1. Introduction

The main purpose of the tool is to easily list the files required by an unreal package (like a map), check which files are missing and allow the downloading from a (redirect) server.
The feature list:

- View dependencies of unreal packages.
- Check which files are missing
- Download the missing files from redirect servers
- Copy all required files to one central directory to allow easy publishing of e.g. a map
- Compress/Decompress uz, uz2 and uz3 files (standalone, no local installation of any game required; uz compression is around 30 times faster than UT99's ucc.exe)

I began writing the tool because I wanted to get to know C# and the general .NET environment, i.e. not because I required such a tool in the first place.
I hope, though, that it might be some use to you ☺

Requirements:

Windows and .NET Framework 3.5

Supported games:

I don't really know… I am pretty sure that all games up to UT2004 (3369) are supported, because I based my implementation of the package-reading on Arcodero's UnrealDelphi-Library.
Furthermore, Eliot's UnrealLibrary is used, so more games might be supported. Just try it out.

I only tested UT2004 and UT99, though.

Installation

Just drop all files any directory on your hard-drive and start the tool.

2. First start

When you start the tool for the first time and if you don't want to only use the compression/decompression feature, you have to create a "game profile" first.

A game profile contains information specific to a game, like the uz-version or the paths to the unreal packages.

So, open up the settings dialog by clicking on "Modify and add game profiles" and add a new profile by clicking on "Add new game profile".
If you have UT2004 or UT99 on your local hard-drive, all you need to do to configure the profile is to click on "Use prefab…" and choose the desired game. The tool will read the path to the game's directory from the Windows registry (or ask you, in case it fails to do so) and chooses the required settings. The package-types' paths are read from the game's IniFile.

In case you want/need to create the profile yourself, here is a short description of each field:

- Profile name: Almost anything you want; has to be an unique name, though
- Minimum and maximum package version: When a package is opened its version is checked. In case it isn't a value from the min-max-range, an error occurs and the file is skipped. You can deactivate this check by using a 0 for both the max. and min. version.
  Here are some package versions:
  - UE1: 0-79
  - UT2004: 68-128 (68 is required; e.g. Announcer.uax has got this version
  - UT3: around 512
  - Unreal Developer Kit: 648 - 737 (July 2010)
- uz-Version: Choose the version of the game's redirect file format.
  - uz_ut99: Used by UT99. The uz-files start with the numeric value 1234
  - uz_5678: Used e.g. by Postal 2. The uz files start with the numeric value 5678
  - uz2: Used by UT2004
  - uz3: Used by UT3
- Modify known packages button: The package-types of the opened files are being guessed by looking at their content, but sometimes, the tool chooses the incorrect type. Here comes the "known packages" into play: Before the package-type is guessed the tool checks this list for the package's name and, in case it is found, uses that one.
  Note that the tool comes with standard known-packages-files for UT99 and UT2004 (which basically contain all the game's standard filenames).
- Excluded files path: In the Copy-dialog you can specify files which are excluded from the copy operations. Their names are then saved in the file specified by this setting (you should be fine by using e.g. "Excluded_GameName.ufl" (.ufl is the extension I use for the files; may be anything though).
  Note that the tool comes with standard excluded-files-files for UT99 and UT2004 (which basically contain all the game's standard filenames).
- Known packages path: Where the known packages' names are saved to (you should be fine by using e.g. "Known_GameName.ufl").
  Note that the tool comes with standard known-packages-files for UT99 and UT2004 (which basically contain all the game's standard filenames).
- Filtered files path: You can specify filtered files in the main dialog of the tool. These are used to separate "certain" files from the others. This list is saved in the specified file (you should be fine by using e.g. "Filtered_GameName.ufl").
- Supported package types: Each game has certain package types like e.g. Code, Textures, Maps, etc. and these are normally placed in certain directories and have got a certain extensions. You can specify these here.

3. First steps

After you created a game profile you can click on “OK” in the start-up-dialog, to open the main window.

To open an unreal package, click on “File” in the menu bar and then on “Open packages (replace)” (replaces already opened files) or on “Open packages (append)” (appends the newly opened files to the existing ones).

In the left list all opened packages are displayed, in the one next to it all dependencies of the *selected* package. By hovering over the elements some information is displayed about the element, like the number of missing files or the GUID of the package.

All possible actions are placed on the right side:

- Reopen packages: Re-reads all opened packages again. Useful e.g. when you changed a file or the game profile.
- Copy files: Allows the copying of either the opened packages or of the filtered files (including or excluding “some” dependencies). It is also possible to compress the copied files to the uz-format of the used game.
- List export: Allows you to export the opened packages, the dependencies, missing or filtered files as a list of filenames either to a text-file or to the clipboard.
    - “Simple” means that only the filenames are being copied.
    - “Advanced” allows you to export a list of “guessed” URLs to the files on a redirect server, either as a “normal” list or in the format of an HTML-page. The latter can be used e.g. to download all files using Firefox’s DownThemAll-plugin.
- Download: Allows you download the opened packages, the dependencies, missing or filtered files from redirect servers directly. Especially useful for the missing files.

4. General usage tips

- You can drag & drop files onto most dialogs (where files are expected).
- Be aware that most controls have an associated tooltip.
- Switching or modifying game profiles while packages are opened might cause problems.

5. Possible usage scenarios

- Compressing unreal packages into the uz/uz2/uz3 format, as it is used for redirect servers. Check the Extras menu for the compression-dialog.
- Decompressing uz/uz2/uz3 files. Check the Extras menu for the decompression-dialog.
- Check if you got all packages on your harddisk which are required by another package (like a map for example). Simply open the package in question and check the missing-files-list.
- Download the missing files from a redirect server (in the uz format). To do this, open the package in question, choose the missing files as data-source and click the download-button.
- Create a simple name list of used dependencies or missing files (optionally including a download link to a redirect server). Load the package and export to either the clipboard or a file, using the advanced mode (simple mode will only export the name).
- Quickly get all non-standard files for a package like a map (e.g. if you want to release the map together with all required files). Load the desired package, click the copy-button and exclude the standard files from the copy-operation.
- Check which files required by a package are missing on a FTP server. Open the package, create a filter from the FTP directory (check the filtered files tab in the main dialog) and display all files, which are NOT in the filter.

6. Known problems

To be honest I haven't tested all parts of the program thoroughly. So I guess the tool has got a lot of bugs…
Anyway, known problems/limitations/bugs:

- All the package-type-guessing isn't actually required, as the unreal engine ignores the extension of the files anyway. I realised this 2 weeks ago…
- The Download-features only unpacks files of the game profile's uz-format (not the other ones).
- Error/Exception handling could be improved.
- Redirect-servers could be maintained by each game profile instead of using one global list.
- Some minor problems like sometimes not working tooltips in the main dialog.
- I guess the tool isn't actually that intuitive
- My English
- …

7. Statistics

- I worked on the tool itself for over a year
- 3 different languages were used: C#, C++ and C++/CLI (in my opinion the last two languages have enough differences to be judged as different ones)
- Including comments and “empty lines”, including code generated by the dialog designers, excluding Eliot’s UnrealLibrary, the main tool and the uzLib count about 29100 lines of code.
  The IniFile-library (which I didn’t create exclusively for this tool) has got another 11250 lines of code.
  So, all in all, the whole project consists of around 40300 lines. And that for such a useless tool (well, actually it wasn’t pointless to create as I learned a lot).
- VS2008 SP1 was used

8. The End

Most stuff created by Gugi, 2009-2011
UnrealLibrary created by Eliot (also thanks to Eliot for testing)

Released: 19 January 2011
Version: 0.1.0
Regard the release as a **beta**, i.e. I don’t guarantee for anything.
Contact: Wherever I released the tool

I plan to release at least the uzLib-source. I don’t know yet if I will publish all the code of the tool, though (I guess I will).