

MERMAID

EU Project No: IST-1999-10637

**DAE Data File
Compression and Parameterisation**

Deliverable Nos: D4
(Rev 1)

May 2001



Report Title:	DAE Data File Compression and Parameterisation
Customer:	European Commission, Directorate-General Information Society, IST Programme
BMT Report no:	13300/D/04
Deliverable nos:	D4
Report status:	Rev 1
Date:	May 2001
Contact details:	BMT MARINE INFORMATION SYSTEMS LTD Grove House, 7 Ocean Way, Ocean Village, Southampton, Hampshire, SO14 3TJ. United Kingdom. Tel: +44 (0) 2380 232222 Fax: +44 (0) 2380 232891 e-mail: mis@bmtmis.demon.co.uk Website: http://www.bmtmis.com

	Name	Signature	Date
Author:	Chris Rawlings		
Approved by:	Dr Andrew Tyler		

This report is commercial-in-confidence. Any information contained herein should not be communicated to any third party without prior permission of BMT Marine Information Systems Ltd.

Contributors

Company	Name
BMT	Chris Rawlings
	Paul Goddard
	Paul Taylor
TXT	Matteo Villa

Distribution List

Company	Name	Number of Copies
EU Commission (DG-XIII)	Guy Weets	6
BMT	Paul Taylor	1
TXT	Matteo Villa	1
CEDRE	Vincent Gouriou	1
Met. Office	Jack Hopkins	1
IMGW-OM	Wlodek Kryzminski	1
NC	Koen DeHulsters	1

Revision

Revision Number	Date	Author	Purpose of Revision
0	26/03/2001	Chris Rawlings	Initial Internal Release
1	25/05/2001	Paul Taylor	Reformatting

Table of Contents

1	INTRODUCTION	6
1.1	References.....	6
2	COMPONENTS OF A ZIP FILE.....	6
3	COMPRESSING DATA TO A ZIP FILE	6
3.1	Step 1 – Create the zip output stream.....	6
3.2	Step 2 – Open the source data file	7
3.3	Step 3: Create the zip entry	7
3.4	Step 4: Put the zip entry into the archive.....	7
3.5	Step 5 – Read source and write data to the zip output stream	7
3.6	Step 6 – Close the zip entry and other open streams	8
4	CONCLUSION.....	8

1 Introduction

This document forms part of the design documentation for the MERMAID Data Broker, specifying the built in functionality of compression within the JAVA API for use by the Data Access Engine (DAE), and is one of the formal deliverables of Workpackage 3 (Data Management).

It was decided by the consortium during the design phase that the most prudent and practical approach would be to utilise a standard, 'off-the-shelf' compression utility, rather than to develop one of our own. This has the advantages of using a tried and tested facility, which will be familiar to many users, whilst requiring only a very minimal amount of development time. The chosen utility has the added advantage of being freely available.

1.1 References

This document was created using the following document:

ZIP Data Compression with Java : By Chád Darby
http://www.j-nine.com/pubs/javazip/Java_Zip.html

2 Components of a zip file

A compressed file is described by a *zip entry*. The zip entry contains information for the compressed file such as the file name, original size, compressed size and additional file details.

3 Compressing data to a zip file

In order to compress data to a ZIP file, you can use the `ZipOutputStream` class. This class provides the functionality of writing the data in a compressed format. There is only a small amount of work required by the developer to create a ZIP file. The simple six-step process is outlined below:

3.1 Step 1 – Create the zip output stream

The `java.util.zip` package provides the class `ZipOutputStream` for writing ZIP method is set to `ZipOutputStream.DEFLATED`.

You also have the option of just storing the files in an uncompressed format. To accomplish this, you pass the value of `ZipOutputStream.STORED` to the `setMethod()` routine. However, when storing files in an uncompressed format, you must specify the CRC-32 checksum and the file size. Please reference the JDK 1.1 API for details on the `CRC32` class files. The constructor for this class accepts an

`OutputStream` object. Basically, you can pass the output stream of the file you are writing to.

Below is an example of how to create a ZIP file titled "modules.zip":

```
fos = new FileOutputStream("modules.zip");  
targetStream = new ZipOutputStream(fos);  
targetStream.setMethod(ZipOutputStream.DEFLATED);
```

Notice the call to `setMethod()`. By passing the value of `DEFLATED`, the `ZipOutputStream` object will store the files in a compressed manner. By default, the compression method is set to `ZipOutputStream.DEFLATED`.

You can also set the level of compression by calling the `setLevel(int aLevel)` method. The compression levels range from 1-9 with 1 being the weakest and 9 being the fastest level of compression.

3.2 Step 2 – Open the source data file

Now that the target zip output stream is created, you can open the source data file. In this code example, the file "java_intro.ppt" is the source data file:

```
String dataFileName = "java_intro.ppt";  
fis = new FileInputStream(dataFileName);  
sourceStream = new BufferedInputStream(fis);
```

3.3 Step 3: Create the zip entry

You will need to create a zip entry for each data file that is read. Recall from an earlier section that a zip entry contains file information such as the file name, original size, compressed size and additional details. Most of the zip entry information will be updated once the data is written to the zip output stream. Here is the code for creating a `ZipEntry` object:

```
theEntry = new ZipEntry(dataFileName);
```

3.4 Step 4: Put the zip entry into the archive

Before you can write information to the zip output stream, you must first put the zip entry object that was created in Step 3.

```
targetStream.putNextEntry(theEntry);
```

3.5 Step 5 – Read source and write data to the zip output stream

Finally, the coast is clear for you to read the source file and write the data. Since you are writing to a zip output stream, the data will be written in a compressed format without any additional work by you.

```
data = new byte[DATA_BLOCK_SIZE];
while((bCnt=sourceStream.read(data, 0, ATA_BLOCK_SIZE)) !=
-1)
{
    targetStream.write(data, 0, bCnt);
}
targetStream.flush();
```

3.6 Step 6 – Close the zip entry and other open streams

You can close the zip entry when you are finished writing the data. By closing the zip entry, the `ZipEntry` object is updated with the compressed file size, uncompressed file size and other file related information. It is also a good idea to close any open streams.

```
targetStream.closeEntry();
targetStream.close();
sourceStream.close();
```

4 Conclusion

As can be seen, the `java.util.zip` package is easy to use. By following a simple six-step process, you can create, read and write ZIP files. You can easily tune the compression level of files to favour speed or size. The package allows you to take advantage of the many benefits of reading and writing ZIP files.