U7. String List Utility Commands

An important subset of ViewIt utility commands are those used to manipulate "string lists" (STR#-type resources or any relocatable block with a similar structure). String lists contain Pascal-type strings packed together in a list:

[2-byte count][length byte][text][length byte][text]...
Although this is a very efficient way to store strings, the only toolbox call, GetIndString, available for use with such string lists simply returns a particular string from an STR#-type resource.

The four ViewIt commands GetStr, SetStr, SrtLst, and DupLst allow you to easily manipulate string lists. For example, to set STR# 1040 equal to the first 10 elements of STR# 1030, you could write,

```
FaceIt(nil,SetStr,1040,-1,0,0); clear list
for i:= 1 to 10 do

begin
get ith string from STR# 1030

  FaceIt(nil,GetStr,1030,i,0,0);
set ith string in STR# 1040

  FaceIt(nil,SetStr,1040,i,0,0);

end;
```

where each string being copied goes through the uString variable. String lists can also be dynamically created and disposed of without the need for STR# resources. For example, to copy the contents of STR# 1030 into a new string list handle, you could write:

```
FaceIt(nil,DupLst,1030,0,0,0);

myList := uResult;
```

where a handle to the new list is returned in uResult and saved in "myList". Such a list handle can then be used with other string list commands. For example,

```
FaceIt(nil,SetStr,myList,0,0,-1);
```

would cause ViewIt to dispose of the dynamically allocated "myList" string list.

In some cases you may need to determine the number of strings which are currently in a list. The first two bytes of the copy of the string list in memory contains this value. For example, using Language Systems FORTRAN, where "n" is the number of strings in the list, "resID" is an STR# ID, and "strHdl" is the handle to a string list,

```
strHdl = GetResource(%val('STR#'),%val(resID))
n = word(long(strHdl))
```

or, using Pascal, where the type "word" is declared as a pointer to an integer,

```
word = ^integer;
...
strHdl := GetResource('STR#',resID);
n := word(strHdl^)^;
```

or, using C,

```
strHdl = GetResource('STR#',resID);
n = *(short*)(*strHdl);
```

The following commands use parameter a to designate a string list to manipulate. Parameter a can refer to either an existing STR#-type resource or to any relocatable block in memory having the structure of a string list. CAUTION: A string list that is not based on an STR# resource must still have the structure of a string list. An "empty" string list, for

example, is not a 0-byte relocatable block created with the toolbox call NewHandle, but rather a 2-byte block containing the value zero.

Name Number Parameters & Variables used

GetStr 491 a,b,c,d,uString,uName

Gets a string or substring from a string list, uString, or uName, returning it in uString.
a = STR# resource ID of an existing string list resource
or a handle to an existing string list block in memory
or 0 = use uString or uName as source string
b = number of string in list to get
or 0 or 1 = use uString (if a = 0)
or 2 = use uName (if a = 0)
or other = address of a Pascal string (if a = 0)
c = number of substring within string to get
or 0 = return entire string (- d leading characters)
d = ASCII character number used to delimit substrings
(“,” = 44, “:” = 58, “;” = 59, etc.)
or number of leading characters to skip (if c = 0)

SetStr 492 a,b,c,d,uString,uResult

Adds, deletes, or inserts strings in string lists. Where necessary, empty strings are added to the string list. Note that uString is preserved across calls to SetStr.
a = STR# resource ID of an existing string list resource
or a handle to an existing string list block in memory
or, if a = 0, a new string list is created and its handle is

returned in uResult (save this handle for later use)
b = scope of changes to make
-1 = clear all strings in the list
0 = don't change any of the strings
n = nth string in list to manipulate
c = type of changes to make
-1 = delete nth string from list
0 = replace nth string in list with uString
1 = insert uString at nth string position
d = memory and disk options (disk operations are skipped if not working with an STR# resource)
-2 = delete copy of string list from memory w/o updating
-1 = update copy of string list in memory and on disk,
then delete the string list from memory
0 = update copy in memory only
1 = update both copy in memory and on disk
You should, in general, minimize the number of calls made to update a copy of an STR# list on disk since this step involves saving the entire STR# resource back to disk. In other words, if you have a loop which results in many SetStr calls, then don't ask ViewIt to update the disk copy (d = 1) until changes to the list in memory are complete. There will often simply be no need to update the disk copy.

SrtLst 493 a,b

Sorts (alphabetizes) the list whose resource ID or handle is given by a. Parameter b can be used to designate the number of leading characters to ignore when comparing strings in the list.

DupLst 494 a,b,c,d,uResult

Copies the list designated by parameter a to that designated by b, where a and b are either STR# IDs or handles to string lists. If b is zero, then a new destination string list block is dynamically allocated and its handle returned in uResult. Parameter c can be used to add or remove leading characters as each string is copied. If c is positive, then c characters whose ASCII value is given by d are added to each string. If c is negative, then c characters are removed from the beginning of each copied string. If a = b, then the source string list is simply modified according to c and d.