visible

The splash screen and title bar will display the Major field with at least one digit and as many as needed. The Minor field is always displayed with two digits, the Build field is always displayed with three digits and Revision is always displayed with four digits. Use this information and establish rules about when to bump each of these numbers.

My suggestion is that Major is not changed and is the release version of the generic launcher. The other three fields should be set based on your internal rules. This is only a suggestion though.

Update the Copyright field to reflect the changes that you make. The delivered version of the field indicates the earliest copyright associated with the launcher. The latest date should reflect the year of your latest changes.

# Class documentation

## WebFileDownloader Class

This file was written and presented as an example of code on how to do a downloader with real time progress status on www.vbforums.com. Since it provided all the functionality that was desired it is used as is. The SWGChoice Launcher did not break the file into chunks and was the equivalent of the DownloadFile function in this class. This version of the launcher will use the DownloadFileWithProgress function in the class. The main program which updates the progress bars with be set up to catch and process the four events that function raises to communicate progress or failure.

The four events are FileDownloadSizeObtained, AmountDownloadedChanged, FileDownloadComplete and FileDownloadFailed.

### DownloadFileWithProgress Function

This function takes two strings. The first string is the full URL to where to get the file that is being copied to the local system. The second path is the path to where to download the file to.

The function passes file size and progress information along using an integer data size. This will limit the maximum size of a single downloaded file to 2,147,483,647 bytes. None of the SOE files come close to this size and none of the files associated with this project come close to that size so there should be no problem. If a server development project decides to dump the SOE files and go with mega files or if the additions to the game are large, then you will have to watch the file size and keep them below this number for everything to function properly.

### FileDownloadSizeObtained Handler

In the example the file size is received from the server as a long data type. This is converted to an integer. The progress bar is initialized so that the maximum value is the full length of the file and the current value is 0. This handler is called once per file and does the initialization of the progress bar.

### AmountDownloadedChanged Handler

This handler receives the current file pointer position as a long date type. This is converted to an integer and the current value of the progress bar is set to how much has been downloaded. This handler is called after every chunk has been downloaded and is what causes the progress bar to move.

### FileDownloadComplete Handler

This handler is called when the entire file has been downloaded. It sets the current value on the progress bar to the maximum value that was set when the progress bar was initialized.

### FileDownloadFailed Handler

The experience with the SWGChoice launcher is that this is a rare occurrence. In nearly a year of operation this error was never observed. But since it is a possibility it needs to be caught. Currently it is coded to pop up a message box with the error text from the system.

It should probably call presentStatus() to tell the user to check network connectivity and restart the launcher and try again.

## textUtils class

This class is a port of helper routines that were part of the SWGChoice launcher. These routines were written to simplify and make consistent updating text information presented to the user. It was originally created to have a consistent look and feel between the SWGChoice launcher and manifest creation tool.

The design behind these functions was that they would be given a text box to write the information, the information and how long to pause for people to read the information before continuing. The Pause time is in milliseconds so 1000 is a one second delay.

### clearStatus(ByVal box As TextBox)

This function only takes the TextBox object as a parameter. It clears the contents of that TextBox and returns immediately.

### presentStatus(ByVal box As TextBox, ByVal text As String, ByVal delay As Double)

This function takes the TextBox, Text to present and a delay time. It combines clearStatus() and extendStatus() into one call. So it clears the content of the TextBox and then writes the Text into it and waits the prescribed amount of time to allow the user to read the text before possibly updating the TextBox.

### extendStatus(ByVal box As TextBox, ByVal text As String, ByVal delay As Double)

This function adds text to the current TextBox. It does no formatting so if a space or newline is desired it needs to be a part of the string passed to this function.

### presentText(ByVal box As TextBox, ByVal text As String, ByVal delay As Double)

This function does some formatting of the text that it receives in that it appends a newline to the end of the text so that the next text sent to the TextBox starts on a new line. It adds the newline to the text and calls presentStatus().

### extendText(ByVal box As TextBox, ByVal text As String, ByVal delay As Double)

This function does some formatting of the text that it receives in that it appends a newline to the end of the text so that the next text sent to the TextBox starts on a new line. It adds the newline to the text and calls extendStatus().

### blankLine(ByVal box As TextBox, ByVal delay As Double)

This function takes no text and simply calls extendStatus() with a newline character as the text to write.

## configUtils Class

This class was written to simplify getting and setting configuration information in the Launchpad. The names of the functions are intended to help the code be self-documenting. The get functions either return a Boolean value or a string with the information. The set functions return nothing and take a string to set the configuration data to. The name of the function implies where to get or set the information.

To simplify modifying the launcher to store information in the registry based on the name of the server all operations on the registry where name in this document has <SWG Emu> this class uses the static string **SWGServer**. When customizing the launcher changing this one variable will adjust every function that acts upon the registry.

### Commercial SWG Functions

This function return from the Registry the location of where the commercial SOE installer put the game files.

#### Function getSWGLocation() As String

This returns the value stored in HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\StarWarsGalaxies\Path or Nothing of it does not exist.

### Launcher Functions

These functions inform the Launcher about what the server does and does not support in the way of functionality. They also tell the launcher about version information and download location relating to the launcher itself.

#### Function getLauncherVersion() As String

This function returns a string with the launcher package version that is currently stored on the system. The term launcher package is used since there may be a need to update server utilities like the EULA or profession calculator but not the actual launcher code. So at compile time the version of the launcher executable will be compiled into it and displayed on the splash screen and toolbar.

The Launcher, Play client and optional Test client version should be presented to the user as part of a request for info.

#### Sub setLauncherVersion(ByVal setValue As String)

This subroutine updates the launcher version in the registry with the value given it. It would be called after a successful update of the launcher package.

#### Function getLauncherURL() As String

This function returns the full URL, with trailing “/”, of the http server that will provide the version, manifest and updated files for the launcher package. The URL for the Launcher, Play Client and Test Client are stored independently so that if needed different locations could be used for each set of files. Generally the same server would be used for all three packages with each in a different subdirectory.

#### Function getLauncherHasTest() As Boolean

This function returns True if the registry entry is set to “Yes” to indicate that there is a separately maintained Test Server Client for this server. If this is set to True the registry entries associated with the Test Server Client must exist and have legitimate values set. If set to False they will return False or Nothing.

#### Function getLauncherHasEULA() As Boolean

This function returns True if the registry entry is set to “Yes” to indicate that the server does require that the players agree to an End User License Agreement in order to play on the server.

#### Function getLauncherReadEULA() As Boolean

This function returns True if the registry entry is set to “Yes” to indicate that the player has pressed the accept button when presented with the EULA to read. The launcher is coded so that if getLauncherHasEULA() returns True then getLauncherReadEULA() must also return True for the launcher to actually launch the game client. In the situation where getLauncherHasEULA() returns True and getLauncherReadEULA() returns false pressing the launch game button will present the user with the read and accept EULA screen instead of launching the game. If getLauncherHasEULA() returns False then getLauncherReadEULA() is not checked.

#### Sub setLauncherReadEula(ByVal setValue As String)

This subroutine is called to set the value to “Yes” when the user presses the accept button to indicate that they have read, understand and accept the EULA. If the launcher is processing a manifest flag 7 entry this function is called to set the value to “No” so that the updated EULA is presented to the player to accept of decline again.

#### Function getLauncherEULA() As String

This function was intended to get the URL that contained the EULA. Given that the players should have a copy of the EULA on their game system the registry entry was removed and this function simply returns the string “EULA.rtf”. It was retained as a function so that if servers wanted to change the name of the file or use a downloaded rtf file it would be easier for them to change the code.

#### Function getLauncherHasCalc() As Boolean

This function returns True if the server has a profession calculator. Currently it is expected that servers will use a local executable file. If they make an online version the code will need to be tweaked to open a browser to that URL.

#### Function getLauncherCalc() As String

This function return the full path to where the profession calculator is installed. The expectation is that most SWGEmu based servers will ship with kodan’s profession calculator and that it will be installed by the Launcher installer into the same directory as the launcher. This registry entry is specified as being the full path so that is can accommodate a separate installer for the profession calculator which could be stored separately. If it is stored separately code would need to be added to fill in the location in this registry entry.

#### Function getLauncherHasForum() As Boolean

This function returns True if the server operates a forum site.

#### Function getLauncherForum() As String

This function returns the full URL to the top page of the forums for the game server.

### Game Server Functions

These functions return information about the play server client install.

#### Function getGameVersion() As String

This function returns a string with the play server client package version that is currently stored on the system.

The Launcher, Play client and optional Test client version should be presented to the user as part of a request for info.

#### Sub setGameVersion(ByVal setValue As String)

This subroutine updates the play server client version in the registry with the value given it. It would be called after a successful update or verification of the play server client package.

#### Function getGameURL() As String

This function returns the full URL, with trailing “/”, of the http server that will provide the version, manifest and updated files for the play server client package. The URL for the Launcher, Play Client and Test Client are stored independently so that if needed different locations could be used for each set of files. Generally the same server would be used for all three packages with each in a different subdirectory.

#### Function getGameManifest() As String

This function returns a complete URL to where the game server client manifest can be downloaded from. This function returns getGameURL() & “Manifest”. If in customization the server operator wants to use a different name for the manifest file, then this function needs to be modified.

#### Function getGamePatchNotes() As String

This function returns a complete URL to where the rtf file containing the current play server Patch Notes can be downloaded. The complete URL is stored in the registry so that it may be stored in a location where it could be shared with the server web site.

#### Function getGameStatus() As String

This function returns a complete URL to where the rtf file containing the current play server Status can be downloaded. The complete URL is stored in the registry so that it may be stored in a location where it could be shared with the server web site.

#### Function getGamePath() As String

This function returns the complete file path including the trailing “\” to where the Play Server Client package has been installed.

#### Sub setGamePath(ByVal setValue As String)

This subroutine stores the complete file path including the trailing “\” to where the Play Server Client package has been installed. The launcher calls this function after it creates this directory during the install process.

### Test Server Functions

These functions return information about the test server client install. The information is only stored in the registry only if getLauncherHasTest() returns True.

#### Function getTestVersion() As String

This function returns a string with the test server client package version that is currently stored on the system.

The Launcher, Play client and optional Test client version should be presented to the user as part of a request for info.

#### Sub setTestVersion(ByVal setValue As String)

This subroutine updates the test server client version in the registry with the value given it. It would be called after a successful update or verification of the test server client package.

#### Function getTestURL() As String

This function returns the full URL, with trailing “/”, of the http server that will provide the version, manifest and updated files for the play server client package. The URL for the Launcher, Play Client and Test Client are stored independently so that if needed different locations could be used for each set of files. Generally the same server would be used for all three packages with each in a different subdirectory.

#### Function getTestManifest() As String

This function returns a complete URL to where the test server client manifest can be downloaded from. This function returns getTestURL() & “Manifest”. If in customization the server operator wants to use a different name for the manifest file, then this function needs to be modified.

#### Function getTestPatchNotes() As String

This function returns a complete URL to where the rtf file containing the current test server Patch Notes can be downloaded. The complete URL is stored in the registry so that it may be stored in a location where it could be shared with the server web site.

#### Function getTestStatus() As String

This function returns a complete URL to where the rtf file containing the current test server Status can be downloaded. The complete URL is stored in the registry so that it may be stored in a location where it could be shared with the server web site.

#### Function getTestPath() As String

This function returns the complete file path including the trailing “\” to where the Play Server Client package has been installed. Since the install location of the Test Server client is relative to the play server this function returns getGamePath() & “Testserver\”. This is why there is no setTestPath() subroutine.

## installUtils class

This class will contain all of the functions and subroutines that are used to validate that there appears to be a legal copy of the commercial version of Star Wars Galaxies on the system.

### Secondary routines

These routines perform the low level functions of the package

#### Function PrintByteArray(ByVal array() As Byte) As String

This function takes the byte array created by the ComputeHash() and converts it to a string.

#### Function getFileMD5(ByVal file As String) As String

This function opens the local file that is passed in and returns the calculated MD5 checksum as a printable string.

#### Function isGoodFile(ByVal file As String, ByVal cSum As String) As Boolean

This function calls getFileMD5() with the file to get the checksum of. If the file does not exist the function returns Nothing. If Nothing has been returned by getFileMD5() the function returns False. If a valid string is returned it is compared to the cSum that was passed in to compare it to. If the checksums do not match the function returns False. If they match it returns True.

### Primary routines

These routines are the high level task functions.