Programming for Artists and Designers

Lecture 2: Introduction to the INT File

Introduction

This lecture aims to introduce the INT file, its format, what it is responsible for and how to use it.

The INT File

INT files are one of the Unreal Engine's file formats. These files are used by the Unreal engine for reference. Instead of searching all the packages made accessible to it to find the resources it's looking for, it simply reads the lists of resources contained in these files.

In addition, they are used for language localization and that's the origin of the file extension .int for "international" as well.

These (and all other localization files like DET, ITT, FRT, etc.) look much like INI files. They consist of one or more sections which start with the section's name in brackets followed by Key=Value pairs.

File Content

Public Section - [Public]

The [Public] section can take two different keys, Object and Preferences. These can be used as often as needed.

Object

General syntax:

```
Object=(Name=Package.ObjectName, Class=ObjectClass,
        MetaClass=Package.MetaClassName, Description="descriptive string")
```

Name: Arbitrary name; generally, must refer to an existing class. (Though that's no engine requirement. If you work with GetNextIntDesc to retrieve Object keys yourself, the Name argument can be anything.)

Class: Class of the object described by this line. In many cases, that's simply Class if the Object line refers to a class.

MetaClass: Common superclass of all objects described in Object lines that belong together; for mutators, that'd be Engine.Mutator, for instance. (This class name is used as GetNextIntDesc's first parameter.)

Description: Arbitrary description (optional), also retrieved by GetNextIntDesc.

In UT2003 the following Object entries are used:

Item	MetaClass	Name	Explanation
Mutator	Engine.Mutator	Mutator subclass	Registers a mutator to make it show up in the Mutators dialog box when starting a game. Description is not used in the GUI, but is nevertheless specified for the default mutators.
Game Type	Engine.GameInfo	GameInfo subclass	Registers a game type to make it show up in the Game Type drop-down box when starting a game. Description is important and takes the form "A|B|C|D|E" with the following meanings: • A = Prefix of the maps used by this game type • B = Gametype's display name. • C = a Tab_InstantActionBaseRules subclass that - loads a class that allows mod author to change the tab for the game type options • D = A MapList class used to save the current map list selection. • E = "true" for teamgames, "false" otherwise
Weapon	Engine.Weapon	Weapon subclass	Registers weapons. The Description value is displayed as the weapon's description in the priority tab. You can use "|" to create line breaks in the description.

Preferences

General syntax:

```
Preferences=(Caption="display name", Parent="display name of parent",
        Class=Package.ClassName, Category=variable group name,
        Immediate=True)
```

This is used to create the options available in the UnrealEd Advanced Options window.

Either Class, Category and Immediate are left out or Class has to be a valid UnrealScript class and Category should be a variable group used in that class. (See Variable Syntax: http://wiki.beyondunreal.com/wiki/Variable_Syntax)

Localization Sections

Used to localize strings to different (natural) languages.

Localization

The Unreal engine supports language localizations (or localisations for the English amongst us) in a very straightforward and easy-to-use way. If you declare a string variable using the localized keyword, the engine will automatically look it up in the selected language's localization file, potentially falling back to the international language localization file first and the value declared in the default properties at last.

The [Localization page] on the [Unreal Technology site] explains all the technical aspects of language localization. Corrections and annotations to this document:
The file extension and language code for German is .det, not .de. (Similar modifications apply to the other language codes; it's also .est, not .esp, and so on — the last character always is a t.)
To switch to a different language, open UnrealTournament.ini and replace the language code in the line "Language=int" by the code you wish to use. If the localization file for the Core package is available and the Language value in the [Language] section is set you can also select the language in UnrealEd Advanced Options → Drivers → Language.

Localization Sections Cont…

To create a class variable with different default values for different languages use this syntax in an UnrealScript class:

```
class aClassName extends aSuperClass;

var localized string Description;
var localized float SoundLength;

defaultproperties
{
     Description="An example class showing localization."
     SoundLength=2.750000
}
```

This class is part of a package Example.u.

You can now create localized versions of this class by writing the corresponding localization files. This will be easier if you use UnrealEd to create the international (English) version of this file first. Open UnrealEd, load Example.u and type "dumpint example" at the console. UnrealEd created the file Example.int for you which contains all localizable variables that have been set in the defaultproperties.

For this example the file will look like this:

```
[aClassName]
Description=An example class showing localization.
SoundLength=2.750000
```

You can copy this file and change the extension to the desired language, e.g. DET. Now you can translate the strings and adjust the other variables to suit the new language:

```
[aClassName]
Description=Eine Beispielklasse, die Localization veranschaulicht.
SoundLength=2.930000
```

Since the name of the new file is Example.det the values in it automatically become the default values of aClassName when this language is selected.

Note: You can't use dumpint in UnrealEd due to a bug in the current UT2003 version. UnrealEd will crash if you try. Use: ucc dumpint <package(s)> instead.

UnrealScript Functions

string GetNextInt(string ClassName, int Num)
Returns the Class string from an Object entry for the MetaClass given by ClassName. Num is an index, starting from 0 (zero), that specifies which (of multiple matching) entries to get. The function returns an empty string if Num exceeds the number of available matching entries. Warning: ClassName must refer to a valid, loaded class, or the game will crash.

GetNextIntDesc(string ClassName, int Num, out string Entry, out string Description)
Like GetNextInt, but retrieves more detailed information about the Object entry, including the content of the Description argument.

string Localize(string SectionName, string KeyName, string PackageName)
Returns the localized string value of the item given by KeyName in the section given by SectionName for the package PackageName. Only rarely needed thanks to the localized keyword in variable declarations that automatically performs this lookup. (Can be very handy for read-only configuration files though.)