About VDSinout

- **VDS extension written by Peter Jacob .**
- **This extension allows reading and writing of character or binary files, completely or partially, in VDS.**
- **Version 2.0, April 1998.**
- For this extension to work:
- The script must start with a <External VDSinout> command.
- The file VDSinout.dll must be accessible. Place it either in your script directory or in your windows directory.
- The binary coding uses two characters per HEX byte and is referred in this file as Character Coded Byte (CCB).
  This system ensures the compatibility between Hex and Character strings, which are the only ones recognised by VDS.
- Inside the functions however, the counts are always expressed in bytes.

New features in Release.

VDSINOUT 2.0 Released 1 April 1998

New Feature in Version 2.0 of VDSINOUT are:

Enrypting files Encrypting files.
Enrypting files Encrypting strings.
Erasing files Wipe out files.

Help corrected:
For certain functions were a 32767 limit described, which was never implemented. The following functions have no 32767 limit:

- FPOSC
- FPSOB
- COMPARE
- APPND
- XTRCT
- ENCRYPT
- DECRYPT
- ERASE
- CREATE

VDSINOUT 1.0 Released 1 February 1998

Input from files.

Syntax:

<variable> = @IO(READC,<file description>,<pos1>,<pos2>,<null>)
<variable> = @IO(READB,<file description>,<pos1>,<pos2>)

Description:

@IO(READC) returns as a string of characters. Null characters (x00) are returned by default as CR (x0D), except if another replacement character is given as the optional parameter <null> or defined by IO NULLC.
@IO(READB) returns as a string of Character Coded Bytes (CCB, 2 characters/byte).

This function returns the content of the first file that matches <file description> if it exists , starting at byte position <pos1> and ending at byte position <pos2> in the source file. The bytes in the file are counted from 1.
<pos2> can also be defined as length, see the command IO TOPARM

If< pos1> is zero, negative or omitted, <pos1> is set to 1.
If< pos2> is zero, negative or omitted, the whole file, starting at <pos1>, is returned to <variable>, up to 32767 bytes.
If the difference <pos2>-<pos1> is greater than the remaining source size from <pos1> or 32767, the bytes are read up to the source size or 32767.

If <pos1> is greater than the size of the file, or if the difference <pos2>-<pos1> is negative or if the source file does not exist, then the function fails, OK is set to false and an error code can be read with @IO(MSNBR) or @IO(MSLIN).

See also IO EXCLUSIVE, IO REDUCE.

Example:

Output to files.

Syntax:

<variable1> = @IO(WRITC,<variable2>,<file description>,<pos1>,<pos2>,<pad>)
<variable1> = @IO(WRITB,<variable2>,<file description>,<pos1>,<pos2>,<pad>)

Description:

@IO(WRITC) Writes the string <variable2> as a string of characters.
@IO(WRITB) Writes the string <variable2> as a CCB string.

This function returns the number of bytes written and writes the string <variable2> to the first file that matches <file description>, starting at byte position <pos1> and ending at byte position <pos2> in the destination file. The bytes are counted from 1. If the file does not exist, it is created and null bytes are patched up to <pos1>.
If <pos1> is greater than the length of the file, null bytes are padded up to <pos1>. The padding character can be changed with the optional parameter <pad>, or with the command IO PADC.
<pos2> can also be defined as length, see the command IO TOPARM.
If the string was read compressed with the command IO REDUCE, then it is decompressed before being written.

If< pos1> is zero, negative or omitted, <pos1> is set to 1.
If< pos2> is zero, negative or omitted, the whole string is written to the file, up to 32767 bytes.
If the difference <pos2>-<pos1> is greater than the remaining source size from <pos1> or 32767, the bytes are read up to the source size or 32767.

If the difference <pos2>-<pos1> is negative or if the source string does not exist, then the function fails, OK is set to false and an error code can be read with @IO(MSNBR) or @IO(MSLIN).

Example:

Input from files to memory.

Syntax:

<variable> = @IO(FETCH,<file description>,<pos1>,<pos2>)

Description:

This function returns the number of bytes fetched to memory and fetches the content of the first file that matches <file description> to memory, starting at byte position <pos1> and ending at byte position <pos2>. The bytes in the file are counted from 1.
<pos2> can also be defined as length, see the command IO TOPARM.

If< pos1> is zero, negative or omitted, <pos1> is set to 1.
If< pos2> is zero, negative or omitted, the whole file, starting at <pos1>, is fetched to memory, up to 32767 bytes.
If the difference <pos2>-<pos1> is greater than the remaining source size from <pos1>, or 32767, the bytes are read up to the source size or 32767.

If <pos1> is greater than the size of the file, or if the difference <pos2>-<pos1> is negative or if the source file does not exist, then the function fails, OK is set to false and an error code can be read with @IO(MSNBR) or @IO(MSLIN).

See also IO EXCLUSIVE.

Example:

Output from memory to files.

Syntax:

<variable> = @IO(STORE,<file description>,<pos1>,<pos2>,<pad>)

Description:

This function returns the number of bytes stored and stores the string previously loaded with @IOFETCH, to the first file that matches <file description>, starting at byte position <pos1> and ending at byte position <pos2>. The bytes are counted from 1. If the file does not exist, it is created and null bytes are padded up to <pos1>.
If <pos1> is greater than the length of the file, null bytes are padded up to <pos1>. The padding character can be changed with the optional parameter <pad>.
<pos2> can also be defined as length, see the command IO TOPARM.

If< pos1> is zero, negative or omitted, <pos1> is set to 1.
If< pos2> is zero, negative or omitted, the whole file in memory, starting at <pos1>, is stored, up to 32767 bytes.
If the difference <pos2>-<pos1> is greater than the remaining source size from <pos1> or 32767, the bytes are read up to the source size or 32767.

If the difference <pos2>-<pos1> is negative or if nothing was previously fetched, then the function fails, OK is set to false and an error code can be read with @IO(MSNBR) or @IO(MSLIN).

Example:

Searches.

Syntax:

<variable> = @IO(FPOSC,<file description>,<pos1>,<pos2>,<string>)
<variable> = @IO(FPOSB,<file description>,<pos1>,<pos2>,<string>)

Description:

@IO(FPOSC) searches with a character string.
@IO(FPOSB) searches with a CCB string.

This function returns the position of the first occurrence of <string> in the first file that matches <file description>, starting at byte position <pos1> and ending at byte position <pos2>. The bytes are counted from 1. see also IO TOPOS
If <string> is not found the function returns 0.
<pos2> can also be defined as length, see the command IO TOPARM.

If< pos1> is zero, negative or omitted, <pos1> is set to 1.
If < pos2> is zero, negative or omitted, the file is searched, starting at <pos1>, up to the end of the file.
If the difference <pos2>-<pos1> is greater than the remaining source size from <pos1>, the file is searched up to the end .

If <pos1> is greater than the size of the file, or if the difference <pos2>-<pos1> is negative or if the source file does not exist, then the function fails, OK is set to false and an error code can be read with @IO(MSNBR) or @IO(MSLIN).

See also IO EXCLUSIVE.

Example:

Appending files.

Syntax:

<variable> = @IO(APPND,<file description1>,<file description2>)

Description:

This function appends very fast <file description1> by <file description2>.
It returns the number of bytes appended.

OK:

Set to false if the function fails.

Example:

%y = @IO(APPND,file1.txt,file2.txt)
Sets file1.txt = file1.txt with file2.txt appended.

Extracting files.

Syntax:

<variable> = @IO(XTRCT,<file description1>,<pos1>,<pos2>,<file description2>)

Description:

This function returns the number of bytes written and reads from <pos1> to <pos2> bytes out of <file description1> and saves them in <file description2>. The bytes in the file are counted from 1.
<pos2> can also be defined as length, see the command IO TOPARM.

If< pos1> is zero, negative or omitted, <pos1> is set to 1.
If< pos2> is zero, negative or omitted, the whole file, starting at <pos1>, is read to <file description2>.

If <pos1> is greater than the size of the source file, or if the difference <pos2>-<pos1> is negative or if the source file does not exist, then the function fails, OK is set to false and an error code can be read with @IO(MSNBR) or @IO(MSLIN).

See also IO EXCLUSIVE.

Example:

Comparing files.

Syntax:

<variable> = @IO(COMPARE,<file description1>,<pos1>,<pos2>,<file description2>)

Description:

This function compares from <pos1> to <pos2> the bytes out of <file description1> to the bytes at the same position in <file description2>. The bytes in the file are counted from 1. It returns the position of the first byte which differs, relative to <pos1>.
<pos2> can also be defined as length, see the command IO TOPARM.

If< pos1> is zero, negative or omitted, <pos1> is set to 1.
If< pos2> is zero, negative or omitted, the whole file, starting at <pos1>, is compared to <file description2>.

If <pos1> is greater than the size of the source file, or if the difference <pos2>-<pos1> is negative or if the source file does not exist, then the function fails, OK is set to false and an error code can be read with @IO(MSNBR) or @IO(MSLIN).

See also IO EXCLUSIVE.

Example:

Encrypting files.

Syntax:

<variable> = @IO(ENCRYPT,<file description1>,<file description2>,pass-phrase)
<variable> = @IO(DECRYPT,<file description1>,<file description2>,pass-phrase)

Description:

@IO(ENCRYPT,<file description1>,<file description2>,pass-phrase)	Encrypts the file
@IO(DECRYPT,<file description1>,<file description2>,pass-phrase)	Decrypts the file

This function encrypts or decrypts <file description1> to <file description2>. For the process a pass phrase must be supplied, which is used to encrypt/decrypt the data.
If the function fails, OK is set to false and an error code can be read with @IO(MSNBR) or @IO(MSLIN).
If function encrypted/decrypted successfully, OK is set to true (1).

!! Warning: there is no way to encrypt data, when the pass phrase is lost.

It might be a good idea to decrypt an encrypted file, to verify it with the source-file. This action can be performed before the source file is wiped out.

See also IO EXCLUSIVE.

Encrypting strings.

Syntax:

<variable> = @io(ENCCCB,string-to-encrypt-in-CCB-format,pass-phrase)
<variable> = @io(DECCCB,string-to-decrypt-in-CCB-format,pass-phrase)
<variable> = @io(ENCSTR,string-to-encrypt-in-normal-format,pass-phrase)
<variable> = @io(DECSTR,string-to-decrypt-in-CCB-format,pass-phrase)

Description:

@io(ENCCCB,string-to-encrypt-in-CCB-format,pass-phrase)
the string to encrypt has CCB format, the function returns the encrypted string also in CCB format.
@io(DECCCB,string-to-decrypt-in-CCB-format,pass-phrase)
the string to decrypt has CCB format, the function returns the decrypted string also in CCB format.
@io(ENCSTR,string-to-encrypt-in-normal-format,pass-phrase)
string to encrypt has normal character format (is readable), the function returns the encrypted string in CCB format.
@io(DECSTR,string-to-decrypt-in-CCB-format,pass-phrase)
string to decrypt has CCB format, the function returns the decrypted string also in normal character format.

Encrypted strings always have CCB format, because encryption may creates unreadable characters.

This functions encrypts or decrypts a string, which is either in CCB or normal string format. For the process a pass phrase must be supplied, which is used to encrypt/decrypt the data.
If the function fails, OK is set to false. If function encrypted/decrypted successfully, OK is set to true (1).

!! Warning: there is no way to encrypt data, when the pass phrase is lost.

Example:

Wipe out files.

Syntax:

<variable> = @io(ERASE, <file description1>)
<variable> = @io(CREATE, <file description1>)

Description:

<variable> = @io(ERASE, <file description1>,erase-string)
Erases an existing file, by overwriting the contents with an erase string.
<variable> = @io(CREATE, <file description1>)
Creates a new file with size 0, if the file exists, its size is reset to 0.

This functions help the Encrypting process to get rid of the original source file. A normal FILE DELETE process may leave the data on disk, because just the FAT entry is reset. Disk editing program may easily recover the data, by reviewing the hard disk.
A more save process of deleting files is:

1. Overwriting the data with a hex combination
2. Reset the file size to 0
3. finally delete the file

The data should not be recoverable.

Example:

```
rem This Script implements a WIPE out utility, to unrecoverable remove
rem files
rem It must be called with the file name as parameter
external vdsinout
%2 = @io(ERASE,%1,FF00FF)
%2 = @io(ERASE,%1,00FF00)
%2 = @io(ERASE,%1,FF)
%2 = @IO(create,%1)
if @ok()
   FILE SETDATE,%1,11:45,15.3.93
   FILE DELETE,%1
end
```

Comparing CCB's.

Syntax:

<variable> = @IO(CLC,<string1>,<string2>,<resultL>,<resultE>,<resultG>)

Description:

This function returns a value as result of the comparison <string1>,<string2>
if string1<string2 resultL is returned
if string1=string2 resultE is returned
if string1>string2 resultG is returned

It can be used it for conditional branches:

This function must be used when comparing two single CCB 's because of a bug in the VDS @equal() function which returns true when comparing 00 or E0 to E9 to different values in the same range.

Optionally, this bug can be avoided with a dummy character:
@equal(00,E0) returns true but
@equal(x00,xE0) returns false

Example:

```
goto @IO(CLC,%a,%b,ltlab,eqlab,gtlab)
…
:ltlab
…executes if %a < %b
:eqlab
…executes if %a = %b
:gtlab
…executes if %a > %b
```

Error messages

Syntax:

<variable> = @IO(MSNBR)
<variable> = @IO(MSLIN)

Description:

Every IO function records is message internally, up to 16 messages may be saved.
The message buffer is cleared whenever a new @IO function is invoked.

To retrieve the recorded message numbers for the last call, you can use: @IO(MSNBR) which returns the message numbers of the last call.
If more than one message is stored, they are returned as concatenated string, separated by |.

To retrieve the recorded messages for the last call, you can use: @IO(MSLIN) which returns the messages of the last call.
If more than one message is stored, they are returned as concatenated string, separated by |.

To check the messages is only necessary, if @OK() is returning false, in this case the very last message is the one, which should indicate, what reason causes the function to abort. The other messages may be treated as informational message.

If @OK() is true, all messages contained in the message buffer, must be treated as informational messages.

Messages:

1,file-name, file not present
2,file-name, already open by other application
3,file-name, open error nnnn
4,file-name, EOF reached;
5,file-name, unknown error in set to start position nnnn
6,file-name, unknown error during write of fill characters;

10,hex-string at position nnnn is an invalid hex combination
11,file-name, offset nnnn, greater than file length
12,nnnn, start offset negative, or not numeric;
13,nnnn, end offset or length negative, or not numeric
14,nnnn, start greater than end offset nnnn
15,nnnn, missing, set to nnnn
16,nnnn, invalid nnnn, set to nnnn
17,nnnn, less than minimum value nnnn
18,nnnn, greater than maximum value nnnn

20,search string not present, or invalid;
21,write string not present, or invalid;

30,String size exceeds 32767 byte;
31,String size exceeds 65534 byte;
32,Pass phrase not present
33,String to encrypt is not in CCB format
34,string to encrypt not present

35,string to decrypt not present
36,string to decrypt is not in CCB format

file-name is the specified file name
nnnn is an integer value
hex-string represent a 2 byte CCB hex string

Example:

%E = @IO(MSLIN)
then %E = a text line.

Position 2 parameter

Syntax:

IO TOPARM,<OFFSET/LENGTH>

Description:

This command defines how the <pos2> parameter in the functions have to be interpreted:

LENGTH means it defines a length

OFFSET means it defines an offset, that is default.

The settings of the command remain active up to the next similar command.

Padding Characters

Syntax:

IO PADC,<character>

Description:

This command defines the padding byte for files newly created or appended files, before offset starts. The settings of the command remain active up to the next similar command.

Null Character

Syntax:

IO NULLC,<character>

Description:

This command defines the translation character used by @IO(READC) for null bytes.
The settings of the command remain active up to the next similar command.

Return from @IO(FPOSC) @IO(FPOSB)

Syntax:

IO TOPOS,<ABSOLUTE,RELATIVE>

Description:

This command defines how the result of @IO(FPOSC) or @IO(FPOSB) should be returned
ABSOLUTE (default), returns the absolute offset in the file
RELATIVE returns the offset counted from the function <pos1> point.
The settings of the command remain active up to the next similar command.

Read from already open files

Syntax:

IO EXCLUSIVE,<OFF,ON>

Description:

When OFF, this command let the DLL ignore the already open message when reading files only.
The settings of the command remain active up to the next similar command.

Compression

Syntax:

IO REDUCE,<ON/OFF>

Description:

When ON, this command activates the compression during @IO(READB) and the decompression during @IO(WRITB)
The settings of the command remain active up to the next similar command.

Version

Syntax:

<variable> = @IO(VERSION)

Description:

This function returns the version of VDSinout.dll.

Character Coded Byte (CCB)

CCB is used in the communications between a binary file and VDS.
The reason is that VDS only recognises character strings and that a null byte (x00) is interpreted as an end of string.

To be able to read successfully binary files, a translation scheme is used which transforms each byte in his hex notation
as two characters.
As example, the byte with a value of 31 is translated in the two characters 1 and F according to his hex representation x1F. The null byte x00 is translated in two characters = 00.
Consequently, an hex string of 32 bytes translates in a character string of 64 characters that can be handled by VDS.
The translations in both directions are implemented automatically by all @IO() functions with a name ending with B.
In a VDS script, these strings can be converted, compressed and decompressed etc. See the annexed subroutines.

Conversion subroutines

By E. de le Court. Expand the window to avoid scrolling.

```
:STR2CCB
rem Character string to CCB string. %1 = input character string.
%2 =
repeat
  %3 = @substr(@hex(@asc(@substr(%1,1))),3,4)
  %2 = %2%3
  %1 = @strdel(%1,1)
until @null(%1)
exit
rem %2 = output CCB string.
rem Changed: %1, %2, %3.
-----------------------------------------------------------------
:CCB2STR
rem CCB string to character string. %1 = input CCB string.
%2 =
repeat
  %3 = @chr($@substr(%1,1,2))
  %2 = %2%3
  %1 = @strdel(%1,1,2)
until @null(%1)
exit
rem %2 = output character string.
rem Changed: %1, %2, %3.
-----------------------------------------------------------------
:CCB2VAL
rem CCB string to value. %1 = input CCB sting, max 8 chrs = 4 bytes
rem most significant byte last.
%2 = @sum($@substr(%1,7,8)@substr(%1,5,6)@substr(%1,3,4)@substr(%1,1,2),0)
exit
rem %2 = decimal value
rem Changed: %2.
-----------------------------------------------------------------
:VAL2CCB
rem Value to CCB string. %1 = input value, integer
%2 = @hex(%1)
%1 = @len(%2)
if @equal(%1,3)@equal(%1,5)@equal(%1,7)
  %2 = 0%2
end
%2 = @substr(%2,7,8)@substr(%2,5,6)@substr(%2,3,4)@substr(%2,1,2)
exit
rem %2 = CCB string, 2 to 4 bytes, most significant byte last.
rem Changed: %1, %2
-----------------------------------------------------------------
:compress
rem Compress CCB string. %1 = input CCB string.
%5 = 1
%3 = @substr(%1,1,2)
repeat
  %2 = %3
  %4 = %5
  %6 = 0
  repeat
    %6 = @succ(%6)
    %5 = @sum(%5,2)
    %3 = @substr(%1,%5,@succ(%5))
  until @not(@equal(x%2,x%3))
```

```
  if @greater(%6,2)
    %1 = @strdel(%1,%4,@pred(%5))
    %1 = @strins(%1,%4,<%6>%2)
    %5 = @diff(@sum(%5,@len(%6),4),@prod(%6,2))
  end
until @nul(%3)
%2 = %1
%1 =
exit
rem %2 = output CCB compressed string idem as with IO REDUCE,ON.
rem Used %1, %2, %3, %4, %5, %6
-------------------------------------------------------------
:decomp
rem Decompress CCB string, compressed with above or IO REDUCE,ON.
rem %1 = input CCB compressed string.
%2 =
repeat
  option fieldsep,<
  parse "%3;%6",%1
  option fieldsep,>
  parse "%6",%6
  %4 = @sum(@len(%3),@len(%6),2)
  %1 = @strdel(%1,1,%4)
  %5 = @substr(%1,1,2)
  %2 = %2%3
  if @not(@zero(%6))
    %6 = @pred(%6)
    repeat
      %2 = %2%5
      %6 = @pred(%6)
    until @zero(%6)
  end
until @nul(%1)
exit
rem %2 = output CCB string, decompressed.
rem Used %1, %2, %3, %4, %5, %6, and fieldsep
```

DBase subroutines.

```
External VDSINOUT
IO EXCLUSIVE,OFF
IO TOPARM,LENGTH
IO REDUCE,OFF

:Basereg
rem %1 = path\name.ext of database
%2 = @IO(READB,%1,9,2),rec pos
if @not(@ok())
  exit
end
%2 = @sum($@substr(%2,3,4)@substr(%2,1,2),0)
%3 = @IO(READB,%1,11,2),rec len
%3 = @sum($@substr(%3,3,4)@substr(%3,1,2),0)
registry write,CURUSER,software\eco\Dbase\@name(%1),~Record,%2|%3
%3 = 33
%4 = 2
repeat
  %5 = @IO(READC,%1,%3,11)
  %5 = @substr(%5,1,@pred(@pos(@chr(13),%5)))
  %3 = @sum(%3,16)
  %6 = @sum($@IO(READB,%1,%3,1),0)
  registry write,CURUSER,software\eco\Dbase\@name(%1),%5,%4|%6
  %4 = @sum(%4,%6)
  %3 = @sum(%3,16)
until @equal(%3,%2)
exit
rem %1 = path\name.ext of database
rem Used: %2,%3,%4,%5,%6
rem OK false if failed.

:Bgetfield
rem %1 = path\name.ext of database (must be registered)
rem %2 = record nummer (from 1)
rem %3 = field name
parse "%5;%6",@regread(CURUSER,software\eco\Dbase\@name(%1),~Record)
parse "%3;%7",@regread(CURUSER,software\eco\Dbase\@name(%1),%3)
%2 = @sum(%5,@prod(%6,@pred(%2)),%3)
%2 = @IO(READC,%1,%2,%7)
exit
rem %1 = path\name.ext of database
rem %2 = field data
rem Used: %2,%3,%5,%6,%7
rem OK false if failed

:Bputfield
rem %1 = path\name.ext of database (must be registered)
rem %2 = record nummer (from 1)
rem %3 = field name
rem %4 = field data
parse "%5;%6",@regread(CURUSER,software\eco\Dbase\@name(%1),~Record)
parse "%3;%7",@regread(CURUSER,software\eco\Dbase\@name(%1),%3)
%2 = @sum(%5,@prod(%6,@pred(%2)),%3)
%4 = @strins(%4,@succ(@len(%4)),"                                    ")
%4 = @strdel(%4,@succ(%7),256)
%2 = @IO(WRITC,%4,%1,%2,%7)
exit
rem %1 = path\name.ext of database
rem %2 = number of bytes written
```

```
rem Used: %2,%3,%4,%5,%6,%7
rem OK false if failed
```

Example

By E. de le Court.

Pack2bin, use it as follows:

1) Start writing a script, using all the BMPs etc.you need from the current directory.
   Test it thoroughly.
2) Save it as: "newscript.dsc"
3) Run Pack2bin. Drag and drop all auxiliary files. Select "newscript.dsc" as destination and pack.
4) The auxiliary files, in CCB, and the unpacking subroutine are inserted at the very beginning of
   "newscript.dsc" to make them available when this script starts. The variables can be reused.
3) Compile "newscript.dsc", move the exe to another directory and test it. The auxiliary files are not
   necessary now.
4) If, after loading them, the script deletes the auxiliary files, they don't appear in the new directory and
   the user can not modify them.

Remark: This script was partially created by itself, to include Ecolbl.bmp, compressed.

```
External VDSinout
IO REDUCE,ON
IO EXCLUSIVE,OFF
List 1,create
List 1,loadtext
"424D5608<6>00360400002800000030000000160000000010008<5>0020040000CE0E0000D80E<
16>00800000800000008080008000000080008000808000000C0C0C000808080000000FF0000FF0
00000FFFF00FF000000FF00FF00FFFF0000FFFFFF005D0070F45B005249B87F<4>FF7CF45B00CD
1CB77F0A00000001<7>000A00000094090000EEF45B008C6E02008C6E0200C009<8>00B1214C00
B900D70000007466020094090000110100000 4E1<18>00A080BA7F<4>001CF45B00D8F45B00B84
BB87F<4>FFBCF45B00261BB77F881B5D00940900000A00000001<7>00A4F45B00128500006336F
7BF940900000A00000001<7>00072EEC8457010000D0F45B002319F7BF<4>005F300100E528F9B
F791AF7BF0C850000F4F85B002E19F7BF072EEC84<4>00EC845F30<4>00460200004A8502002E3
B0000C71600003F273F0100005F303F273F015085FA3BE71604<7>00F91AB77F<4>00010000000
A000000FA3B10<7>003C85D379FFFF5F305085413CE716FFFF5F30<4>005F30570174668C85000
00300F91AB77FE782AF17<6>0001000A00940957010000080000000C7165919B77F
"14F65B0094F55B00074BB87F<4>FFA0F55B00E218B77F14F65B00ACF55B00C24AB87F<4>FFB8F
55B00B0B3B77F14F65B00C4F55B003569B87F<4>FFD0F55B006161410014F65B00DCF55B006E61
4100<4>FFE8F55B00C160410014F65B00F4F55B00DC604100<4>FF00F65B001861410014F65B00
F4F85B0025614100<4>FF00F95B009C5A41001101000001<7>0098A0BA7F01<15>00EC2D120101
<11>0008932280<8>000570<18>004C0000009409<6>0008AE5D<5>0005000000300000074AE5
D0004010000B4F65B0040<7>0060A85D00260808003400380 08CA85D<5>00F2B9B87F<8>00C826
BB7F<8>0065636F2E626D70009809000088 1B5D0020000000ECF65B005D18B77F010000009409<
6>00200000009809000012000002881B5D009CF75B00471CB77F<4>009809000012000002881B5
D008976B77F120000020500582C0000280F00002487507F2F01537F280F010047016687E14A2F0
164134701582C0000280F000066875B004E875B<5>0001004B8001000000280F00006F024701EF
000000404166876487000 00C0000002C0FEF00504E874B4000
"00006F02D70000004041402800008 4B27005A62B200002002E0F51317F3120002E0FB7059E873
04F882C5D00582C0000AA87507F2F01537F582C2979BEB82AAC2AACB705EC87AC116704360F8A0
8<98>09<44>0B<4>090B<43>0B<4>090B0B<11>09<6>0B<5>09<9>0B<5>09<6>0B<4>090B0B<11
>09<4>0B09<8>09<5>0B<9>09<4>0B090909090B0B<11>090B0B0B<10>09<4>0B09<10>090B0B0
B<4>090B0B090909<11>0B090909<4>0B090909090B0B0B<4>090B0B0B<4>090B0B0B<4>090B0B
090909<10>0B090909<6>0B09090B0B0B<4>090B0B0B
"0B0B<4>090B0B<4>090B0B090909<10>0B090909<11>0B090909<7>0B0909090B0B<4>090B0B<
10>090B0B0B090909<11>0B090909<7>0B0909090B0B<4>090B0B<10>090B0B0B090909<11>0B0
90909<7>0B0909090B0B<4>090B0B<10>090B0B0B090909<11>0B090909<7>0B0909090B0B<4>0
90B0B090909<10>0B090909<6>0B09090B0B0B<4>090B<4>0B090909090B0B<4>090B0B090909<
```

```
11>0B090909<4>0B090909090B0B0B<4>090B0B0B<4>090B0B0B<4>090B0B<11>090B0B0B<10>0
9<4>0B09<10>090B0B0B<4>090B0B<11>09<4>0B09<8>09<5>0B<9>09<4>0B090909090B0B<11>
09<6>0B<5>09<9>0B<5>09<6>0B<4>090B<43>0B<4>090B<7>0B
"<36>0B<98>09
%W = ecolbl.bmp
gosub unpack
list 1,clear
list 1,close
goto DSC
:unpack
%I = 1
repeat
  %X = @next(1)
  if %X
    %J = @io(WRITB,%X,%W,%I)
    %I = @sum(%I,%J)
  end
until @null(%X)
exit
:DSC
DIALOG CREATE,Pack to DSC,-1,0,400,216,-
  DLGTYPE(DRAGDROP),-
  BUTTON(BDES;2;10;;;Destination),-
  EDIT(EDES;4;82;248),-
  LIST(LFIL;50;10;374;124),-
  BUTTON(BPACK;180;10;100;;Pack all files),-
  BUTTON(BCOM;180;130;70;;Compile),-
  CHECK(INT;182;206;70;;Integrated;;CLICK),-
  BUTTON(BREM;180;286;100;;Remove Selected),-
  TEXT(TXT;30;10;374;;Drag and drop auxiliary files here),-
  BITMAP(eco;4;336;50;22;ecolbl.bmp)
  file delete,ecolbl.bmp
  dialog set,EDES,@windir()\Desktop\Tmp.dsc
:evloop
  dialog cursor
  wait event
  goto @event()
:BDESbutton
  directory change,@path(@dlgtext(EDES))
  %E = @filedlg(Script file|*.DSC)
  if %E
    if @null(@ext(%E))
      %E = %E.dsc
    end
    dialog set,EDES,%E
  end
  goto evloop
:DRAGDROP
  list LFIL,dropfiles
  goto evloop
:BREMbutton
  %I = @index(LFIL)
  dialog clearsel,LFIL
  if @both(@greater(@count(LFIL),%I),@greater(%I,-1))
      list LFIL,seek,%I
  else
    %I = @pred(%I)
```

```
    if @greater(%I,-1)
      list LFIL,seek,%I
    end
  end
  goto evloop
:BPACKbutton
  dialog cursor,wait
  %C = @count(LFIL)
  If @greater(%C,0)
    list LFIL,seek,0
  else
    goto evloop
  end
  list 1,create
  list 1,add,"External VDSinout"
  list 1,add,"IO REDUCE,ON"
  list 1,add,"List 1,create"
  repeat
    %W = @item(LFIL)
    list LFIL,delete
    %V = @file(%W,Z)
    list 1,add,"List 1,loadtext"
    %I = 0
    repeat
      %J = @sum(%I,500)
      if @greater(%J,%V)
        %J = %V
      end
      %I = @succ(%I)
      %X = @IO(READB,%W,%I,%J)
      list 1,add,@chr(34)%X
      %I = %J
    until @equal(%J,%V)
    list 1,add,"%W = "@name(%W).@ext(%W)
    list 1,add,"gosub unpack"
    list 1,add,"list 1,clear"
    %C = @pred(%C)
  until @zero(%C)
  list 1,add,"list 1,close"
  list 1,add,"goto DSC"
  list 1,add,":unpack"
  list 1,add,"%I = 1"
  list 1,add,"repeat"
  list 1,add,"  %X = @next(1)"
  list 1,add,"  if %X"
  list 1,add,"    %J = @io(WRITB,%X,%W,%I)"
  list 1,add,"    %I = @sum(%I,%J)"
  list 1,add,"  end"
  list 1,add,"until @null(%X)"
  list 1,add,"exit"
  list 1,add,":DSC"
  %D = @dlgtext(EDES)
  if @file(%D)
    %F = ~tmp.tmp
    list 1,savefile,%F
    %B = @io(APPND,%F,%D)
    file delete,%D
```

```
    file rename,%F,%D
  else
    list 1,savefile,%D
  end
  list 1,close
  dialog set,TXT,Saved to %D
  %Z = @event()
  goto evloop
:INTclick
  if @dlgtext(INT)
    dialog set,INT,1
  else
    dialog set,INT,""
  end
  goto evloop
:BCOMbutton
  if @file(%D)
    %E = @shortname(C:\Program Files\Visual DialogScript\DS.EXE)
    %D = @shortname(%D)
    %C = %E" "/C" "%D
    if @dlgtext(INT)
      %C = %C" /INT"
    end
    run %C
  end
  goto evloop
:close
exit
```