PKZIP Your Files Using APL and .NET

Gianluigi Quario

When dealing with ZIP files you have a few choices: use native APIs from third party Dlls, java APIs or dotNet APIs.

I gave up with Java, even if I was surprised to find out that we can find dotNET support for multi-file archives in dotNET buried in J# assemblies that offers parity with Java APIs.
I tried to use dotNet APIs from System.IO.Compress dotNET namespace, but I was very disappointed. For reasons only Microsoft knows, the support is limited to streams only and lacks completely for multi-files archives.

During 2007 the dotNet framework 3.0 delivered a new System.IO.Packaging dotNET namespace , probably inside the same stream as MS Office 2007.

Windows Presentation Foundation (WPF) uses packages to store content, resources, and relationships for pages and documents using a standard ZIP file by default. As with any ZIP file, your application can use the System.IO.Packaging classes to store and optionally protect any type or number of data files in a single efficient-to-access container.

But when I started to envisage a solution to the problem of substituting the executables PKZIP/PKUNZIP exploited by the application software I was working on, I could only revolve round dotNet 2.0

This was the reason why the choice of a third party dotNET library like SharpZipLib cropped up.

ICSharpCode.SharpZipLib.Dll is a dotNET compression library that supports ZIP files using PKZIP 2.0 style encryption, with GNU long filename extensions. It is written entirely in C# for the dotNET platform. It is implemented as an assembly , and thus can easily be incorporated into other projects (in any dotNET language).

The library is released under the GPL with the following exception: Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License cover the whole combination.

It can be implemented readily in a manner not covered by patents, and hence can be practiced and distributed freely.

I shall present three Dyalog APL functions that create and use objects that are instances of dotNET classes derived from this library and other dotNET base libraries.

These functions allow the compression and decompression of ZIP files. The functions call a few simple cover functions of System.IO dotNet namespace.

They are something more than a draft, as they are a service for APL Italiana's software. Nevertheless they lack many features provided by mature utilities like WinZip; for example, they don't worry to manage nested directories nor large files: we need to handle files with an uncompressed size of no more than 10 MBytes.

The performances are sometimes poor, but that constraint is not a problem when dealing with medium size files. I didn't take the matter further; maybe it is a problem connected to my poor knowledge of the capabilities of the SharpZip library.

I met these demands:

to archive, to compress, to protect, to transport, to decompress the data managed by APL Italiana's software

Primary goals:

- To archive a set of files to keep related files together
- To encrypt confidential files so that they can't be used without a password
- To check that the original ZIP file has not been replaced
- To deploy a compliant compressor producing data sets that conform to all popular archive and compression software.

Secondary goals:

- To make ftp, transporting, e-mailing faster and more efficient
- To save disk space for stored files that are seldom used

My way of working:

build only what you need now, not what you might need later

In consequence:

- Performance was less of a consideration than other things.
- Neglecting to deflate other formats of archive files
- No GUI interface
- No compression of nested directories,

- No zipping of a ZIP file
- No splitting of ZIP File to make large ZIP files manageable,
- No Extraction of a single stored file from a ZIP file,
- No change the name of files stored inside a ZIP file nor delete a stored file from a ZIP file

The APL functions:

- ZipNetIsZip
- ZipNetZip
- ZipNetUnzip

The requirements:

- Dyalog APL V11 or higher
- Dyalog APL .NET interface
- Microsoft .NET Framework SDK V1.1 or V2.0
- ICSharpCode.SharpZipLib.Dll V2.2: free software licensed

  This .DLL file must be saved in the same directory as DYALOG.EXE; alternatively may be saved elsewhere: you need to modify the three ZIP functions by substituting the line:

  ```
  'SOF'≡¯3↑,uflib:lib25
  ```

  with the line e.g. :

  ```
  'c:\myDirectory\mySubDirectory'
  ```

**** ZipNetIsZip ****

verify that a ZIP file can be managed by ZipNetZip and ZipNetUnzip functions.

syntax : ZipNetIsZip <filename>
filename is a full filename, it includes the directory path
right argument: simple character vector
returns: (return code:scalar integer)(message:char_vector)(CRCs:vector integer)(filenames: vector of char_vectors)

return code: is 1 if the file is valid for our ZIP functions
Warning: it happens sometimes that the file is checked as valid, but the ZipNetUnzip shall fail to inflate the compressed files (large file, unknown compression method, etc.)

message :	a short message (in case of error or not valid file)
CRCs :	a integer vector with the CRC's of compressed files
Filenames:	a vector of [full]names of compressed files

*** *ZipNetZip* ***

create a ZIP file including compressed files

```
syntax       :       ZipNetZip <filename(s) to be compressed>
               or
                   ZipNetZip <OUTPUT filename> <filename(s) to be compressed> [<password>]
right argument:  a)simple char_vector or simple char_matrix
              or
               b) vector of 2 char_arrays
                  first  array: simple char_vector
                  second array: simple char_vector or simple char_matrix or vector of
                              simple char_vectors
              or
               c) vector of 3 char_arrays
                  first  array: simple char_vector
                  second array: simple char_vector or simple char_matrix or vector of
                              simple char_vectors
                  third  array: simple char_vector
             in case a) the ZIP file is created inside the directory where the file(s)
              to be compressed lay

returns:     (return code:scalar integer)(message:char_vector)(CRCs:vector integer)(filenames:
            vector of char_vectors)

          return code: is 1  when OK, the ZIP file has been created
            a short message (in case of error or not valid file)
```

*** *ZipNetUnzip* ***

inflate all files included in a ZIP file

```
syntax       :      ZipNetUnzip <filename> [<output directory> [<password>]]
                    filename is a full filename, it includes the directory path
                    output directory is a full directory name, it includes the path
right argument:  a)simple char_vector of filename of ZIP file to be inflated
             or
               b)vector of 2 simple char_vectors:
                   filename of Zip file to be inflated
                   output directory
             or
               c)vector of 3 simple char_vectors:
                   filename of ZIP file to be inflated
                   output directory
                   password
             in case a) the compressed files are inflated inside the directory of
                        ZIP file to be inflated
returns:     (return code:scalar integer)(message:char_vector)(CRCs:vector integer)(filenames:
            vector of char_vectors)

            return code: is 1  when OK, the inflation has been performed
            a short message (in case of error or not valid file)
            CRCs :        a integer vector with the CRC's of compressed files
            Filenames:   a vector of [full]names of compressed files
```

NB. the ZIP file cannot be a container for directories; the inflation cannot succeed when it includes a directory (and its children files).

NB. the files are inflated inside the selected directory.
In the event that the compressed file were stored inside the ZIP file with fully qualified names, ZipNetUnzip shall ignore the full path.

My personal remarks:

The internal structure of a ZIP file is quite complicated. It should be unwise to handle it directly by means of APL and disregard an external library.

The dotNET platform is wide and I didn't want to become an expert about it before I could do something with it.

I achieved to use the base dotNET libraries SYSTEM.IO by a method like "go and try". Something now is more clear about those libraries, but something is still working without my understanding.

I found easy to use of a third party library (ICSharpCode.SharpZipLib.Dll) that has a good online help file. My major difficulty has been to inspect and handle the exceptions generated by it. My final solution was to trap (try and catch) nearly every call to the services of this library.

When dealing with an error, it is difficult to figure out what are the resources (external to APL) still engaged.