TN234

File Compression
(Using #zimport)

This document gives a brief description of the Dynamic C `#zimport` compression utility feature (available starting with Dynamic C version 8.00) and details its performance characteristics. The `#zimport` compiler directive is similar to `#ximport` with one important difference: the file is compressed before it is downloaded to the target device. This compression happens during compilation by invoking an external compression utility. The external utility may be replaced with a user-designed executable.

Compressed files are accessed in the same manner as #ximport files. To support the compression for user programs, several libraries are supplied (see below).

The #zimport Directive

The `#zimport` directive extends the functionality of `#ximport` to include file pre-processing by an external utility. The default utility supplied with Dynamic C is the `zcompress.exe` compression utility. The name of the utility is a project file option, and may be changed to reflect the needs of individual projects.

The syntax for `#zimport` is similar to `#ximport`:

```
#zimport "filename" symbolname
```

Where `filename` is the input file, and `symbolname` represents the 20-bit physical address of the downloaded file. The first 32 bits of the file is the length field, which contains the length (in bytes) of the file. The file is accessed in the same manner as a #ximported file, with one difference: the top bit of the length field (the most significant bit of the 32-bit length value) is set to indicate a compressed (pre-processed) file. The `zimport.lib` compression support library defines a mask `ZIMPORT_MASK` which can be used to extract the indicator bit or the length of the file.

The Compression Utility

The `#zimport` directive invokes a utility program to pre-process a file before it is downloaded to the target. The utility implements an LZ77 compression algorithm for use with the compression support libraries.

The utility's path is defined in the current project file, and may be changed to suit the needs of that project. This allows a user to replace `zcompress.exe` with a utility of his or her own. To edit the project file, open it in NotePad with Dynamic C closed. Find the following line:

```
Zimport External Utility=ZCOMPRESS.EXE
```

Substitute the name of your utility for `ZCOMPRESS.EXE`. Then save the project file. When you open Dynamic C and the project file you just edited, your utility will be invoked when `#zimport` is used.

A user replacement for the utility must have the following characteristics:

- The program must be in the same directory as the executable file for Dynamic C, `Dcrab_xxx.exe`.
- The program must be a console application (Windows EXE) and take a single parameter, which is the file path of the input file given in the `#zimport` statement.
- The program must output a file to the same directory as the input file. The file must have the same name, and the extension appended with `.DCZ`. E.g., `test.txt` becomes `test.txt.dcz`.
- The program must return 0 on success and non-zero on failure, so Dynamic C can handle errors in the utility program.

Compression Support Libraries

Several libraries are included with Dynamic C to support the compression utility and the `#zimport` directive. These are found in the `\LIB\ZIMPORT` directory. User programs need only include the `zimport.lib` library. Sample programs can be found in the `\SAMPLES\ZIMPORT` directory. A sample program named `zimport.c` that demonstrates different ways to use compressed files with the HTTP server can be found at `\SAMPLES\TCPIP\HTTP`.

The support libraries allow for on-the-fly decompression of `#zimport` files, as well as utilities for compressing to and decompressing from FS2 file system files. See the *Dynamic C User Manual* and the sample programs for more information.

Compression Ratio Examples

The LZ77 algorithm achieves an average compression ratio of better than 50% for text files, but is very dependent upon the input file. Some input files may exhibit better compression ratios, some far worse. For example, some files, such as JPEG graphics files (which are already compressed), may actually become larger when recompressed. To demonstrate the compression ratios for different types of files, we present some examples, summarized in the table below in order of decreasing benefit:

Table 1. Compression Ratios for Various File Types

File	File Type	Uncompressed Size (bytes)	Compressed Size (bytes)	Ratio (Compressed / Uncompressed)
RABBIT.HTML	HTML Page	25493	8120	32.204%
TCP_TIME.C	Dynamic C Source File	16479	7660	46.483%
LICENSE.TXT	Basic Windows Text	10858	5778	53.214%
ZCOMPRESS.EXE	Windows Executable	133632	77022	57.637%
LICENSEPDF.PDF	PDF File	14925	10680	71.557%
COFFEE BEAN.BMP	Windows BMP file	17062	15752	92.322%
PARADISE.JPG	JPEG File	62417	69411	111.205%

As can be seen from the compression ratios in the table, text files compress well, whereas graphics files benefit little, or, as in the case of JPEG, suffer from the additional compression. Also note that compressing small files (< 1 KB) provides little benefit, due to the small space savings and large decompression overhead.

Code and Data Space Considerations

In order to realize the benefits of this compression, the size of the support library code must be taken into consideration. For on-the-fly decompression straight from a `#zimport` file, the code footprint is about 2.5KB (actual size varies, depending on compiler options and compiler version). The code size for FS2 compression support using the function `CompressFile()` is about 6.6 KB. None of the compression support library code is forced to root.

Table 2. Library Code Size

Code Functionality	Code Footprint
Decompression only	2.5 KB
Compression and decompression	6.6 KB

The data space used by the libraries is a user-controlled option. All the data used by the compression support libraries resides in xmem, except for a small fixed array of pointers in root memory. For decompression, the minimum space required is 4 KB for the LZSS window. Compression requires an additional 24 KB. If multiple files are to be compressed/decompressed concurrently, each additional file will require a 4 KB buffer for decompression, and 24 KB for compression. The actual number of these buffers is controlled by 2 macros, which can be defined by the user at the beginning of a program.

`OUTPUT_COMPRESSION_BUFFERS` (default value = 0, must be at least 1 if compression is used)

`INPUT_COMPRESSION_BUFFERS` (default value = 1)

Performance

The performance of the decompression is based upon the compression ratio of the file being decompressed, and the speed of the processor. On a 44 MHz Rabbit 3000, the decompression algorithm achieves a throughput of 10-20 KB/second (size of uncompressed file/total time to decompress). Compression (from a `#ximport` file to an FS2 file) achieves a rate of 500-1000 bytes/second The performance characteristics are summarized in the table below:

Table 3. Throughput

Action	Speed
Decompression	10-20 KB/sec
Compress from `#ximport` to FS2 file	500-1000 bytes/sec