## U6. String Utility Commands

*ViewIt supports several commands that perform common string-related operations: setting substrings, trimming strings, setting parameter text, converting strings from one format to another, finding strings within text blocks, and interconverting strings and numbers.*

Name Number Parameters & Variables used

SetSub 451 a,b,c,d,uString,uName

*Moves the contents of uString into uName beginning at character a of uName and ending at character b (i.e. sets a substring in uName). Undefined characters are replaced with spaces so that SetSub works even if uName is less than b characters in length, or uString is smaller than the substring. You can optionally use parameter c to pass the address of a Pascal-type source string (instead of using uString), and/or d to pass the address of a Pascal-type destination string (instead of using uName).*

*NOTE: You can get substrings using the command GetStr described in the "String Lists" topic.*

TrmStr 452 a,b,uString,uName

*Moves or removes leading or trailing spaces within the string uString, uName, or some other string designated by a, according to the type of adjustment specified by b.*

*a = string to adjust*
*0 or 1 = uString*
*2 = uName*
*other = address of a Pascal string*
*b = type of adjustment to make*
*1 = removes (trims) trailing spaces*
*0 = removes both leading and trailing spaces*
*-1 = removes leading spaces*
*-2 = moves leading spaces to end of string*

SetPrm 453 a,b,c,d,uString

*Resets the four parameter text strings (^0, ^1, ^2, and ^3 items in dialogs and alerts) to the strings designated by a, b, c, and d where,*

*0 = empty string*
*1 to 255 = size of substring from uString*
*other = address of a Pascal string*

*If using substrings from uString, these are obtained in succession, and trimmed of both leading and trailing spaces before being assigned to parameter text. For example, the call "FaceIt(nil,SetPrm,10,20,10,30)" would set the ^0 item to the first 10 characters of uString, ^1 to the next 20 characters, ^2 to the next 10 characters, ^3 to the next 30 characters (= 70 total characters of uString used). This approach can be mixed with that of passing Pascal string addresses. SetPrm is primarily provided for those programmers who do not have access to the toolbox routine ParamText, but is also useful when "writing" multiple parameter text items to a single string.*

*NOTE: This command is largely obsolete now since the basic control driver used with ViewIt provides a more powerful "parameter text" scheme based on the use of string lists.*

CnvStr 454 a,b,c,d,uString,uName

*Converts the string designated by parameter c from string type a to type b. Parameter d designates the total number of bytes occupied by the string variable (its "storage size") which is usually larger than the number of characters in the string (the "string length").*

*a = source type (0 = Pascal, 1 = C, 2 = FORTRAN)*
*b = converted type (0 = Pascal, 1 = C, 2 = FORTRAN)*
*c = source string (0 or 1 = uString, 2 = uName, other = string address)*
*d = source string storage size (0 ≤ d ≤ 256 bytes)*
*(if d = 0, ViewIt assumes d = 256 bytes)*

*CnvStr preserves all fRec scratch variables, so you don't need to worry about it clobbering other values in fRec.*

FndTxt 455 a,b,c,d,uResult

Searches within the text block defined by a and b for the string defined by c and d. Note that the text block must be locked in memory during the search since FndTxt can move heap memory. uResult returns the position of the found string as a byte offset from a, or -1 if not found.

a = address of text block to search
b = size of text block to search (bytes)
c = address of search text
d = size of search text (use -d for case sensitive search)

NumToS 471 a,b,c,d,uString

*Converts a number from the variable designated by b to the string designated by a, using the format indicated by c and d. Infinities will appear as the character "∞".*

*a = destination string*
*0 = uString*
*other = address of a Pascal string*
*b = source of number (as a data type)*
*1 = ul1 5 = uR4*
*2 = ul2 6 = uR8*
*3 = ul4 7 = uR10*
*4 = ul8 8 or 12 = uR12 (12 = THINK C "universal")*
*c = format*
*0 = general (= fixed point unless very large or small)*
*1 = floating point*
*2 = fixed point*
*d = digits (use -d to also trim trailing decimal zeroes)*
*if c = 0 or 1, d = significant figures (-4 used if d = 0)*
*if c = 2, d = decimals to display*

*C & Pascal Programmers: The "FaceStorXC" files declare uR8, uR10, and uR12 as arrays of 2-byte integers:*

```
short uR8[4];
short uR10[5];
short uR12[6];
```

*The "FaceStorXP" files define these variables similarly:*

```
uR8 : array [1..4] of integer;
uR10 : array [1..5] of integer;
uR12 : array [1..6] of integer;
```

*The reason that these fRec elements are defined as integer arrays is that we needed to ensure that the size of these variables was always 8, 10, or 12 bytes, respectively, but could not use "double" or "extended" since the meaning of these depends on both the compiler in use and on compiler options (see the "Numbers" topic in the "About Compilers" program for more information about this issue).*

*To work with uR8, uR10, and uR12, you will either need to "fix" the "FaceStorXY" file to declare these variables using numerical types corresponding to 8, 10, and 12-byte reals, or type cast these variables to the proper types in expressions involving real numbers. The following lines, for example, will not compile due to a type mismatch,*

```
uR10 := myReal;
myReal := uR10;
```

*but can be quickly fixed by applying type casts,*

```
extended(uR10) := myReal;
myReal := extended(uR10);
```

*which assumes, of course, that the compiler is interpreting "extended" as a reference to a 10-byte real, and that the program variable "myReal" can be assigned or assigned to a variable of type extended.*

SToNum 481 a,b,uString,uResult,fI1Err...

*Converts a string designated by a to an integer or real number in the variable designated by b. The "∞" character can be used in strings to represent infinities. Leading and trailing spaces are ignored.*

*a = source string*
*0 = uString*
*other = address of a Pascal string*
*b = destination variable (as a data type)*
*1 = ul1  5 = uR4*
*2 = ul2  6 = uR8*
*3 = ul4  7 = uR10*
*4 = ul8  8 or 12 = uR12  (12 = THINK C "universal")*

*If no error occurs, then uResult is set equal to zero, else uResult returns -1 and the variable (ul1...uR12) is set equal to the corresponding error value in fRec (fl1Err... fR12Err). The default error values are zero, but these can be changed at any time to other values if you prefer a special non-zero value returned when an error occurs.*

*Conditions causing an error include an empty string or a string that cannot be evaluated as a number (an NAN). A number that is out of the range of values represented by the destination variable's numerical type is either set equal to the maximum or minimum integer (for integer types), or to ±∞ or 0 (for real types).*

*C and Pascal Programmers: See note above accompanying NumToS that describes type-casting that may be needed if working with uR8, uR10, or uR12.*