See Also

Control Panel Resolution on Windows on page III-456, **ScreenResolution**, **Control Panel Expansion** on page III-443

ParamIsDefault

`ParamIsDefault(pName)`

The ParamIsDefault function determines if an optional user function parameter *pName* was specified during the function call. It returns 1 when *pName* is default (not specified) or it returns 0 when it was specified.

Details

ParamIsDefault works only in the body of a user function and only with optional parameters. The variable *pName* must be valid at compile time; you can not defer lookup to runtime with $.

See Also

Optional Parameters on page IV-33 and **Using Optional Parameters** on page IV-60.

ParseFilePath

`ParseFilePath(mode, pathInStr, separatorStr, whichEnd, whichElement)`

The ParseFilePath function provides the ability to manipulate file paths and to extract sections of file paths.

Parameters

The meaning of the parameters depends on *mode*.

mode	**Information Returned**
0	Returns the element specified by *whichEnd* and *whichElement*. *whichEnd* is 0 to select an element relative to the beginning of *pathInStr*, 1 to select an element relative to the end. *whichElement* is zero-based. Pass ":" if *pathInStr* is a Macintosh HFS path, "\\" if it is a Windows path. See **Path Separators** on page III-451 for details about Macintosh versus Windows paths.
1	Returns the entire *pathInStr*, up to but not including the element specified by *whichEnd* and *whichElement*. *whichEnd* is 0 to select an element relative to the beginning of *pathInStr*, 1 to select an element relative to the end. *whichElement* is zero-based. Pass ":" if *pathInStr* is a Macintosh HFS path, "\\" if it is a Windows path. See **Path Separators** on page III-451 for details about Macintosh versus Windows paths.
2	Returns the entire *pathInStr* with a trailing separator added if it is not already there. This is useful when you have a path to a folder and want to tack on a file name. Pass ":" if *pathInStr* is a Macintosh HFS path, "\\" if it is a Windows path. See **Path Separators** on page III-451 for details about Macintosh versus Windows paths. *whichEnd* and *whichElement* are ignored. Pass 0 for them.
3	Returns the last element of *pathInStr* with the extension, if any, removed. The extension is anything after the last dot in *pathInStr*. *whichEnd* and *whichElement* are ignored. Pass 0 for them.
4	Returns the extension in *pathInStr* or `""` if there is no extension. The extension is anything after the last dot in *pathInStr*. Pass ":" if *pathInStr* is a Macintosh HFS path, "\\" if it is a Windows path. See **Path Separators** on page III-451 for details about Macintosh versus Windows paths. *whichEnd* and *whichElement* are ignored. Pass 0 for them.
5	Returns the entire *pathInStr* but converts it to a format determined by *separatorStr*.

mode	**Information Returned**
	separatorStr = ":" Converts the path to Macintosh HFS style if it is Windows style. Does nothing to a Macintosh HFS path. *separatorStr* = "\\" Converts the path to Windows style if it is Macintosh style. Does nothing to a Windows path. *separatorStr* = "*" Converts the path to the native style of the operating system Igor is running on. Does nothing to a native path. For historical reasons, on Macintosh “native” means colon-separated HFS path, not UNIX path. *separatorStr* = "/" Macintosh-only: Converts the Macintosh-style *pathInStr* input to a Posix (UNIX) path. Unlike the other conversions, the directory or file to which *pathInStr* refers must exist, otherwise "" is returned. To generate a Posix path for a non-existent file, generate the path for the existing folder and append the file name. This always returns "" on Windows. *whichEnd* and *whichElement* are ignored. Pass 0 for them.
6	UNC volume name ("\\Server\Share") if *pathIn* starts with a UNC volume name or "" if not. Pass "*" for *separatorStr*. *whichEnd* and *whichElement* are ignored. Pass 0 for them.
7	UNC server name ("Server" from "\\Server\Share") if *pathIn* starts with a UNC volume name or "" if not. Pass "*" for *separatorStr*. *whichEnd* and *whichElement* are ignored. Pass 0 for them.

mode	**Information Returned**
8	UNC share name ("Share" from "\\Server\Share") if *pathIn* starts with a UNC volume name or "" if not. Pass "*" for *separatorStr*. *whichEnd* and *whichElement* are ignored. Pass 0 for them.
9	Macintosh only. On Windows this mode returns an error. Returns a Posix version of *pathInStr* which must be a full HFS path pointing to an existing volume, directory or file. This is the same as mode 5 except that *separatorStr* must be "*". *whichEnd* and *whichElement* are ignored. Pass 0 for them. You would typically use this mode when you are about to execute a Unix command, which requires Posix paths, via ExecuteScriptText. This mode was created in Igor Pro 7.00 to provide an alternative to the obsolete HFSToPosix function provided by the HSFAndPosix XOP. With HFSToPosix, if the input path referred to a directory, the output always ended with a slash. With ParseFilePath(9), the output will end with a slash only if the input path ends with a colon.
10	Macintosh only. On Windows this mode returns an error. Returns the HFS path corresponding to the Posix path in pathInStr . *pathInStr* must be a full Posix path starting with a slash character. It does not need to point to an existing directory or file. The returned path may or may not refer to an existing volume, folder or file, depending on pathInStr . Pass "*" for *separatorStr*. *whichEnd* and *whichElement* are ignored. Pass 0 for them. You would typically use this mode when you receive a Posix path from a Unix command executed via ExecuteScriptText and you want to use that path in Igor. This mode was created in Igor Pro 7.00 to provide an alternative to the obsolete PosixToHFS function provided by the HSFAndPosix XOP.

Details

When dealing with Windows paths, you need to be aware that Igor treats the backslash character as an escape character. When you want to put a backslash in a literal string, you need to use two backslashes. See **Escape Sequences in Strings** on page IV-14 and **Path Separators** on page III-451 for details.

On Windows two types of file paths are used: drive-letter paths and UNC (“Universal Naming Convention”) paths. For example:

```
// This is a drive-letter path.
C:\Program Files\WaveMetrics\Igor Pro Folder\WaveMetrics Procedures

// This is a UNC path.
\\BigServer\SharedApps\WaveMetrics\Igor Pro Folder\WaveMetrics Procedures
```

In this example, ParseFilePath considers the volume name to be `C:` in the first case and `\\BigServer\SharedApps` in the second. The volume name is treated as one element by ParseFilePath, except for modes 7 and 8 which permit you to extract the components of the UNC volume name.

Except for the leading backslashes in a UNC path, ParseFilePath modes 0 and 1 internally strip any leading or trailing separator (as defined by the *separatorStr* parameter) from `pathInStr` before it starts parsing. So if you pass `":Igor Pro Folder:WaveMetrics Procedures:"`, it is the same as if you had passed `"Igor Pro Folder:WaveMetrics Procedures"`.

If there is no element corresponding to *whichElement* and *mode* is 0, ParseFilePath returns `""`.

If there is no element corresponding to *whichElement* and *mode* is 1, ParseFilePath returns the entire *pathInStr*.