HOMEWORLD2

Archive Tool DOCUMENTATION

Document Version 1.01

0 Introduction

What is the Archive Tool? The archive tool gives modders the ability to package their data for distribution to end-users. It also allows modders to create scenario packs and add-ons. The archive tool is a command line tool.

This document will describe how to use the archive tool.

1 Command Line Arguments

The following table lists the command line arguments for the tool as well as a description of what they do. The parameters to the tool are specified with angle brackets; replace the brackets and their contents with the appropriate argument.

Argument	Description
-a <archive file>	This option specifies the archive file. Substitute <archive file> with the path and filename to the archive.
-c <buildfile> -r <rootpath>	These options are used to create an archive. The archive specified with the –a option will be created. The <buildfile> option should be set to the path and filename of the build script file. The <rootpath> option should be set to the folder where the files in the build script are located. See section 4 for info on the build script syntax.

-l	List the contents of the archive file specified with the –a option.
-e <extract location>	Extract the contents of the archive file specified with the –a option. It will extract all of the files (including folders) to the location specified by <extract location>
-t	Test the archive file specified with the –a option. This will calculate the CRC for each stored file and compare with the stored version.
-hash	Output the hash on the archive file. This can be used to uniquely identify the archive.
-v	Use verbose output. This can be used with any of the above options. It will display all possible warnings/errors.

2 Archive Tool Operations

Using the specified command line arguments listed above there are four main operations that you can perform with the archive tool. These operations are listed with samples in the following table.

Operation	Example
Create Archive	-a c:\archives\archive.sga -c c:\archives\buildarchive.txt -r c:\IC\Data This example will create the archive c:\archives\archive.sga using the build script c:\archives\buildarchive.txt. The files listed in the build script will be found using the root path c:\IC\Data. More info on building an archive is found on section 4.
Extract Archive	-a c:\archives\archive.sga -e c:\archives\archivecontents This example will extract the contents of the archive file c:\archives\archive.sga to the path c:\archives\archivecontents. It will extract all of the files and re-build the original folder tree.
List Archive contents	-a c:\archives\archive.sga -l This example will list the contents of the archive file c:\archives\archive.sga. It will list the folder and file name as well as the original and compressed file sizes. The method of file storage will also be listed. Adding the text "> <destfile>" after the argument will cause the output to be written to the location <destfile>, useful for examining in Excel.
Test the Archive	-a c:\archives\archive.sga -t This example will test the archive. If a file CRC doesn't match the version stored the test will fail. Useful when the archive has been transferred to verify its integrity.

3 Building an Archive

Building an Archive is the most complicated operation as a build script is parsed by the tool to determine what files to add and how to add them. This section will describe the syntax of the build script and a sample usage.

3.1 Build Script Syntax

The build script is composed of 4 parts, the Archive info, the Table of Contents info, the File Settings, and finally the list of files to store in the archive. Comments may be added to the build script two forward slashes // must be placed at the start of the line. Empty lines are also ignored during parsing.

Conventions

Grey italic font indicates required script syntax

< bold text inside of angle brackets is user supplied content >

Normal text describes what the line does.

Syntax:

*Archive name="***<archive name>***"*

Tells the tool to create an archive. The archive will have the name **<archive name>** embedded inside the archive header. This name is only used for identification purposes.

*TOCStart name="***<name>***" alias="***<alias>***" relativeroot="***<relative folder>***"*

Tells the tool to create a new Table of Contents (TOC). The TOC is used by the game to quickly retrieve what files are in the archive as well as basic file into, e.g. size. The **<name>** is used to identify the TOC contents; it is for identification purposes only. The **<alias>** tells the game how to map the archive into the game's file system; this should always be set to data. The **<relative folder>** is used in conjunction with the -r command line option to locate the files listed, this can optionally be empty if the -r specifies the root folder fully.

*FileSettingsStart defcompression="***<default storage>***"*

Starts the File Settings section. This section contains the settings for how different files should be stored in the archive as well as the option to not add files to the archive. The **<default storage>** parameter should be set to the way you would like files stored by default. The following table lists the types of storage available:

Ordinal	Storage Method
0	No compression just stores the files raw.
1	Compressed, files are decompressed, as they are read. Should use this for very large files.
2	Compressed, files are decompressed in one step then read back from memory, useful for small files, and especially .LUA files.

*Override wildcard="***<wildcard>***" minsize="***<minbytes>***" maxsize="***<maxbytes>***" ct="***<storage>***"*

This directive can occur multiple times for each type of override you would like to set to change the default storage type. The first matching override directive that is found for a file will be used, so the order is important. The file is matched against the three criteria specified. The **<wildcard>** argument can be used to select a certain subset of files, * and ? wildcards are supported. The **<minbytes>** argument sets the minimum file size in bytes to match this override to; -1 means there is no minimum file size. The **<maxbytes>** argument sets the maximum file size in bytes to match this override to; -1 means there is no max size. The **<storage>** argument is used to specify the custom storage to use for files that match the criteria.

*SkipFile wildcard="***<wildcard>** *" minsize="***<minbytes>** *" maxsize="***<maxbytes>** *"*

This directive can occur multiple times to optionally exclude files from the archive. This can be useful to remove certain subsets of files. The **<wildcard>** argument can be used to select a certain subset of files, * and ? wildcards are supported. The **<minbytes>** argument sets the minimum file size in bytes to match this override to; -1 means there is no minimum file size. The **<maxbytes>** argument sets the maximum file size in bytes to match this override to; -1 means there is no max size. If the file matches these criteria it isn't added to the archive.

FileSettingsEnd

This tells the tool that all of the Settings have been specified and the file list is next.

< file listing … >

The file list can contain as many files as you like. The files can be specified in two ways.

1) Relative to the path that is made up of the root folder specified with the -r command line, and the **<relative folder>** from the *TOCStart* directive.

2) Fully qualified path name. The path must match the Root folder comprised of the path passed using the -r command line and the **<relative folder>** specified in the TOCStart directive.

A dos command of "dir /s /a-d /b *.* > <filelist>" on the directory that you would like files added to an archive from will generate a filelist that can be inserted into a build script. This can be used to generate a complete file listing since we don't support file additions using wildcards.

TOCEnd

This specified that the Files have all been listed and the archive should be built. Check out Appendix A for a sample folder listing and a script that will create an archive.

4 Appendix A Sample Script

Here is a directory listing of files:

```
Volume in drive C is unlabeled      Serial number is
 Directory of  C:\temp\test\*.*

 2/03/03   0:02         <DIR>    .
 2/03/03   0:02         <DIR>    ..
 2/03/03   1:35         <DIR>    TextFiles
 2/01/03  15:54       2,965,745  SampleData.test
      2,965,745 bytes in 1 file and 3 dirs    2,969,600 bytes allocated

 Directory of  C:\temp\test\TextFiles\*.*

 2/03/03   1:35         <DIR>    .
 2/03/03   1:35         <DIR>    ..
 2/01/03  12:38              36  TestFile.txt
            36 bytes in 1 file and 2 dirs    4,096 bytes allocated

    Total for:  C:\temp\test\*.*
      2,965,781 bytes in 2 files and 5 dirs    2,973,696 bytes allocated
```

Here is a build file that will create an archive with these files inside of it.

Contents of c:\temp\Test.build.lst

```
Archive name="ICTestArchive"

TOCStart name="ICTestData" alias="Data" relativeroot=""

FileSettingsStart defcompression="1"

        // Anything less than 100 bytes, just store, don't compress
        Override wildcard="*.*" minsize="-1" maxsize="100" ct="0"

        // Any of these file types we just store, don't compress
        Override wildcard="*.mp3" minsize="-1" maxsize="-1" ct="0"
        Override wildcard="*.wav" minsize="-1" maxsize="-1" ct="0"
        Override wildcard="*.jpg" minsize="-1" maxsize="-1" ct="0"

        // Lua files we always compress, and then decompress in one shot
        Override wildcard="*.lua" minsize="-1" maxsize="-1" ct="2"

        // Skip all files that meet this criteria
        SkipFile wildcard="*emptyfile.txt" minsize="-1" maxsize="-1"

FileSettingsEnd

// File specified relative to the root location passed in using the -r option
TextFiles\TestFile.txt
// File specified as a fully qualified path name
C:\Temp\test\SampleData.test

TOCEnd
```

Here is the output that the archive tool records when executing this script.

```
[c:\icpc\bin]Archive.exe -a c:\temp\test.sga -c "c:\temp\Test.build.lst" -r c:\temp\test
Parsing Build File 'c:\Temp\Test.build.lst'
Creating TOC Entry Name:'ictestdata'  Alias:'data'
Parsing Compression Overrides
Adding All files to TOC
Writing Archive Header
Writing Archive Root Info
Writing TOC Entries
Writing Folder Entries
```

```
Writing File Entries
Writing String Data Base
Writing File Data
FileName
Compression Type
testfile.txt
Store
sampledata.test
Compress Stream
Calculating Hash

Build Operation took 0.80 seconds.
```

5 Appendix B

Homeworld2 Game Rules mods and Total Conversion mods are not subject to auto download of content. All players that are to join the game need to have installed them prior to launching the game.

5.1 Game Rules

Game Rules mods are automatically loaded as long as they are present in your bin\GAMERULES folder.

5.2 Total Conversion

For total conversion mods you need to use the following command line argument:

-mod nameofyourmod ,;nameofyoursecondmod

example:

-mod modifiedUI.big ,;modifiedAI.big

You can also list all of your mods in a text file that lists in the mods along the following format

nameofyourmod

nameofyoursecondmod

example:

modifiedUI.Big

modifiedAI.Big

in which case the text file is loaded like this

-mod @mylistofmod.txt