Details

The SplitWave operation is in some ways the inverse of the Concatenate operation. *srcWave* is decomposed into waves of lower dimensionality.

Splitting a 2D 10x15 wave results in 15 waves of 10 rows each.

Splitting a 3D 10x15x4 using /SDIM=2 results in 4 2D waves of dimension 10x15.

Splitting a 3D 10x15x4 using /SDIM=1 results in 60 1D waves of 10 rows each.

The SplitWave operation works on all wave types. *srcWave* must be 2D or greater.

The operation creates the string variable S_waveNames which contains a semicolon separated list of the names of the output waves. However if you use /FREE then S_waveNames will be empty as free waves can not be accessed by name; use /OREF to access the created waves.

Examples

```
// Create sample input
Make/N=(5,4,3,2) wave1 = p + 10*q + 100*r + 1000*s

// Split chunks into 2 3D waves and store them in data folder Chunks
SplitWave/DDF=Chunks/N=chunk wave1

// Split layers into 6 2D waves and store them in data folder Layers
SplitWave/DDF=Layers/N=Layers/SDIM=2 wave1

// Split into 24 1D waves and store them in data folder Columns
SplitWave/DDF=Columns/N=Columns/SDIM=1 wave1
```

See Also

Duplicate, **Redimension**, **Concatenate**

sprintf

sprintf ***stringName*, *formatStr*** [**, *parameter***]…

The sprintf operation is the same as printf except it prints the formatted output to the string variable *stringName* rather than to the history area.

Parameters

formatStr	See printf.
parameter	See printf.
stringName	Results are "printed" into the named string variable.

See Also

The **printf** operation for complete format and parameter descriptions and **Creating Formatted Text** on page IV-259.

sqrt

sqrt(*num*)

The sqrt function returns the square root of *num* or NaN if *num* is negative.

In complex expressions, *num* is complex, and sqrt(*num*) returns the complex value x + *i*y.

sscanf

sscanf *scanStr*, *formatStr*, *var* [**, *var***]

The sscanf operation is useful for parsing text that contains numeric or string data. It is based on the C sscanf function and provides a subset of the features available in C.

Here is a trivial example:

```
Variable v1
sscanf "Value= 1.234", "Value= %f", v1
```

This skips the text "Value=" and the following space and then converts the text "1.234" (or whatever number appeared there) into a number and stores it in the local variable v1.

The sscanf operation sets the variable V_flag to the number of values read. You can use this as an initial check to see if the *scanStr* is consistent with your expectations.

Note: The sscanf operation is supported in user functions only. It is not available using the command line, using a macro, or using the Execute operation.

Parameters

scanStr contains the text to be parsed.

formatStr is a format string which describes how the parsing is to be done.

formatStr is followed by the names of one or more local numeric or string variables or NVARs (references to global numeric variables) or SVARs (references to global string variables), which are represented by *var* above.

sscanf can handle a maximum of 100 *var* parameters.

Details

The format string consists of the following:

- Normal text, which is anything other than a percent sign ("%") or white space.
- White space (spaces, tabs, linefeeds, carriage returns).
- A percent ("%") character, which is the start of a conversion specification.

The trivial example illustrates all three of these components.

```
Variable v1
sscanf "Value= 1.234", "Value= %f", v1
```

sscanf attempts to match normal text in the format string to the identical normal text in the scan string. In the example, the text "Value=" in the format string skips the identical text in the scan string.

sscanf matches a single white space character in the format string to 0 or more white space characters in the scan string. In the example, the single space skips the single space in the scan string.

When sscanf encounters a percent character in the format string, it attempts to convert the corresponding text in the scan string into a number or string, depending on the conversion character following the percent, and stores the resulting number or string in the corresponding variable in the parameter list. In the example, "%f" converts the text "1.234" into a number which it stores in the local variable v1.

A conversion specification consists of:

- A percent character ("%").
- An optional "*", which is a conversion suppression character.
- An optional number, which is a maximum field width.
- A conversion character, which specifies how to interpret text in the scan string.

Don't worry about the suppression character and the maximum width specification for now. They will be explained later.

The sscanf operation supports a subset of the conversion characters supported by the C sscanf operation. The supported conversion characters, which are case-sensitive, are:

d	Converts text representing a decimal number into an integer numeric value.
i	Converts text representing a decimal, octal or hexadecimal number into an integer value. If the text starts with "0x" (zero-x), it is interpreted as hexadecimal. Otherwise, if it starts with "0" (zero), it is interpreted as octal. Otherwise it is interpreted as decimal.
o	Converts text representing an octal number into an integer numeric value.
u	Converts text representing an unsigned decimal number into an integer numeric value.
x	Converts text representing a hexadecimal number into an integer numeric value.
c	Converts a single character into an integer value which is the ASCII code representing that character.
e	Converts text representing a decimal number into a floating point numeric value.
f	Same as e.

g	Same as e.
s	Stores text up to the next white space into a string.
[	Stores text that matches a list of specific characters into a string. The list consists of the characters inside the brackets ("%[abc]"). If the first character is "^", this means to match any character that is **not** in the list. You can specify a range of characters to match. For example `"%[A-Z]"` matches all of the upper case letters and `"%[A-Za-z]"` matches all of the upper and lower case letters.

Here are some simplified examples to illustrate each of these conversions.

```
Variable v1
String s1
```

Convert text representing a decimal number to an integer value:

```
sscanf "1234", "%d", v1
```

Convert text representing a decimal, octal, or hexadecimal number:

```
sscanf "1.234", "%i", v1          // Convert from decimal.
sscanf "01234", "%i", v1          // Convert from octal.
sscanf "0x123", "%i", v1          // Convert from hex.
```

Convert text representing an octal number:

```
sscanf "1234", "%o", v1
```

Convert text representing an unsigned decimal number:

```
sscanf "1234", "%u", v1
```

Convert text representing a hexadecimal number:

```
sscanf "1FB9", "%x", v1
```

Convert a single ASCII character:

```
sscanf "A", "%c", v1
```

Convert text representing a decimal number to an floating point value:

```
sscanf "1.234", "%e", v1
sscanf "1.234", "%f", v1
sscanf "1.234", "%g", v1
```

Copy a string of text up to the first white space:

```
sscanf "Hello There", "%s", s1
```

Copy a string of text matching the specified characters:

```
sscanf "+4.27", "%[+-]", s1
```

In a C program, you will sometimes see the letters "l" (ell) or "h" between the percent and the conversion character. For example, you may see "%lf" or "%hd". These extra letters are not needed or tolerated by Igor's sscanf operation.

When sscanf matches the format string to the scan string, it reads from the scan string until a character that would be inappropriate for the section of the format string that sscanf is trying to match. In the following example, sscanf stops reading characters to be converted into a number when it hits the first character that is not appropriate for a number.

```
Variable v1
String s1, s2
sscanf "1234Volts DC", "%d%s %s", v1, s1, s2
```

sscanf stops matching text for "%d" when it hits "V" and stores the converted number in v1. It stops matching text for the first "%s" when it hits white space and stores the matched text in s1. It then skips the space in the scan string because of the corresponding space in the format string. Finally, it matches the remaining text to the second "%s" and stores the text in s2.

The maximum field width must appear just before the conversion character ("d" in this case).

```
Variable v1, v2
sscanf "12349876", "%4d%4d", v1, v2
```

The suppression character ("*") is used in a conversion specification to skip values in the scan string. It parses the value, but sscanf does not store the value in any variable. In the following example, we read one