# Z/Install documentation addendum

This addendum describes corrections and additions to the Z/Install user's manual. Please review it thoroughly.

## Release 1.23 - August 21st, 1992

Enhancements:

- The maximum number of script labels has been increased to 500. This will only increase memory requirements for the compiler, and not the installation program itself.

Additions to the Installation Executable:

- Z/Install now handles internal errors with grace. For the following error conditions, a window is popped up with an explanation. Unfortunately, these 'fatal internal errors' must cause the installation to abort. Faliure to do so would result in a system crash.

  1) Stack overflow: The GoSub stack exceeded the maximum of 500. This can occur when a label is 'GoSubbed' many times without a return statement. The following example will produce the error:

```
Label Forever
GoSub Forever
```

  2) Integer/String overflow: The maximum number of integers or strings was exceeded. This error is unlikely to occur, since a script file may contain up to 128 of each type of variable.

  These types of errors will return the user to DOS and report the following:

"A fatal error has occured. Please contact the company and report this code: [n]"

'n' will be the internal error code:

1 - Stack overflow

2 - Integer overflow
3 - String overflow

Release 1.22 - August 20th, 1992

Additions to ZPACK.EXE:

- New command line action [u]. This action (update pack) is functionally the same as the freshen command (see user's manual), with the following differences:

  1) When the command is issued, it will freshen all the files in the pack. Upon completion, it will add any files in the current directory that are not already in the pack.

  2) When used with the new option [/d], the update command will also delete any files from the pack that are not in the current directory.

- These new commands are useful for developers who constantly update their distribution files. The commands effectively 'image' the current directory in the pack.

Bug fixes:

- The Installation Executable would not report a CRC error prior to this version. Now, it pops up an error window with the text "Error: Compressed file is corrupt (s)", where 's' is the corrupt file name.

  This error can occur after calling BeginInstall. If there is an error during BeginInstall, the global variable 'InstalledOK' is set to zero.

- The function 'ViewFile' would not print the document when 'P' was pressed. This has been corrected.

Enhancements:

- The function 'ViewFile' now has emulation for Vern Buerg's LIST program, which seems to be popular among the BBS community. This makes the function more user-friendly.

Documentation update:

- The following should be noted about the function 'ViewFile':

  When printing, the function will not format the document in any way before sending it to the printer. It will simply send each line and a cr/lf combination until the EOF has been reached. Therefore, the files loaded should be printer-ready.

Release 1.21 - August 19th, 1992

PLEASE NOTE:

- The function "UseHighIntensityBGColors" has been renamed to "HiBack". The reason for this change is that the name of the function became too large for the parsing table. Likewise, to maintain consistency, the function "UseBlink" has been renamed to "Blink".

Changes to ZINSTALL.EXE:

- ZINSTALL.EXE has two new command line parameters. They are as follows:

  /v Verbose Error Logging. When this option is specified, Z/Install will give a detailed report of each error, including the text of the offending line, and the correct syntax.

  /c Create refenrence chart. This option will create ZINSTALL.REF, which is a list of all the functions, their return values and their parameters.

Other changes:

- There is now a maximum of 128 string and 128 integer variables, up from the previous limit of 50.

- When "BeginInstall" is called, it only creates the paths that are going to be used in the installation. This is only effective when file grouping is used and the user has turned off the installation of one or more groups.

- Decreased the size of the installation program by 4k, with no other losses.

New global integer variables:

EMSAvail (formerly EMSMemory):

Holds the available amount of EMS memory available on the user's system (in 1K blocks).

EMSTotal:

Holds the total amound of EMS memory (in 1K blocks).

EMSVersion:

Holds the LIM EMS version of the memory manager in use on the user's system. For example, if this is set to 310, the driver version is 3.10. If this number is set to zero, there is no LIM EMS driver in use.

New global string variable:

BootDrv:

Holds the drive that the user's system was started from at last bootup, eg: "C".

One new function has been added:

StrLen
---

Parms : StrLen <STR string>

Function: Calculates the length of 'string'.

```
Returns : The length of string, as an integer.

Example : int len = StrLen "Hello"

          print "The length of 'Hello' is ", len, "\n"
```

## Release 1.20 - August 15th, 1992

The file access functions have been replaced with a new and improved set. These new functions allow up to 25 files being open simultaneously, using the 'file handle' technique.

### OpenFile

```
Parms   : OpenFile <INT mode>, <FORMAT filename>

Function: Opens 'filename' for access.  The access mode depends
          on the parameter 'mode', which can be equal to one of
          the following:

            0 - Create a new file (overwrites any existing files)
            1 - Open for READ-ONLY access
            2 - Open for writing (same as mode 0)
            3 - Open for writing and appending.  This mode
                positions the file pointer at the end of the
                file before writing any data.

Returns : Integer:  A file handle from 1 to 25.
          The function returns 0 if the file was not found (mode
          2 only), or if write access was denied (mode 0, 1 and
          3).

          0 may also be returned because the user's system has
          run out of file handles.  Remember, the FILES= setting
          in CONFIG.SYS denotes the maximum number of files that
          can be open at once.

Example : Int handle = OpenFile 1, "\\AUTOEXEC.BAT" ;  Open for
                                                    ;  reading
```

```
If handle = 0
  Goto faliure
Else
  Goto success
EndIf
```

ReadLine
---

Parms : ReadLine <INT handle>

Function: Reads a line of text from a file and returns the line read. This function is only valid for files that have been opened with mode 1 (read access).

ReadLine reads from the file until it encounters the linefeed character. It will not stop at carriage returns. The cr/lf pair is kept in the string upon completion of the function.

Note that this function should only be used on ASCII files with line lengths of less than 255.

Returns : String: the text read from the file. This string is blank (var = "") if the function encounters the EOF before reading any characters.

Example :

```
int Handle = OpenFile 1, "\\AUTOEXEC.BAT"

Str text

Label loop
  text = ReadLine Handle

  If text = ""    ; Got to EOF
      goto exitloop
  EndIf

  print text
Goto loop

Label exitloop
```

WriteLine
---

Parms : WriteLine <INT handle>, <FORMAT str>

Function: Writes the contents of 'str' to the file associated with 'handle'. This function is only valid with files that have been opened with modes 0, 1 or 3. The function does not append any cr/lf codes, nor does it do any translation before writing the string.

Returns : Nothing

Example :
```
int Handle = OpenFile 2, "\\AUTOEXEC.NEW"

WriteLine Handle, "FILES=10"
WriteLine Handle, "SET ZINSTALL=", Z_Env_Var

CloseFile Handle
```

CloseFile
---

Parms : CloseFile <INT handle>

Function: Closes the file associated with 'handle'.

Returns : Nothing

Example :
```
int Handle = OpenFile 1, "\\AUTOEXEC.BAT"
 ...
CloseFile Handle
```

Release 1.10 - August 12th, 1992

This release incorporates 20 new script functions, used for parsing strings and doing string conversions.

StrCmp and StrCmpI
---

```
Parms   : StrCmp <STR str1>, <STR str2>

Function: Compare two strings.

Returns : 1 - Strings are the same.
          2 - Str2 is less than Str1.
          3 - Str2 is greater than Str1.

Example : int ret = StrCmp "Hello, World!", "Helow, World!"

          If ret = 1
            Print "Strings are the same!\n"
          Elif ret = 2
            Print "Str2 is less than Str1!\n"
          Else
            Print "Str2 is greater than Str1!\n"
          Endif

Notes   : StrCmpI is the case-insensitive version of StrCmp.
```

## StrNCmp and StrNCmpI

```
Parms   : StrNCmp <STR Str1>, <STR Str2>, <INT len>

Function: Compares the first 'len' bytes of Str1 and Str2.

Returns : 1 - Strings are the same.
          2 - Str2 is less than Str1.
          3 - Str2 is greater than Str1.

Example : int ret = StrNCmp "Hello, World!", "Helow, World!", 5

          If ret = 1
            Print "Strings are the same!\n"
          Elif ret = 2
            Print "Str2 is less than Str1!\n"
          Else
            Print "Str2 is greater than Str1!\n"
          Endif

Notes   : StrNCmpI is the case-insensitive version of StrNCmp.
```

StrInStr and StrInStrI
---

```
Parms   : StrInStr <STR bigstr>, <STR smallstr>

Function: Checks if 'bigstr' contains 'smallstr'.

Returns : 256 - Check negative.   This value is returned because
                no string is longer than 255 bytes.  Therefore,
                256  will  never  be  returned  if the check is
                positive.

     position - Check positive.  Position will be from 0 to 255,
                0 being the start of the string.

Example : int pos = StrInStr "Hello, World!", "Hello"

          If pos = 256
            Print "Check is negative!\n"
          Else
            Print "Check is positive!\n"
          Endif

Notes   : StrInStrI is the Case-Insensitive version of StrInStr.
```

StrNCpy
---

```
Parms   : StrNCpy <STRVAR dest>, <STR src>, <INT len>

Function: Copies 'len' bytes of 'src' into 'dest'.   If 'len' is
          longer than the length of 'src', 'dest' will be padded
          with spaces.

Returns : Nothing

Example : Str dest
          StrNCpy dest, "Hello, World!", 5
```

StrCCpy
---

```
Parms   : StrCCpy <STRVAR dest>, <STR src>, <STR char_str>
```

```
Function:  Copies 'src' into 'dest' until one of the  characters
          in 'char_str' is reached.

Returns : Nothing

Example : Str dest
          Str src = "Hello, World!"
          StrCCpy dest, src, ","
```

AtoI

```
Parms   : AtoI <STR string>

Function: Converts 'string' into an integer and returns the value.

Returns : The integer value of 'string'.

Example : Str ten_str = "10"
          Int ten_int = AtoI ten_str
```

StrDelete and StrDeleteI

```
Parms   : StrDelete <STRVAR str>, <STR del_str>

Function: Deletes the FIRST occurance of 'del_str' from 'str'.

Returns : Nothing

Example : Str hello = "Hello, World!"
          StrDelete hello, ", World"

          ; 'hello' now contains "Hello!".

Notes   : StrDeleteI is the case-insensitive version of StrDelete.
```

StrChangeStr and StrChangeStrI

```
Parms   : StrChangeStr <STRVAR dest>, <STR old>, <STR new>
```

```
Function: Changes all occurances of the string 'old' to 'new' in
          the variable 'dest'.

Returns : Nothing

Example : Str apples = "I have three apples"

          StrChangeStr apples, "three", "two"

          ; 'apples' now contains "I have two apples".

Notes   :  StrChangeStrI  is  the  case-insensitive  version  of
          StrChangeStr
```

StrCut

---

```
Parms   : StrCut <STR src>, <INT pos>, <INT len>

Function:  Returns  a  string  containing  'len'  bytes of 'src'
          starting at the position 'pos'.   Remember,  the FIRST
          character in 'src' is position ZERO, not ONE.

Returns : A string, detailed in the Function description.

Example : Str hello = "Hello, World!"
          Str dest

          dest = StrCut hello, 7, 5

          ; 'dest' now contains the string "World"
```

StrUpr and StrLwr

---

```
Parms   : StrUpr or StrLwr <STRVAR str>

Function: StrUpr converts 'str' to all-uppercase, and
          StrLwr converts 'str' to all-lowercase.

Returns : Nothing.
```

```
Example : Str UpDown = "ThIs Is A fUnKy StRiNg!"
          StrUpr UpDown

          ; 'UpDown' now contains "THIS IS A FUNKY STRING!"
```

StrInsert
---

```
Parms   : StrInsert <STRVAR dest>, <STR ins_str>, <INT pos>

Returns : Nothing.

Function:  Inserts  'ins_str'  into  'dest'  at  position 'pos'.
          Remember,  the FIRST byte in 'dest' is position  ZERO,
          not ONE.

Example : Str dest = ", World!"

          StrInsert dest, "Hello", 0

          ; 'dest' now contains "Hello, World!"
```

ItoA
---

```
Parms   : ItoA <INT n>

Function: Returns the string version of the integer 'n'.

Returns : See function description.

Example : int ten_int = 10
          Str ten_str = ItoA ten_int

          ; 'ten_str' now contains "10".
```

Release 1.03 - August 5th, 1992

New script functions for getting/putting [PATHS] contents:

```
GetPathString                                                        Str
---------------------------------------------------------------------

Parms   : GetPathString <INT path_number>

Returns : String: The path corresponding to path_number,  as set
                  in [PATHS].

Function:  GetPathString returns a string containing the path as
          set in [PATHS].  It returns the full string, including
          the root path.

Example : [PATHS]
          0:\\ZINSTALL
          1:\\DOCS
          [END]

          Str PathStr = GetPathString 1

          print PathStr, "\n"

Output:   \ZINSTALL\DOCS (linefeed)


PutPathString                                                       None
---------------------------------------------------------------------

Parms   : PutPathString <INT path_number>, <STR path>

Returns : None

Function:  PutPathString puts the string 'path' into the [PATHS]
          section,  under the path number  'path_number'.    The
          string  is  automatically  formatted  correctly before
          being put in.

Example : [PATHS]
          0:\\ZINSTALL
          1:\\DOCS
          [END]

          Str PathStr = GetPathString 1

          ... Edit PathStr here ...
```

PutPathString 1, PathStr

Revision 1.01 - July 21st, 1992

ZPack command line - new option:

- The ZPack program (ZPACK.EXE) has a new command-line option. It is used as follows:

  /e<filename>

  This option will exclude all files matching 'filename' (wildcards are allowed) from the selected action. For example:

  ZPACK a /e*.bak docs *.*

  This would add every file in the current directory to DOCS.ZPK except all files matching '*.BAK'. The /e option must be the last option on the command line and the first before the archive name.

Change to script function 'Dialog':

- Prior to this release, the script function 'Dialog' had no means by which to insert preceeding spaces in the dialog text. This release supports this using optional surrounding quotation marks in the text, as follows:

  Dialog 0, 0

      " This has a preceeding space character"
      This line does not.

  EndDialog

  Please see the user's guide for more information on the Dialog function.

Correction to the user's guide

- On page 9 of the user's guide, the [PATHS] section of the sample script file is written incorrectly. Each entry in the section should have two backslash characters where, in the manual, there is only one. Please take note of this correction as it will cause the script to malfunction if only one backslash character is used (Please see section 2.6 in the user's guide for information on backslash codes).

## Greater script command versatility

- When a script function takes an integer as a parameter, it is now acceptable to use one of three different integer formats. For example, the function SetBackChar sets the background character and takes an integer as its parameter. Before this release, the user would have to type in the ASCII code of the character (eg 32 for the space character). In this release, all of the methods below are acceptable:

```
SetBackChar    32        ; 32 = space character
SetBackChar    0x20      ; 20 hex = space character
SetBackChar    ' '       ; Actual space character
SetBackChar    '\x20'    ; Space hex code in single quotes
```

  The single quote character denotes an integer variable, and the quotes can only contain one character (Backslash codes are accepted).

## New [PATHS] format

- If your installation requires multiple directories to be made from the root (x:\), simply place a star symbol (*) as the first character of each root directory. For example:

```
[PATHS]
0:\\MYAPP\\          ; Path 0 will is always off the root
1:\\DOCS\\           ; \MYAPP\DOCS
2:\\SAMPLES\\        ; \MYAPP\SAMPLES
3:*\\DRIVERS\\       ; \DRIVERS
4:\\VGA\\            ; \DRIVERS\VGA
[END]
```

  This release also supports PATH NAMING, for use with the

GetInstallPaths function. It allows the developer to define an optional descriptive name for each path. For example:

```
[PATHS]
0:\\ZINSTALL\\:Z/Install root directory
1:\\DOCS\\:Z/Install documentation
2:\\SAMPLES\\:Z/Install samples
[END]
```

The new script function 'GetInstallPaths' will allow the user to edit each path from a bar menu. The paths that the user edits are actual physical paths, not subdirectories from anything. For example, if the user chose to edit path 1 from the example above, the initial setting would be:

```
\ZINSTALL\DOCS
```

The user could change the path to:

```
\ZI-DOCS
```

if they so desired. If you don't want the user to edit a certain path, simply skip the descriptive name and it will not be shown in the bar menu. For example:

```
[PATHS]
0:\\ZINSTALL\\:Z/Install root directory      ; editable
1:\\DOCS\\                                   ; NOT editable
2:\\SAMPLES\\:Z/Install sample scripts       ; editable
etc...
[END]
```

The function 'GetInstallPaths' is called with no parameters.

## New script functions for editing ASCII files

- There are three new script functions, used for editing ASCII files (normally the user's boot files). They are:

InterLoad Int

---

Parms : InterLoad <FORMAT Filename>

```
Returns : Integer: 0 - Couldn't load file (Not enough memory
                       or file not found)
                   1 - Loaded file OK

Function:  Loads  'filename'  into  the interactive editor's
           buffer.   This function must be  called  at  least
           once  before  using  any  of  the  other  Inter...
           functions.    Each call to 'InterEdit' must have a
           preceeding call to 'InterLoad'.

Example : InterLoad "AUTOEXEC.BAT"
          InterPut  "SET ZINSTALL=C:\\ZINSTALL"
          InterEdit


InterPut                                                  None
-------------------------------------------------------------

Parms   : InterPut <FORMAT string>

Returns : None

Function:  Places 'string' in  the Interactive Editor's file
           buffer.   When 'InterEdit' is called,  the  string
           will  be the first line in the scroll buffer.   Up
           to ten lines  may  be  inserted  into the editor's
           buffer.   'InterLoad' must be called  before  this
           function.

Example : InterLoad "AUTOEXEC.BAT"
          InterPut  "SET ZINSTALL=C:\\ZINSTALL"
          InterEdit


InterEdit                                                 None
-------------------------------------------------------------

Parms   : None

Returns : None

Function:  Starts up the Interactive Editor.   A window will
           appear  on  the  screen  with  a scroll bar.   The
           standard movement keys can be used to move the bar
```

around. When the user presses <ENTER>, the line under the scroll bar will be 'Picked Up', and the user can then move it around. When the user hits <ENTER> again, the line will be dropped in the current position.

The <INS> and <DEL> keys can be used to insert and delete lines in the file, respectively. If the user presses any alphabetical or numeric character, he/she will be able to edit the line under the scroll bar.

When the user presses <F10>, the editor will remove itself from the screen and the file will be saved. If the user presses <ESC>, the installation program will confirm that the changes be abandoned.

All of the above information is available to the user by pressing the <F1> key when editing the file.

After this function exits, all memory allocated with 'InterLoad' and 'InterPut' is freed. Therefore, 'InterLoad' must be called again before 'InterEdit' (even when loading the same file).

```
Example : InterLoad "AUTOEXEC.BAT"
          InterPut  "SET ZINSTALL=C:\\ZINSTALL"
          InterEdit
```