Unreal Tournament Package File Format

Version 1.6

If you find any bug in this document or you know some other information please tell me at acordero@acordero.org

Color	Meaning
Blue	This color flags any comment that could be wrong.
Violet	This color flags unknown fields.
Red	This color is used for messages that flag differences between package versions.

Thanks to:
Half (from Unreal Services).
Erik de Neve & Tim Sweeney from Epic Games.
Scott Martin from Ion Storm.

Revision History

1.6

- Fixed type of AnimSeqs_Count in Mesh class.
- Fixed Sound class in packages with version >=63 (thanks to Richard Wynne).
- Some small changes about compressed textures.
- Made clearer the use of LODMesh data.
- Added SkeletalMesh and Animation classes.
- Added two script opcodes for old version packages.

1.5

- Changes in package header. Added LicenseeMode value and clarified the meaning of some fields.
- Changed property interpretation.
- Changed Null Class interpretation and added Field, Struct, Function, State, Property (and its different types) classes.
- Added script decompilation section.

Global Header

DWORD	Signature	0x9E2A83C1
DWORD	PackageVersion	Low order WORD is the file version, high order WORD is the *LicenseeMode* in newer versions (seems unused).
DWORD	Package Flags	See “Package Flags”
DWORD	NameCount	Number of names in the Name Table
DWORD	NameOffset	Offset of the Name Table from the beginning of the file
DWORD	ExportCount	Number of objects in the Export Table
DWORD	ExportOffset	Offset of the Export Table from the beginning of the file
DWORD	ImportCount	Number of objects in the Import Table
DWORD	ImportOffset	Offset of the Import Table from the beginning of the file
If PackageVersion<68 then		
DWORD	HeritageCount	Number of values in the Heritage Table
DWORD	HeritageOffset	Offset of the Heritage Table from the beginning of the file
Else		
16 Bytes	GUID	
DWORD	GenerationCount	Unknown meaning of “Generation”. Seems to be related to recompilation.
For each generation		
DWORD	ExportCount	
DWORD	NameCount	
EndIf		

Name Table

NameCount elements with this format:

NAME	Object Name	
DWORD	Object Flags	See “Object Flags”

Export Table

ExportCount elements with this format:

INDEX	Class	Class of the Object. See “object references”.
INDEX	Super	Parent of the Object (from which it inherits). See “object references”.
DWORD	Package	Package this Object resides in. Could be an internal package (a group). See “object references”.
INDEX	Object Name	The Object name. It’s an index into the Name Table.
DWORD	Object Flags	See “Object Flags”
INDEX	Serial Size	Size of the object inside the file.
INDEX	Serial Offset	Offset of the object inside the file. This field only exists if SerialSize>0

Import Table

ImportCount elements with this format:

INDEX	Class Package	Package of the Class. It’s an index into the Name Table.
INDEX	Class Name	The Class of the Object. It’s an index into the Name Table.
DWORD	Package	The Package this object resides in. See “object references”.
INDEX	Object Name	The Object name. It’s an index into the Name Table.

Heritage Table

HeritageCount elements with this format:

16 Bytes	GUID

Package Flags

PKG_AllowDownload	0x0001	Allow downloading package
PKG_ClientOptional	0x0002	Purely optional for clients
PKG_ServerSideOnly	0x0004	Only needed on the server side
PKG_BrokenLinks	0x0008	Loaded from linker with broken import links
PKG_Unsecure	0x0010	Not trusted
PKG_Need	0x8000	Client needs to download this package

Object Flags

RF_Transactional	0x00000001	Object is transactional.
RF_Unreachable	0x00000002	Object is not reachable on the object graph.
RF_Public	0x00000004	Object is visible outside its package.
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 Or RF_EliminateObject(?)	0x00000400	A hardcoded name which should be syntax-highlighted.
RF_InSingularFunc Or RF_RemappedName(?)	0x00000800	In a singular function.
RF_Suppress Or RF_StateChanged(?)	0x00001000	Suppressed log name.
RF_InEndState	0x00002000	Within an EndState call.
RF_Transient	0x00004000	Don't save object.
RF_PreLoading	0x00008000	Data is being preloaded from file.
RF_LoadForClient	0x00010000	In-file load for client.
RF_LoadForServer	0x00020000	In-file load for client.
RF_LoadForEdit	0x00040000	In-file load for client.
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.
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

Some indices do not refer to a name in the Name Table, but to other objects in the Export or Import tables. They work in this way:

If the index is zero the object referenced is null.
If the index<0 the object is in the Import table in the position (–index-1).
If the index>0 the object is in the Export table in the position (index-1).

NAME Type Format

The NAME type changed between versions of the package format.

If PackageVersion<64 then the type is like an ASCIIZ string.

But if PackageVersion>=64 then the type also saves first the length of the string plus one for the zero byte. So for example, the name “Unreal” would be saved:
0x07 “U” “n” “r” “e” “a” “l” 0x00

INDEX Type Format

The INDEX type is used as a way of reducing file size. It is a DWORD saved with as less bytes as possible. The first byte tells if the number is positive or negative (bit 7, B && 0x80; 1 means negative). Any byte has a bit that means that there is another byte following, in the first byte this is bit 6 (B && 0x40) and in the following is bit 7 (B && 0x80). For example for number –12345:

0x00003039 (in positive, the sign is flagged at the end of the conversion)

00000000 00000000 00110000 00111001 (in binary)

0000001 1000000 111001 (grouping, 6 bits for the most significant byte, 7 bits for others)

0x01 0x40 0x39 (in hex)

0x01 0xC0 0xF9 (added the bits for the sign and for the continuation flags)

Saved as 0xF9 0xC0 0x01, three bytes instead of four.

The Epic package format document calls this data type Compact Indices and it says the following:

```
Compact indices exist so that small numbers can be stored efficiently.
An index named "Index" is stored as a series of 1-5 consecutive bytes
with the following C++ code. Basically, the "Ar << B0" type code
serializes the byte stored in the variable B0.  Serialize can mean
read or write, depending on the internal implementation of the archive
object Ar.
//
// FCompactIndex serializer.
//
FArchive& operator<<( FArchive& Ar, FCompactIndex& I )
{
      INT Original = I.Value;
      DWORD V = Abs(I.Value);
      BYTE B0 = ((I.Value>=0) ? 0 : 0x80) +
                ((V < 0x40) ? V : ((V & 0x3f)+0x40));
      I.Value = 0;
      Ar << B0;
      if( B0 & 0x40 ) {
        V >>= 6;
        BYTE B1 = (V < 0x80) ? V : ((V & 0x7f)+0x80);
        Ar << B1;
        if( B1 & 0x80 ) {
          V >>= 7;
          BYTE B2 = (V < 0x80) ? V : ((V & 0x7f)+0x80);
          Ar << B2;
          if( B2 & 0x80 ) {
            V >>= 7;
            BYTE B3 = (V < 0x80) ? V : ((V & 0x7f)+0x80);
            Ar << B3;
            if( B3 & 0x80 ) {
              V >>= 7;
              BYTE B4 = V;
              Ar << B4;
              I.Value = B4;
            }
            I.Value = (I.Value << 7) + (B3 & 0x7f);
          }
          I.Value = (I.Value << 7) + (B2 & 0x7f);
        }
        I.Value = (I.Value << 7) + (B1 & 0x7f);
      }
      I.Value = (I.Value << 6) + (B0 & 0x3f);
      if( B0 & 0x80 ) I.Value = -I.Value;
      if( Ar.IsSaving() && I.Value!=Original )
        appErrorf("Mismatch: %08X %08X",I.Value,Original);
      return Ar;
}
```

Objects format

Each object has a different format depending on its class. The format depends on the Serialize function or the << operator of the class and the class ancestors (they save their data first). The best way to check for the object format is to read the available source code to see if they are published for that class and its ancestors. Sometimes you can get the format knowing the variables that are saved (they are published) and some investigation with sample objects.

An object usually have a header, a list of properties (ended with property “None”) and specific data.

Properties format

A property starts with an INDEX that specify its name (in the Name Table). If the name is “None” the properties list has finalized. The next BYTE is an info byte for the property (includes its type).

The info byte is composed of several parts. Bits 0 to 3 is the type, bits 4 to 6 is the size and bit 7 is the array flag.

The property type value means:

1 = ByteProperty
2 = IntegerProperty
3 = BooleanProperty
4 = FloatProperty
5 = ObjectProperty
6 = NameProperty
7 = StringProperty
8 = ClassProperty
9 = ArrayProperty
10 = StructProperty
11 = VectorProperty
12 = RotatorProperty
13 = StrProperty
14 = MapProperty
15 = FixedArrayProperty

If the type is an struct then the struct name follows.

The size value is interpreted in the following way:

0 = 1 byte
1 = 2 bytes
2 = 4 bytes
3 = 12 bytes
4 = 16 bytes
5 = a byte follows with real size
6 = a word follows with real size
7 = an integer follows with real size

It bit 7 is set and its not a boolean property, the property is part of an array and the index in this array is specified just at this point. This only happens for array indices greater than 0, that is, the first element (0-based) of the array does not have bit 7 set. The array index value is coded in this way: If i<128 then it is saved as a byte. If i<16384 then it is saved as a word with the most significant byte OR-ed with 0x80. With a greater value, it is saved as an integer with the most significant byte OR-ed with 0xC0.

If the property is a boolean, bit 7 is the value.

Then the property value follows (except for booleans).

Here is how to interpret the values for each type:

Type	*Value Format*	*Comments*
0x01 (ByteProperty)	BYTE	
0x02 (IntegerProperty)	DWORD	
0x03 (BooleanProperty)		The real value is in bit 7 of the info byte.
0x04 (FloatProperty)	DWORD	A 4-byte float.
0x05 (ObjectProperty)	INDEX	Object Reference value. See “Object References”.
0x06 (NameProperty)	INDEX	Name Reference value. Index in to the Name Table.
0x07 (StringProperty)	Unknown	
0x08 (ClassProperty)		See below for some known classes.
0x09 (ArrayProperty)	Unknown	
0x0A (StructProperty)		See below for some known structs.
0x0B (VectorProperty)	Unknown	
0x0C (RotatorProperty)	Unknown	
0x0D (StrProperty)	INDEX length ASCIIZ text	Length field includes null terminator.
0x0E (MapProperty)	Unknown	
0x0F (FixedArrayProperty)	Unknown	

These are known classes and their format:

Class Name	*Value Format*	*Comment*
ADrop or ASpark	WORD unknown1 BYTE x BYTE y DWORD unknown2	Used in wave/wet/fire textures. Only known values are the x and y values of the drop. Seem to be divided by 2 respect to the texture size.

These are the known structs and their value format:

Struct Name	*Format*	*Comment*
Color	BYTE r BYTE g BYTE b BYTE a	
Vector	DWORD x DWORD y DWORD z	x,y,z are floating point values.
Rotator	DWORD pitch DWORD yaw DWORD roll	Pitch, yaw, roll are integer values.
Scale	DWORD x DWORD y DWORD z DWORD sheerrate BYTE sheeraxis	x,y,z are floating point values.
PointRegion	INDEX zone DWORD ileaf BYTE zonenumber	Zone is an object reference. See “Object References”.

Some examples:

“MipZero” would be coded:

<MipZero name index><0x2A><Color struct name index><R><G><B><A>

“InternalTime[0]” and “InternalTime[1]” would be coded :
(look in the second part bit 7 of info byte and following the position in the array)

<InternalTime name index><0x22><integer>
<InternalTime name index><0xA2><0x01><integer>

Generic Objects

All classes except the native ones inherit from the Object class that have the following header.

Only if object has RF_HasStack flag.			
	INDEX	StateFrame.Node	
	INDEX	StateFrame.StateNode	
	QWORD	StateFrame.ProbeMask	
	DWORD	StateFrame.LatentAction	
	INDEX	Offset	Only if StateFrame.Node<>0
EndIf			
Properties. (only if object is not a Class)			

Documented exceptions (native classes) are noted in each class section.

Class Field

This is an abstract class, but we need it to explain the following classes since they inherit from it. It is also the ancestor for Const, Enum and all Property classes. It inherits from the generic object.

INDEX	SuperField	Parent object. Object Reference.
INDEX	Next	Next object in list. Object Reference.

Class Const

This class inherits from Field.

INDEX	Size	This includes the ending null character.
ASCIIZ	Constant	

Class Enum

This class inherits from Field.

INDEX	ArraySize	
For each element		
INDEX	ElementName	Name Reference.

Class Property

This class inherits from Field.

WORD	ArrayDimension	
WORD	ElementSize	
DWORD	PropertyFlags	
INDEX	Category	Name Reference.
WORD	ReplicationOffset	Only if PropertyFlags include CPF_Net

Property Flags

CPF_Edit	0x00000001	Property is user-settable in the editor.
CPF_Const	0x00000002	Actor's property always matches class's default actor property.
CPF_Input	0x00000004	Variable is writable by the input system.
CPF_ExportObject	0x00000008	Object can be exported with actor.
CPF_OptionalParm	0x00000010	Optional parameter (if CPF_Param is set).
CPF_Net	0x00000020	Property is relevant to network replication (not specified in source code)
CPF_ConstRef	0x00000040	Reference to a constant object.
CPF_Parm	0x00000080	Function/When call parameter
CPF_OutParm	0x00000100	Value is copied out after function call.
CPF_SkipParm	0x00000200	Property is a short-circuitable evaluation function parm.
CPF_ReturnParm	0x00000400	Return value.
CPF_CoerceParm	0x00000800	Coerce args into this function parameter
CPF_Native	0x00001000	Property is native: C++ code is responsible for serializing it.
CPF_Transient	0x00002000	Property is transient: shouldn't be saved, zero-filled at load time.
CPF_Config	0x00004000	Property should be loaded/saved as permanent profile.
CPF_Localized	0x00008000	Property should be loaded as localizable text
CPF_Travel	0x00010000	Property travels across levels/servers.
CPF_EditConst	0x00020000	Property is uneditable in the editor
CPF_GlobalConfig	0x00040000	Load config from base class, not subclass.
CPF_OnDemand	0x00100000	Object or dynamic array loaded on demand only.
CPF_New	0x00200000	Automatically create inner object
CPF_NeedCtorLink	0x00400000	Fields need construction/destruction (not specified in source code)

Another one is the “private” flag that is however saved in the object flags (!RF_Public).

Class ByteProperty

This class inherits from Property.

INDEX	EnumType	0 if it is a normal byte, other for a reference to the Enum type object.

Class ObjectProperty

This class inherits from Property.

INDEX	ObjectType	Reference to the Object type object.

Class FixedArrayProperty

This class inherits from Property.

INDEX	Inner	Reference to the element object.
INDEX	Count	Size of the array.

Class ArrayProperty

This class inherits from Property.

INDEX	Inner	Reference to the element object.

Class MapProperty

This class inherits from Property.

INDEX	Key	
INDEX	Value	

Class ClassProperty

This class inherits from ObjectProperty.

INDEX	Class	Reference to class object.

If Class is “Object” then the type is the inherited object else the type is object<class>.

Class StructProperty

This class inherits from Property.

INDEX	StructType	Reference to the Struct object.

Classes IntProperty, BoolProperty, FloatProperty,

These classes inherits from Property and do not add any new data.

Classes NameProperty, StrProperty, StringProperty

These classes inherits from Property and do not add any new data.

Class Struct

This is the class for UnrealScript struct definitions. It inherits from Field.

INDEX	ScriptText	Object Reference.
INDEX	Children	First object inside the struct. Object Reference.
INDEX	FriendlyName	Name of the struct. Name Reference.
DWORD	Line	
DWORD	TextPos	
DWORD	ScriptSize	
? BYTEs	Script	Script Code

The real script size is not exactly the same as the value above since some statements count more than they actually occupy (any index counts four instead of the real index size). To know where the Script ends it is necessary to traverse it (decompile).

The format of the script code is documented later in the document.

Class Function

Inherits from Struct.

INDEX	ParmsSize	Only if Package_Version<=63
WORD	iNative	Native function index
INDEX	NumParms	Only if Package_Version<=63
BYTE	OperatorPrecedence	
INDEX	ReturnValueOffset	Only if Package_Version<=63
DWORD	FunctionFlags	
WORD	ReplicationOffset	Only if FUNC_Net flag in FunctionFlags.

To know its parameters, result type and local variables you should traverse its children objects.

Function Flags

FUNC_Final	0x00000001	Function is final (prebindable, non-overridable function).
FUNC_Defined	0x00000002	Function has been defined (not just declared). Not used in source code.
FUNC_Iterator	0x00000004	Function is an iterator.
FUNC_Latent	0x00000008	Function is a latent state function.
FUNC_PreOperator	0x00000010	Unary operator is a prefix operator.
FUNC_Singular	0x00000020	Function cannot be reentered.
FUNC_Net	0x00000040	Function is network-replicated. Not used in source code.
FUNC_NetReliable	0x00000080	Function should be sent reliably on the network. Not used in source code.
FUNC_Simulated	0x00000100	Function executed on the client side.
FUNC_Exec	0x00000200	Executable from command line.
FUNC_Native	0x00000400	Native function.
FUNC_Event	0x00000800	Event function.
FUNC_Operator	0x00001000	Operator function.
FUNC_Static	0x00002000	Static function.
FUNC_NoExport	0x00004000	Don't export intrinsic function to C++.
FUNC_Const	0x00008000	Function doesn't modify this object.
FUNC_Invariant	0x00010000	Return value is purely dependent on parameters; no state dependencies or internal state changes.

Class State

This class inherits from Struct.

QWORD	ProbeMask	
QWORD	IgnoreMask	
WORD	LabelTableOffset	Offset of the Label Table into the script.
DWORD	StateFlags	

State Flags

STATE_Editable	0x00000001	State should be user-selectable in UnrealEd.
STATE_Auto	0x00000002	State is automatic (the default state).
STATE_Simulated	0x00000004	State executes on client side.

Class null (or “None”)

This class inherits from State and it is the “class” of the actual classes in UnrealScript.

DWORD	OldClassRecordSize	Only if Package_Version<=61
DWORD	ClassFlags	
GUID	ClassGuid	
INDEX	Dependencies_Count	
INDEX	Class	Object Reference.
DWORD	Deep	
DWORD	ScriptTextCRC	
INDEX	PackageImports_Count	
INDEX	PackageImport	Object Reference.
If Package_Version>=62		
INDEX	ClassWithin	Object Reference.
INDEX	ClassConfigName	Name Reference.
EndIf		
Properties (see above to decode)		

This class is the main object in any package, excluding other native classes. All other objects are structured inside them.

Class Flags

CLASS_Abstract	0x00001	Class is abstract and can't be instantiated directly.
CLASS_Compiled	0x00002	Script has been compiled successfully.
CLASS_Config	0x00004	Load object configuration at construction time.
CLASS_Transient	0x00008	This object type can't be saved; null it out at save time
CLASS_Parsed	0x00010	Successfully parsed.
CLASS_Localized	0x00020	Class contains localized text. Not used in source code.
CLASS_SafeReplace	0x00040	Objects of this class can be safely replaced with default or NULL.
CLASS_RuntimeStatic	0x00080	Objects of this class are static during gameplay.
CLASS_NoExport	0x00100	Don't export to C++ header.
CLASS_NoUserCreate	0x00200	Don't allow users to create in the editor.
CLASS_PerObjectConfig	0x00400	Handle object configuration on a per-object basis, rather than per-class.
CLASS_NativeReplication	0x00800	Replication handled in C++

Another flag is “Native” read from the object flags (RF_Native).

Class Texture

This is a native class that does not inherit from the generic object header.

Properties (see above to decode)		
BYTE	MipMapCount	Number of MipMaps in the texture object.
DWORD	WidthOffset	Offset in file of the following Width DWORD.Only if PackageVersion is >=63
INDEX	MipMapSize	Size of the graphic block
MipMapSize BYTEs	MipMapData	Image. One byte per pixel.
DWORD	Width	Width in pixels
DWORD	Height	Height in pixels
BYTE	BitsWidth	Number of bits of Width (8 bits for 256 pixels)
BYTE	BitsHeight	Number of bits of Height

If the Format property exists it tell which format the mipmaps have, else use TEXF_P8 (see below).

Some textures have compressed MipMaps after the normal MipMaps. This happens when the bHasComp property is True. The structure is the same as in the above table except that the MipMapData has a data format based on the CompFormat property.

The Format and CompFormat properties have these values:

0x00	TEXF_P8	8 bits, paletted
0x01	TEXF_RGBA7	7 bits for RGBA ?
0x02	TEXF_RGB16	16 bits for RGB ?
0x03	TEXF_DXT1	DirectX-1 format. Search for “Opaque and One-bit Alpha Textures” in the Microsoft Platform SDK documentation.
0x04	TEXF_RGB8	8 bits for RGB ?
0x05	TEXF_RGBA8	8 bits for RGBA ?

Only TEXF_P8 and TEXF_DXT1 are known at this time.

Class Palette

This is a native class that does not inherit from the generic object header.

Properties (see above to decode)		
INDEX	PaletteSize	Number of colors following
BYTE	Red	
BYTE	Green	
BYTE	Blue	
BYTE	Alpha	Maybe an Alpha value?

Class Font

This is a native class that does not inherit from the generic object header.

The Font objects describe the position and size of each font in a texture object.

Properties (see above to decode)		
BYTE	TextureCount	Number of textures used
INDEX	Texture	See “Object References”
INDEX	CharCount	Number of characters in this texture.
DWORD	X	Horizontal position in texture.
DWORD	Y	Vertical position intexture.
DWORD	Width	Width in texture.
DWORD	Height	Height in texture.

Class TextBuffer

This is a native class that does not inherit from the generic object header.

Properties (see above to decode). Seem to be only “None”.		
DWORD	Pos	Always zero.
DWORD	Top	Always zero.
INDEX	TextSize	
TextSize BYTEs	TextData	Text block, usually in UnrealScript.
BYTE	NULL	Only if TextSize>0

Class Sound

This is a native class that does not inherit from the generic object header.

Properties (see above to decode)		
INDEX	SoundFormat	Index into the Name Table of the sound format extension (WAV).
DWORD	OffsetNext	Offset in file of next object in package. .Only if PackageVersion is >=63
INDEX	SoundSize	
SoundSize BYTEs	SoundData	Sound in WAV format.

Class Music

This is a native class that does not inherit from the generic object header.

Music packages have only one Music object inside them, or at least this is the case for the original UT packages (UnrealED does not allows more than one object per package). The Music format (file extension) is the first name in the Name Table. The format is one of the several Tracker formats (IT, XM, MOD, etc). This objects do not have properties.

WORD	ChunkCount?	Always 1. Could also be the index into the name table for the format.
DWORD	Unknown	If package version>61 it's the position of the byte next to ChunkData.
INDEX	ChunkSize	
ChunkSize BYTEs	ChunkData	

Type Vector

Float	X
Float	Y
Float	Z

Type Rotator

Float	Pitch
Float	Yaw
Float	Roll

16384 correspond to 90 degress.

Type BoundingBox

Vector	Min
Vector	Max
BYTE	IsValid

Type BoundingSphere

Vector	Position	
Float	W	Only if Package Version > 61

Class Mesh

This is a native class that does not inherit from the generic object header.

Properties (see above to decode). Seem to be only “None”.		
BoundingBox	Primitive.BoundingBox	Part of UPrimitive ancestor
BoundingSphere	Primitive.BoundingSphere	Part of UPrimitive ancestor
DWORD	Verts_Jump	Only if Package Version > 61
INDEX	Verts_Count	Vertex array. All frames.
	UT variation: This values do not give a Front view, but not all models have the same orientation. I don’t know how UT knows where is the front.	

	DWORD	XYZ	x=(xyz && 0x7FF)/8
			y=((xyz >> 11) && 0x7FF)/8
			if (y>128) { y=y-256; }
			y=-y
			if (x>128) { x=x-256; }
			x=-x
			z=((xyz >> 22) && 0x3FF)/4
			if (z>128) { z=z-256; }

	DeusEx variation:		

	QUADWORD	XYZ	x=(xyz && 0xFFFF)/256
			y=((xyz >> 16) && 0xFFFF)/256
			if (y>128) { y=y-256; }
			y=-y
			if (x>128) { x=x-256; }
			x=-x
			z=((xyz >> 32) && 0xFFFF)/256
			if (z>128) { z=z-256; }

DWORD	Tris_Jump	Only if Package Version > 61
INDEX	Tris_Count	
	WORD	VertexIndex1	
	WORD	VertexIndex2	
	WORD	VertexIndex3	
	BYTE	Vertex1_U	
	BYTE	Vertex1_V	
	BYTE	Vertex2_U	
	BYTE	Vertex2_V	
	BYTE	Vertex3_U	
	BYTE	Vertex3_V	
	DWORD	Flags	
	DWORD	TextureIndex	
INDEX	AnimSeqs_Count	
	INDEX	Name	
	INDEX	Group	
	DWORD	Start_Frame	
	DWORD	Num_Frames	
	INDEX	Function_Count	

	DWORD	Time	
	INDEX	Function	Object Reference.
FLOAT		Rate	
DWORD		Connects_Jump	
INDEX		Connects_Count	
DWORD		NumVertTriangles	
DWORD		TriangleListOffset	
BoundingBox		BoundingBox	
BoundingSphere		BoundingSphere	
DWORD		VertLinks_Jump	
INDEX		VertLinks_Count	
DWORD		VertLink	
INDEX		Textures_Count	
INDEX		Texture	Object Reference.
INDEX		BoundingBoxes_Count	
BoundingBox		BoundingBoxes	
INDEX		BoundingSpheres_Count	
BoundingSphere		BoundingSpheres	
DWORD		FrameVerts	
DWORD		AnimFrames	
DWORD		ANDFlags	
DWORD		ORFlags	
Vector		Scale	
Vector		Origin	
Rotator		RotOrigin	
DWORD		CurPoly	
DWORD		CurVertex	
If Package_Version = 65			
FLOAT		TextureLOD?	
ElseIf Package_Version >=66			
INDEX		TextureLOD_Count	
FLOAT		TextureLOD	
EndIf			

Class LodMesh

This class inherits from Mesh. The Tris array is empty in a LodMesh object.

INDEX		CollapsePointThus_Count	
	WORD	CollapsePointThus	
INDEX		FaceLevel_Count	
	WORD	FaceLevel	
INDEX		Faces_Count	
	WORD	WedgeIndex1	
	WORD	WedgeIndex2	
	WORD	WedgeIndex3	
	WORD	MaterialIndex	
INDEX		CollapseWedgeThus_Count	
	WORD	CollapseWedgeThus	
INDEX		Wedges_Count	
	WORD	VertexIndex	
	BYTE	S	=U
	BYTE	T	=255-V
INDEX		Materials_Count	
	DWORD	Flags	
	DWORD	TextureIndex	
INDEX		SpecialFaces_Count	Weapon?
	WORD	WedgeIndex1	
	WORD	WedgeIndex2	
	WORD	WedgeIndex3	
	WORD	MaterialIndex	
DWORD		ModelVerts	
DWORD		SpecialVerts	
FLOAT		MeshScaleMax	
FLOAT		LODHysteresis	
FLOAT		LODStrength	
DWORD		LODMinVerts	
FLOAT		LODMorph	
FLOAT		LODZDisplace	
INDEX		ReMapAnimVerts_Count	
	WORD	ReMapAnimVerts	
DWORD		OldFrameVerts	

To read the mesh you will need to get the Verts, Textures, Wedges, Faces and Materials arrays and the FrameVerts, AnimFrames and SpecialVerts values.

The WedgeIndex1, 2, 3 values of the Faces array tell you which Wedges uses the face. Each Wedge has a VertexIndex that added to SpecialVerts+selected_frame*FrameVerts gives you the vertex index into the Verts array.

The SpecialFaces array has the weapon polygon. The WedgeIndex1, 2, 3 of this face has to be used a bit differently because in this case we don't have to add the

SpecialVerts value. So to find the correct vertex index just add selected_frame*FrameVerts to the VertexIndex value.

The wedge S and T values give you the texture coordinates (U=S/255, V=1-T/255) and the faces MaterialIndex points you to the TextureIndex value in the Materials array that points you to the Texture object in the Textures array.

The AnimFrames value gives you the number of frames. And you can also read the AnimSeqs array to get sequence names, position and duration.

Class SkeletalMesh

This class inherits from LodMesh.

INDEX	ExtWedges_Count	
WORD	iVertex	
WORD	Flags	
FLOAT	U	
FLOAT	V	
INDEX	Points_Count	
FLOAT	X	
FLOAT	Y	
FLOAT	Z	
INDEX	RefSkeleton_Count	
INDEX	Name	Name reference.
DWORD	Flags	
FLOAT	BonePos.Orientation.X	
FLOAT	BonePos.Orientation.Y	
FLOAT	BonePos.Orientation.Z	
FLOAT	BonePos.Orientation.W	
FLOAT	BonePos.Position.X	
FLOAT	BonePos.Position.Y	
FLOAT	BonePos.Position.Z	
FLOAT	BonePos.Length	
FLOAT	BonePos.Xsize	
FLOAT	BonePos.Ysize	
FLOAT	BonePos.Zsize	
DWORD	NumChildren	
DWORD	ParentIndex	
INDEX	BoneWeightIdx_Count	
WORD	WeightIndex	
WORD	Number	
WORD	DetailA	
WORD	DetailB	
INDEX	BoneWeights_Count	
WORD	PointIndex	
WORD	BoneWeight	
INDEX	LocalPoints_Count	

	FLOAT	X	
	FLOAT	Y	
	FLOAT	Z	
DWORD		SkeletalDepth	
INDEX		DefaultAnimation	Object Reference.
DWORD		WeaponBoneIndex	
FLOAT		WeaponAdjust.Origin.X	
FLOAT		WeaponAdjust.Origin.Y	
FLOAT		WeaponAdjust.Origin.Z	
FLOAT		WeaponAdjust.Xaxis.X	
FLOAT		WeaponAdjust.Xaxis.Y	
FLOAT		WeaponAdjust.Xaxis.Z	
FLOAT		WeaponAdjust.Yaxis.X	
FLOAT		WeaponAdjust.Yaxis.Y	
FLOAT		WeaponAdjust.Yaxis.Z	
FLOAT		WeaponAdjust.Zaxis.X	
FLOAT		WeaponAdjust.Zaxis.Y	
FLOAT		WeaponAdjust.Zaxis.Z	

This mesh can be interpreted as a LodMesh except that instead of the Verts array you have to use the Points array for the vertices.

Also, you don't have frames here, just the reference model, because the animation is at the assigned Animation object that animates the bone skeleton.

Class Animation

Properties (see above to decode)			
INDEX		RefBones_Count	
	INDEX	Name	Name reference
	DWORD	Flags	
	DWORD	ParentIndex	
INDEX		Moves_Count	
	FLOAT	RootSpeed3D.X	
	FLOAT	RootSpeed3D.Y	
	FLOAT	RootSpeed3D.Z	
	FLOAT	TrackTime	
	DWORD	StartBone	
	INDEX	BoneIndices_Count	
	DWORD	BoneIndex	
	INDEX	AnimTracks_Count	Array of AnalogTracks
	DWORD	Flags	
	INDEX	KeyQuat_Count	
	FLOAT	KeyQuat.X	
	FLOAT	KeyQuat.Y	

	FLOAT	KeyQuat.Z	
	FLOAT	KeyQuat.W	
INDEX		KeyPos_Count	
	FLOAT	KeyPos.X	
	FLOAT	KeyPos.Y	
	FLOAT	KeyPos.Z	
INDEX		KeyTime_Count	
	FLOAT	KeyTime	
DWORD		RootTrack.Flags	
INDEX		RootTrack.KeyQuat_Count	
	FLOAT	RootTrack.KeyQuat.X	
	FLOAT	RootTrack.KeyQuat.Y	
	FLOAT	RootTrack.KeyQuat.Z	
	FLOAT	RootTrack.KeyQuat.W	
INDEX		RootTrack.KeyPos_Count	
	FLOAT	RootTrack.KeyPos.X	
	FLOAT	RootTrack.KeyPos.Y	
	FLOAT	RootTrack.KeyPos.Z	
INDEX		RootTrack.KeyTime_Count	
	FLOAT	RootTrack.KeyTime	

This object contains the animations for skeletal meshes. The RootTrack is the same type as the elements of the previous array AnimTracks.

Script Format

The format of a struct, function, state or class script is complex. The code statements are compiled into tokens. Here are the tokens and their interpretation:

OpCode Name	Value	Follows	Decompiled into
EX_LocalVariable	0x00	INDEX Object	ObjectName
EX_InstanceVariable	0x01	INDEX Object	ObjectName
EX_DefaultVariable	0x02	INDEX Object	Default.ObjectName
?	0x03		
EX_Return	0x04	TOKEN Result	Return Result
EX_Switch	0x05	BYTE Size TOKEN Condition	Switch (Condition) { Size is the size of the Condition expression. The end of the switch is unknown except maybe for some hints inside it.
EX_Jump	0x06	WORD Offset	Goto Offset
EX_JumpIfNot	0x07	WORD Offset TOKEN Condition	If (!Condition) Goto Offset
EX_Stop	0x08		Stop End of State.
EX_Assert	0x09	WORD Line TOKEN Condition	Assert (Condition)
EX_Case	0x0A	WORD NextOffset {TOKEN Value}	Case Value: The Value does not exist if NextOffset is 0xFFFF meaning a “default” case (last one).
EX_Nothing	0x0B		This token is used as filler or as a placeholder in a not used optional parameter. You should return an empty string as its result.
EX_LabelTable	0x0C	Array of INDEX Name DWORD Offset	The array ends when Name is “None”. This token follows an EX_Stop in an state and wil be aligned to 4 (filled with EX_Nothing). It is the last token in the script.
EX_GotoLabel	0x0D	TOKEN Value	goto (Value)
EX_EatString	0x0E	TOKEN Value	Value
EX_Let	0x0F	TOKEN LeftSide TOKEN RightSide	LeftSide=RightSide
EX_DynArrayElement	0x10	TOKEN Index TOKEN Array	Array[Index]
EX_New	0x11	TOKEN Value1 TOKEN Value2 TOKEN Value3 TOKEN Value4	new (Value1,Value2,Value3,Value4)
EX_ClassContext	0x12	INDEX Class	ClassName.ObjectName

		WORD wSkip BYTE bSize INDEX Object	
EX_MetaCast	0x13	INDEX Class TOKEN Value	Class<ClassName>(Value) The “Class” word at the start is fixed.
EX_LetBool	0x14	TOKEN LeftSide TOKEN RightSide	LeftSide=RightSide In version 61 packages seems to be a label table at the start of the script.
Unknown	0x15	[TOKEN Value]	Only seen in version 61 packages. If at the end of the script it does not have any parameter. In other places it resolves to the token value.
EX_EndFunctionParms	0x16		)
EX_Self	0x17		Self
EX_Skip	0x18	WORD Skip TOKEN Value	Value
EX_Context	0x19		
EX_ArrayElement	0x1A	TOKEN Index TOKEN Array	Array[Index]
EX_VirtualFunction	0x1B	INDEX Name	Name (Parameters follow until EX_EndFunctionParms.
EX_FinalFunction	0x1C	INDEX Object	ObjectName (Parameters follow until EX_EndFunctionParms. Depending on the relation between the current function and this funcion it could be preceded with “Super.” or “Super(class).”
EX_IntConst	0x1D	DWORD Value	Value
EX_FloatConst	0x1E	FLOAT Value	Value
EX_StringConst	0x1F	ASCIIZ Value	“Value”
EX_ObjectConst	0x20	INDEX Object	ObjectClass’ObjectName’
EX_NameConst	0x21	INDEX Name	‘Name’
EX_RotationConst	0x22	DWORD Pitch DWORD Yaw DWORD Roll	rot(Pitch,Yaw,Roll)
EX_VectorConst	0x23	FLOAT x FLOAT y FLOAT z	vect(x,y,z)
EX_ByteCont	0x24		
EX_IntZero	0x25		0
EX_IntOne	0x26		1
EX_True	0x27		True
EX_False	0x28		False
EX_NativeParm	0x29	INDEX Object	ObjectName

EX_NoObject	0x2A		None
Unknown	0x2B	BYTE Unknown TOKEN Value	Only seen in version 61 packages. Seems to be a type cast. Resolves to the token value.
EX_IntConstByte	0x2C	TOKEN Value	Value
EX_BoolVariable	0x2D	TOKEN Value	Value
EX_DynamicCast	0x2E	INDEX Class TOKEN Value	<ClassName>(Value)
EX_Iterator	0x2F	TOKEN Value WORD Offset	ForEach Value { Offset points to the EX_IteratorNext (end of foreach)
EX_IteratorPop	0x30		This occurs when the iterator must jump to the next element. Does not show in source code and precedes an EX_IteratorNext or appears before a Goto if a Continue was used.
EX_IteratorNext	0x31		} End of ForEach loop.
EX_StructCmpEq	0x32	INDEX Struct TOKEN Value1 TOKEN Value2	Value1 == Value2
EX_StructCmpNe	0x33	INDEX Struct TOKEN Value1 TOKEN Value2	Value1 != Value2
EX_UnicodeStringConst	0x34	UNICODEZ Value	“Value”
?	0x35		
EX_StructMember	0x36	INDEX Object TOKEN Value	Value.ObjectName
?	0x37		
EX_GlobalFunction	0x38	INDEX Name	Global.Name (Parameters follow until EX_EndFunctionParms.
EX_RotatorToVector	0x39	TOKEN Value	vector(Value)
EX_ByteToInt	0x3A	TOKEN Value	Value
EX_ByteToBool	0x3B	TOKEN Value	Value
EX_ByteToFloat	0x3C	TOKEN Value	Value
EX_IntToByte	0x3D	TOKEN Value	Value
EX_IntToBool	0x3E	TOKEN Value	Value
EX_IntToFloat	0x3F	TOKEN Value	Value
EX_BoolToByte	0x40	TOKEN Value	Value
EX_BoolToInt	0x41	TOKEN Value	Value
EX_BoolToFloat	0x42	TOKEN Value	Value
EX_FloatToByte	0x43	TOKEN Value	Value
EX_FloatToInt	0x44	TOKEN Value	Value
EX_FloatToBool	0x45	TOKEN Value	Value
EX_StringToName	0x46	TOKEN Value	name(Value) Not defined in UT source, but used in unrealscript.

EX_ObjectToBool	0x47	TOKEN Value	bool(Value)
EX_NameToBool	0x48	TOKEN Value	bool(Value)
EX_StringToByte	0x49	TOKEN Value	byte(Value)
EX_StringToInt	0x4A	TOKEN Value	int(Value)
EX_StringToBool	0x4B	TOKEN Value	bool(Value)
EX_StringToFloat	0x4C	TOKEN Value	float(Value)
EX_StringToVector	0x4D	TOKEN Value	vector(value)
EX_StringToRotator	0x4E	TOKEN Value	rotator(Value)
EX_VectorToBool	0x4F	TOKEN Value	bool(Value)
EX_VectorToRotator	0x50	TOKEN Value	rotator(Value)
EX_RotatorToBool	0x51	TOKEN Value	bool(Value)
EX_ByteToString	0x52	TOKEN Value	string(Value)
EX_IntToString	0x53	TOKEN Value	string(Value)
EX_BoolToString	0x54	TOKEN Value	string(Value)
EX_FloatToString	0x55	TOKEN Value	string(Value)
EX_ObjectToString	0x56	TOKEN Value	string(Value)
EX_NameToString	0x57	TOKEN Value	string(Value)
EX_VectorToString	0x58	TOKEN Value	string(Value)
EX_RotatorToString	0x59	TOKEN Value	string(Value)
?	0x5A to 0x5F		
EX_ExtendedNative	0x60	See below.	
EX_FirstNative	0x70		

Any offset is in Script units, that is, taking into account that any INDEX value in the tokens adds four bytes instead of the real serialized size.

Any value >= EX_ExtendedNative should be treated as a Native function call.
If (b && 0xF0) == EX_ExtendedNative then it is an extended native function, you should read another byte and calculate its value with:

Native=(token-EX_ExtendedNative)<<8 + SecondToken

Native should be >=EX_FirstNative.

If the Native function is a “Function” or “Event” the result is NativeName(parameters) as the EX_VirtualFunction token.

If it is a “PreOperator” you should read another token and the result is NativeName+Token (the two joined, without the plus sign). An EX_EndFunctionParms token follows, discard it.

If it is a “PostOperator” you should read another token and the result is Token+NativeName (the two joined, without the plus sign). An EX_EndFunctionParms token follows, discard it.

If it is an “Operator” you should read two tokens and the result is Token1+NativeName+Token2 (the three joined, without the plus signs). An EX_EndFunctionParms token follows, discard it.

To know the name and type of the native functions you will have to find them in all the packages of the game and extract this information (friendlyname and functionflags at least).