Numeric and String Variables

Overview

This chapter discusses the properties and uses of global numeric and string variables. For the fine points of programming with global variables, see **Accessing Global Variables and Waves** on page IV-59.

Numeric variables are double-precision floating point and can be real or complex. String variables can hold an arbitrary number of bytes. Igor stores all global variables when you save an experiment and restores them when you reopen the experiment.

Numeric variables or numeric expressions containing numeric variables can be used in any place where literal numbers are appropriate including as operands in assignment statements and as parameters to operations, functions, and macros.

When using numeric variables in operation flag parameters, you need parentheses. See **Reference Syntax Guide** on page V-15.

String variables or string expressions can be used in any place where strings are appropriate. String variables can also be used as parameters where Igor expects to find the name of an object such as a wave, variable, graph, table or page layout. For details on this see **Converting a String into a Reference Using $** on page IV-57.

In Igor7 or later, Igor assumes that the contents of string variables are encoded as UTF-8. If you store non-ASCII text in string variables created by Igor6 or before, you need to convert it for use in Igor7 or later. See **String Variables and Text Encodings** on page II-98 for details.

Creating Global Variables

There are 20 built-in numeric variables (K0 … K19), called system variables, that exist all the time. Igor uses these mainly to return results from the CurveFit operation. We recommend that you refrain from using system variables for other purposes.

All other variables are user variables. User variables can be created in one of two ways:

- Automatically in the course of certain operations
- Explicitly by the user, via the Variable/G and String/G operations

When you create a variable directly from the command line using the Variable or String operation, it is always global and you can omit the /G flag. You need /G in Igor procedures to make variables global. The /G flag has a secondary effect — it permits you to overwrite existing global variables.

Uses For Global Variables

Global variables have two properties that make them useful: globalness and persistence. Since they are global, they can be accessed from any procedure. Since they are persistent, you can use them to store settings over time.

Using globals for their globalness creates non-explicit dependencies between procedures. This makes it difficult to understand and debug them. Using a global variable to pass information from one procedure to another when you could use a parameter is bad programming and should be avoided except under rare circumstances. Consequently, you should use global variables when you need persistence.

A legitimate use of a global variable for its globalness is when you have a value that rarely changes and needs to be accessed by many procedures.

Variable Names

Variable names consist of 1 to 31 bytes. The first character must be alphabetic. The remaining characters can be alphabetic, numeric or the underscore character. Names in Igor are case insensitive.

Variable names must be standard names, not liberal names. See **Object Names** on page III-443 for details.

Variable names must not conflict with the names of other Igor objects, functions or operations.

You can rename a variable using the Rename operation, or the Rename Objects dialog via the Misc menu.

System Variables

System variables are built in to Igor. They are mainly provided for compatibility with older versions of Igor and are not recommended for general use. You can see a list of system variables and their values by choosing the Object Status item in the Misc menu.

There are 20 system variables named K0,K1...K19 and one named veclen. The K variables are used by the curve fitting operations.

The `veclen` variable is present for compatibility reasons. In previous versions of Igor, it contained the default number of points for waves created by the Make operation. This is no longer the case. Make will always create waves with 128 points unless you explicitly specify otherwise using the /N=(<number of points>) flag.

Although the CurveFit operation stores results in the K variables, it does so only for compatibility reasons and it also creates user variables and waves to store the same results.

However, the CurveFit operation does use system variables for the purpose of setting up initial parameter guesses if you specify manual guess mode. You can also use a wave for this purpose if you use the kwCWave keyword. See the **CurveFit** operation on page V-105.

It is best to not rely on system variables unless necessary. Since Igor writes to them at various times, they may change when you don't expect it.

The Data Browser does not display system variables.

System variables are stored on disk as single precision values so that they can be read by older versions of Igor. Thus, you should store values that you want to keep indefinitely in your own global variables.

User Variables

You can create your own global variables by using the `Variable/G` (see **Numeric Variables** on page II-96) and `String/G` operations (see **String Variables** on page II-97). Variables that you create are called "user variables" whether they be numeric or string. You can browse the global user variables by choosing the Object Status item in the Misc menu. You can also use the Data Browser window (Data menu) to view your variables.

Global user variables are mainly used to contain persistent settings used by your procedures.

Special User Variables

In the course of some operations, Igor automatically creates special user variables. For example, the Curve-Fit operation creates the user variable `V_chisq` and others to store various results generated by the curve fit. The names of these variables always start with the characters "V_" for numeric variables or "S_" for string variables. The meaning of these variables is documented along with the operations that generate them in Chapter V-1, **Igor Reference**.

In addition, Igor sometimes checks for `V_` variables that you can create to modify the default operation of certain routines. For example, if you create a variable with the name V_FitOptions, Igor will use that to control the CurveFit, FuncFit and FuncFitMD operations. The use of these variables is documented along with the operations that they affect.

When an Igor operation creates V_ and S_ variables, they are global if the operation was executed from the command line and local if the operation was executed in a procedure. See **Accessing Variables Used by Igor Operations** on page IV-115 for details.

Numeric Variables

You create numeric user variables using the `Variable` operation from the command line or in a procedure. The syntax for the Variable operation is:

```
Variable [flags] varName [=numExpr] [,varName [=numExpr]]...
```

There are three optional flags:

/C	Specifies complex variable.
/D	Obsolete. Used in previous versions to specify double-precision. Now all variables are double-precision.
/G	Specifies variable is to be global and overwrites any existing variable.

The variable is initialized when it is created if you supply the initial value with a numeric expression using `=numExpr`. If you create a numeric variable and specify no initializer, it is initialized to zero.

You can create more than one variable at a time by separating the names and optional initializers for multiple variables with a comma.

When used in a procedure, the new variable is local to that procedure unless the /G flag is used. When used on the command line, the new variable is always global.

Here is an example of a variable creation with initialization:

```
Variable v1=1.1, v2=2.2, v3=3.3*sin(v2)/exp(v1)
```

Since the /C flag was not specified, the data type of v1, v2 and v3 is double-precision real.

Since the /G flag was not specified, these variables would be global if you invoked the `Variable` operation directly from the command line or local if you invoked it in a procedure.

`Variable/G` varname can be invoked whether or not a variable of the specified name already exists. If it does exist as a variable, its contents are not altered by the operation unless the operation includes an initial value for the variable.

To assign a value to a complex variable, use the `cmplx()` function:

```
Variable/C cv1 = cmplx(1,2)
```

You can kill (delete) a global user variable using the Data Browser or the `KillVariables` operation. The syntax is:

```
KillVariables [flags] [variableName [,variableName]...]
```

There are two optional *flags*:

/A	Kills all global variables in the current data folder. If you use /A, omit variableName.
/Z	Doesn't generate an error if a global variable to be killed does not exist.

For example, to kill global variable `cv1` without worrying about whether it was previously defined, use the command:

```
KillVariables/Z cv1
```

Killing a global variable reduces clutter and saves a bit of memory. You can not kill a system variable or local variable.

To kill all global variables in the current data folder, use KillVariables/A/Z.

String Variables

You create user string variables by calling the `String` operation from the command line or in a procedure. The syntax is:

```
String [/G] strName [=strExpr] [,strName [=strExpr]... ]
```

The optional /G flag specifies that the string is to be global, and it overwrites any existing string variable.

The string variable is initialized when it is created if you supply the initial value with a string expression using =*strExpr*. If you create a string variable and specify no initializer it is initialized to the empty string ("").

When you call String from the command line or from a macro, the string variable is initialized to the specified initial value or to the empty string ("") if you provide no initial value.

When you declare a local string variable in a user-defined function, it is null (has no value) until you assign a value to it, via either the initial value or a subsequent assignment statement. Igor generates an error if you use a null local string variable in a user-defined function.

When you call String in a procedure, the new string is local to that procedure unless you include the /G flag. When you call String from the command line, the new string is always global.

You can create more than one string variable at a time by separating the names and optional initializers for multiple string variables with a comma.

Here is an example of variable creation with initialization:

```
String str1 = "This is string 1", str2 = "This is string 2"
```

Since /G was not used, these strings would be global if you invoked String directly from the command line or local if you invoked it in a procedure.

`String/G strName` can be invoked whether or not a variable of the given name already exists. If it does exist as a string, its contents are not altered by the operation unless the operation includes an initial value for the string.

You can kill (delete) a global string using the Data Browser or the `KillStrings` operation. The syntax is:

```
KillStrings [flags] [stringName [,stringName ]...]
```

There are two optional *flags*:

A	Kills all global strings in the current data folder. If you use /A, omit stringName.
/Z	Doesn't generate an error if a global string to be killed does not exist.

For example, to kill global string `myGlobalString` without worrying about whether it was previously defined, use the command:

```
KillStrings/Z myGlobalString
```

Killing a string reduces clutter and saves a bit of memory. You can not kill a local string.

To kill all global strings in the current data folder, use KillStrings/A/Z.

Local and Parameter Variables in Procedures

You can create variables in procedures as parameters or local variables. These variables exist only while the procedure is running. They can not be accessed from outside the procedure and do not retain their values from one invocation of the procedure to the next. See **Local Versus Global Variables** on page IV-56 for more information.

String Variables and Text Encodings

Igor Pro 7 uses Unicode internally. Older versions of Igor used non-Unicode text encodings such as Mac-Roman, Windows-1252 and Shift JIS.

If you have string variables containing non-ASCII text in old experiments, they will be misinterpreted by Igor7. For a discussion of this issues, see **String Variable Text Encodings** on page III-428.