Parameters

stringList is a semicolon-separated string list of waves specified using full paths or partial paths relative to the current data folder.

options is a bit field that is 0 by default. Set bit 0 to if you want the function to generate an error (see **GetRTError**) if any of the list elements does not specify an existing wave. Other bits are reserved for future use and must be cleared.

Example

```
Function Test()
   NewDataFolder/O/S root:TestDF     // Create empty data folder as current DF
   Make/O aaa                        // aaa will be the first wave in list
   Make/O bbb
   String strList = WaveList("*", ";", "")
   Wave/WAVE wr = ListToWaveRefWave(strList, 0)   // Returns a free wave
   Wave w = wr[0]
   Print GetWavesDataFolder(w, 2)
   Wave w = wr[1]
   Print GetWavesDataFolder(w, 2)
End
```

See Also

WaveRefWaveToList, **ListToTextWave**, **Wave References** on page IV-71

ln

`ln(`*`num`*`)`

The ln function returns the natural logarithm of *num*, -INF if *num* is 0, or NaN if *num* is less than 0. In complex expressions, *num* is complex, and ln(*num*) returns a complex value.

To compute a logarithm base n use the formula:

$$\log_n(x) = \frac{\log(x)}{\log(n)}.$$

See Also

The **log** function.

LoadData

`LoadData` `[`***`flags`***`]` ***`fileOrFolderNameStr`***

The LoadData operation loads data from the named file or folder. “Data” means Igor waves, numeric and string variables and data folders containing them. The specified file or folder must be an Igor packed experiment file or a folder containing Igor binary data, such as an Igor unpacked experiment folder or a folder in which you have stored Igor binary wave files.

In Igor Pro 9.00 or later, LoadData can load data from a standard packed experiment file (.pxp) or from an HDF5 packed experiment file (.h5xp). Previous versions support .pxp only. The term “packed experiment file” below refers to either .pxp or .h5xp.

LoadData loads data objects into memory and they become part of the current Igor experiment, disassociated from the file from which they were loaded.

If loading from a file-system folder, the data (waves, variables, strings) in the folder, including any subfolders if /R is specified, is loaded into the current Igor data folder.

If loading from a packed Igor experiment file, the data in the file, including any packed sub-data folders if /R is specified, is loaded into the current Igor data folder.

Use LoadData to load experiment data using Igor procedures. To load experiment data interactively, use the Data Browser (Data menu).

Parameters

If you use a full or partial path for *fileOrFolderNameStr*, see **Path Separators** on page III-451 for details on forming the path.

If *fileOrFolderNameStr* is omitted you get to locate the file (if /D is omitted) or the folder (if /D is present) via a dialog.

Flags

/D

If present, loads from a file-system folder (a directory). If omitted, LoadData loads from an Igor packed experiment file.

/GREP={*regExpStr*, *grepMode*, *objectTypeMask*, *grepFlags*}

Applies a regular expression to determine what objects are loaded. /GREP was added in Igor Pro 8.00.

See **Using Regular Expressions With LoadData** on page V-503 for details.

/I

Interactive. Forces LoadData to present a dialog.

/J=*objectNamesStr*

Loads only the objects named in the semicolon-separated list of object names.

/L=*loadFlags*

Controls what kind of data objects are loaded with a bit for each data type:

loadFlags	**Bit Number**	**Loads this Object Type**
1	0	Waves
2	1	Numeric Variables
4	2	String Variables

To load multiple data types, sum the values shown in the table. For example, /L=1 loads waves only, /L=2 loads numeric variables only, and /L=3 loads both waves and numeric variables. See **Setting Bit Parameters** on page IV-12 for further details about bit settings.

If no /L is specified, all object types are loaded. This is equivalent to /L=7. All other bits are reserved and should be set to zero.

/O[=*overwriteMode*]

If /O alone is used, overwrites existing data objects in case of a name conflict.

overwriteMode is defined as follows:

0:	No overwrite, as if there were no /O.
1:	Normal overwrite. In the event of a name conflict, objects in the incoming file replace the conflicting objects in memory. Incoming data folders completely replace any conflicting data folders in memory.
2:	Mix-in overwrite. In the event of a name conflict, objects in the incoming file replace the conflicting objects in memory but nonconflicting objects in memory are left untouched.

See **Details** for more about overwriting.

/P=*pathName*

Specifies folder to look in for the specified file or folder. *pathName* is the name of an existing Igor symbolic path. See **Symbolic Paths** on page II-22 for details.

/Q

Suppresses the normal messages in the history area.

/R

Recursively loads sub-data folders.

/S=*subDataFolderStr*

Specifies a sub-data folder within a packed experiment file to be loaded. See **Loading a Specific Data Folder from a Packed Experiment** on page V-502 for details.

/T[=*topLevelName*]

If /T=*topLevelName* is specified, it creates a new data folder in the current data folder with the specified name and places the loaded data in the new data folder.

If just /T is specified, it creates a new data folder in the current data folder with a name derived from the name of the unpacked experiment folder, packed experiment file or packed sub-data folder being loaded.

Details

If /T is present, LoadData loads the top level data folder and its contents. If /T is omitted, it loads just the contents of the top level data folder and not the data folder itself. This distinction has an analogy in the desktop. You can drag the contents of disk folder A into folder B or you can drag folder A itself into folder B.

Because LoadData can load from a complex packed Igor experiment file or from a complex hierarchy of file-system folders, it does not set the variables normally set by a file loader: S_path, S_fileName, and S_waveNames. The variable V_flag is set to the total number of objects loaded, or to -1 if the user cancelled the open file dialog. To find what objects were created by LoadData, you can use the **CountObjects** and **GetIndexedObjName** functions.

Overwriting Existing Data

If you do a load of a data folder, overwriting an existing data folder of the same name, the behavior of LoadData depends on whether you use /J. If you do not use /J, the entire data folder and all of its contents are replaced. If you do use /J, just the specified objects in the data folder are replaced, leaving any other preexisting objects in the data folder unchanged.

If you do not use the /O (overwrite) flag or if you use /O=0 and there is a conflict between objects or data folders in the current data folder and objects or data folders in the file or folder being loaded, LoadData presents a dialog to ask you if you want to replace the existing data. However, LoadData can not replace an object with an object of a different type and will refuse to do so.

You can overwrite an object that is in use, such as a wave that is displayed in a graph or table. You can also overwrite a data folder that contains objects that are in use. This is a powerful feature. Imagine that you define a data structure consisting of waves, variables and possibly sub-data folders. You can display the data in graphs and tables and you can display these in a layout. You can then overwrite the data with an analogous data structure from a packed experiment file and Igor will automatically update any graphs, tables or layouts that need to be updated.

Controlling Which Objects Are Loaded

LoadData provides several ways to control whether a given object is loaded or is skipped. These method, which are described in the following sections, include:

- Using /L to control whether waves, numeric variables, and string variables are loaded
- Specifying a sub-data folder using /S
- Specifying object names using /J
- Applying a regular expression using /GREP. /GREP requires Igor Pro 8.00 or later.

You can use more than one of these flags. For example, if you use /S and /J, then LoadData loads only objects in the data folder specified by /S with names listed using /J. In most cases, if you use /GREP, you will not need to combine it with /S or /J.

Loading a Specific Data Folder from a Packed Experiment

If present, /S=*subDataFolderStr* specifies the sub-data folder within the packed experiment file from which the load is to start. For example:

```
LoadData/P=Path1/S="root:Folder A:Folder B" "aPackedExpFile.pxp"
```

This starts loading from data folder "root:Folder A:Folder B" in the packed experiment file. *subDataFolderStr* is the full data folder path to the desired data folder. A trailing semicolon is allowed but not required. Since this parameter is specified as a string, you must not use single quotes to quote liberal names.

Prior to Igor Pro 7.00, the "root:" part of the data folder path was implicit and you had to omit it from *subDataFolderStr*. Now you can provide the full path including "root:" and this is recommended for clarity. For backward compatibility, if *subDataFolderStr* does not start with "root:", LoadData automatically prepends it.

/S has no effect if you are loading from an unpacked file system folder rather than from a packed experiment file.

Specifying /S="" acts like no /S at all.

Loading Specific Objects Based on Object Name

If /J=*objectNamesStr* is used, then only the objects named in objectNamesStr are loaded into the current experiment. For example, `/J="wave0;wave1;"` will load only the two named waves, ignoring any other data in the file or folder being loaded.

Assume that you have an experiment that contains 5 runs where each run, stored in a separate data folder in a packed experiment file, consists of data acquired from four channels from an ADC card. Using the /J flag, you can load just one specific channel from each run. You can use this to compare that channel's data from all runs without loading the other channels.

The list of object names used with /J must be semicolon-separated. A semicolon after the last object name in the list recommended but optional. Since objectNamesStr is specified as a string, you must not use single quotes to quote liberal names.

objectNamesStr is limited to 2000 bytes.

Specifying /J="" acts like no /J at all.

If you load a hierarchy of data folders, using the /R flag, with /J in effect, LoadData will create each data folder in the hierarchy even if it contains none of the named objects. This behavior is necessary to avoid loading a sub-data folder without loading its parent, as well as other such complications.

Using Regular Expressions With LoadData

If you include the /GREP={*regExpStr*,*grepMode*,*objectTypeMask*,*grepFlags*} flag, LoadData applies a regular expression (see **Regular Expressions** on page IV-176) to determine if a given object is to be loaded.

Before we get into the details, here are some simple examples. These examples assume that we are loading a packed experiment file named "Packed.pxp" and have an Igor symbolic path named Data.

```
// Load waves named wave1 from all data folders
LoadData/P=Data/Q/R/L=1/GREP={"(?i)^wave1$",1,1,0} "Packed.pxp"

// Load root:DataFolder0B and all objects in it
LoadData/P=Data/Q/R/L=7/GREP={"(?i)^root:DataFolder0B:$",3,8,0} "Packed.pxp"

// Load all data folders whose paths are of the form root:DataFolder<suffix>
LoadData/P=Data/Q/R/L=7/GREP={"(?i)^root:DataFolder.+:.*$",3,8,0} "Packed.pxp"
```

See the **Examples** on page V-504 for more examples.

regExpStr is the regular expression.

GREP is by default case-sensitive but Igor names are case-insensitive. For this reason, we recommend that you start *regExpStr* with "(?i)" which turns GREP's case-sensitivity off.

When looking for the appropriate value for *regExpStr*, you may find it helpful to test if a given name or path is matched by a given regular expression using **GrepString**. For example:

```
Print GrepString("root:DataFolder0A:str0", "(?i).*0$")   // Prints 1 if path is matched
```

The *grepMode* parameter selects one of three modes of using the regular expression.

If *grepMode* is 1, LoadData matches the name of each object of types specified by *objectTypeMask* against the specified regular expression. If there is a match, the object passes this test.

If *grepMode* is 2, LoadData matches the full data folder path to the object's parent data folder for objects of types specified by *objectTypeMask* against the regular expression. The full data folder path starts from "root:" and includes a trailing colon. If there is a match, the object passes this test. By definition, the parent of the root: data folder is "" (empty string) which can be expressed using the regular expression "^$".

If *grepMode* is 3, LoadData matches the full data folder path to the object against the regular expression. The full data folder path starts from "root:" and includes a trailing colon for data folder objects only. If there is a match, the object passes this test.

When loading unpacked data, modes 2 and 3 are not supported and are ignored. They apply when loading data from a packed experiment file only.

objectTypeMask is a bitwise parameter with each bit controlling whether the regular expression is applied to a given object type. Bits 0, 1, 2, and 3 control whether the regular expression is applied against waves, numeric variables, string variables and data folders respectively. You can apply a given regular expression to more than one type of object by setting more than one bit in *objectTypeMask*. See **Setting Bit Parameters** on page IV-12 for details about bit settings.

grepFlags is a bitwise parameter. Currently only bit 0 is used so its value must be 0 or 1. Normally, when *grepFlags*=0, objects that do not match the regular expression are skipped by LoadData. If *grepFlags*=1, the meaning of a match is reversed and objects that do match the regular expression are skipped by LoadData. For example:

```
// Load all objects whose name ends with "2"
LoadData/P=Data/Q/R/L=7/GREP={"(?i)[2]$",1,7,0} "Packed.pxp"    // Normal mode

// Load all objects except those whose name ends with "2"
LoadData/P=Data/Q/R/L=7/GREP={"(?i)[2]$",1,7,1} "Packed.pxp"    // Reverse mode
```

It is often not necessary, but you can use /GREP more than once in a given LoadData command, each time with a different combination of *grepMode*, object type, and reverse mode. A given object is loaded only if it passes all tests as well as any tests specified by /L, /J and /S. For example:

```
// Load root:DataFolder0B and all objects in it, except objects whose name ends with "2"
LoadData/P=Data/Q/R/L=7/GREP={"(?i)^root:DataFolder0B:$",3,8,0}/GREP={"(?i)[2]$",1,7,1} "Packed.pxp"
```

In this example, the first /GREP flag restricts which data folders are loaded and the second /GREP flag restricts which waves and variables are loaded.

If no test is applied to a particular kind of object then all objects of that type are loaded. For example, this command

```
LoadData/P=<pathName>/GREP={"(?i)^wave1$",1,1,0} <file name>
```

specifies that only waves named "wave1" should be loaded but it says nothing about numeric variables, string variables, or data folders so they are all loaded regardless of name. If you want to exclude all numeric and string variables, add the /L flag:

```
LoadData/P=<pathName>/L=1/GREP={"(?i)^wave1$",1,1,0} <file name>
```

As LoadData progresses through the data folders of the source, it has to create each data folder before it applies regular expression tests to objects in that data folder and its descendants. Consequently, it would be possible to end up with empty data folders because no data objects were loaded into them or their descendants. To eliminate the clutter that this would create, if you use /GREP, after loading all of the data, LoadData deletes any empty data folders that it created along the way. Data folders that existed before LoadData ran are not deleted even if they are empty. For backward compatibility, this pruning of empty data folders is done only if you use /GREP.

For each combination of *grepMode*, object type, and reverse mode, only one regular expression is applied. If you specify, for example, /GREP={"(?i)first",1,1,0} /GREP={"(?i)second",1,1,0}, the second /GREP flag overrides the first because they both apply a name test (*grepMode*=1) to wave objects (*objectTypeMask*=1) using normal mode (*grepFlags*=0). By contrast, /GREP={"(?i)first",1,1,0} /GREP={"(?i)second",1,2,0} results in no overriding because the first /GREP flag applies to waves (*objectTypeMask*=1) while the second applies to numeric variables (objectTypeMask=2). Similarly, /GREP={"(?i)first",1,1,0} /GREP={"(?i)second",1,1,1} results in no overriding because the first /GREP flag uses normal mode (*grepFlags*=0) while the second uses reverse mode (*grepFlags*=1).

Examples

These examples assume that we are loading a packed experiment file named "Packed.pxp" and have an Igor symbolic path named Data. The file contains this data hierarchy:

```
root:
   wave0,wave1,wave2,var0,var1,var2,str0,str1,str2
   DataFolder0A
      wave0,wave1,wave2,var0,var1,var2,str0,str1,str2
   DataFolder0B
      wave0,wave1,wave2,var0,var1,var2,str0,str1,str2

// Load everything
LoadData/P=Data/Q/R "Packed.pxp"

// Load all objects named wave1
LoadData/P=Data/Q/R/J="wave1;" "Packed.pxp"

// Load all objects whose name ends with "2"
LoadData/P=Data/Q/R/L=7/GREP={"(?i)[2]$",1,7,0} "Packed.pxp"

// Load all numeric variables whose name starts with "var"
LoadData/P=Data/Q/R/L=2/GREP={"(?i)^var",1,2,0} "Packed.pxp"

// Load all string variables whose name starts with "str" and ends with "2"
LoadData/P=Data/Q/R/L=4/GREP={"(?i)^str.*2$",1,4,0} "Packed.pxp"

// Load all objects whose path contains "DataFolder", case-insensitive
LoadData/P=Data/Q/R/L=7/GREP={"(?i)DataFolder",2,7,0} "Packed.pxp"

// Load root:DataFolder0B and all objects in it using /S
LoadData/P=Data/Q/R/L=7/S="root:DataFolder0B"/T "Packed.pxp"
```