Keyword	Information
NAME	The name of the reference function or "" if the FUNCREF variable has not been assigned to point to a function.
ISPROTO	0 if the FUNCREF variable has been assigned to point to a function. 1 if it has not been assigned and therefore still points to the prototype function.
ISXFUNC	0 if it points to a user-defined function. 1 if the FUNCREF points to an external function.

Function

`Function` `[[`**`/C /D /S /DF /WAVE`**`]` ***`functionName`*****`(`**`[`***`parameters`***`]`**`)`**

The Function keyword introduces a user-defined function in a procedure window.

The optional flags specify the return value type, if any, for the function.

Flags

/C	Returns a complex number.
/D	Returns a double-precision number. Obsolete, accepted for backward compatibility.
/S	Returns a string.
/DF	Returns a data folder reference. See **Data Folder Reference Function Results** on page IV-81.
/WAVE	Returns a wave reference. See **Wave Reference Function Results** on page IV-76.

Details

If you omit all flags, the result is a scalar double-precision number.

The /D flag is not needed because all numeric return values are double-precision.

See Also

Chapter IV-3, **User-Defined Functions** and **Function Syntax** on page IV-31 for further information.

FunctionInfo

`FunctionInfo(`***`functionNameStr`*** `[,` ***`procedureWinTitleStr`***`]`**`)`**

The FunctionInfo function returns a keyword-value pair list of information about the user-defined or external function name in *functionNameStr*.

Parameters

functionNameStr a string expression containing the name or multipart name of a user-defined or external function. *functionNameStr* is usually just the name of a function, or "" to return information about the function that called FunctionInfo.

To return information about a static function, supply both the module name and the function name in MyModule#MyFunction format (see **Regular Modules** on page IV-236), or specify the function name and *procedureWinTitleStr* (see below).

To return information about a function in a different independent module, supply the independent module name in addition to any module name and function name (a double or triple name):

Name	What It Refers To
MyIndependentModule#MyFunction	Refers to a non-static function in an independent module.
MyIndependentModule#MyModule#MyFunction	Refers to a static function in a procedure file with `#pragma moduleName=MyModule` in an independent module.

(See **Independent Modules** on page IV-238 for details on independent modules.)

The optional *procedureWinTitleStr* can be the title of a procedure window (such as "Procedure" or "File Name Utilities.ipf") in which to search for the named user-defined function. The information about the named function in the specified procedure window is returned.

The *procedureWinTitleStr* parameter makes it possible to select one of several static functions with identical names among different procedure windows, even if they do not contain a `#pragma moduleName=myModule` statement.

The *procedureWinTitleStr* parameter can also be a title followed by an independent module name in brackets to return information about the named function in the procedure window of the given title that belongs to named independent module. You can use this syntax in an independent module when inquiring about a function not in that independent module. For example, use "Procedure [ProcGlobal]" to return information about functions in the main procedure window.

procedureWinTitleStr can also be just an independent module name in brackets to return information about the named nonstatic function in any procedure window that belongs to named independent module.

Omit the procedureWinTitleStr parameter or set it to "" when functionNameStr is "".

Details

The returned string contains several groups of information. Each group is prefaced by a keyword and colon, and terminated with the semicolon. The keywords are as follows:

User-Defined and External Functions

Keyword	**Information Following Keyword**
`NAME`	The name of the function. Same as contents of *functionNameStr* in most cases. Just the function name if you use the `module#function` format.
`TYPE`	Value is "UserDefined" or "XFunc".
`THREADSAFE`	Either "yes" or "no". See **ThreadSafe Functions** on page IV-106.

`RETURNTYPE`	The return type or types of the function. See **Return Type and Parameter Type Codes** on page V-281.
	For functions that return a single value, the return type is a simple number. For example, for a function that returns a single string:
	`RETURNTYPE:8192;`
	For functions that return multiple values, the return types are listed in the order of their declaration in brackets separated by commas. For example, for a function that returns a number and a string:
	`RETURNTYPE:[4100,12288];`

`N_PARAMS`	Number of parameters for this function.
`PARAM_`*n*`_TYPE`	Number encoding the type of each parameter. There will be N of these keywords, one for each parameter. The part shown as *n* will be a number from 0 to N-1.

See **Examples** for a method for decoding these keywords.

User-Defined Functions Only

Keyword	**Information Following Keyword**	
PROCWIN	Title of procedure window containing the function definition.	
MODULE	Module containing function definition (see **Regular Modules** on page IV-236).	
INDEPENDENTMODULE	Independent module containing function definition (see **Independent Modules** on page IV-238).	
SPECIAL	The value part has one of three values:	
	no:	Not a “special” function (not static or override).

User-Defined Functions Only

Keyword	Information Following Keyword
	static: Function is a static function. You must use the module#function format to get info about static functions.
	override: Function is an override function. See **Function Overrides** on page IV-106.
SUBTYPE	The function subtype, for instance **FitFunc**. See **Procedure Subtypes** on page V-13 for others.
PROCLINE	Line number within the procedure window of the function definition.
VISIBLE	Either "yes" or "no". Set to "no" in the unlikely event that the function is defined in an invisible file.
N_OPT_PARAMS	Number of optional parameters. Usually zero.

External Functions Only

Keyword	Information Following Keyword
XOP	Name of the XOP module containing the function.

Return Type and Parameter Type Codes

Type	Code	Code in Hex
Complex	1	0x1
Single Precision	2	0x2
Variable	4	0x4
Double Precision	4	0x4
Byte	8	0x8
16-bit Integer	16	0x10
32-bit Integer	32	0x20
Unsigned	64	0x40
/WAVE	128	0x80
Data folder reference	256	0x100
Structure	512	0x200
Function reference	1024	0x400
Pass by reference parameter	4096	0x1000
String	8192	0x2000
Wave	16384	0x4000
/Z	32768	0x8000

Igor functions can return a numeric value, a string value, a wave reference, or a data folder reference.

A returned numeric value is always double precision and may be complex. The return type for a normal numeric function is 4, for a complex function (Function/C) is 5 (4 for number +1 for complex).

A string function (Function/S) is 8192 (string). A Function/WAVE has a return type of 16384.

The return and parameter codes may be combined to indicate combinations of attributes. For instance, the code for a variable is 4 and the code for complex is 1. Consequently, the code for a complex variable parameter is 5. The code for a complex variable parameter passed by reference is (4+1+4096) = 4101.

Variables are always double-precision, hence the code of 4.

Waves may have a variety of codes. Numeric waves will combine with one of the number type codes such as 2 or 16. This does not reflect the numeric type of any actual wave, but rather any flag you may have used in the Wave reference. Thus, if the beginning of your function looks like

```
Function myFunc(w)
    Wave w
```

the code for the parameter w will be 16386 (16384 + 2) indicating a single-precision wave. You can use a numeric type flag with the Wave reference:

```
Function myFunc(w)
    Wave/I w
```

In this case, the code will be 16416 (16384 + 32).

Such codes are not very useful, as it is very rare to use a numeric type flag because the numeric type will be resolved correctly at runtime regardless of the flag.

A text wave has no numeric type, so its code is exactly 16384 or 49152 if /Z is also specified. Thus, the numeric type part of the code for a numeric wave serves to distinguish a numeric wave from a text wave. And a Wave/WAVE (a wave that contains references to other waves) has a code of 16512 (16384 + 128), unless /Z is also specified, which adds 32768, resulting in a code of 49280.

When a function uses the multi-valued return syntax, the return value types indicate that they are passed by reference. For example, the type code for a string returned this way is 0x2000 + 0x1000 = 0x3000 (12288 decimal).

Examples

This function formats function information nicely and prints it in the history area in an organized fashion. You can copy it into the Procedure window to try it out. It uses the function `InterpretType()` below to print a human-readable version of the parameter and return types. To try `PrintFuncInfo()`, you will need to copy the code for `InterpretType()` as well.

```
Function PrintFuncInfo(functionName)
    String functionName

    String infostr = FunctionInfo(functionName)
    if (strlen(infostr) == 0)
        print "The function \""+functionName+"\" does not exist."
        return -1
    endif

    print "Name: ", StringByKey("NAME", infostr)

    String typeStr = StringByKey("TYPE", infostr)
    print "Function type: ", typeStr
    Variable IsUserDefined = CmpStr(typeStr, "UserDefined")==0

    // It's not really necessary to use an IF statement here;
    // it simply prevents lines with blank information being
    // printed for an XFUNC.

    if (IsUserDefined)
        print "Module: ", StringByKey("MODULE", infostr)
        print "Procedure window: ", StringByKey("PROCWIN", infostr)
        print "Subtype: ", StringByKey("SUBTYPE", infostr)
        print "Special? ", StringByKey("SPECIAL", infostr)

        // Note use of NumberByKey to get a numeric key value
        print "Line number: ", NumberByKey("PROCLINE", infostr)
    endif

    // See function InterpretType() below for example of
    // interpreting type information.

    Variable returnType = NumberByKey("RETURNTYPE", infostr)

    String returnTypeStr = InterpretType(returnType, 1)
    printf "Return type: %d (0x%X) %s\r", returnType, returnType, returnTypeStr

    Variable nparams = NumberByKey("N_PARAMS", infostr)
    print "Number of Parameters: ", nparams

    Variable nOptParams = 0
    if (IsUserDefined)
        nOptParams = NumberByKey("N_OPT_PARAMS", infostr)
        print "Optional Parameters: ", nOptParams
    endif

    Variable i
    for (i = 0; i < nparams; i += 1)
        // Note how the PARAM_n_TYPE keyword string is constructed here:
        String paramKeyStr = "PARAM_"+num2istr(i)+"_TYPE"
        Variable ptype = NumberByKey(paramKeyStr, infostr)

        String ptypeStr = InterpretType(ptype,0)
        String format = "Parameter %d; type as number: %g (0x%X); type as string: %s"
        String output
        sprintf output, format, i, ptype, ptype, pTypeStr
        print output
    endfor

    return 0
End
```

Function that creates a human-readable string with information about parameter and return types. Note that various attributes of the type info is tested using the bitwise AND operator (&) to test for individual bits. The constants are expressed as hexadecimal values (prefixed with “0x”) to make them more readable (at least to a programmer). Otherwise, 0x4000 would be 16384; at least, 0x4000 is clearly a single-bit constant.

```
Function/S InterpretType(type, isReturnType)
    Variable type
    Variable isReturnType      // 0: type is parameter type; 1: type is return type.

    String typeStr = ""

    // limit type to unsigned 16-bit values (remove sign extensions caused by 0x8000)
    type = type & 0xFFFF

    // isNumeric is flag to tell whether to print out "complex" and "real";
    // we don't want that information on strings, text waves or wave of wave references.
    Variable isNumeric = 1

    if (type & 0x4000)         // test for WAVE bit set
        typeStr += "Wave"

        if( !isReturnType )
            if (type & 0x80)    // test for WAVE/WAVE bit set
                typeStr += "/WAVE"
                // don't print "real" or "complex" for wave waves
                isNumeric = 0
            endif
            if (type & 0x8000)  // test for WAVE/Z bit set
                typeStr += "/Z"
            endif
        endif
        typeStr += " "

        if( (type == 0x4000) || (type == (0x4000 | 0x8000)) )    // WAVE/T or WAVE/Z/T
            if( !isReturnType )
                // For parameter types, if no numeric bits are set, it is a text wave.
                // A numeric wave has some other bits set causing the value
                // to be different from 0x4000 or 0xC000.
                typeStr += "text "
            endif
            // Function/WAVE doesn't (cannot) specify whether the returned wave
            // is text or numeric.
            // Don't print "real" or "complex" for text or unknown wave types.
            isNumeric = 0
        endif
    elseif (type & 0x2000)     // test for STRING bit set
        typeStr += "String "
```