data2pas : Reference Manual

for data2pas version 0.8.0
July 2015

Yann Mérignac

This manual is for `data2pas` version 0.8.0 (last updated 29 July 2015).

Table of Contents

1 Introduction

1.1 About this document

This document is the reference manual for `data2pas` version 0.8.0.

Chapter 2 explains how to compile and install `data2pas`. (Chapter 2 [Compiling and Installing data2pas], page 2)

Chapters 3 explains how to write a data file for `data2pas`. (Chapter 3 [Writing a Data Description File], page 3)

Chapter 4 describes in detail the `data2pas` data file syntax. (Chapter 4 [Data Description File Syntax], page 6)

1.2 Conventions

Commands that are entered by the user are shown preceded by '`~$`'.

```
~$ fpc myprog.pas
~$ ./myprog
```

Examples of `data2pas` input are set out from the normal text, and shown in a fixed width font, like this

```
1 FILE 'data.inc';
2 BEGIN
3   LazarusDescription : String = '';
4 END;
```

The line numbers at the beginning of the lines must not be typed. They are present to serve as references in the explanations.

1.3 About `data2pas`

`data2pas` is a command line tool to convert text or binary data into pascal constants. It takes in entry a text file describing the data and outputs one or more pascal file that you can include in your own source code. It's a tool similar to `data2inc` but using a more 'pascalish' syntax.

2 Compiling and Installing data2pas

2.1 Compiling

In order to compile data2pas you need :

- the Free Pascal compiler : http://www.freepascal.org/
- the data2pas sources : http://yann.merignac.free.fr/data2pas.html

When the data2pas sources have been uncompressed enter the data2pas source directory and run the following commands:

```
~$ fpc pasconf
~$ ./pasconf
~$ make
```

For extra configuration options try running ./pasconf -h.

2.2 Installing

To install data2pas on a Unix like system run make install.

On Windows just copy data2pas.exe to a directory listed in your PATH environment variable.

3 Writing a Data Description File

In this chapter we will generate a pascal file named "data.inc" wich will contain two constants. The first one will be a string extending over several lines. This kind of string is usally difficult to edit in a pascal source. The second constant will be an array of byte initialized with the content of a png file.

Let's start by creating the data description file with a simple text editor. I have decided to name it "data.d2p" but you can name it as you want. data2pas does not expect data description files to have a particular extension.

The first thing to do is to create a *FILE* section to indicate that we want to generate the file "data.inc":

Example: data.d2p
```
1 FILE 'data.inc';
2 BEGIN
3 END;
```

It looks a lot like pascal right ? Note that the file name can be either a relative name (as in our example) or an absolute name (eg "/home/user/data.inc").

Now we will declare our first constant :

Example: data.d2p
```
1 FILE 'data.inc';
2 BEGIN
3   LazarusDescription : String = '';
4 END;
```

If you know the Pascal language you should understand that we have declared a string constant named *LazarusDescription*. So far this string is empty. Let's see how to give it a more interresting value. We can use a standard pascal string. This has no interest but it works:

Example: data.d2p
```
1 FILE 'data.inc';
2 BEGIN
3   LazarusDescription : String =
4     'Lazarus is a Delphi compatible cross-platform IDE for Rapid '#10  +
5     'Application Development. It has variety of components ready for ' +
6     'use'#10'and a graphical form designer to easily create complex '  +
7     'graphical user'#10'interfaces.';
8 END;
```

We can also import a text file to initialize the string.

Example: data.d2p
```
1 FILE 'data.inc';
2 BEGIN
3   LazarusDescription : String = TEXT('lazdesc.txt');
4 END;
```

Importing an external file can be useful for a very long text but still requires you to add a file to your project that is already too big. Another solution is to use the heredoc syntax :

Example: data.d2p
```
1 FILE 'data.inc';
2 BEGIN
3   LazarusDescription : String = <<EOS
4 Lazarus is a Delphi compatible cross-platform IDE for Rapid
5 Application Development. It has variety of components ready for use
```

```
 6 and a graphical form designer to easily create complex graphical user
 7 interfaces.
 8 EOS>>;
 9 END;
```

We now focus on the creation of binary constant. So let's create the *LazarusIcon* constant and initialize it with the contents of a `png` file:

Example: `data.d2p`

```
 1 FILE 'data.inc';
 2 BEGIN
 3   LazarusDescription : String = <<EOS
 4 Lazarus is a Delphi compatible cross-platform IDE for Rapid
 5 Application Development. It has variety of components ready for use
 6 and a graphical form designer to easily create complex graphical user
 7 interfaces.
 8 EOS>>;
 9
10   LazarusIcon : Array of Byte = DATA('lazarus16x16.png');
11
12 END;
```

Now we can generate our `"data.inc"` file. Just go to the folder containing the file `"data.d2p"` and run:

```
~$ data2pas data.d2p
```

You should get a file `"data.inc"` that you can include in your pascal sources. It should look like this:

Example: `data.inc`

```
 1  const
 2    LazarusDescription = 'Lazarus is a Delphi compatible cross-platform' +
 3      ' IDE for Rapid'#10'Application Development. It has variety of co' +
 4      'mponents ready for use'#10'and a graphical form designer to easi' +
 5      'ly create complex graphical user'#10'interfaces.'#10;
 6
 7  const
 8    LazarusIcon : array [0..554] of Byte = (
 9      $89, $50, $4E, $47, $0D, $0A, $1A, $0A, $00, $00, $00, $0D, $49,
10      $48, $44, $52, $00, $00, $00, $10, $00, $00, $00, $10, $08, $06,
11      $00, $00, $00, $1F, $F3, $FF, $61, $00, $00, $00, $01, $73, $52,
12      $47, $42, $00, $AE, $CE, $1C, $E9, $00, $00, $00, $06, $62, $4B,
13      $47, $44, $00, $FF, $00, $FF, $00, $FF, $A0, $BD, $A7, $93, $00,
14      $00, $00, $09, $70, $48, $59, $73, $00, $00, $0B, $13, $00, $00,
15      $0B, $13, $01, $00, $9A, $9C, $18, $00, $00, $00, $07, $74, $49,
16      $4D, $45, $07, $DC, $06, $14, $0F, $1F, $36, $F0, $AF, $9F, $A6,
17      $00, $00, $01, $AB, $49, $44, $41, $54, $38, $CB, $95, $92, $41,
18      $68, $13, $41, $14, $86, $BF, $D9, $6E, $EC, $B1, $D0, $93, $57,
19      $AF, $5E, $24, $22, $48, $68, $D1, $83, $30, $11, $7B, $F5, $60,
20      $4E, $1A, $C1, $A6, $EB, $A1, $88, $50, $09, $85, $F4, $D0, $80,
21      $2C, $22, $22, $8B, $68, $5B, $C1, $D6, $53, $2E, $B9, $28, $49,
22      $21, $0B, $A1, $45, $10, $1B, $5A, $10, $3C, $45, $BD, $E7, $92,
23      $43, $C1, $83, $D0, $E2, $CE, $F3, $B0, $DD, $71, $D3, $26, $D8,
24      $FE, $30, $CC, $CC, $9B, $FF, $CD, $9B, $7F, $DE, $AF, $18, $81,
25      $4B, $D7, $1E, $3C, $17, $D4, $5D, $47, $61, $32, $2E, $6B, $BB,
```

```
26      $ED, $B5, $0A, $A7, $C5, $54, $BE, $D4, $CC, $69, $4F, $BA, $BD,
27      $48, $BA, $BD, $48, $72, $DA, $93, $EB, $B7, $4A, $1F, $4E, $95,
28      $7C, $FE, $E2, $9D, $89, $74, $F2, $FF, $2E, $51, $47, $15, $5B,
29      $46, $1C, $9D, $04, $4B, $8B, $CF, $C8, $5E, $18, $67, $3C, $33,
30      $66, $89, $3B, $3F, $7E, $B3, $FA, $F4, $89, $DD, $3B, $CA, $84,
31      $9F, $5B, $AB, $79, $35, $95, $2F, $B5, $44, $D0, $EF, $36, $5E,
32      $73, $16, $DC, $BF, $F7, $30, $52, $8A, $B6, $0B, $20, $38, $00,
33      $14, $66, $AE, $58, $C2, $D5, $1B, $B7, $79, $F4, $B8, $0C, $C0,
34      $CB, $17, $3E, $9D, $76, $DD, $9E, $D5, $9A, $7B, $08, $CE, $98,
35      $C2, $C4, $81, $9C, $F6, $24, $68, $F4, $A5, $DB, $8B, $24, $9B,
36      $CD, $8A, $48, $3C, $77, $7E, $1E, $C8, $FB, $AD, $FD, $81, $58,
37      $D0, $E8, $4B, $D0, $E8, $4B, $4E, $7B, $02, $E0, $02, $7C, $09,
38      $DF, $28, $40, $F4, $C6, $2B, $6A, $CD, $3D, $BC, $05, $1F, $80,
39      $CE, $F7, $5F, $B6, $AA, $B7, $E0, $53, $AC, $86, $F1, $0B, $82,
40      $A5, $24, $E7, $E8, $ED, $C7, $D0, $69, $D7, $2D, $19, $A0, $58,
41      $0D, $07, $24, $A4, $E1, $24, $12, $0A, $F3, $CB, $27, $0E, $D7,
42      $2B, $9A, $F5, $8A, $3E, $11, $2F, $CC, $2F, $63, $25, $C4, $2D,
43      $04, $7D, $79, $92, $C3, $3F, $C2, $F6, $B7, $FD, $B3, $18, $AE,
44      $E5, $C6, $66, $30, $03, $C9, $C5, $6A, $C8, $D7, $FA, $A2, $95,
45      $91, $5E, $FF, $33, $90, $19, $6D, $A4, $61, $72, $92, $CF, $3B,
46      $6E, $A4, $A1, $C4, $A4, $AD, $E9, $91, $68, $4E, $50, $2E, $97,
47      $47, $77, $C1, $51, $26, $4C, $57, $AB, $05, $4B, $9C, $73, $CD,
48      $C7, $34, $C7, $F7, $7D, $B1, $12, $86, $61, $FA, $E6, $5C, $23,
49      $32, $6A, $06, $20, $E3, $CA, $DB, $4F, $9B, $2B, $B3, $C3, $78,
50      $7F, $01, $CE, $48, $DF, $B7, $D4, $C3, $75, $30, $00, $00, $00,
51      $00, $49, $45, $4E, $44, $AE, $42, $60, $82
52    ); // LazarusIcon
```

4 Data Description File Syntax

Conventions

module
: denotes a non terminal

‘--’
: denotes a literal

[x]
: denotes zero or one occurrences of x.

{ x }
: denotes zero or more occurrences of x.

(x | y)
: means one of either x or y.

Data Description File

DataDescriptionFile → { *FileSection* }

You can insert comment in a data description file the same way you do it in pascal.

The following words are reserved : `ARRAY`, `BEGIN`, `BYTE`, `CHAR`, `COMPRESSED`, `DATA`, `END`, `FILE`, `OF`, `STRING`, `TEXT`.

File Section

FileSection → ‘`FILE`’ *FileName* ‘`;`’ ‘`BEGIN`’ *ConstantDeclarations* ‘`END`’ ‘`;`’

FileName → *StringConst*

`data2pas` will generate a pascal file called *FileName* for each file section.

Constants

ConstantDeclarations → *ConstantDeclaration* { *ConstantDeclaration* }

ConstantDeclaration → *Identifier* ‘`:`’ *Type* ‘`=`’ *Expression* ‘`;`’

Type → [‘`COMPRESSED`’] ‘`STRING`’ | ‘`RESOURCESTRING`’ | [‘`COMPRESSED`’] ‘`ARRAY`’ ‘`OF`’ ‘`CHAR`’ | [‘`COMPRESSED`’] ‘`ARRAY`’ ‘`OF`’ ‘`BYTE`’

If the `COMPRESSED` keyword is present the constant will be zipped before being outputed to pascal file. For decompression see the `FPC` *zstream* unit.

Array of byte are zero-based while array of char are one-based.

Expressions

Expression → *SimpleExpression* | *SimpleExpression* ‘`+`’ *Expression*

SimpleExpression → *StringConst* | *IntegerConst* | *DataImport* | *TextImport*

The plus sign is always treated as a concatenation operator. So if you write an expression like `65 + 66 + 67` the result is the 3 bytes length array `[65, 66, 64]` wich, for `data2pas` is strictly equivalent to the string `'ABC'`.

Binary Data Import

DataImport → ‘`DATA`’ ‘`(`’ *FileName* ‘`)`’

FileName → *StringConst*

The *DATA* function imports the file *FileName*.

Text Import

TextImport → 'TEXT' '(' *FileName* [',' *EOLString*] ')'

FileName → *StringConst*

EOLString → *StringConst*

The *TEXT* function used with a single parameter loads the file *FileName* replacing all end of line marks by the system end of line. With the 2 parameters version end of line marks are replaced by *EOLString*.

String Constants

String constants can be written like in pascal. But, they can also use double quotes. Control chars (ex : '#9', '#10', '#13') can also be used. In addition `data2pas` accepts some extra control chars.

#NUL = #0	#SOF = #1	#STX = #2	#ETX = #3
#EOT = #4	#ENQ = #5	#ACK = #6	#BEL = #7
#BS = #8	#HT, #TAB = #9	#LF = #10	#VT = #11
#NP = #12	#CR = #13	#SO = #14	#SI = #15
#DLE = #16	#DC1 = #17	#DC2 = #18	#DC3 = #19
#DC4 = #20	#NAK = #21	#SYN = #22	#ETB = #23
#CAN = #24	#EM = #25	#SUB = #26	#ESC = #27
#FS = #28	#GS = #29	#RS = #30	#US = #31
#SP = #32	#CRLF = #13#10	#LFCR = #10#13	#EOL = OS line ending

Heredoc

When you have to deal with long multi-line strings you can use the heredoc notation. Such a string starts with '<<Identifier' and ends with 'Identifier>>'.

Example: Simple heredoc

```
 1   Str1  : String = <<EOS
 2 line 1
 3 line 2
 4 line 3
 5 EOS>>;
 6
 7  Str2  : String = <<EOS
 8 This
 9 is
10 a
11 long
12 stringEOS>>;
```

If you want to indent a heredoc string you can start it with '<<-' rather than '<<'.

Example: Indented heredoc

```
 1   Str1  : String = <<-EOS
 2     |line 1
 3     |line 2
 4     |line 3
 5     |EOS>>;
 6
```

```
 7   Str2  : String = <<-EOS
 8     :This
 9     :is
10     :a
11     :long
12     :stringEOS>>;
```

By default all line breaks find in a heredoc string are replaced by the line ending of the system on wich `data2pas` is running. If you want to use a different line ending mark you can start your string with '`<<Identifier:Mark`'. Where *Mark* can take one the following values:

NUL = #0	SOF = #1	STX = #2	ETX = #3
EOT = #4	ENQ = #5	ACK = #6	BEL = #7
BS = #8	HT, TAB = #9	LF = #10	VT = #11
NP = #12	CR = #13	SO = #14	SI = #15
DLE = #16	DC1 = #17	DC2 = #18	DC3 = #19
DC4 = #20	NAK = #21	SYN = #22	ETB = #23
CAN = #24	EM = #25	SUB = #26	ESC = #27
FS = #28	GS = #29	RS = #30	US = #31
SP = #32	CRLF = #13#10	LFCR = #10#13	EOL = OS line ending

Example: Heredoc with line ending mark

```
 1   Str1  : String = <<EOS:LF
 2     |line 1
 3     |line 2
 4     |line 3
 5     |EOS>>;
 6
 7   Str2  : String = <<-EOS:CRLF
 8     :This
 9     :is
10     :a
11     :long
12     :stringEOS>>;
```

Integer Constants

Integer constants are limited in the range `[0..255]`. Like in pascal they can be written in decimal format, hexadecimal format (with '`$`'), octal format (with '`&`') and binary format (with '`%`').

5 Running data2pas

Running data2pas is very simple. The following command will generate the pascal files described in "datafile.d2p":

```
~$ data2pas datafile.d2p
```

data2pas options are:

`-h, --help`
: Print a usage message, then exit.

`-v, --version`
: Print the version number, then exit.

`-i, --indent=INDENT`
: Set indent size [1..40] (default 2).

`-w, --width=WIDTH`
: Set maximum line width [40..1024] (default 77).

`-o, --output-dir=OUTDIR`
: Set output directory.

Index