Importing and Exporting Data

Loading Waves ............................................................................................................. 141
Load Waves Submenu ................................................................................................. 141
Number Formats .......................................................................................................... 143
The End of the Line ...................................................................................................... 143

Loading Delimited Text Files .................................................................................... 143
Date/Time Formats ...................................................................................................... 144
  Custom Date Formats ................................................................................................. 144
Column Labels ............................................................................................................ 146
Examples of Delimited Text ......................................................................................... 146
The Load Waves Dialog for Delimited Text — 1D .................................................. 147
Editing Wave Names .................................................................................................. 148
Set Scaling After Loading Delimited Text Data ....................................................... 149
The Load Waves Dialog for Delimited Text — 2D .................................................. 149
2D Label and Position Details .................................................................................... 149
Loading Text Waves from Delimited Text Files ..................................................... 150
Delimited Text Tweaks ............................................................................................... 151
Troubleshooting Delimited Text Files ....................................................................... 152
Loading Fixed Field Text Files .................................................................................. 152
The Load Waves Dialog for Fixed Field Text .......................................................... 153
Loading General Text Files ....................................................................................... 153
Examples of General Text ......................................................................................... 154
Comparison of General Text, Fixed Field and Delimited Text ................................ 154
The Load Waves Dialog for General Text — 1D .................................................... 155
Editing Wave Names for a Block .............................................................................. 155
The Load Waves Dialog for General Text — 2D .................................................... 156
Set Scaling After Loading General Text Data .......................................................... 156
General Text Tweaks ................................................................................................. 156
Troubleshooting General Text Files ......................................................................... 157
Loading Igor Text Files ............................................................................................. 158
Examples of Igor Text ............................................................................................... 158
Igor Text File Format .................................................................................................. 159
Setting Scaling in an Igor Text File .......................................................................... 160
The Load Waves Dialog for Igor Text ..................................................................... 160
Loading MultiDimensional Waves from Igor Text Files ....................................... 161
Loading Text Waves from Igor Text Files ............................................................. 161
The Igor Text File Type Code and File Extension .................................................... 162
Loading UTF-16 Files ............................................................................................... 162
Loading Igor Binary Data ......................................................................................... 162
The Igor Binary File .................................................................................................. 163
The Load Waves Dialog for Igor Binary ................................................................. 163
The LoadData Operation ........................................................................................... 164
Sharing Versus Copying Igor Binary Files ............................................................. 165
Loading Image Files ................................................................................................. 165
## Chapter II-9 — Importing and Exporting Data

The Load Image Dialog ................................................................. 165  
Image Loading Details ................................................................. 166  
Loading Other Files ........................................................................ 167  
Loading Row-Oriented Text Data .................................................. 168  
Loading Sound Files ....................................................................... 168  
Loading HDF Data ......................................................................... 169  
Loading GIS Data ........................................................................... 169  
Loading Very Big Binary Files ....................................................... 169  
Loading Waves Using Igor Procedures ........................................ 170  
  Variables Set by the LoadWave Operation ................................. 170  
  Loading and Graphing Waveform Data .................................... 170  
  Loading and Graphing XY Data ............................................... 173  
  Loading All of the Files in a Folder ......................................... 174  
Saving Waves ............................................................................... 175  
  Saving Waves in a Delimited Text File .................................... 176  
  Saving Waves in a General Text File ....................................... 177  
  Saving Waves in an Igor Text File ......................................... 177  
  Saving Waves in Igor Binary Files ....................................... 178  
  Saving Waves in Image Files ................................................. 178  
  Saving Sound Files ................................................................. 178  
Exporting Text Waves ................................................................. 178  
Exporting MultiDimensional Waves ............................................ 179  
Accessing SQL Databases ............................................................ 179
Loading Waves

Most Igor users create waves by loading data from a file created by another program. The process of loading a file creates new waves and then stores data from the file in them. Optionally, you can overwrite existing waves instead of creating new ones. The waves can be numeric or text and of dimension 1 through 4.

Igor provides a number of different routines for loading data files. There is no single file format for numeric or text data that all programs can read and write.

There are two broad classes of files used for data interchange: text files and binary files. Text files are usually used to exchange data between programs. Although they are called text files, they may contain numeric data, text data or both. In any case, the data is encoded as plain text that you can read in a text editor. Binary files usually contain data that is efficiently encoded in a way that is unique to a single program and can not be viewed in a text editor.

The closest thing to a universally accepted format for data interchange is the “delimited text” format. This consists of rows and columns of numeric or text data with the rows separated by carriage return characters (Macintosh), linefeed return characters (Unix), or carriage return/linefeed (Windows) and the columns separated by tabs or commas. The tab or comma is called the “delimiter character”. Igor can read delimited text files written by most programs.

FORTRAN programs usually create fixed field text files in which a fixed number of characters is used for each column of data with spaces as padding between columns. The Load Fixed Field Text routine is designed to read these files.

Text files are convenient because you can create, inspect or edit them with any text editor. In Igor, you can use a notebook window for this purpose. If you have data in a text file that has an unusual format, you may need to manually edit it before Igor can load it.

Text files generated by scientific instruments or custom programs often have “header” information, usually at the start of the file. The header is not part of the block of data but contains information associated with it. Igor’s text loading routines are designed to load the block of data, not the header. The Load General Text routine can usually automatically skip the header. The Load Delimited Text and Load Fixed Field Text routines needs to be told where the block of data starts if it is not at the start of the file.

An advanced user could write an Igor procedure to read and parse information in the header using the Open, FReadLine, StrSearch, sscanf and Close operations as well as Igor’s string manipulation capabilities. Igor includes an example experiment named Load File Demo which illustrates this.

If you will be working on a Macintosh, and loading data from files on a PC, or vice-versa, you should look at File System Issues on page III-400.

The following table lists the six types of built-in data loading routines in Igor and their salient features.

<table>
<thead>
<tr>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delimited text</td>
<td>Created by spreadsheets, database programs, data acquisition programs, text editors, custom programs. This is the most commonly used format for exchanging data between programs.</td>
</tr>
<tr>
<td></td>
<td><strong>Row Format</strong>: <code>&lt;data&gt;&lt;delimiter&gt;&lt;data&gt;&lt;CR&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Contains one block of data with any number of rows and columns. A row of column labels is optional.</td>
</tr>
<tr>
<td></td>
<td>Can load numeric, text, date, time, and date/time columns.</td>
</tr>
<tr>
<td></td>
<td>Can load columns into 1D waves or blocks into 2D waves.</td>
</tr>
<tr>
<td></td>
<td>Columns may be equal or unequal in length.</td>
</tr>
</tbody>
</table>
In addition, extensions to Igor are available to load data from other types of files, including Excel, Matlab, HDF, HDF5, JCAMP, DEM, DLG, Nicolet, various sound formats and general binary files. See **Loading Other Files** on page II-167 for details.

### Load Waves Submenu

You access all of these routines via the Load Waves submenu of the Data menu.
The Load Waves item in this submenu leads to the Load Waves dialog. This dialog invokes all of the built-in loading routines except for the image loader and accesses all available options.

The Load Igor Binary, Load Igor Text, Load General Text, and Load Delimited Text items in the Load Waves submenu are shortcuts that access the respective file loading routines with default options. We recommend that you start with the Load Waves item so that you can see what options are available. There are no shortcut items for loading fixed field text or image data because these formats require that you specify certain parameters.

The Load Image item leads to the Load Image dialog which provides the means to load various kinds of image files.

The remaining items are provided by Igor File-Loader Extensions. These are plug-in software modules that can be installed or removed easily as described under Loading Other Files on page II-167.

All of the built-in file loaders can load numeric data. The delimited text and fixed field text loaders can also load string text, date, time and date/time data.

**Number Formats**

A number has the following form:

```
[+/-] <digits> [.<digits>] [e/E[+/-] <exponent>]
```

An example is “-17.394e+3”. Some FORTRAN programs write “d” or “D” instead of “e” or “E” to introduce the exponent. Igor recognizes this.

**The End of the Line**

Different computer systems use different characters to mark the end of a line in a text file. The Macintosh uses the carriage-return character (CR). Unix uses linefeed (LF). Windows uses a carriage-return and linefeed (CRLF) combination. When loading waves, Igor treats a single CR, a single LF, or a CRLF as the end of a line. This allows Igor to load text data from file servers on a variety of computers without translation.

**Loading Delimited Text Files**

A delimited text file consists of rows of values separated by tabs or commas with a carriage return, linefeed or carriage return/linefeed combination at the end of the row. There may optionally be a row of column labels. Igor can load each column in the file into a separate 1D wave or it can load all of the columns into a single 2D wave. There is no limit to the number of rows or columns except that all of the data must fit in available memory.

In addition to numbers and text, the delimited text file may contain dates, times or date/times. The Load Delimited Text routine attempts to automatically determine which of these formats is appropriate for each column in the file. You can override this automatic determination if necessary.
Chapter II-9 — Importing and Exporting Data

A numeric column can contain, in addition to numbers, NaN and [±]INF. NaN means “Not a Number” and is the way Igor represents a blank or missing value in a numeric column. INF means “infinity”. If Igor finds text in a numeric or date/time column that it can’t interpret according to the format for that column, it treats it as a NaN.

If Igor encounters, in any column, a delimiter with no data characters preceding it (i.e., two tabs in a row) it takes this as a missing value and stores a blank in the wave. In a numeric wave, a blank is represented by a NaN. In a text wave, it is represented by an element with zero characters in it.

Date/Time Formats

The Load Delimited Text routine can handle dates in many formats. A few “standard” formats are supported and in addition, you can specify a “custom” format (see Custom Date Formats on page II-144).

The standard date formats are:

- **mm/dd/yy** (month/day/year)
- **mm/yy** (month/year)
- **dd/mm/yy** (day/month/year)

To use the **dd/mm/yy** format instead of **mm/dd/yy**, you must set a tweak. See Delimited Text Tweaks on page II-151.

You can also use a dash or a dot as a separator instead of a slash.

Igor can also handle times in the following forms:

- **[+]h:mm:ss [AM PM]** (hours, minutes, seconds)
- **[+]h:mm:ss.ff [AM PM]** (hours, minutes, seconds, fractions of seconds)
- **[+]h:mm [AM PM]** (hours, minutes)
- **[+]hh:mm:ss.ff** (hours, minutes, seconds, fractions of seconds)

As of Igor Pro 6.23, Igor also accepts a colon instead of a dot before the fractional seconds.

The first three forms are time-of-day forms. The last one is the elapsed time. In an elapsed time, the hour is in the range 0 to 9999.

The year can be specified using two digits (99) or four digits (1999). If a two digit year is in the range 00 … 39, Igor treats this as 2000 … 2039. If a two digit year is in the range 40 … 99, Igor treats this as 1940 … 1999.

The Load Delimited Text routine can also handle date/times which consist of one of these date formats, a single space or the letter T, and then one of the time formats.

Custom Date Formats

If your data file contains dates in a format other than the “standard” format, you can use Load Delimited Text to specify exactly what date format to use. You do this using the Delimited Text Tweaks dialog which you access through the Tweaks button in the Load Waves dialog. Choose Other from the Date Format pop-up menu. This leads to the Date Format dialog.
By clicking the Use Common Format radio button, you can choose from a pop-up menu of common formats. After choosing a common format, you can still control minor properties of the format, such as whether to use 2 or 4 digits years and whether to use leading zeros or not.

In the rare case that your file’s date format does not match one of the common formats, you can use a full custom format by clicking the Use Custom Format radio button. It is best to first choose the common format that is closest to your format and then click the Use Custom Format button. Then you can make minor changes to arrive at your final format.
When you use either a common format or a full custom format, the format that you specify must match the date in your file exactly.

When loading data as delimited text, if you use a date format containing a comma, such as “October 11, 1999”, you must make sure that LoadWave operation will not treat the comma as a delimiter. You can do this using the Delimited Text Tweaks dialog.

When loading a date format that consists entirely of digits, such as 991011, you should use the LoadWave/B flag to specify that the data is a date. Otherwise, LoadWave will treat it as a regular number. The /B flag can not be generated from the dialog — you need to use the LoadWave operation from the command line.

Another approach is to use the dialog to generate a LoadWave command without the /B flag and then specify that the column is a date column in the Loading Delimited Text dialog that appears when the LoadWave operation executes.

**Column Labels**

Each column may optionally have a column label. When loading 1D waves, if you read wave names and if the file has column labels, Igor will use the column labels for wave names. Otherwise, Igor will automatically generate wave names of the form wave0, wave1 and so on.

Igor considers text in the label line to be a column label if that text can not be interpreted as a data value (number, date, time, or datetime) or if the text is quoted using single or double quotes.

When loading a 2D wave, Igor optionally uses the column labels to set the wave’s column dimension labels. The wave name does not come from column labels but is automatically assigned by Igor. You can rename the wave after loading if you wish.

Igor expects column labels to appear in a row of the form:

```
<label><delimiter><label><delimiter>...<label><CR> (or CRLF or LF)
```

where `<column label>` may be in one of the following forms:

- `<label>` (label with no quotes)
- `"<label>"` (label with double quotes)
- `'<label>'` (label with single quotes)

The default delimiter characters are tab and comma. There is a tweak (see Delimited Text Tweaks on page II-151) for using other delimiters.

Igor expects that the row of column labels, if any, will appear at the beginning of the file. There is a tweak (see Delimited Text Tweaks on page II-151) that you can use to specify if this is not the case.

Igor will clean up column labels found in the file, if necessary, so that they are legal wave names using standard name rules. The cleanup consists of converting illegal characters into underscores and truncating long names to the maximum of 31 characters.

**Examples of Delimited Text**

Here are some examples of text that you might find in a delimited text file. These examples are tab-delimited.

**Simple delimited text**

```
ch0  ch1  ch2  ch3 (optional row of labels)
2.97055  1.95692  1.00871  8.10685
3.09921  4.08008  1.00016  7.53136
3.18934  5.91134  1.04205  6.90194
```

Loading this text would create four waves with three points each or, if you specify loading it as a matrix, a single 3 row by 4 column wave.
Delimited text with missing values

<table>
<thead>
<tr>
<th>ch0</th>
<th>ch1</th>
<th>ch2</th>
<th>ch3</th>
<th>(optional row of labels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.97055</td>
<td>1.95692</td>
<td>8.10685</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.09921</td>
<td>4.08008</td>
<td>1.00016</td>
<td>7.53136</td>
<td></td>
</tr>
<tr>
<td>5.91134</td>
<td>1.04205</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Loading this text as 1D waves would create four waves. Normally each wave would contain three points but there is an option to ignore blanks at the end of a column. With this option, ch0 and ch3 would have two points. Loading as a matrix would give you a single 3 row by 4 column wave with blanks in columns 0, 2 and 3.

Delimited text with a date column

<table>
<thead>
<tr>
<th>Date</th>
<th>ch0</th>
<th>ch1</th>
<th>ch2</th>
<th>(optional row of labels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/22/93</td>
<td>2.97055</td>
<td>1.95692</td>
<td>1.00871</td>
<td></td>
</tr>
<tr>
<td>2/24/93</td>
<td>3.09921</td>
<td>4.08008</td>
<td>1.00016</td>
<td></td>
</tr>
<tr>
<td>2/25/93</td>
<td>3.18934</td>
<td>5.91134</td>
<td>1.04205</td>
<td></td>
</tr>
</tbody>
</table>

Loading this text as 1D waves would create four waves with three points each. Igor would convert the dates in the first column into the appropriate number using the Igor system for storing dates (number of seconds since 1/1/1904). Loading as a matrix would give you a single 3 row by 4 column wave with column 0 containing dates encoded as numbers.

Delimited text with a nonnumeric column

<table>
<thead>
<tr>
<th>Sample</th>
<th>ch0</th>
<th>ch1</th>
<th>ch2</th>
<th>(optional row of labels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ge</td>
<td>2.97055</td>
<td>1.95692</td>
<td>1.00871</td>
<td></td>
</tr>
<tr>
<td>Si</td>
<td>3.09921</td>
<td>4.08008</td>
<td>1.00016</td>
<td></td>
</tr>
<tr>
<td>GaAs</td>
<td>3.18934</td>
<td>5.91134</td>
<td>1.04205</td>
<td></td>
</tr>
</tbody>
</table>

Loading this text as 1D waves would normally create four waves with three points each. The first wave would be a text wave and the remaining would be numeric. You could also load this as a single 3x3 matrix, treating the first row as column labels and the first column as row labels for the matrix. If you loaded it as a matrix but did not treat the first column as labels, it would create a 3 row by 4 column text wave, not a numeric wave.

The Load Waves Dialog for Delimited Text — 1D

To load a delimited text file as 1D waves, invoke the Load Waves dialog by choosing the Load Waves menu item.

The basic process of loading 1D data from a delimited text file is as follows:
1. Bring up the Load Waves dialog.
Chapter II-9 — Importing and Exporting Data

2. Choose Delimited Text from the File Type pop-up menu.
3. Click the File button to select the file containing the data.
4. Click Do It.

When you click Do It, the LoadWave operation runs. It executes the Load Delimited Text routine which goes through the following steps:
1. Optionally, determine if there is a row of column labels.
2. Determine the number of columns.
3. Determine the format of each column (number, text, date, time or date/time).
4. Optionally, present another dialog allowing you to confirm or change wave names.
5. Create waves.
6. Load the data into the waves.

Igor looks for a row of labels only if you enable the “Read wave names” option. If you enable this option and if Igor finds a row of labels then this determines the number of columns that Igor expects in the file. Otherwise, Igor counts the number of data items in the first row in the file and expects that the rest of the rows have the same number of columns.

In step 3 above, Igor determines the format of each column by examining the first data item in the column. Igor will try to interpret all of the remaining items in a given column using the format that it determines from the first item in the column.

If you choose Load Delimited Text from the Load Waves submenu instead of choosing Load Waves, Igor will display a dialog from which you can select the delimited text file to load directly. This is a shortcut that skips the Load Waves dialog and uses default options for the load. This will always load 1D waves, not a matrix.

Before you use this shortcut, take a look at the Load Waves dialog so you can see what options are available.

Editing Wave Names

The “Auto name & go” option is used mostly when you’re loading 1D data under control of an Igor procedure and you want everything to be automatic. When loading 1D data manually, you normally leave the “Auto name & go” option deselected. Then Igor presents an additional dialog in which you can confirm or change wave names.

The context area gives you feedback on what Igor is about to load. You can’t edit the file here. If you want to edit the file, abort the load and open the file as an Igor notebook or open it in a word processor.
Set Scaling After Loading Delimited Text Data

If your 1D numeric data is uniformly spaced in the X dimension then you will be able to use the many operations and functions in Igor designed for waveform data. You will need to set the X scaling for your waves after you load them, using the Change Wave Scaling dialog.

**Note:** If your 1D data is uniformly spaced it is very important that you set the X scaling of your waves. Many Igor operations depend on the X scaling information to give you correct results.

If your 1D data is not uniformly spaced then you will use XY pairs and you do not need to change X scaling. You may want to use Change Wave Scaling to set the data units.

The Load Waves Dialog for Delimited Text — 2D

To load a delimited text file as a 2D wave, choose the Load Waves menu item. Then, select the “Load columns into matrix” checkbox.

When you load a matrix (2D wave) from a text file, Igor creates a single wave. Therefore, there is no need for a second dialog to enter wave names. Instead, Igor automatically names the wave based on the base name that you specify. After loading, you can then rename the wave if you want.

To understand the row/column label/position controls, you need to understand Igor’s view of a 2D delimited text file:

<table>
<thead>
<tr>
<th>Optional row positions</th>
<th>Optional column labels</th>
<th>Optional column positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional row labels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row 0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>Row 1</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>Row 2</td>
<td>0.2</td>
<td></td>
</tr>
</tbody>
</table>

In the simplest case, your file has just the wave data — no labels or positions. You would indicate this by deselecting all four label/position checkboxes.

2D Label and Position Details

If your file does have labels or positions, you would indicate this by selecting the appropriate checkbox. Igor expects that row labels appear in the first column of the file and that column labels appear in the first line of the file unless you instruct it differently using the Tweak subdialog (see Delimited Text Tweaks on page II-151). Igor loads row/column labels into the wave’s dimension labels (described in Chapter II-6, Multidimensional Waves).
Chapter II-9 — Importing and Exporting Data

Igor can treat column positions in one of two ways. It can use them to set the dimension scaling of the wave (appropriate if the positions are uniformly-spaced) or it can create separate 1D waves for the positions. Igor expects row positions to appear in the column immediately after the row labels or in the first column of the file if the file contains no row labels. It expects column positions to appear immediately after the column labels or in the first line of the file if the file contains no column labels unless you instruct it differently using the Tweaks subdialog.

A row position wave is a 1D wave that contains the numbers in the row position column of the file. Igor names a row position wave “RP_” followed by the name of the matrix wave being loaded. A column position wave is a 1D wave that contains the numbers in the column position line of the file. Igor names a column position wave “CP_” followed by the name of the matrix wave being loaded. Once loaded (into separate 1D waves or into the matrix wave’s dimension scaling), you can use row and column position information when displaying a matrix as an image or when displaying a contour of a matrix.

If your file contains header information before the data, column labels and column positions, you need to use the Tweaks subdialog to specify where to find the data of interest. The “Line containing column labels” tweak specifies the line on which to find column labels. The “First line containing data” tweak specifies the first line of data to be stored in the wave itself. The first line in the file is considered to be line zero.

If you instruct LoadWave to read column positions, it determines which line contains them in one of two ways, depending on whether or not you also instructed it to read column labels. If you do ask LoadWave to read column labels, then LoadWave assumes that the column positions line immediately follows the column labels line. If you do not ask LoadWave to read column labels, then LoadWave assumes that the column positions line immediately precedes the first data line.

Loading Text Waves from Delimited Text Files

With regard to text columns, the Load Delimited Text operation can work in one of three ways: auto-identify column type, treat all columns as numeric, treat all columns as text. You can specify which method you want to use using the Tweaks subdialog of the Load Delimited Text dialog.

In the “auto-identify column type” method, Igor attempts to determine whether a column is numeric or text by examining the file. This is the default method when you choose Data→Load Waves→Load Delimited Text. Igor looks for the first nonblank value in each column and determines if the value is numeric or not. If it is numeric, Igor loads the column into a numeric wave which could be plain numeric, date, time or date/time as appropriate. If it is not numeric, Igor loads the column into a text wave.

In the “treat all columns as numeric” method, Igor loads all columns into numeric waves. This is the default method when you use the LoadWave/J operation from the command line or from an Igor procedure. We made LoadWave/J behave this way by default for backward-compatibility reasons. In ensures that Igor procedures will work the same in Igor Pro 3.0 and later as they did before. To use the “auto-identify column type” method, you need to use LoadWave/J/K=0.

In the “treat all columns as text” method, Igor loads all columns into text waves. This method may have use in rare cases in which you want to do text-processing on a file by loading it into a text wave and then using Igor’s string manipulation capabilities to massage it.

There are a few issues relating to special characters that you may need to deal with when loading data into text waves.

By default, the Load Delimited Text operation considers comma and tab characters to be delimiters which separate one column from the next. If the text that you are loading may contain commas or tabs as values rather than as delimiters, you will need to change the delimeter characters. You can do this using the Tweaks subdialog of the Load Delimited Text dialog.

The Load Delimited Text operation always considers carriage return and linefeed characters to mark the end of a line of text. It would be quite unusual to find a data file that uses these characters as values. In the extremely rare case that you need to load a carriage return or linefeed as a value, you can use an escape sequence. Replace the carriage return value with “\r” (without the quotes) and the linefeed value with “\n”. Igor will convert these to carriage return and linefeed and store the appropriate character in the text wave.
In addition to “\r” and “\n”, Igor will also convert “\t” into a tab value and do other escape sequence conversions (see *Escape Characters in Strings* on page IV-13). These conversions create a possible problem which should be quite rare. You may want to load text that contains “\r”, “\n” or “\t” sequences which you do not want to be treated as escape sequences. To prevent Igor from converting them into carriage return and tab, you will need to replace them with “\r”, “\n” and “\t”.

Igor does not remove quotation marks when loading data from delimited text files into text waves. If necessary, you can do this by opening the file as a notebook and doing a mass replace before loading or by displaying the loaded waves in a table and using Edit → Replace.

**Delimited Text Tweaks**

There are many variations on the basic form of a delimited text file. We’ve tried to provide tweaks that allow you to guide Igor when you need to load a file that uses one of the more common variations. To do this, use the Tweaks button in the Load Waves dialog. Most people will not need to use the tweaks.

The Tweaks dialog can specify the space character as a delimiter. Use the LoadWave operation to specify other delimiters as well.

The main reason for allowing space as a delimiter is so that we can load files that use spaces to align columns. This is a common format for files generated by FORTRAN programs. Normally, you should use the fixed field text loader to load these files, not the delimited text loader. If you do use the delimited text loader and if space is allowed as a delimiter then Igor treats any number of consecutive spaces as a single delimiter. This means that two consecutive spaces do not indicate a missing value as two consecutive tabs would.

When loading a delimited file, Igor normally expects the first line in the file to contain either column labels or the first row of data. There are several tweaks that you can use for a file that doesn’t fit this expectation.

Lines and columns in the tweaks dialog are numbered starting from zero.

Using the “Line containing column labels” tweak, you can specify on what line column labels are to be found if not on line zero. Using this and the “First line containing data” tweak, you can instruct Igor to skip garbage, if any, at the beginning of the file.

The “First line containing data”, “Number of lines containing data”, “First column containing data”, and “Number of columns containing data” tweaks are designed to allow you to load any block of data from anywhere within a file. This might come in handy if you have a file with hundreds of columns but you are only interested in a few of them.
If “Number of lines containing data” is set to “auto” or 0, Igor will load all lines until it hits the end of the file. If “Number of columns containing data” is set to “auto” or 0, Igor will load all columns until it hits the last column in the file.

The proper setting for the “Ignore blanks at the end of a column” tweak depends on the kind of 1D data stored in the file. If a file contains some number of similar columns, for example four channels of data from a digital oscilloscope, you probably want all of the columns in the file to be loaded into waves of the same length. Thus, if a particular column has one or more missing values at the end, the corresponding points in the wave should contain NaNs to represent the missing value. On the other hand, if the file contains a number of dissimilar columns, then you might want to ignore any blank points at the end of a column so that the resulting waves will not necessarily be of equal length. If you enable the “Ignore blanks at the end of a column” tweak then LoadWave will not load blanks at the end of a column into the 1D wave. If this option is enabled and a particular column has nothing but blanks then the corresponding wave is not loaded at all.

Troubleshooting Delimited Text Files

You can examine the waves created by the Load Delimited Text routine using a table. If you don’t get the results that you expected, you will need to try other LoadWave options or inspect and edit the text file until it is in a form that Igor can handle. Remember the following points:

• Igor expects the file to consist of numeric values, text values, dates, times or date/times separated by tabs or commas unless you set tweaks to the contrary.
• Igor expects a row of column labels, if any, to appear in the first line of the file unless you set tweaks to the contrary. It expects that the column labels are also delimited by tabs or commas unless you set tweaks to the contrary. Igor will not look for a line of column labels unless you enable the Read Wave Names option for 1D waves or the Read Column Labels options for 2D waves.
• Igor determines the number of columns in the file by inspecting the column label row or the first row of data if there is no column label row.

If merely inspecting the file does not identify the problem then you should try the following troubleshooting technique.

• Copy just the first few lines of the file into a test file.
• Load the test file and inspect the resulting waves in a table.
• Open the test file as a notebook.
• Edit the file to eliminate any irregularities, save it and load it again. Note that you can load a file as delimited text even if it is open as a notebook. Make sure that you have saved changes to the notebook before loading it.
• Inspect the loaded waves again.

This process usually sheds some light on what aspect of the file is irregular. Working on a small subset of your file makes it easier to quickly do some trial and error investigation.

If you are unable to get to the bottom of the problem, email a small segment of the file to support@wavemetrics.com along with a description of the problem. Do not send the segment as plain text because email programs may strip out or replace unusual control characters in the file. Instead, send a compressed version of the file.

Loading Fixed Field Text Files

A fixed field text file consists of rows of values, organized into columns, that are a fixed number of characters wide with a carriage return, linefeed, or carriage return/linefeed combination at the end of the row. Space characters are used as padding to ensure that each column has the appropriate number of characters. In some cases, a value will fill the entire column and there will be no spaces after it. FORTRAN programs typically generate fixed field text files.

Igor’s Load Fixed Field Text routine works just like the Load Delimited Text routine except that, instead of looking for a delimiter character to determine where a column ends, it counts the number of characters in the column. All of the features described in the section Loading Delimited Text Files on page II-143 apply also to loading fixed field text.
Chapter II-9 — Importing and Exporting Data

The Load Waves Dialog for Fixed Field Text

To load a fixed field text file, invoke the Load Waves dialog by choosing the Load Waves menu item.

![Load Waves Dialog](image)

The dialog is the same as for loading delimited text except for three additional items.

In the **Number of Columns** item, you must enter the total number of columns in the file. In the **Field Widths** item, you must enter the number of characters in each column of the file, separated by commas. The last value that you enter is used for any subsequent columns in the file. If all columns in the file have the same number of characters, just enter one number.

If you select the **All 9’s Means Blank** checkbox then Igor will treat any column that consists entirely of the digit 9 as a blank. If the column is being loaded into a numeric wave, Igor sets the corresponding wave value to NaN. If the column is being loaded into a text wave, Igor sets the corresponding wave value to "" (empty string).

## Loading General Text Files

We use the term “general text” to describe a text file that consists of one or more blocks of numeric data. A block is a set of rows and columns of numbers. Numbers in a row are separated by one or more tabs or spaces. One or more consecutive commas are also treated as white space. A row is terminated by a carriage return character, a linefeed character, or a carriage return/linefeed combination.

The Load General Text routine handles numeric data only, not date, time, date/time or text. Use Load Delimited Text or Load Fixed Field Text for these formats. Load General Text can handle 2D numeric data as well as 1D.

The first block of data may be preceded by header information which the Load General Text routine will automatically skip.

If there is a second block, it is usually separated from the first with one or more blank lines. There may also be header information preceding the second block which Igor will also skip.
When loading 1D data, the Load General Text routine loads each column of each block into a separate wave. It treats column labels as described above for the Load Delimited Text routine, except that spaces as well as tabs and commas are accepted as delimiters. When loading 2D data, it loads all columns into a single 2D wave.

The Load General Text routine determines where a block starts and ends by counting the number of numbers in a row. When it finds two rows with the same number of numbers, it considers this the start of a block. The block continues until a row which has a different number of numbers.

Examples of General Text

Here are some examples of text that you might find in a general text file.

Simple general text

```
ch0  ch1  ch2  ch3  (optional row of labels)
2.97055  1.95692  1.00871  8.10685
3.09921  4.08008  1.00016  7.53136
3.18934  5.91134  1.04205  6.90194
```

The Load General Text routine would create four waves with three points each or, if you specify loading as a matrix, a single 3 row by 4 column wave.

General text with header

```
Date: 3/2/93
Sample: P21-3A
ch0  ch1  ch2  ch3  (optional row of labels)
2.97055  1.95692  1.00871  8.10685
3.09921  4.08008  1.00016  7.53136
3.18934  5.91134  1.04205  6.90194
```

The Load General Text routine would automatically skip the header lines (Date: and Sample:) and would create four waves with three points each or, if you specify loading as a matrix, a single 3 row by 4 column wave.

General text with header and multiple blocks

```
Date: 3/2/93
Sample: P21-3A
ch0_1  ch1_1  ch2_1  ch3_1  (optional row of labels)
2.97055  1.95692  1.00871  8.10685
3.09921  4.08008  1.00016  7.53136
3.18934  5.91134  1.04205  6.90194
```

```
Date: 3/2/93
Sample: P98-2C
ch0_2  ch1_2  ch2_2  ch3_2  (optional row of labels)
2.97055  1.95692  1.00871  8.10685
3.09921  4.08008  1.00016  7.53136
3.18934  5.91134  1.04205  6.90194
```

The Load General Text routine would automatically skip the header lines and would create eight waves with three points each or, if you specify loading as a matrix, two 3 row by 4 column waves.

Comparison of General Text, Fixed Field and Delimited Text

You may wonder whether you should use the Load General Text routine, Load Fixed Field routine or the Load Delimited Text routine. Most commercial programs create simple tab-delimited files which these routines can handle. Files created by scientific instruments, mainframe programs, custom programs, or exported from spreadsheets are more diverse. You may need to try these routines to see which works better. To help you decide which to try first, here is a comparison.

Advantages of the Load General Text compared to Load Fixed Field and to Load Delimited Text:

- It can automatically skip header text.
Chapter II-9 — Importing and Exporting Data

- It can load multiple blocks from a single file.
- It can tolerate multiple tabs or spaces between columns.

Disadvantages of the Load General Text compared to Load Fixed Field and to Load Delimited Text:
- It can not handle blanks (missing values).
- It can not tolerate columns of nonnumeric text or nonnumeric values in a numeric column.
- It can not load text values, dates, times or date/times.
- It can not handle comma as the decimal point (European number style).

The Load General Text routine can load missing values if they are represented in the file explicitly as “NaN” (Not-a-Number). It can not handle files that represent missing values as blanks because this confounds the technique for determining where a block of numbers starts and ends.

The Load Waves Dialog for General Text — 1D

To load a general text file as 1D waves, invoke the Load Waves dialog by choosing the Load Waves menu item. The dialog appears as shown above for delimited text.

The basic process of loading data from a general text file is as follows:
1. Bring up the Load Waves dialog.
2. Choose General Text from the File Type pop-up menu.
3. Click the File button to select the file containing the data.
4. Click Do It.

When you click Do It, Igor’s LoadWave operation runs. It executes the Load General Text routine which goes through the following steps:
1. Locate the start of the block of data using the technique of counting numbers in successive lines. This step also skips the header, if any, and determines the number of columns in the block.
2. Optionally, determine if there is a row of column labels immediately before the block of numbers.
3. Optionally, present another dialog allowing you to confirm or change wave names.
4. Create waves.
5. Load data into the waves until the end of the file or until a row that contains a different number of numbers.
6. If not at the end of the file, go back to step one to look for another block of data.

Igor looks for a row of column labels only if you enable the “Read wave names” option. It looks in the line immediately preceding the block of data. If it finds labels and if the number of labels matches the number of columns in the block, it uses these labels as wave names. Otherwise, Igor will automatically generate wave names of the form wave0, wave1 and so on.

If you choose the Load General Text item from the Load Waves submenu instead of the Load Waves item, Igor will display a dialog from which you can select the general text file to load directly. This is a shortcut that skips the Load Waves dialog and uses default options for the load. This will always load 1D waves, not a matrix.

Before you use this shortcut, take a look at the Load Waves dialog so you can see what options are available.

Editing Wave Names for a Block

In step 3 above, the Load General Text routine presents a dialog in which you can change wave names. This works exactly as described above for the Load Delimited Text routine except that it has one extra button: “Skip this block”.

II-155
Use “Skip this block” to skip one or more blocks of a multiple block general text file.

Click the Skip Column button to skip loading of the column corresponding to the selected name box. Shift-click the button to skip all columns except the selected one.

**The Load Waves Dialog for General Text — 2D**

Igor can load a 2D wave using the Load General Text routine. However, Load General Text does not support the loading of row/column labels and positions. If the file has such rows and columns, you must load it as a delimited text file.

The main reason to use the Load General Text routine rather than the Load Delimited Text routine for loading a matrix is that the Load General Text routine can automatically skip nonnumeric header information. Also, Load General Text treats any number of spaces and tabs, as well as one comma, as a single delimiter and thus is tolerant of less rigid formatting.

**Set Scaling After Loading General Text Data**

If your 1D data is uniformly spaced in the X dimension then you will be able to use the many operations and functions in Igor designed for waveform data. You will need to set the X scaling for your waves after you load them, using the Change Wave Scaling dialog.

*Note:* If your data is uniformly spaced it is very important that you set the X scaling of your waves. Many Igor operations depend on the X scaling information to give you correct results.

If your 1D data is not uniformly spaced then you will use XY pairs and you do not need to change X scaling. You may want to use Change Wave Scaling to set the waves’ data units.

**General Text Tweaks**

The Load General Text routines provides some tweaks that allow you to guide Igor as it loads the file. To do this, use the Tweaks button in the Load Waves dialog. Most people will not need to use these tweaks.
Chapter II-9 — Importing and Exporting Data

The items at the top of the dialog are hidden because they apply to the Load Delimited Text routine only. Load General Text always skips any tabs and spaces between numbers and will also skip a single comma. The “decimal point” character is always period and it can not handle dates.

The items relating to column labels, data lines and data columns have two potential uses. You can use them to load just a part of a file or to guide Igor if the automatic method of finding a block of data produces incorrect results.

Lines and columns in the tweaks dialog are numbered starting from zero.

Igor interprets the “Line containing column labels” and “First line containing data” tweaks differently for general text files than it does for delimited text files. For delimited text, zero means “the first line”. For general text, zero for these parameters means “auto”.

Here is what “auto” means for general text. If “First line containing data” is auto, Igor starts the search for data from the beginning of the file without skipping any lines. If it is not “auto”, then Igor skips to the specified line and starts its search for data there. This way you can skip a block of data at the beginning of the file. If “Line containing column labels” is auto then Igor looks for column labels in the line immediately preceding the line found by the search for data. If it is not auto then Igor looks for column labels in the specified line.

If the “Number of lines containing data” is not “auto” then Igor will stop loading after the specified number of lines or when it hits the end of the first block, whichever comes first. This behavior is necessary so that it is possible to pick out a single block or subset of a block from a file containing more than one block.

If a general text file contains more than one block of data and if “Number of lines containing data” is “auto” then, for blocks after the first one, Igor maintains the relationship between the line containing column labels and first line containing data. Thus, if the column labels in the first block were one line before the first line containing data then Igor will expect the same to be true of subsequent blocks.

You can use the “First column containing data” and “Number of columns containing data” tweaks to load a subset of the columns in a block. If “Number of columns containing data” is set to “auto” or 0, Igor will load all columns until it hits the last column in the block.

Troubleshooting General Text Files

You can examine the waves created by the Load General Text routine using a table. If you don’t get the results that you expected, you will need to inspect and edit the text file until it is in a form that Igor can handle. Remember the following points:

- Load General Text can not handle dates, times, date/times, commas used as decimal points, or blocks of data with nonnumeric columns. Try Load Delimited Text for this.
- It skips any tabs or spaces between numbers and will also skip a single comma.
Chapter II-9 — Importing and Exporting Data

- It expects a line of column labels, if any, to appear in the first line before the numeric data unless you set tweaks to the contrary. It expects that the labels are also delimited by tabs, commas or spaces. It will not look for labels unless you enable the Read Wave Names option.
- It works by counting the number of numbers in consecutive lines. Some unusual formats (e.g., 1,234.56 instead of 1234.56) can throw this count off, causing it to start a new block prematurely.
- It can not handle blanks or nonnumeric values in a column. Each of these will start a new block of data.
- If it detects a change in the number of columns, it starts loading a new block into a new set of waves.

If merely inspecting the file does not identify the problem then you should try the technique of loading a subset of your data. This is described under Troubleshooting Delimited Text Files on page II-152 and often sheds light on the problem. In the same section, you will find instructions for sending the problem file to WaveMetrics for analysis, if necessary.

Loading Igor Text Files

An Igor Text file consists of keywords, data and Igor commands. The data can be numeric, text or both and can be of dimension 1 to 4. Many Igor users have found this to be an easy and powerful way to import data from their own custom programs into Igor.

The file name extension for an Igor Text file is “.itx”. Old versions of Igor used “.awav” and this is still accepted.

Examples of Igor Text

Here are some examples of text that you might find in an Igor Text file.

Simple Igor Text

IGOR
WAVES/D unit1, unit2
BEGIN
  19.7  23.9
  19.8  23.7
  20.1  22.9
END
X SetScale x 0,1, "V", unit1; SetScale d 0,0, "A", unit1
X SetScale x 0,1, "V", unit2; SetScale d 0,0, "A", unit2

Loading this would create two double-precision waves named unit1 and unit2 and set their X scaling, X units and data units.

Igor Text with extra commands

IGOR
WAVES/D/O xdata, ydata
BEGIN
  98.822    486.528
  109.968   541.144
  119.573   588.21
  133.178   654.874
  142.906   702.539
END
X SetScale d 0,0, "V", xdata
X SetScale d 0,0, "A", ydata
X Display ydata vs xdata; DoWindow/C TempGraph
X ModifyGraph mode=2,lsize=5
X CurveFit line ydata /X=xdata /D
X Textbox/A=LT/X=0/Y=0 "ydata= \{W_coef[0]\}+\{W_coef[1]\}*xdata"
X PrintGraphs TempGraph
X DoWindow/K TempGraph // kill the graph
X KillWaves xdata, ydata, fit_ydata // kill the waves
Chapter II-9 — Importing and Exporting Data

Loading this would create two double-precision waves and set their data units. It would then make a graph, do a curve fit, annotate the graph and print the graph. The last two lines do housekeeping.

Igor Text File Format

An Igor Text file starts with the keyword IGOR. The rest of the file may contain blocks of data to be loaded into waves or Igor commands to be executed and it must end with a blank line.

A block of data in an Igor Text file must be preceded by a declaration of the waves to be loaded. This declaration consists of the keyword WAVES followed by optional flags and the names of the waves to be loaded. Next the keyword BEGIN indicates the start of the block of data. The keyword END marks the end of the block of data.

A file can contain any number of blocks of data, each preceded by a declaration. If the waves are 1D, the block can contain any number of waves but waves in a given block must all be of the same data type. Multidimensional waves must appear one wave per block.

A line of data in a block consists of one or more numeric or text items with tabs separating the numbers and a carriage return at the end of the line. Each line should have the same number of items.

You can’t use blanks, dates, times or date/times in an Igor Text file. To represent a missing value in a numeric column, use “NaN” (not-a-number). To represent dates or times, use the standard Igor date format (number of seconds since 1/1/1904).

There is no limit to the number of waves or number of points except that all of the data must fit in available memory.

The WAVES keyword accepts the following optional flags:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>/N=(…)</td>
<td>Specifies size of each dimension for multidimensional waves.</td>
</tr>
<tr>
<td>/O</td>
<td>Overwrites existing waves.</td>
</tr>
<tr>
<td>/R</td>
<td>Makes waves real (default).</td>
</tr>
<tr>
<td>/C</td>
<td>Makes waves complex.</td>
</tr>
<tr>
<td>/S</td>
<td>Makes waves single precision floating point (default).</td>
</tr>
<tr>
<td>/D</td>
<td>Makes waves double precision floating point.</td>
</tr>
<tr>
<td>/I</td>
<td>Makes waves 32 bit integer.</td>
</tr>
<tr>
<td>/W</td>
<td>Makes waves 16 bit integer.</td>
</tr>
<tr>
<td>/B</td>
<td>Makes waves 8 bit integer.</td>
</tr>
<tr>
<td>/U</td>
<td>Makes integer waves unsigned.</td>
</tr>
<tr>
<td>/T</td>
<td>Specifies text data type.</td>
</tr>
</tbody>
</table>

Normally you should make single or double precision floating point waves. Integer waves are normally used only to contain raw data acquired via external operations. They are also appropriate for storing image data.

The /N flag is needed only if the data is multidimensional but the flag is allowed for one-dimensional data, too. Regardless of the dimensionality, the dimension size list must always be inside parentheses. Examples:

\[
\text{WAVES/N=(5) wave1D} \\
\text{WAVES/N=(3,3) wave2D} \\
\text{WAVES/N=(3,3,3) wave3D}
\]

Integer waves are signed unless you use the /U flag to make them unsigned.

If you use the /C flag then a pair of numbers in a line supplies the real and imaginary value for a single point in the resulting wave.
Chapter II-9 — Importing and Exporting Data

If you specify a wave name that is already in use and you don’t use the overwrite option, Igor will display a dialog so that you can resolve the conflict.

The /T flag makes text rather than numeric waves. See Loading Text Waves from Igor Text Files on page II-161.

A command in an Igor Text file is introduced by the keyword X followed by a space. The command follows the X on the same line. When Igor encounters this while loading an Igor Text file it executes the command.

Anything that you can execute from Igor’s command line is acceptable after the X. Introduce comments with “X //”. There is no way to do conditional branching or looping. However, you can call an Igor procedure defined in a built-in or auxiliary procedure window.

Commands, introduced by X, are executed as if they were entered on the command line or executed via the Execute operation. Such command execution is not thread-safe. Therefore, you can not load an Igor text file containing a command from an Igor thread.

Setting Scaling in an Igor Text File

When Igor writes an Igor Text file, it always includes commands to set each wave’s scaling, units and dimension labels. It also sets each wave’s note.

If you write a program that generates Igor Text files, you should set at least the scaling and units. If your 1D data is uniformly spaced in the X dimension, you should use the SetScale operation to set your waves X scaling, X units and data units. If your data is not uniformly spaced, you should set the data units only. For multidimensional waves, use SetScale to set Y, Z and T units if needed.

The Load Waves Dialog for Igor Text

To load an Igor Text file, invoke the Load Waves dialog by choosing the Load Waves menu item.

The basic process of loading data from an Igor Text file is as follows:
1. Bring up the Load Waves dialog.
2. Choose Igor Text from the File Type pop-up menu.
3. Click the File button to select the file containing the data.
4. Click Do It.

When you click Do It, Igor’s LoadWave operation runs. It executes the Load Igor Text routine which loads the file.
If you choose the Load Igor Text item from the Load Waves submenu instead of the Load Waves item, Igor will display a dialog from which you can select the Igor Text file to load directly. This is a shortcut that skips the Load Waves dialog.

**Loading MultiDimensional Waves from Igor Text Files**

In an Igor Text file, a block of wave data is preceded by a WAVES declaration. For multidimensional data, you must use a separate block for each wave. Here is an example of an Igor Text file that defines a 2D wave:

```igor
IGOR
WAVES/D/N=(3,2) wave0
BEGIN
  1  2
  3  4
  5  6
END
```

The “/N=(3,2)” flag specifies that the wave has three rows and two columns. The first line of data (1 and 2) contains data for the first row of the wave. This layout of data is recommended for clarity but is not required. You could create the same wave with:

```igor
IGOR
WAVES/D/N=(3,2) wave0
BEGIN
  1 2 3 4 5 6
END
```

Igor merely reads successive values and stores them in the wave, storing a value in each column of the first row before moving to the second row. All white space (spaces, tabs, return and linefeed characters) are treated the same.

When loading a 3D wave, Igor expects the data to be in column/row/layer order. You can leave a blank line between layers for readability but this is not required.

Here is an example of a 3 rows by 2 columns by 2 layers wave:

```igor
IGOR
WAVES/D/N=(3,2,2) wave0
BEGIN
  1  2
  3  4
  5  6
  11 12
  13 14
  15 16
END
```

The first 6 numbers define the values of the first layer of the 3D wave. The second 6 numbers define the values of the second layer.

When loading a 4D wave, Igor expects the data to be in column/row/layer/chunk order. You can leave a blank line between layers and two blank lines between chunks for readability but this is not required.

If loading a multidimensional wave, Igor expects that the dimension sizes specified by the /N flag are accurate. If there is more data in the file than expected, Igor ignores the extra data. If there is less data than expected, some of the values in the resulting waves will be undefined. In either of these cases, Igor will print a message in the history area to alert you to the discrepancy.

**Loading Text Waves from Igor Text Files**

Loading text waves from Igor Text files is similar to loading them from delimited text files except that in an Igor Text file you declare a wave’s name and type. Also, text strings are quoted in Igor Text files as they are in Igor’s command line. Here is an example of Igor Text that defines a text wave:

```igor
IGOR
WAVES/D/N=(3,2) wave0
BEGIN
  1  2
  3  4
  5  6
END
```
IGOR
WAVES/T textWave0, textWave1
BEGIN
   "This"    "Hello"
   "is"      "out"
   "a test"  "there"
END

All of the waves in a block of an Igor Text file must have the same number of points and data type. Thus, you
can not mix numeric and text waves in the same block. You can have any number of blocks in one Igor Text file.

As this example illustrates, you must use double quotes around each string in a block of text data. If you
want to embed a quote, tab, carriage return or linefeed within a single text value, use the escape sequences
\", \t, \r or \n. Use \\
 to embed a backslash. For less common escape sequences, see Escape Characters in Strings on page IV-13.

The Igor Text File Type Code and File Extension

On Macintosh, Igor recognizes files of type IGTX as Igor Text. The file type can also be TEXT. If you are
writing a program that generates Igor text files, use file type IGTX, creator code IGR0 (last character is zero)
and the file name extension “.itx”.

On Windows, just use the file name extension “.itx”.

Loading UTF-16 Files

The LoadWave operation can load data from UTF-16 (two-byte Unicode) text files. It does not recognize
non-ASCII characters, but does ignore the byte-order mark at the start of the file (BOM) and null bytes con-
tained in UTF-16 text files. Consequently it can load data from UTF-16 files containing just numeric data
and ASCII text.

Loading Igor Binary Data

This section discusses loading Igor Binary data into memory. Igor stores Igor Binary data in two ways: one
wave per Igor Binary file in unpacked experiments and multiple waves within a packed experiment file.

When you open an experiment, Igor automatically loads the Igor Binary data to recreate the experiment’s
waves. The main reason to explicitly load an Igor Binary file is if you want to load data from another
program that knows how to create an Igor Binary file. The easiest way to load data from another experiment
is to use the Data Browser (see Data Browser on page II-130).

Warning: You can get into trouble if two Igor experiments load data from the same Igor Binary file. See
Sharing Versus Copying Igor Binary Files on page II-165 for details.

There are a number of ways to load Igor Binary data into the current experiment in memory. Here is a sum-
mary. For most users, the first and second methods — which are simple and easy to use — are sufficient.

<table>
<thead>
<tr>
<th>Method</th>
<th>Loads</th>
<th>Action</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Experiment</td>
<td>Packed and unpacked files</td>
<td>Restores experiment to the state in which it was last saved.</td>
<td>To restore experiment.</td>
</tr>
<tr>
<td>(Chapter II-3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Browser</td>
<td>Packed and unpacked files</td>
<td>Copies data from one experiment to another.</td>
<td>To collect data from different sources for comparison.</td>
</tr>
<tr>
<td>(Chapter II-8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browse Waves Dialog</td>
<td>Unpacked files only</td>
<td>Copies data from one experiment to another or shares between experiments.</td>
<td>To collect data from different sources for comparison.</td>
</tr>
<tr>
<td>(Chapter II-5)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter II-9 — Importing and Exporting Data

The Igor Binary File

The Igor Binary file format is Igor’s native format for storing waves. This format stores one wave per file very efficiently. The file includes the numeric contents of the wave (or text contents if it is a text wave) as well as all of the auxiliary information such as the dimension scaling, dimension and data units and the wave note. In an Igor packed experiment file, any number of Igor Binary wave files can be packed into a single file.

The file name extension for an Igor Binary file is “.ibw”. Old versions of Igor used “.bwav” and this is still accepted. The Macintosh file type code is IGBW and the creator code is IGR0 (last character is zero).

The name of the wave is stored inside the Igor Binary file. It does not come from the name of the file. For example, wave0 might be stored in a file called “wave0.ibw”. You could change the name of the file to anything you want. This does not change the name of the wave stored in the file.

The Igor Binary file format was designed to save waves that are part of an Igor experiment. In the case of an unpacked experiment, the Igor Binary files for the waves are stored in the experiment folder and can be loaded using the LoadWave operation. In the case of a packed experiment, data in Igor Binary format is packed into the experiment file and can be loaded using the LoadData operation.

Some Igor users have written custom programs that write Igor Binary files which they load into an experiment. Igor Technical Note #003, “Igor Binary Format”, provides the details that a programmer needs to do this. See also Igor Pro Technical Note PTN003.

The Load Waves Dialog for Igor Binary

To load an Igor Binary file, invoke the Load Waves dialog by choosing the Load Waves menu item.

<table>
<thead>
<tr>
<th>Method</th>
<th>Loads</th>
<th>Action</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop Drag and Drop</td>
<td>Unpacked files only</td>
<td>Copies data from one experiment to another or shares between experiments.</td>
<td>To collect data from different sources for comparison.</td>
</tr>
<tr>
<td>(Chapter II-3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Waves Dialog</td>
<td>Unpacked files only</td>
<td>Copies data from one experiment to another or shares between experiments.</td>
<td>To create a LoadWave command that can be used in an Igor procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LoadWaves Operation</td>
<td>Unpacked files only</td>
<td>Copies data from one experiment to another or shares between experiments.</td>
<td>To automatically load data using an Igor Procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LoadData Operation</td>
<td>Packed and unpacked</td>
<td>Copies data from one experiment to another.</td>
<td>To automatically load data using an Igor Procedure.</td>
</tr>
<tr>
<td></td>
<td>files</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Load Waves Dialog for Igor Binary

To load an Igor Binary file, invoke the Load Waves dialog by choosing the Load Waves menu item.
Chapter II-9 — Importing and Exporting Data

The basic process of loading data from an Igor Binary file is as follows:
1. Bring up the Load Waves dialog.
2. Choose Igor Binary from the File Type pop-up menu.
3. Click the File button to select the file containing the data.
4. Set the “Copy to home” checkbox.
5. Click Do It.

When you click Do It, Igor’s LoadWave operation runs. It executes the Load Igor Binary routine which loads the file. If the wave that you are loading has the same name as an existing wave or other Igor object, Igor will present a dialog in which you can resolve the conflict.

Notice the “Copy to home” checkbox. It is very important.

If it is selected, Igor will disassociate the wave from its source file after loading it into the current experiment. When you next save the experiment, Igor will store a new copy of the wave with the current experiment. The experiment will not reference the original source file. We call this “copying” the wave to the current experiment.

If “Copy to home” is not selected, Igor will keep the connection between the wave and the file from which it was loaded. When you save the experiment, it will contain a reference to the source file. We call this “sharing” the wave between experiments.

We strongly recommend that you copy waves rather than share them. See Sharing Versus Copying Igor Binary Files on page II-165 for details.

If you choose the Load Igor Binary item from the Load Waves submenu instead of the Load Waves item, Igor will display a dialog from which you can select the Igor Binary file to load directly. This is a shortcut that skips the Load Waves dialog. When you take this shortcut, you lose the opportunity to set the “Copy to home” checkbox. Thus, during the load operation, Igor will present a dialog from which you can choose to copy or share the wave.

The LoadData Operation

The LoadData operation provides a way for Igor programmers to automatically load data from packed Igor experiment files or from a file-system folder containing unpacked Igor Binary files. It can load not only waves but also numeric and string variables and a hierarchy of data folders that contains waves and variables.

The Data Browser’s Browse Expt button provides interactive access to the LoadData operation and permits you to drag a hierarchy of data from one Igor experiment into the current experiment in memory. To achieve the same functionality in an Igor procedure, you need to use the LoadData operation directly. See the LoadData operation (see page V-388).

LoadData, accessed from the command line or via the Data Browser, has the ability to overwrite existing waves, variables and data folders. Igor automatically updates any graphs and tables displaying the overwritten waves. This provides a very powerful and easy way to view sets of identically structured data, as would be produced by successive runs of an experiment. You start by loading the first set and create graphs and tables to display it. Then, you load successive sets of identically named waves. They overwrite the preceding set and all graphs and tables are automatically updated.
Sharing Versus Copying Igor Binary Files

There are two reasons for loading a binary file that was created as part of another Igor experiment: you may want your current experiment to share data with the other experiment or, you may want to copy data to the current experiment from the other experiment.

There is a potentially serious problem that occurs if two experiments share a file. The file can not be in two places at one time. Thus, it will be stored with the experiment that created it but separate from the other. The problem is that, if you move or rename files or folders, the second experiment will be unable to find the binary file.

Here is an example of how this problem can bite you.

Imagine that you create an experiment at work and save it as an unpacked experiment file on your hard disk. Let’s call this “experiment A”. The waves for experiment A are stored in individual Igor Binary files in the experiment folder.

Now you create a new experiment. Let’s call this “experiment B”. You use the Load Igor Binary routine to load a wave from experiment A into experiment B. You elect to share the wave. You save experiment B on your hard disk. Experiment B now contains a reference to a file in experiment A’s home folder.

Now you decide to take experiment B to another computer. You copy it to a CD and go to the other computer. When you try to open experiment B, Igor can’t find the file it needs to load the shared wave. This file is back on the hard disk of the original computer.

A similar problem occurs if, instead of moving experiment B to another computer, you change the name or location of experiment A’s folder. Experiment B will still be looking for the shared file under its old name or in its old location and Igor will not be able to load the file when you open experiment B.

Because of this problem, we recommend that you avoid file sharing as much as possible. If it is necessary to share a binary file, you will need to be very careful to avoid the situation described above.

The Data Browser always copies when transferring data from disk into memory.

For more information on the problem of sharing files, see References to Files and Folders on page II-37.

Loading Image Files

You can load PICT, TIFF, JPEG, PNG, GIF, Photoshop, SGI, Sun Raster, BMP, and Targa image files into Igor Pro using the Load Image dialog. The same file types are supported both on the Macintosh and on Windows.

Loading the following types requires that you have Apple’s QuickTime software installed on your computer: PICT (on Windows only), JPEG, GIF, PhotoShop, SGI and Targa. All Mac OS X machines have QuickTime installed. Windows users who want to load these types of files can download QuickTime from <http://www.apple.com/quicktime/>.

The Load Image Dialog

To load an image file into an Igor matrix wave, invoke the Load Image dialog by choosing the Load Image menu item in the Load Waves submenu.
Chapter II-9 — Importing and Exporting Data

This dialog looks and works much the same as the other Igor file loading dialogs.

When you choose a particular type of image file from the File Type pop-up menu, you are setting a file filter that is used when displaying the image file selection dialog. If you are not sure that your image file has the correct file type or file name extension, choose “Any” from the File pop-up menu so that the filter does not restrict your selection. Note that when you choose “Any” QuickTime will be used to load the file and therefore you can only load images from file formats supported by QuickTime.

Names for the loaded matrix waves can be the name of the file or a name that you specify. If you enter a matrix wave name in the dialog that conflicts with an existing wave name and you have not selected the Overwrite Existing Waves checkbox, Igor will append a numeric suffix to the new wave names.

Image Loading Details

Except for certain kinds of TIFF and Sun Raster files, images are loaded into a 3D RGB, RGBA, or CMYK wave. See the ImageLoad operation (see page V-301) for further details.

The wave is of type unsigned byte with layer 0 containing the red channel, layer 1 the green channel and layer 2 the blue channel. The wave may contain four layers if you load a CMYK image or if you load an image that has an alpha channel in addition to the RGB information. Grayscale TIFF and Sun Raster images are loaded as 2D waves. If you load a TIFF or Sun Raster image that contains a colormap, Igor creates (in addition to the image wave) a colormap wave (usually with the suffix “_CMap”). You can display images using the NewImage command or convert image waves into other forms using the ImageTransform operation.

There are two menu choices for the PNG format: Raw PNG and PNG. When Raw PNG is selected, the data is read directly from the file into the wave. When PNG is selected, the file is loaded into memory, and off-screen image is created, and the wave data is set by reading the offscreen image. In nearly all cases, you should choose Raw PNG.

When you choose TIFF from the File Type pop-up menu, an additional checkbox appears: Load Multiple Images From File. If your TIFF file contains a stack of images, select this checkbox. You can then set the number of the first image to load (zero-based) and the number of images to load from the TIFF stack.
Chapter II-9 — Importing and Exporting Data

If just one image is loaded from the TIFF file then Igor creates a single 2D wave. If more than one image is loaded, Igor creates a single 3D wave, each layer of which contains the data from one of the images in the stacked TIFF file. Reading a TIFF image stack into a single 3D wave is supported only for images that are 8, 16 or 32-bit/pixel deep.

You can convert a number of 2D image waves into a 3D stack using the ImageTransform operation (stackImages keyword).

HDF images can be loaded only by the HDF or HDF5 XOPs, see Loading HDF Data on page II-169 for further details.

Loading Other Files

WaveMetrics provides a number of extensions that add additional file-loading capabilities to Igor. Most of these file loaders add a menu item to the Load Waves submenu and an entry in the Open or Load File Dialog’s list so you can use it interactively. They also usually add a command line operation so you can use them from an Igor procedure.

The following table lists many of the file loaders included with Igor Pro. Some more obscure file loaders are also available.

<table>
<thead>
<tr>
<th>File Loader/Writer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBLoadWave</td>
<td>Loads numeric data from “general binary” files. The XOP can load 8, 16 and 32 bit integer data and 32 and 64 bit IEEE data from a binary file. It can also load a subset of the file. It can handle numerous kinds of files including interleaved and byte-swapped files. You must know the format of the binary file precisely.</td>
</tr>
<tr>
<td>GISLoadWave</td>
<td>Loads Digital Elevation Model (DEM) and Digital Line Graph (DLG) data for standard U.S. Geological Survey (USGS) format quadrangles. Such geographic data are the basic elements of digital mapping. See Loading GIS Data on page II-169.</td>
</tr>
<tr>
<td>GWLoadWave</td>
<td>Loads an old Macintosh-only file format from GW Instruments.</td>
</tr>
<tr>
<td>HDF Loader</td>
<td>Loads HDF (Hierarchical Data Format) version 4 and earlier files. See Loading HDF Data on page II-169.</td>
</tr>
<tr>
<td>HDF5 XOP</td>
<td>Loads HDF version 5 files. See Loading HDF Data on page II-169.</td>
</tr>
<tr>
<td>IgorGIS</td>
<td>Reads and writes various GIS files including shapefiles, GeoTIFF and many others. It also supports transformations between spatial reference systems and creates underlay images from vector data for use in fills. See Loading GIS Data on page II-169.</td>
</tr>
<tr>
<td>JCAMPLoadWave</td>
<td>Loads JCAMP files, used in spectroscopy.</td>
</tr>
<tr>
<td>LoadWAVfile</td>
<td>Windows only. Adds operations to load and save WAV sound files.</td>
</tr>
<tr>
<td>MLLoadWave</td>
<td>Loads data from Matlab binary files. WaveMetrics thanks Yves Peysson and Bernard Saoutic for this file loader.</td>
</tr>
<tr>
<td>NILoadWave</td>
<td>Loads numeric data from files produced by a number of scientific instruments from Nicolet Instruments.</td>
</tr>
<tr>
<td>SndLoadSaveWave</td>
<td>Loads a variety of sound files on Macintosh and Windows. See Loading Sound Files on page II-168.</td>
</tr>
<tr>
<td>TDM XOP</td>
<td>Loads data from National Instruments TDM files.</td>
</tr>
<tr>
<td>XLLoadWave</td>
<td>Loads numeric and text data from an Excel spreadsheet file. You need to know the cells containing the numeric data, for example, B10 - D25.</td>
</tr>
</tbody>
</table>

If you are a C programmer, you can write your own extension to load data into Igor. To do this you need the Igor External Operations Toolkit, available from WaveMetrics.
Chapter II-9 — Importing and Exporting Data

The Igor installer puts file loaders and other extensions in "Igor Pro Folder/Igor Extensions" and "Igor Pro Folder/More Extensions". To use an extension, put an alias (Macintosh) or shortcut (Windows) for it in "Igor Pro User Files/Igor Extensions" (see Igor Pro User Files on page II-46 for details) and then relaunch Igor.

Each file loader has an associated Igor help file. The help file provides all the information you need to use the file loader.

**Loading Row-Oriented Text Data**

All of the built-in text file loaders are column-oriented — they load the columns of data in the file into 1D waves. There is a row-oriented format that is fairly common. In this format, the file represents data for one wave but is written in multiple columns. Here is an example:

```
350  2.97  1.95  1.00  8.10  2.42
351  3.09  4.08  1.90  7.53  4.87
352  3.18  5.91  1.04  6.90  1.77
```

In this example, the first column contains X values and the remaining columns contain data values, written in row/column order.

Igor Pro does not have a file-loader extension to handle this format, but there is a WaveMetrics procedure file for it. To use it, use the Load Row Data procedure file in the “WaveMetrics Procedures:File Input Output” folder. It adds a Load Row Data item to the Macros menu. When you choose this item, Igor will present a dialog that presents several options. One of the options treats the first column as X values or as data. If you specify treating the column as X values, Igor will use it to determine the X scaling of the output wave, assuming that the values in the first column are evenly spaced. This is usually the case.

**Loading Sound Files**

The SndLoadSaveWave XOP loads a variety of sound files on Macintosh and Windows. It adds the SndLoadWave, SndSaveAIFF and SndSaveWAV operations.

On Windows you must install QuickTime to use the SndLoadSaveWave XOP.

See the SndLoadSaveWave help file in the More Extensions:File Loaders folder for details.

See Sound on page IV-224 for general information on Igor’s sound-related features.
Chapter II-9 — Importing and Exporting Data

Loading HDF Data

HDF stands for “Hierarchical Data Format”. HDF is a complex and powerful format and you will need to understand it as well as the structure of your HDF files to conveniently use it. Information on HDF is available via the World Wide Web from:

<http://www.hdfgroup.org/>

The current version of HDF is HDF5. Igor Pro includes an HDF5XOP that can read and write HDF5 files. HDF5XOP is documented in the “HDF5 Help.ihf” file in “Igor Pro Folder:More Extensions:File Loaders”. An HDF5 browser based on HDFXOP is also provided and documented in the help file.

Igor Pro also includes an older XOP that supports HDF version 3 and version 4 files. This HDF Loader XOP is documented in “HDF Loader Help.ihf” file in the same folder.

Loading GIS Data

GIS stands for “Geographic information system”.

The IgorGIS package reads and writes various GIS files including shapefiles, GeoTIFF and many others. It also supports transformations between spatial reference systems and creating underlay images from vector data for use in fills. For details see the “IgorGIS Help.ihf” file.

The GISLoadWave XOP loads data from USGS Digital Elevation Model (DEM) and Digital Line Graph (DLG) files into Igor waves. The GISLoadWave XOP is used with the GIS Utilities package. For details choose File→Example Experiments→Feature Demos→GIS Utilities Demo.

Loading Very Big Binary Files

Binary data files can be loaded using the GBLoadWave operation or the FBInRead operation.

Most binary data files are not so large as to present issues for Igor. However, if your data file approaches hundreds of millions or billions of bytes, size and memory issues arise.

GBLoadWave and FBInRead can handle very large files, up to hundreds of trillions of bytes, in theory. However, other constraints put a limit on the amount of data you can load into Igor.

First there is the maximum amount of virtual memory that Igor can handle on your machine - between 2 and 4 GB. For details, see Memory Management on page III-427.

Even if you maximize the amount of virtual memory accessible by Igor, you still need as much physical memory as possible to avoid slowing your computer to a crawl. 2 GB is good, 4 GB is better, if your computer supports it.

Even if Igor can theoretically address 4 GB, this does not mean that you can create a 4 GB wave. You are further limited by memory fragmentation, also discussed under Memory Management on page III-427.

If you want GBLoadWave or FBInRead to convert the type of the data, for example from 16-bit signed to 32-bit floating point, this requires an extra buffer during the load process which takes more memory.

Furthermore, Igor itself currently cannot create a wave with more than 2 billion points because of the use of signed longs throughout the program.

Finally, there is very little that you can do in Igor with a 1 billion point wave that won’t take forever. Consequently you need to load your data a piece at a time using the GBLoadWave or FBInRead.

Some experimentation will be necessary to determine how to deal with very large files. It is a good idea to start with a reasonably-sized chunk of data, say 100 million bytes.
Loading Waves Using Igor Procedures

One of Igor’s strong points is that it you can write procedures to automatically load, process and graph data. This is useful if you have accumulated a large number of data files with identical or similar structures or if your work generates such files on a regular basis.

The input to the procedures is one or more data files. The output might be a printout of a graph or page layout or a text file of computed results.

Each person will need procedures customized to his or her situation. In this section, we present some examples that might serve as a starting point.

Variables Set by the LoadWave Operation

The LoadWave operation uses the numeric variable V_flag and the string variables S_fileName, S_path, and S_waveNames to provide information that is useful for procedures that automatically load waves. When used in a function, the LoadWave operation creates these as local variables.

LoadWave sets the string variable S_fileName to the name of the file being loaded. This is useful for annotating graphs or page layouts.

LoadWave sets the string variable S_path to the full path to the folder containing the file that was loaded. This is useful if you need to load a second file from the same folder as the first.

LoadWave sets the variable V_flag to the number of waves loaded. This allows a procedure to process the waves without knowing in advance how many waves are in a file.

LoadWave also sets the string variable S_waveNames to a semicolon-separated list of the names of the loaded waves. From a procedure, you can use the names in this list for subsequent processing.

Loading and Graphing Waveform Data

Here is a very simple example designed to show the basic form of an Igor function for automatically loading and graphing the contents of a data file. It loads a delimited text file containing waveform data and then makes a graph of the waves.

In this function, we make the assumption that the files that we are loading contain three columns of waveform data. Tailoring the function for a specific type of data file allows us to keep it very simple.

Function LoadAndGraph(fileName, pathName)

String fileName // Name of file to load or "" to get dialog
String pathName // Name of path or "" to get dialog

// Load the waves and set the local variables.
LoadWave/J/D/O/P=$pathName fileName
if (V_flag==0) // No waves loaded. Perhaps user canceled.
    return -1
endif

// Put the names of the three waves into string variables
String s0, s1, s2
s0 = StringFromList(0, S_waveNames)
s1 = StringFromList(1, S_waveNames)
s2 = StringFromList(2, S_waveNames)

Wave w0 = $s0 // Create wave references.
Wave w1 = $s1
Wave w2 = $s2

// Set waves' X scaling, X units and data units
SetScale/P x, 0, 1, "s", w0, w1, w2
SetScale d 0, 0, "V", w0, w1, w2
Display w0, w1, w2 // Create a new graph

// Annotate graph
Textbox/N=TBFileName/A=LT "Waves loaded from " + S_fileName
return 0 // Signifies success.

s0, s1 and s2 are local string variables into which we place the names of the loaded waves. We then use the $ operator to create a reference to each wave, which we can use in subsequent commands.

Once the function is entered in the procedure window, you can execute it from the command line or call it from another function. If you execute

LoadAndGraph("", ")

the LoadWave operation will display a dialog allowing you to choose a file. If you call LoadAndGraph with the appropriate parameters, LoadWave will load the file without presenting a dialog.

You can add a “Load And Graph” menu item by putting the following menu declaration in the procedure window:

Menu "Macros"
  "Load And Graph...", LoadAndGraph("", ")
End

Because we have not used the “Auto name & go” option for the LoadWave operation, LoadWave will put up another dialog in which you can enter names for the new waves. If you want the macro to be more automatic, use /A or /N to turn “Auto name & go” on. If you want the procedure to specify the names of the loaded waves, use the /B flag. See the description of the LoadWave operation (see page V-393) for details.

To keep the function simple, we have hard-coded the X scaling, X units and data units for the new waves. You would need to change the parameters to the SetScale operation to suit your data. For more flexibility, you would add additional parameters to the function.

It is possible to write LoadAndGraph so that it can handle files with any number of columns. This makes the function more complex but more general.

For more advanced programmers, here is the more general version of LoadAndGraph.

Function LoadAndGraph(fileName, pathName)
    String fileName // Name of file to load or "" to get dialog
    String pathName // Name of path or "" to get dialog

    // Load the waves and set the variables.
    LoadWave/J/D/O/P=$pathName fileName
    if (V_flag==0) // No waves loaded. Perhaps user canceled.
        return -1
    endif
    Display // Create a new graph

    String theWave
    Variable index=0
    do // Now append waves to graph
      theWave = StringFromList(index, S_waveNames) // Next wave
      if (strlen(theWave) == 0) // No more waves?
        break // Break out of loop
      endif
      Wave w = $theWave
      SetScale/P x, 0, 1, "s", w // Set X scaling
      SetScale d 0, 0, "V", w // Set data units
      AppendToGraph w
The do-loop picks each successive name out of the list of names in S_waveNames and adds the corresponding wave to the graph. S_waveNames will contain one name for each column loaded from the file.

There is one serious shortcoming to the LoadAndGraph function. It creates a very plain, default graph. There are three approaches to overcoming this problem:

- Use preferences.
- Use a style macro.
- Overwrite data in an existing graph.

Normally, Igor does not use preferences when a procedure is executing. To get preferences to take effect during the LoadAndGraph function, you would need to put the statement “Preferences 1” near the beginning of the function. This turns preferences on just for the duration of the function. This will cause the Display and AppendToGraph operations to use your graph preferences.

Using preferences in a function means that the output of the function will change if you change your preferences. It also means that if you give your function to a colleague, it will produce different results. This dependence on preferences can be seen as a feature or as a problem, depending on what you are trying to achieve. We normally prefer to keep procedures independent of preferences.

Using a style macro is a more robust technique. To do this, you would first create a prototype graph and create a style macro for the graph (see Graph Style Macros on page II-307). Then, you would put a call to the style macro at the end of the LoadAndGraph macro. The style macro would apply its styles to the new graph.

The last approach is to overwrite data in an existing graph rather than creating a new one. The simplest way to do this is to always use the same names for your waves. For example, imagine that you load a file with three waves and you name them wave0, wave1, wave2. Now you make a graph of the waves and set everything in the graph to your taste. You now load another file, use the same names and use LoadWave’s overwrite option. The data from the new file will replace the data in your existing waves and Igor will automatically update the existing graph. Using this approach, the function simplifies to this:

```plaintext
Function LoadAndGraph(fileName, pathName)
    String fileName // Name of file to load or "" to get dialog
    String pathName // Name of path or "" to get dialog

    // load the waves, overwriting existing waves
    LoadWave/J/D/O/N/P=$pathName fileName
    if (V_flag==0) // No waves loaded. Perhaps user canceled.
        return -1
    endif

    Textbox/C/N=TBFileName/A=LT "Waves loaded from " + S_fileName
    return 0     // Signifies success.
End
```

There is one subtle change here. We have used the /N option with the LoadWave operation, which autonames the incoming waves using the names wave0, wave1, and wave2.

You can see that this approach is about as simple as it can get. The downside is that you wind up with uninformative names like wave0. You can use the LoadWave /B flag to provide better names.
Chapter II-9 — Importing and Exporting Data

If you are loading data from Igor Binary files or from packed Igor experiments, you can use the LoadData operation instead of LoadWave. This is a powerful operation, especially if you have multiple sets of identically structured data, as would be produced by multiple runs of an experiment. See The LoadData Operation on page II-164 above.

Loading and Graphing XY Data

In the preceding example, we treated all of the columns in the file the same: as waveforms. If you have XY data then things change a bit. We need to make some more assumptions about the columns in the file. For example, we might have a collection of files with four columns which represent two XY pairs. The first two columns are the first XY pair and the second two columns are the second XY pair.

Here is a modified version of our function to handle this case.

Function LoadAndGraphXY(fileName, pathName)
    String fileName // Name of file to load or "" to get dialog
    String pathName // Name of path or "" to get dialog

    // load the waves and set the globals
    LoadWave/J/D/O/P=$pathName fileName
    if (V_flag==0) // No waves loaded. Perhaps user canceled.
        return -1
    endif

    // Put the names of the waves into string variables.
    String sx0, sy0, sx1, sy1
    sx0 = StringFromList(0, S_waveNames)
    sy0 = StringFromList(1, S_waveNames)
    sx1 = StringFromList(2, S_waveNames)
    sy1 = StringFromList(3, S_waveNames)

    Wave x0 = $sx0 // Create wave references.
    Wave y0 = $sy0
    Wave x1 = $sx1
    Wave y1 = $sy1

    SetScale d 0, 0, "s", x0, x1 // Set wave data units
    SetScale d 0, 0, "V", y0, y1

    Display y0 vs x0 // Create a new graph
    AppendToGraph y1 vs x1

    Textbox/A=LT "Waves loaded from " + S_fileName // Annotate graph
    return 0 // Signifies success.
End

The main difference between this and the waveform-based LoadAndGraph function is that here we append waves to the graph as XY pairs. Also, we don’t set the X scaling of the waves because we are treating them as XY pairs, not as waveforms.

It is possible to write a more general function that can handle any number of XY pairs. Once again, adding generality adds complexity. Here is the more general version of the function.

Function LoadAndGraphXY(fileName, pathName)
    String fileName // Name of file to load or "" to get dialog
    String pathName // Name of path or "" to get dialog

    // Load the waves and set the globals
    LoadWave/J/D/O/P=$pathName fileName
    if (V_flag==0) // No waves loaded. Perhaps user canceled.
        return -1
    endif

    // Put the names of the waves into string variables.
    String sx0, sy0, sx1, sy1
    sx0 = StringFromList(0, S_waveNames)
    sy0 = StringFromList(1, S_waveNames)
    sx1 = StringFromList(2, S_waveNames)
    sy1 = StringFromList(3, S_waveNames)

    Wave x0 = $sx0 // Create wave references.
    Wave y0 = $sy0
    Wave x1 = $sx1
    Wave y1 = $sy1

    SetScale d 0, 0, "s", x0, x1 // Set wave data units
    SetScale d 0, 0, "V", y0, y1

    Display y0 vs x0 // Create a new graph
    AppendToGraph y1 vs x1

    Textbox/A=LT "Waves loaded from " + S_fileName // Annotate graph
    return 0 // Signifies success.
End
Display // Create a new graph

String sxw, syw
Variable index=0
do // Now append waves to graph
  sxw=StringFromList(index, S_waveNames) // Next name
  if (strlen(sxw) == 0) // No more?
    break // break out of loop
  endif
  syw=StringFromList(index+1, S_waveNames) // Next name
  Wave xw = $sxw // Create wave references.
  Wave yw = $syw
  SetScale d 0, 0, "s", xw // Set x wave's units
  SetScale d 0, 0, "V", yw // Set y wave's units
  AppendToGraph yw vs xw
  index += 2
while (1) // Unconditionally loop back up to “do”

// Annotate graph
Textbox/A=LT "Waves loaded from " + S_fileName
return 0 // Signifies success.
End

Loading All of the Files in a Folder

In the next example, we assume that we have a folder containing a number of files. Each file contains three columns of waveform data. We want to load each file in the folder, make a graph and print it. This example uses the LoadAndGraph function as a subroutine.

Function LoadAndGraphAll(pathName)
  String pathName // Name of symbolic path or "" to get dialog
  String fileName
  String graphName
  Variable index=0
  
  if (strlen(pathName)==0) // If no path specified, create one
    NewPath/O temporaryPath // This will put up a dialog
    if (V_flag != 0)
      return -1 // User cancelled
    endif
    pathName = "temporaryPath"
  endif
  
  Variable result
  do // Loop through each file in folder
    fileName = IndexedFile($pathName, index, ".dat")
    if (strlen(fileName) == 0) // No more files?
      break // Break out of loop
    endif
    result = LoadAndGraph(fileName, pathName)
    if (result == 0) // Did LoadAndGraph succeed?
      // Print the graph.
      graphName = WinName(0, 1) // Get the name of the top graph
      String cmd
      sprintf cmd, "PrintGraphs $s", graphName
      Execute cmd // Explained below.
      DoWindow/K $graphName // Kill the graph
      KillWaves/A/Ż // Kill all unused waves
    endif
  endwhile
Chapter II-9 — Importing and Exporting Data

```plaintext
index += 1
while (1)
  if (Exists("temporaryPath")) // Kill temp path if it exists
    KillPath temporaryPath
  endif
  return 0 // Signifies success.
End
```

This function relies on the IndexedFile function to find the name of successive files of a particular type in a particular folder. The last parameter to IndexedFile says that we are looking for files with a “.dat” extension. On Macintosh, if we changed the last parameter to “TEXT”, IndexedFile would return all files of type TEXT, regardless of their extension.

Once we get the file name, we pass it to the LoadAndGraph function. After printing the graph, we kill it and then kill all the waves in the current data folder so that we can start fresh with the next file. A more sophisticated version would kill only those waves in the graph.

To print the graphs, we use the PrintGraphs operation. PrintGraphs is one of a few built-in operations that can not be directly used in a function. Therefore, we put the PrintGraphs command in a string variable and call Execute to execute it.

If you are loading data from Igor Binary files or from packed Igor experiments, you can use the LoadData operation. See The LoadData Operation on page II-164 above.

### Saving Waves

Igor automatically saves the waves in the current experiment on disk when you save the experiment. Many Igor users load data from files into Igor and then make and print graphs or layouts. This is the end of the process. They have no need to explicitly save waves.

You can save waves in an Igor packed experiment file for archiving using the SaveData operation or using the Save Copy button in the Data Browser. The data in the packed experiment can then be reloaded into Igor using the LoadData operation or the Load Expt button in Data Browser. Or you can load the file as an experiment using File → Open Experiment. See the SaveData operation on page V-607 for details.

The main reason for saving a wave separate from its experiment is to export data from Igor to another program. To explicitly save waves to disk, you would use Igor’s Save operation.

The following table lists the four types of built-in data saving routines in Igor and their salient features.

<table>
<thead>
<tr>
<th>File type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Delimited text | Used for archiving results or for exporting to another program.  
                | Row Format: `<data><tab><data><terminator>`  
                | Contains one block of data with any number of rows and columns. A row of column labels is optional.  
                | Columns may be equal or unequal in length.  
                | Can export 1D or 2D waves.                      |
| General text   | Used for archiving results or for exporting to another program.  
                | Row Format: `<number><tab><number><terminator>`  
                | Contains one or more blocks of numbers with any number of rows and columns. A row of column labels is optional.  
                | Columns in a block must be equal in length.  
                | Can export 1D or 2D waves.                      |
You can access all of the built-in routines via the Save Waves submenu of the Data menu.

### File type Description

<table>
<thead>
<tr>
<th>Igor Text</th>
<th>Used for archiving waves or for exporting waves from one Igor experiment to another. Format: See Igor Text File Format on page II-159 above. Contains one or more wave blocks with any number of waves and rows. A given block can contain either numeric or text data. Consists of special Igor keywords, numbers and Igor commands. Can export waves of dimension 1 through 4.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Igor Binary</td>
<td>Used for exporting waves from one Igor experiment to another. Contains data for one Igor wave. Format: See Igor Technical Note #003, “Igor Binary Format”.</td>
</tr>
</tbody>
</table>

* `<terminator>` can be carriage return, linefeed or carriage return/linefeed. You would use carriage return for exporting to a Macintosh program, carriage return/linefeed for Windows systems, and linefeed for Unix systems.

You can access all of the built-in routines via the Save Waves submenu of the Data menu.

### Saving Waves in a Delimited Text File

To save a delimited text file, invoke the Save Delimited Text dialog via the Save Waves submenu of the Data menu.

The Save Delimited Text routine writes a file consisting of numbers separated by tabs with a selectable line terminator at the end of each line of text. When writing 1D waves, it can optionally include a row of column names.
labels. When writing a matrix, it can optionally write row labels as well as column labels plus row and column position information.

Save Delimited Text can save waves of any dimensionality. Multidimensional waves are saved one wave per block. Data is written in row/column/layer/chunk order. Multidimensional waves saved as delimited text can not be loaded back into Igor as delimited text because the Load Delimited Text routine does not support multiple blocks. They can be loaded back in as general text. However, for data that is intended to be loaded back into Igor later, the Igor Text, Igor Binary or Igor Packed Experiment formats are preferable.

The order of the columns in the file depends on the order in which the wave names appear in the Save command. This dialog generates the wave names based on the order of the waves in the dialog list which in turn depends on the order in which the waves were created. If you want to change the order then you should click the To Cmd Line button instead of the Do It button and edit the command in Igor’s command line.

By default, the Save operation writes numeric data using the “%.15g” format for double-precision data and “%.7g” format for data with less precision. These formats give you up to 15 or 7 digits of precision in the file.

To use different numeric formatting, create a table of the data that you want to export. Set the numeric formatting of the table columns as desired. Be sure to display enough digits in the table because the data will be written to the file as it appears in the table. In the Save Delimited Text dialog, select the “Use table formatting” checkbox. When saving a multi-column wave (1D complex wave or multi-dimensional wave), all columns of the wave are saved using the table format for the first table column from the wave.

The wfPrintf command line operation can also be used to save waves to text files using a specific numeric format.

The Save operation is capable of appending to an existing file, rather than overwriting the file. This is useful for accumulating results of a analysis that you perform regularly in a single file. You can also use this to append a block of numbers to a file containing header information that you generated with the fPrintf operation. The append option is not available through the dialog. If you want to do this, see the discussion of the Save operation (see page V-604).

**Saving Waves in a General Text File**

Saving waves in a general text file is very similar to saving a delimited text file. The Save General Text dialog is identical to the Save Delimited Text dialog.

All of the columns in a single block of a general text file must have the same length. The Save General Text routine writes as many blocks as necessary to save all of the specified waves. For example, if you ask it to save two 1D waves with 100 points and two 1D waves with 50 points, it will write two blocks of data. Multidimensional waves are written one wave per block.

**Saving Waves in an Igor Text File**

Saving waves in an Igor Text file is also very similar to saving a delimited text file. The Save Igor Text dialog is identical to the Save Delimited Text dialog.

The Igor Text format is capable of saving not only the numeric contents of a wave but its other properties as well. It saves each wave’s dimension scaling, units and labels, data full scale and units and the wave’s note, if any. All of this data is saved more efficiently as binary data when you save as an Igor packed experiment using the SaveData operation.

As in the general text format, all of the columns in a single block of an Igor Text file must have the same length. The Save Igor Text routine handles this requirement by writing as many blocks as necessary.

Save Igor Text can save waves of any dimensionality. Multidimensional waves are saved one wave per block. The /N flag at the start of the block identifies the dimensionality of the wave. Data is written in row/column/layer/chunk order.
Chapter II-9 — Importing and Exporting Data

Saving Waves in Igor Binary Files
Igor’s Save Igor Binary routine saves waves in Igor Binary files, one wave per file. Most users will not need to do this since Igor automatically saves waves when you save an Igor experiment. You might want to save a wave in an Igor Binary file to send it to a colleague.

The Save Igor Binary dialog is similar to the Save Delimited Text dialog. There is a difference in file naming since, in the case of Igor Binary, each wave is saved in a separate file. If you select a single wave from the dialog’s list, you can enter a name for the file. However, if you select multiple waves, you can not enter a file name. Igor will use default file names of the form “wave0.ibw”.

When you save an experiment in a packed experiment file, all of the waves are saved in Igor Binary format. The waves can then be loaded into another Igor experiment using the Data Browser (see page II-130) or The LoadData Operation (see page II-164).

Saving Waves in Image Files
You can save some types of multidimensional waves as image files. The two main limitations of image files are that they usually support only 8 bit depth and that some formats (e.g., JPEG) rely on lossy compression. To avoid compression loss you should choose either TIFF or PNG file formats. At present, the extended TIFF file format is a bit more flexible in that you can save in 8, 16, or 32 bits per sample and you can use image stacks to support 3D and 4D waves. See the ImageSave operation on page V-314 for more details.

Saving Sound Files
You can save waves as sound files using the SndLoadSaveWave XOP. See the corresponding help file in the More Extensions:File Loaders folder.

Exporting Text Waves
Igor does not quote text when exporting text waves as a delimited or general text file. It does quote text when exporting it as an Igor Text file.

Certain special characters, such as tabs, carriage returns and linefeeds, cause problems during exchange of data between programs because most programs consider them to separate one value from the next or one line of text from the next. Igor Text waves can contain any character, including special characters. In most cases, this will not be a problem because you will have no need to store special characters in text waves or, if you do, you will have no need to export them to other programs.

When Igor writes a text file containing text waves, it replaces the following characters, when they occur within a wave, with their associated escape codes:

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
<th>ASCII Code</th>
<th>Escape Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>carriage return</td>
<td>13</td>
<td>\r</td>
</tr>
<tr>
<td>LF</td>
<td>linefeed</td>
<td>10</td>
<td>\n</td>
</tr>
<tr>
<td>tab</td>
<td>tab</td>
<td>9</td>
<td>\t</td>
</tr>
<tr>
<td>\</td>
<td>backlash</td>
<td>92</td>
<td>\ \</td>
</tr>
</tbody>
</table>

Igor does this because these would be misinterpreted if not changed to escape sequences. When Igor loads a text file into text waves, it reverses the process, converting escape sequences into the associated ASCII code.

This use of escape codes can be suppressed using the /E flag of the Save operation (see page V-604). This is necessary to export text containing backslashes to a program that does not interpret escape codes.
Exporting MultiDimensional Waves

When exporting a multidimensional wave as a delimited or general text file, you have the option of writing row labels, row positions, column labels and column positions to the file. Each of these options is controlled by a checkbox in the Save Waves dialog. There is a discussion of row/column labels and positions under 2D Label and Position Details on page II-149.

Igor writes multidimensional waves in column/row/layer/chunk order.

Accessing SQL Databases

Igor Pro includes an XOP, called SQL XOP, which provides access to relational databases from IGOR procedures. It uses ODBC (Open Database Connectivity) libraries and drivers on Mac OS X and Windows to provide this access.

For details on configuring and using SQL XOP, open the SQL Help file in “Igor Pro Folder:More Extensions:Utilities”.