Unreal Tournament Package File Format

by Sapphire

> "Abandon all hope ye who try to parse this file format."
>
> —Tim Sweeney, *Unreal Packages*

After having recently finished writing a basic JavaScript plugin to parse this file format, I felt a little frustrated at what I considered to be the lack of easily understandable information when I began researching this topic.

Of the few resources out there, I couldn't find any which actually go through an existing package step by step, which I think would have helped me a lot when starting out. As a result, I've decided to write this brief guide in the hope that it may help someone.

As stated, there are a few existing guides, some parts of which I have copied/paraphrased here; the rest I have written myself.

The package used in this example is CTF-Face, aka "Facing Worlds".

Contents

Overview

The UT package format has the following structure:

Section	Description
Package Header	Contains information about the package, such as the size and offset of tables within the package.
Name Table	Contains a list of human-readable Unreal names (which correspond to the UnrealScript "Name" data type).
Import Table	Contains a list of objects in other packages which this package refers to.
Export Table	Contains a list of objects contained in (aka "exported by") this package.
Data	The actual package data itself (textures, sounds, brushes, scripts, etc).

Long int/DWORD data types are signed, 4 bytes long, and stored in little-endian format.

Package Header

The package header always begins at offset 0. If the package version is less than 68, "Heritage" count/offset values may also be present instead of the GUID.

Offset		Size (bytes)	Name	Description
0	`0x00`	4	Signature	Always `0x9E2A83C1`.
4	`0x04`	4	File Version	Version of the engine which saved the file.
8	`0x08`	4	Package Flags	Bitflags describing the package (see Package Flags section).
12	`0x0C`	4	Name Count	Number of names stored in the name table. Always >= 0.
16	`0x10`	4	Name Offset	Offset into the file of the name table, in bytes.
20	`0x14`	4	Export Count	Number of exported objects in the export table. Always >= 0.
24	`0x18`	4	Export Offset	Offset into the file of the export table, in bytes.
28	`0x1C`	4	Import Count	Number of imported objects in the import table. Always >= 0.
32	`0x20`	4	Import Offset	Offset into the file of the import table, in bytes.
36	`0x24`	16	GUID	Globally unique identifier; a unique number used for caching and avoiding package naming/version conflicts.

Offset		Size (bytes)	Name	Description
52	0x34	4	Generation Count	Unknown.
-	-	4	Generation *n* Export Count	-
-	-	4	Generation *n* Name Count	-

Example

The package header for CTF-Face is shown below. Hover over the table rows to highlight the corresponding bytes within the raw header data.

Name	Value
Signature	`0xC1832A9E`
File Version	`0x44000000` → 68
Package Flags	`0x01000000` → `PKG_AllowDownload`
Name Count	`0x43050000` → 1,347
Name Offset	`0x40000000` → 64
Export Count	`0x31040000` → 1,073
Export Offset	`0x2AAD0E00` → 961,834
Import Count	`0x8F000000` → 143
Import Offset	`0xDBA70E00` → 960,475
GUID	`0x642E72689581D311A26B00E0811032D2`*
Generation Count	`0x01000000` → 1
Export Count	`0x31040000` → 1,073
Name Count	`0x43050000` → 1,347

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x00	C1	83	2A	9E	44	00	00	00	01	00	00	00	43	05	00	00
0x10	40	00	00	00	31	04	00	00	2A	AD	0E	00	8F	00	00	00
0x20	DB	A7	0E	00	64	2E	72	68	95	81	D3	11	A2	6B	00	E0
0x30	81	10	32	D2	01	00	00	00	31	04	00	00	43	05	00	00

***Fig. 1**—Package header data for CTF-Face.unr; 64 bytes total.*

*When downloading a package in-game, files are saved using the GUID as their filename. With each of the four long int values read in little-endian format, CTF-Face would be saved as "68722E6411D38195E0006BA2D2321081.uxx".

Name Table

The name table contains all unique object names within the package. The structure is as follows:

Name	Size (bytes)	Description
Object Name Length	1	String length of the null-terminated object name (including the null byte), in bytes.
Object Name	*n*	The name of the object. Objects may share names (e.g. "Texture").
Object Flags	4	Bitflags describing the object (see Object Flags section).

Example

As indicated in the package header, the name table for CTF-Face begins at byte 64. The first five values are shown below:

#	Offset		Object Name Length		Object Name		Object Flags	
	Hex	Dec	Hex	Dec	Hex	Value	Hex	Value
1	0x40	64	0x05	5	0x4E6F6E65	"None"	0x10040704	RF_Public RF_NeedLoad RF_HighlightedName RF_LoadForEdit RF_ErrorShutdown
2	0x4A	74	0x07	7	0x566563746F72	"Vector"	0x10040704	RF_Public RF_NeedLoad RF_HighlightedName RF_LoadForEdit RF_ErrorShutdown
3	0x56	86	0x08	8	0x4F555453494445	"OUTSIDE"	0x10000700	RF_NeedLoad RF_HighlightedName RF_ErrorShutdown
4	0x63	99	0x0C	12	0x506F696E74526567696F6E	"PointRegion"	0x10000700	RF_NeedLoad RF_HighlightedName RF_ErrorShutdown
5	0x74	116	0x07	7	0x526567696F6E	"Region"	0x10000700	RF_NeedLoad RF_HighlightedName RF_ErrorShutdown

```
     00  01  02  03   04  05  06  07   08  09  10  11   12  13  14  15
0x40 05  4E  6F  6E   65  00  10  04   07  04  07  56   65  63  74  6F
0x50 72  00  10  04   07  04  08  4F   55  54  53  49   44  45  00  10
0x60 00  07  00  0C   50  6F  69  6E   74  52  65  67   69  6F  6E  00
0x70 10  00  07  00   07  52  65  67   69  6F  6E  00   10  00  07  00
```

***Fig. 2**—Name table data for CTF-Face.unr; 64 bytes are shown, representing five name table entries.*

Import Table

The import table contains *n* entries, where *n* is the import count specified in byte 28 of the header. Each table entry is structured as follows:

Name	Data Type	Description
Class Package	Compact Index*	Name table index for the package which this object's class object resides in.
Class Name	Compact Index	Name table index for this object's class.
Package Index**	Long Int	The index of the package this object resides in. This can be an index into the import table itself, or *NULL*.
Object Name	Compact Index	Name table index for this object.

*See the Compact Index section to understand how values are ascertained from this data type.

**See Object References.

Example

The first five entries of CTF-Face's import table are as follows:

#	Class Package		Class Name		Package Index		Object Name	
	Bytes	Value	Bytes	Value	Bytes	Value	Bytes	Value
1	0x610A	673 → "Core"	0x6B06	427 → "Package"	0x00000000	*NULL*	0x7602	182 → "Engine"
2	0x610A	673 → "Core"	0x6B06	427 → "Package"	0x00000000	*NULL*	0x440E	900 → "SkyCity"
3	0x610A	673 → "Core"	0x6B06	427 → "Package"	0x00000000	*NULL*	0x4112	1,153 → "ShaneChurch"
4	0x610A	673 → "Core"	0x700A	688 → "Class"	0xFFFFFFFF	-1 → 0 → "Engine"	0x0C	12 → "Brush"
5	0x610A	673 → "Core"	0x6B06	427 → "Package"	0xFEFFFFFF	-2 → 1 → "SkyCity"	0x3B	59 → "Base"

```
          00  01  02  03   04  05  06  07   08  09  10  11   12  13  14  15
0x0EA7DB  61  0A  6B  06   00  00  00  00   76  02  61  0A   6B  06  00  00
0x0EA7EB  00  00  44  0E   61  0A  6B  06   00  00  00  00   41  12  61  0A
0x0EA7FB  70  0A  FF  FF   FF  FF  0C  61   0A  6B  06  FE   FF  FF  FF  3B
```

***Fig. 3**—Import table data for CTF-Face.unr; 48 bytes are shown, representing five objects.*

Export Table

As with the import table, the export table contains *n* entries, where *n* is the export count specified in byte 20 of the header. Each table entry is structured as follows:

Name	Data Type	Description
Class Index	Compact Index	Points to the class object describing the class of this object.
Super Index	Compact Index	If this is a field (a struct, class, property, or another field subclass), points to the superfield object of the field.
Package Index	Long Int	Points to the package object describing the package this object resides in.
Object Name	Compact Index	This object's name.
Object Flags	Long Int	Bitflags describing the object.
Serial Size	Compact Index	Size of the object's serialised data, in bytes.
Serial Offset	Compact Index	If >= 0, offset into this file of the start of the object's serialised data.

Example

The first three entries of CTF-Face's export table are as follows:

#	Class Index		Super Index		Package Index		Object Name		Object Flags		Serial Size		Serial Offset	
	Bytes	Value	Bytes	Value	Bytes	Value	Bytes	Value	Bytes	Value	Bytes	Value	Bytes	Value
1	0xF801	-120 → "LevelInfo"	0x00	*NULL*	0x00000000	*NULL*	0x6F02	175 → "LevelInfo0"	0x01000702	RF_Unreachable RF_NeedLoad RF_HighlightedName RF_NeedPostLoad	0x4E04	270 bytes	0x6BB302	19,691 or 0x4CEB
2	0x8E	-14 → "Light"	0x00	*NULL*	0x00000000	*NULL*	0x7903	249 → "Light0"	0x01000702	RF_Transactional RF_Public RF_Suppress RF_InEndState RF_LoadForClient RF_Native RF_ErrorShutdown RF_DebugPostLoad	0x5001	80 bytes	0x79B702	19,961 or 0x4DF9
3	0x8E	-14 → "Light"	0x00	*NULL*	0x00000000	*NULL*	0x7C03	252 → "Light5"	0x01000702	RF_Transactional RF_Public RF_Suppress RF_InEndState RF_LoadForClient RF_Native RF_ErrorShutdown RF_DebugPostLoad	0x5001	80 bytes	0x49B902	20,041 or 0x4E49

```
          00  01  02  03   04  05  06  07   08  09  10  11   12  13  14  15
0x0EAD2A  F8  01  00  00   00  00  00  6F   02  01  00  07   02  4E  04  6B
0x0EAD3A  B3  02  8E  00   00  00  00  00   79  03  01  00   07  02  50  01
0x0EAD4A  79  B7  02  8E   00  00  00  00   00  7C  03  01   00  07  02  50
0x0EAD5A  01  49  B9  02   --  --  --  --   --  --  --  --   --  --  --  --
```

***Fig. 4**—Export table data for CTF-Face.unr; 52 bytes are shown, representing three objects.*

Package Flags

Packages may have several "flags" or properties, as detailed in the table below. The package flags value for CTF-Face is `0x01` which corresponds to the PKG_AllowDownload flag.

Name	Value	Description
PKG_AllowDownload	`0x0001`	Allow downloading package. *"[This should be] disabled for game publisher distribution (so that people aren't able to download retail files freely)."*
PKG_ClientOptional	`0x0002`	Purely optional for clients.
PKG_ServerSideOnly	`0x0004`	Only needed on the server side. *"For example, the IpDrv.u package is tagged as ServerSideOnly, because it contains classes like the server-uplink which only need to run on the server, and will be frequently updated."*
PKG_BrokenLinks	`0x0008`	Loaded from linker with broken import links.
PKG_Unsecure	`0x0010`	Not trusted.
PKG_Need	`0x8000`	Client needs to download this package.

A package's flags can be read by performing a bitwise AND on its flag value with each of the pre-defined flag values. For example, if a package had a package flag of 11 (or `0x0B` in hex):

```
0x0B & 0x01 // 1 - PKG_AllowDownload flag present
0x0B & 0x02 // 2 - PKG_ClientOptional flag present
0x0B & 0x04 // 0 - PKG_ServerSideOnly flag NOT present
0x0B & 0x08 // 1 - PKG_BrokenLinks flag present
```

The PKG_Unsecure and PKG_Need flags aren't present as their values are greater than `0x0B`.

Object Flags

As with the package file itself, objects (i.e. the data contained within the package) can have several flags as well:

Name	Value	Description
RF_Transactional	`0x00000001`	Object is transactional (i.e. supports editor undo/redo).
RF_Unreachable	`0x00000002`	Object is not reachable on the object graph.
RF_Public	`0x00000004`	Object is visible outside its package (i.e. can be referenced by external package files).
RF_TagImp	`0x00000008`	Temporary import tag in load/save.
RF_TagExp	`0x00000010`	Temporary export tag in load/save.
RF_SourceModified	`0x00000020`	Modified relative to source files.
RF_TagGarbage	`0x00000040`	Check during garbage collection.
RF_NeedLoad	`0x00000200`	During load, indicates object needs loading.
RF_HighlightedName	`0x00000400`	A hardcoded name which should be syntax-highlighted.
RF_InSingularFunc	`0x00000800`	In a singular function.
RF_Suppress	`0x00001000`	Suppressed log name.
RF_InEndState	`0x00002000`	Within an EndState call.
RF_Transient	`0x00004000`	Can't be saved or loaded.
RF_PreLoading	`0x00008000`	Data is being preloaded from file.
RF_LoadForClient	`0x00010000`	In-file load for client.
RF_LoadForServer	`0x00020000`	In-file load for server.
RF_LoadForEdit	`0x00040000`	In-file load for edit.
RF_Standalone	`0x00080000`	Keep object around for editing even if unreferenced.
RF_NotForClient	`0x00100000`	Don't load this object for the game client.
RF_NotForServer	`0x00200000`	Don't load this object for the game server.
RF_NotForEdit	`0x00400000`	Don't load this object for the editor.
RF_Destroyed	`0x00800000`	Object Destroy has already been called.

Name	Value	Description
RF_NeedPostLoad	`0x01000000`	Object needs to be postloaded.
RF_HasStack	`0x02000000`	Has execution stack.
RF_Native	`0x04000000`	Native (UClass only).
RF_Marked	`0x08000000`	Marked (for debugging).
RF_ErrorShutdown	`0x10000000`	ShutdownAfterError called.
RF_DebugPostLoad	`0x20000000`	For debugging Serialize calls.
RF_DebugSerialize	`0x40000000`	For debugging Serialize calls.
RF_DebugDestroy	`0x80000000`	For debugging Destroy calls.

Object References

Objects within a package may reference other objects. To differentiate between an object in the export table and the import table, a positive value is given for export entries, and negative for the import. To "convert" a negative value to a usable index, remove the negative sign and subtract 1 (e.g. -5 → 5 → 4). This can be done easily using the bitwise NOT operator.

Example

Part of the decoration in the red base uses a rune texture:

***Fig. 5**—Red team base in CTF-Face; texture highlighted in red is found as an object in the import table.*

The name of this texture is "runeSgn2", part of the "Deco" group within the texture package SkyCity. This object's package index value is -72, meaning it's index #71 in the import table. This is the Deco object, which in turn has a package index of -2, i.e. the SkyCity object. As SkyCity is its own package (SkyCity.utx), this object has a package index of 0 or *NULL*, i.e. it has no "parent" package.

Compact Index Format

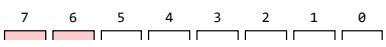
Tim Sweeney, the format's creator, states *"Compact indices exist so that small numbers can be stored efficiently"*.

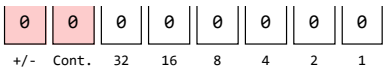
Rather than storing small numbers in a fixed 4-byte "long int"/DWORD, the compact index format is variable in size, ranging from 1-5 bytes. A comparison highlighting the difference in size is shown below:

Value	Long Int/DWORD	Compact Index	Difference
1	`0x00000001`	`0x01`	-3 bytes
100	`0x00000064`	`0x6401`	-2 bytes
10,000	`0x00002710`	`0x509C01`	-1 byte
10,000,000	`0x00989680`	`0x40DAC409`	0 bytes
1,000,000,000	`0x3B9ACA00`	`0x40A8D6B907`	+1 byte

Converting a value to a compact index is achieved by using 1-2 bits of each byte to store information *about* the value, and the remaining bits to store the value itself.

The **first** byte of a compact index is structured as follows:

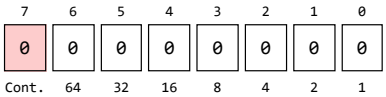




Bits 6 and 7 are essentially used as flags:

- when **bit 6** is set, the value continues into next byte;
- when **bit 7** is set, the value is negative.

All other bytes, if any, are structured as follows:

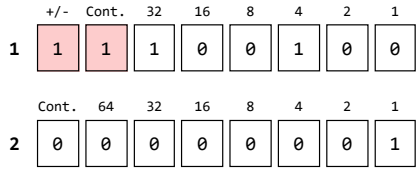


The maximum value that can be stored is `0x7FFFFFFF` (2,147,483,647) as defined in the Object class.

Example

This example will show how to represent the number -100 as a compact index.

The *absolute* value, 100, is represented in binary as `01100100`. As bits 6 and 7 are required to indicate continuation and the negative sign, the last two bits (`0` and `1`) have to be moved along to the next byte:



Put back together, the binary value is now `0000000111100100` or `0xE401` in hex, saving two bytes compared to a regular long int value.

Interactive Example

Enter a number below to see the value as represented by the compact index format.

0

DWORD (4 bytes) **Compact Index**

Byte #	Decimal	Hexadecimal	Binary

References

1. *Unreal Packages* by Tim Sweeney, 1999.
2. *Unreal Tournament Package File Format, Version 1.6* by Antonio Cordero Balcázar, 2001.
3. *Package File Format* by Jesco, 2003. Available from http://www.unrealtexture.com/Unreal/Downloads/3DEditing/UnrealEd/Tutorials/unrealwiki-offline/package-file-format.html

Page last updated: 8th March 2020