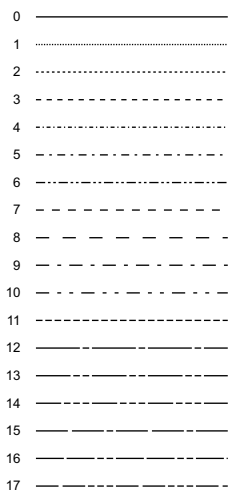
Chapter III-17

Miscellany

Dashed Lines

You can display traces in graphs, as well as drawn lines, rectangles, ovals, and polygons, using various line styles. This table, generated from the ColorsMarkerSLinesPatterns.pxp example experiment, shows Igor's default line styles:



It is usually not necessary, but you can edit the built-in dashed line styles using the Dashed Lines dialog by choosing Misc→Dashed Lines.

Dashed line 0 (the solid line) cannot be edited. If you need to create a custom dashed line pattern, we recommend that you modify the high numbered dashed lines, leaving the low number ones in their default state. This ensures that the low numbered patterns will be the same for everyone.

You can also change dashed lines with the **SetDashPattern** operation.

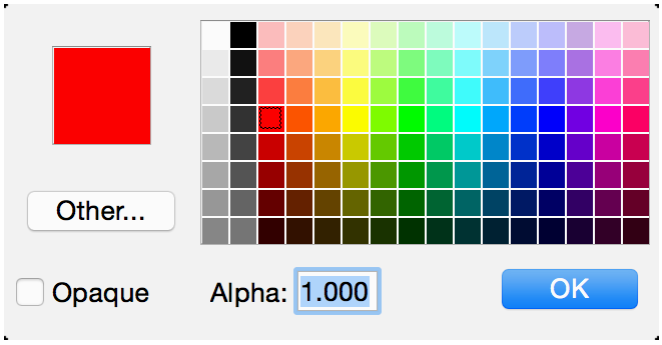
Dashed lines are stored with the experiment, so each experiment can have different dashed lines. You can capture the current dashed lines as the preferred dashed lines for new experiments.

The Color Environment

Igor has a main color palette that contains colors that you can use for traces in graphs, text, axes, backgrounds and so on. The main color palette appears as a pop-up menu in a number of dialogs, such as the Modify Trace Appearance dialog. This section discusses this palette.

Igor also has color tables and color index waves you can select among when displaying contour plots and images. These are discussed in Chapter II-14, **Contour Plots**, and Chapter II-15, **Image Plots**.

You can select a color from the colors presented in a color palette:



You can use the Other button to select colors that are not in the palette. As you use Igor, colors are added to the palette in the Recent Colors area.

The recent colors are remembered by Igor when it quits and restored when it restarts if you have selected the "Save and restore color palette's recent colors" checkbox in the Miscellaneous Settings dialog's Color Settings category.

Color Blending

In places where you can supply a color in RGB format, you can optionally provide a fourth parameter called "alpha". Alpha, like the R, G and B values, can range from 65535 (100% opaque) down to 0 (100% transparent). Intermediate values provide translucency. For example:

```
Make jack=sin(x/8),sam=cos(x/6); Display jack, sam
ModifyGraph mode=7,hbFill=2
```

```
ModifyGraph rgb(jack)=(65535,32768,32768)
ModifyGraph rgb(sam)=(0,0,65535,40000)       // Translucent
```

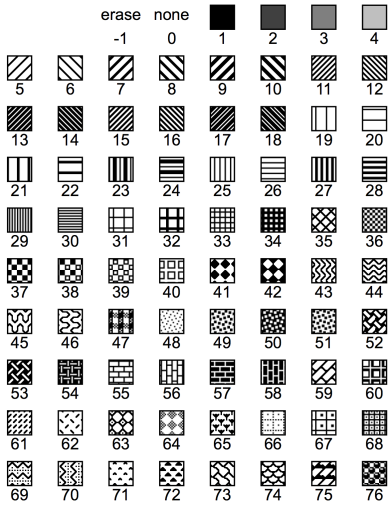
Color blending was added in Igor Pro 7.

Color blending does not work on Windows in the old GDI graphics mode explained under **Graphics Technology on Windows** on page III-445.

Color blending does not work in graphics exported as EPS. Use PDF instead.

Fill Patterns

Fill patterns can be used in graphs and drawing layers. This table, generated from the ColorsMarkersLinesPatterns.pxp example experiment, shows the available fill patterns:



Gradient Fills

As of Igor Pro 7, you can specify gradient fills for drawing elements, graph trace fills, and graph window and plot area background fills. A gradient fill is a gradual change of color.

Programmatically you specify gradients using two keywords, gradient and gradientExtra, when calling the ModifyGraph, ModifyLayout or SetDrawEnv operations.

The syntax for the gradient keyword has two forms. The first form provides overall control of the gradient while the second form controls the color details.

The first form for the gradient keyword is:

```
gradient = 0, 1, 2, or 3
```

0 deletes the gradient entirely.

1 turns on an existing gradient or creates a new one with default values.

2 turns off a gradient but keeps the settings.

3 is used in combination with the ModifyLayout operation and signals that a particular page should not display a gradient even when a layout global gradient is set.

The second form for the gradient keyword is:

```
gradient = {type, x0, y0, x1, y1 [,color0 [, color1 ] ]}
```

type is a bitfield.

Bits 0 through 3 specify the gradient mode. 0 is linear, 1 is radial and other values are reserved.

Bits 4 through 7 specify the coordinate mode. 0 is object rect, 1 is window rect.

To construct a type parameter signifying a radial gradient in window rect mode you could write:

```
Variable gradientType = 1 | (1*16)
```

The following commands illustrate the difference between object rect and window rect mode. Execute these commands and then drag the rectangles around:

```
Display /W=(150,50,474,365)
SetDrawLayer UserFront
SetDrawEnv xcoord=abs,ycoord=abs
SetDrawEnv save
SetDrawEnv fillfgc=(16385,16388,65535),fillbgc=(65535,16385,16385)
SetDrawEnv gradient={16, 0, 0, 1, 1}      // Window rect mode
DrawRect 25,38,95,138
SetDrawEnv gradient={1, 0.5, 0.5, 0, 1, (0,65535,0), (65535,0,65535)}
DrawRect 130,150,273,289
ShowTools/A
```

The *x0, y0, x1,* and *y1* parameters specify normalized locations that define the gradient. (0,0) is the upper left corner of the bounding rectangle while (1,1) is the lower right corner. For a linear gradient, (*x0, y0*) defines the start while (*x1, y1*) defines the end. Only the slope of the line is used, not the actual extent. For a radial gradient, (*x0, y0*) defines the center of a bounding circle while (*x1, y1*) is not used at present.

The optional color0 and color1 keywords are specified as (r,g,b) or (r,g,b,a) but use of alpha is not fully supported on all platforms (Quartz PDF on Macintosh for example). When omitted or with a value of (0,0,0) the actual color used is a default for the given circumstance. The default values are specified in the individual operation documentation.

The gradientExtra keyword adds an additional color change point. You can add as many change points as desired.The syntax for the gradientExtra keyword is:

```
gradientExtra = {loc, color}
```

loc is in the range 0.0 to 1.0. Values of exactly 0 or 1 replace the original *color0* or *color1* values as specified by the gradient keyword.

color is specified as (r,g,b) or (r,g,b,a) but use of alpha is not fully supported on all platforms (Quartz PDF on Macintosh for example).

The following operations support the gradient and gradientExtra keywords.

SetDrawEnv

If you specify a gradient it overrides the fill for drawing elements.

If you omit the *color0* parameter or specify (0,0,0,0) then the current pattern background color is used.

If you omit the *color1* parameter or specify (0,0,0,0) then the current foreground color is used.

All gradientExtra keywords must be on the same line as the gradient keyword itself.

ModifyLayout

You can specify gradients for individual pages by combining the gradient and gradientExtra keywords with the /PAGE=*pageNum* flag.

Page numbers start from 1. Use /PAGE=0 to use the currently active page.

/PAGE=-1 causes the operation to modify the layout global gradient which applies to any pages in the targeted page layout for which no explicit gradient has been set.

ModifyGraph (traces)

If you specify a gradient it overrides the fill for traces.

The gradient and gradientExtra keywords apply to all traces unless you specify a trace name like this:

```
gradient(<trace name>) = {...}
gradientExtra(<trace name>) = {...}
```

If you omit the *color0* parameter or specify (0,0,0,0) then the current pattern background color is used.

If you omit the *color1* parameter or specify (0,0,0,0) then the plusRGB color is used.

When the *type* parameter specifies a window rect, the plus and neg areas are used automatically and the *color1* is away from the zero level.

ModifyGraph (colors)

The wbGradient and wbGradientExtra keywords control the window background gradient, if any.

The gbGradient and gbGradientExtra keywords control the graph plot area background gradient, if any.

The "Demo Experiment #1" and "Demo Experiment #2" example experiments demonstrate these gradients. You can turn the on and off using the Macros menu.

Miscellaneous Settings

You can customize many aspects of how Igor works using the Miscellaneous Settings dialog. Choose Misc→Miscellaneous Settings to display the dialog.

Most of the settings are self-explanatory. Many have tool tips that describe what they do.

Object Names

Every Igor object has a name which you give to the object when you create it or which Igor automatically assigns. You use an object's name to refer to it in dialogs, from commands and from Igor procedures. The named objects are:

Waves	Data folders	Variables (numeric and string)
Windows	Symbolic paths	Pictures
Annotations	Controls	Rulers

In Igor Pro, the rules for naming waves and data folders are not as strict as the rules for naming all other objects, including string and numeric variables, which are required to have standard names. These sections describe the standard and liberal naming rules.

Standard Object Names

Here are the rules for standard object names:

- May be 1 to 31 bytes in length.
- Must start with an alphabetic character (A-Z or a-z).
- May include alphabetic or numeric characters or the underscore character.
- Must not conflict with other names (of operations, functions, etc.).

All names in Igor are case insensitive. wave0 and WAVE0 refer to the same wave.

Characters other than letters and numbers, including spaces and periods, are not allowed. We put this restriction on names so that Igor can identify them unambiguously in commands, including waveform arithmetic expressions.

Liberal Object Names

The rules for liberal names are the same as for standard names except that almost any character can be used in a liberal name. Liberal name rules are allowed for waves and data folders only.

If you are willing to expend extra effort when you use liberal names in commands and waveform arithmetic expressions, you can use wave and data folder names containing almost any character. If you create liberal names then you will need to enclose the names in single (not curly) quotation marks whenever they are used in commands or waveform arithmetic expressions. This is necessary to identify where the name ends. Liberal names have the same rules as standard names except you may use any character except control characters and the following:

```
"  '  :  ;
```

Here is an example of the creation and use of liberal names:

```
Make 'wave 0';    // 'wave 0' is a liberal name
'wave 0' = p
Display 'wave 0'
```

Note: Providing for liberal names requires extra effort and testing on the part of Igor programmers (see **Programming with Liberal Names** on page IV-157) so you may occasionally experience problems using liberal names with user-defined procedures.

Name Spaces

When you refer to an object by name, in a user function for example, each object must be referenced unambiguously. In general, an object must have a unique name so. Sometimes the object type can be inferred from the context, in which case the name can be the same as objects of other types. Objects whose names can be the same are said to be in different name spaces.

Data folders are in their own name space. Therefore the name of a data folder can be the same as the name of any other object, except for another data folder at the same level of the hierarchy.

Waves and variables (numeric and string) are in the same name space and so Igor will not let you create a wave and a variable in a single data folder with the same name.

An annotation is local to the window containing it. Its name must be unique only among annotations in the same window. The same applies for controls and rulers. Data folders, waves, variables, windows, symbolic paths and pictures are global objects, not associated with a particular window.

The names of global objects, except for data folders, are required to be distinct from the names of macros, functions (built-in, external or user-defined) and operations (built-in or external).

Here is a summary of the four global name spaces:

Name Space	Requirements
Data folders	Names must be distinct from other data folders at the same level of the hierarchy.
Waves, variables, windows	Names must be distinct from other waves, variables (numeric and string), windows.
Pictures	Names must be distinct from other pictures.
Symbolic paths	Names must be distinct from other symbolic paths.

Renaming Objects

You can use Misc→Rename Objects or Data→Rename to rename waves, variables, strings, symbolic paths, and pictures. Both of these invoke the Rename Objects dialog.

Graphs, tables, page layouts, notebooks, control panels, Gizmo windows, and XOP target windows are renamed using the **DoWindow** operation (see page V-147) which you can build using the Window Control dialog (see **The Window Control Dialog** on page II-44).

You can use the Data Browser to rename data folders, waves, and variables. See **The Data Browser** on page II-106.

Graphics Technology

As of version 7, Igor Pro is based on the cross-platform Qt framework and, by default, Igor uses Qt for graphics. However, for special purposes, Igor provides access to platform-native graphics.

You should avoid native graphics and stick with Qt graphics if possible because it is the focus for future development. Native graphics is provided mainly for emergency use.

You can select native graphics on a global basis (all windows are affected) in two ways:

1. From the Miscellaneous category of the Miscellaneous Settings dialog (Misc→Miscellaneous Settings menu item).
2. By executing

   `SetIgorOption GraphicsTechnology=`*`n`*

   where *n* is 0, 1, 2 or 3 for default, new, old and Qt. Unlike most other SetIgorOption cases, this change is saved to preferences on disk and applies to future Igor sessions.

In addition to changing the global graphics technology setting, you can change individual windows using

`SetWindow` *`winName`*`, graphicsTech=`*`n`*

where n is the same as above. *winName* can be kwTopWin or the actual name of a window. This not saved with the window.

Over time, it is our intention that Qt graphics will replace the other technologies and the SetIgorOption GraphicsTechnology option may no longer be supported.

Graphics Technology on Windows

In general, you should use the Qt graphics (GraphicsTechnology=3). If you experience problems with graphics, you might try GraphicsTechnology=1 or 2. High-resolution displays are supported using Qt graphics only.

On Windows, GraphicsTechnology=1 utilizes the GDI+ interface. This provides support for transparency and other advanced features. Unfortunately, GDI+ is very slow, especially when rendering text.

If speed is an issue, you can try GraphicsTechnology=2 to use the older GDI interface. However, transparency is not honored - colors will be opaque regardless of the alpha value (see **Color Blending** on page III-440 for a discussion of alpha).

In GraphicsTechnology=1 mode, EMF export uses a hybrid format, called “dual EMF”, that contains both GDI+ and, for compatibility, the older GDI. In GraphicsTechnology=2 mode, only the older GDI-only style is exported. In Qt graphics mode (GraphicsTechnology=3), GDI+ is used for rendering the EMF for export, resulting in the dual EMF format. Some applications don’t work well with EMF+ or dual EMF. In that case, try setting GraphicsTechnology=2 (old GDI mode) to export an EMF without the EMF+ component.

When you import or paste an EMF picture, the method by which Igor renders the EMF depends on the chosen graphics technology:

Technology	EMF	EMF+	Dual EMF
Qt (GraphicsTechnology=3)	GDI	GDI+	GDI+
GDI+ (GraphicsTechnology=2)	GDI+	GDI+	GDI+
GDI (GraphicsTechnology=1)	GDI	Not supported	GDI+

Our experience indicates that plain EMF is rendered better by GDI mode. In Qt graphics, the best rendering technology is chosen based on the contents of the picture you are pasting.

Regardless of the selected technology, exporting as EMF on Windows uses native interfaces.

Graphics Technology on Macintosh

On Macintosh, there is only one native format, Core Graphics, also known as Quartz, and it works the same as in Igor Pro 6.3. The native picture format is PDF.

High-resolution screen graphics on Retina displays is supported in Qt graphics technology mode only.

Regardless of the chosen graphics technology, PDF export is drawn using Core Graphics.

PDF pictures pasted into Igor are also rendered using Core Graphics. This produces the same picture regardless of the chosen Igor graphics technology, except on a Retina display, in which case the PDF is rendered on-screen at high-resolution only when using Qt graphics.

SVG Graphics

In Qt mode (GraphicsTechnology=3), SVG (Scalable Vector Graphics) is available as an alternative object-based picture format. This can replace EMF and PDF whenever the destination program supports it. In Qt mode, other imported object-based pictures (PDF on Macintosh, EMF on Windows) are rendered as high-resolution bitmap images.

In native graphics modes, imported SVG pictures are rendered as high-resolution bitmap images using Qt graphics.

Regardless of the selected technology, exporting as SVG format is rendered using Qt.

High-Resolution Displays

High-resolution screen graphics is supported in Qt graphics technology mode only.

Macintosh Retina and Windows high-resolution displays, also called Retina, 4K, 5K, Ultra HD, and Quad HD, bring a more pleasing visual experience but also present performance challenges. Depending on the operating system, graphics technology, and your actual data, graph updates can be thousands of times slower. This situation is expected to be temporary as operating systems optimize high-resolution graphics. See the “Graphs and High-Resolution Displays” topic in the release notes for the latest information.

On Windows, prior to Igor Pro 7, control panels were drawn using pixels for coordinates. Now, when the resolution exceeds 96 DPI, those coordinates are taken to be points. This prevents panels from appearing tiny on high-resolution displays. See **Control Panel Resolution on Windows** on page III-405 for more information.

Windows High-DPI Recommendations

High-DPI (dots-per-inch) displays have resolutions substantially higher than the Windows standard 96 DPI. These displays sometimes go by the names Retina, 4K, 5K, Ultra HD and Quad HD.

Since the pixels of a high-DPI monitor are typically about one-half the size of pixels on standard monitors, software running on a high-DPI monitor must make adjustments. These adjustments are made by some combination of the operating system and the program itself.

On Macintosh, with its “Retina” high-DPI screens, the operating system handles most resolution issues and there are few problems with Igor on Retina monitors.

On Windows, both the operating system and the Qt framework that Igor uses introduce complexities that sometimes result in less than perfect behavior. This section provides guidance for Windows user to minimize these issues.

To get the best experience when using a high-DPI display, we recommend that you use Windows 10 or later. We do not test or explicitly support high-DPI features on older Windows versions.

There are two factors that may affect your experience on high-DPI displays.

The first factor is the "style" that is used. By default, Igor uses a style that is specific to the operating system. This allows user interface elements such as title bars and dialog controls to look native. However, as of this writing (early 2016), the style used by Qt on Windows has not been completely updated to support high-DPI displays. Therefore, you may notice that elements, such as the arrows on scroll bar buttons and the check marks of checkboxes, are pixelated, that the window icons at the left end of the window title bar are misaligned, etc.

As an alternative to the default style, you can configure Igor to use the "Fusion" style. This style looks better on high-DPI displays though most user interface elements do not look native. To turn use the Fusion style, open the Miscellaneous Settings dialog, select the Miscellaneous category, and check the Use Fusion Style checkbox. We expect that the native style will improve over time and it may eventually be unnecessary to use the Fusion style.

The second factor is the way that Windows and the Qt framework scale user interface elements such as dialogs and windows. You may encounter undesirable behavior when using multiple display configurations in which there is one high-DPI display, such as a laptop monitor, and one or more additional standard-resolution displays such as an external monitor.

If you only have one monitor, such as a laptop with a high-DPI monitor, and no external displays are connected, you should not need to make any changes to Igor's default configuration to achieve the correct behavior. You may wish to use the Fusion theme, as described above, but it's not necessary to do so.

If you have multiple displays connected, Windows requires that you designate one of the displays as your main display. As of this writing, our experience indicates that acceptable behavior on a mixed display system can be obtained only by making the external (standard-DPI) display the main display.

When using Igor on a machine with multiple displays, we suggest you follow these steps to get the best results:

1. Connect the external standard-resolution display.
2. Open the Display settings page. You can do this by clicking the Start menu, clicking Settings, clicking System, and selecting Display in the lefthand pane.
3. Click the Identify link below the diagram of your displays to confirm which display is which. For these instructions, assume that display #1 is the high-DPI display of a laptop and display #2 is a standard-DPI external monitor.
4. Click the box representing the external monitor (display #2 for this example).
5. Check the Make This My Main Display check box.
6. In the Multiple Displays setting, select Extend These Displays.
7. Click the Apply button.

8. Reboot your machine **twice**.
9. To get the best results, you must reboot your machine **2 times** after any changes to the display settings.
10. After your machine restarts **twice**, start Igor.
11. Move Igor's outer frame window to monitor #2 (the external monitor).
12. Open the Miscellaneous Settings dialog and select the Miscellaneous category. Check the Disable High-DPI Scaling checkbox. Also check the Use Fusion Style checkbox.
13. Restart Igor.

If you later remove the standard-DPI display and wish to use Igor with only a high-DPI display connected, do the following:

1. Reboot the machine **twice**.
2. Start Igor. Open the Miscellaneous Settings dialog and select the Miscellaneous category. Uncheck the Disable High-DPI Scaling checkbox. You can check or uncheck the Use Fusion Style checkbox depending on your preference.
3. Restart Igor.

Pictures

Igor can import pictures from other programs for display in graphs, page layouts and notebooks. It can also export pictures from graphs, page layouts and tables for use in other programs. Exporting is discussed in Chapter III-5, **Exporting Graphics (Macintosh)**, and Chapter III-6, **Exporting Graphics (Windows)**. This section discusses how you can import pictures into Igor, what you can do with them and how Igor stores them.

For information on importing images as data rather than as graphics, see **Loading Image Files** on page II-138.

Importing Pictures

There are three ways to import a picture.

- Pasting from the Clipboard into a graph, layout, or notebook
- Using the Pictures dialog (Misc menu) to import a picture from a file or from the Clipboard
- Using the **LoadPICT** operation (see page V-441) to import a picture from a file or from the Clipboard

Each of these methods, except for pasting into a notebook, creates a *named*, global picture object that you can use in one or more graphs or layouts. Pasting into a notebook creates a picture that is local to the notebook.

This table shows the types of graphics formats from which Igor can import pictures:

Format	Notes
PDF (Portable Document Format)	Supported in Macintosh native graphics only.
EMF (Enhanced Metafile)	Supported in Windows native graphics only.
	See **Graphics Technology on Windows** on page III-445 for information about different types of EMF pictures.
BMP (Windows bitmap)	Supported in on Windows only.
	BMP is sometimes called DIB (Device Independent Bitmap).
PNG (Portable Network Graphics)	Lossless cross-platform bitmap format
JPEG	Lossy cross-platform bitmap format
TIFF (Tagged Image File Format)	Lossless cross-platform bitmap format

Format	Notes
EPS (Encapsulated PostScript)	High resolution vector format. Requires PostScript printer. On Windows and on Macintosh in Qt graphics mode, a screen preview is displayed on screen.
SVG (Scalable Vector Graphics)	Cross-platform vector and bitmap format. Always rendered using Qt graphics. In other graphics technology modes, it is drawn into a high-resolution btmap which is then drawn on the screen, exported, or printed.

Formats that are not supported on the current platform are drawn as gray boxes.

See also **Picture Compatibility** on page III-399 for a discussion of Macintosh graphics on Windows and vice-versa.

The Picture Gallery

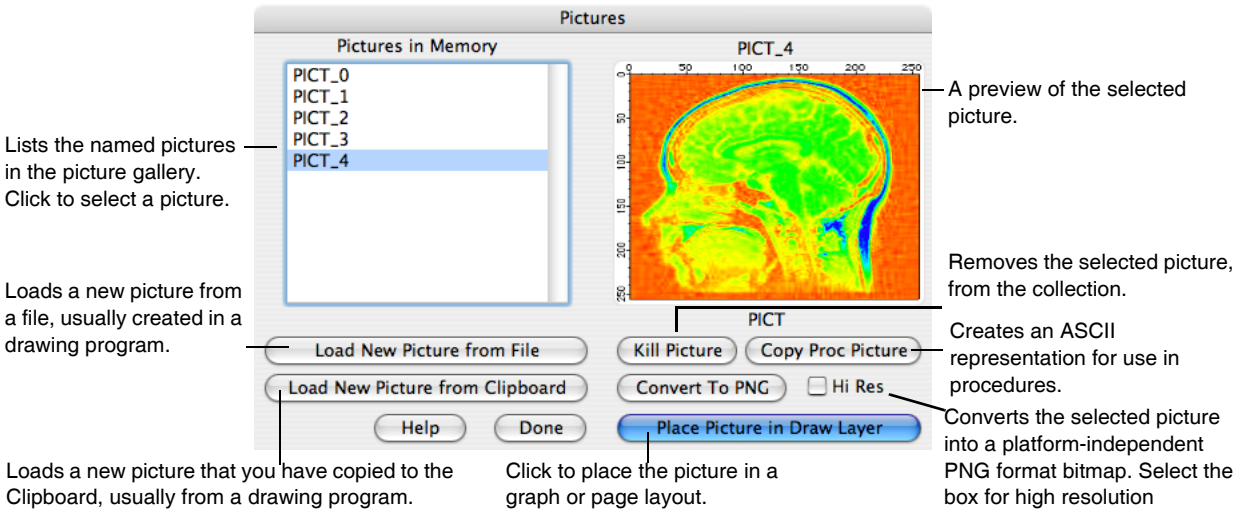
When you create a named picture using one of the techniques listed above, Igor stores it in the current experiment's **picture gallery**. When you save the experiment, the picture gallery is stored in the experiment file. You can inspect the collection using the Pictures dialog via the Misc menu.

Igor gives names to pictures so they can be referenced from an Igor procedure. For example, if you paste a picture into a layout, Igor assigns it a name of the form "PICT_0" and stores it in the picture gallery. If you then close the layout and ask Igor to create a recreation macro, the macro will reference the picture by name.

You can rename a named picture using the Pictures dialog in the Misc menu, the Rename Objects dialog in the Misc menu, the Rename dialog in the Data menu, or the **RenamePICT** operation (see page V-677). You can kill a named picture using the Pictures dialog or the **KillPICTs** operation (see page V-410).

Pictures Dialog

The Pictures dialog permits you to view the picture gallery, to add pictures, to remove pictures and to place a picture into a graph or page layout. It also can place a copy of a picture into a formatted notebook. To invoke it, choose Pictures from the Misc menu.



The Kill This Picture button will be dimmed if the selected picture is used in a currently open graph or layout.

Note: Igor determines if a picture is in use by checking to see if it is used in an *open* graph or layout window.

If you kill a graph or layout that contains a picture and create a recreation macro, the recreation macro will *refer* to the picture by name. However, Igor does not check for this. It will consider the picture to be unused and will allow you to kill it. If you later run the recreation macro, an error will occur when the macro attempts to append the picture to the graph or layout. Therefore, don't kill a picture unless you are sure that it is not needed.

NotebookPictures

When you paste a picture into a formatted notebook, you create a notebook picture. These work just like pictures in a word processor document. You can copy and paste them. These pictures will not appear in the Pictures or Rename Objects dialogs.

Igor Extensions

Igor includes a feature that allows a C or C++ programmer to extend its capabilities. An Igor extension is called an "XOP" (short for "external operation"). The term XOP comes from that fact that, originally, adding a command line operation was all that an extension could do. Now extensions can add operations, functions, menus, dialogs and windows.

WaveMetrics XOPs

The "Igor Pro 7 Folder/Igor Extensions" and "Igor Pro 7 Folder/More Extensions" folders contain XOPs that we developed at WaveMetrics. These add capabilities such as file-loading and instrument control to Igor and also serve as examples of what XOPs can do. These XOPs range from very simple to rather elaborate. Most XOPs come with help files that describe their operation.

The WaveMetrics XOPs are described in the XOP Index help file, accessible through the Help→Help Windows submenu.

Third Party Extensions

A number of Igor users have written XOPs to customize Igor for their particular fields. Some of these are freeware, some are shareware and some are commercial programs. WaveMetrics publicizes third party XOPs through our Web page. User-developed XOPs are available from http://www.igorexchange.com.

Activating Extensions

The Igor installer creates a folder called Igor Extensions inside the Igor Pro 7 Folder. The Igor installer puts some XOP files in this folder. XOPs in this folder are loaded when Igor starts up.

The installer puts less-frequently used XOPs in "Igor Pro 7 Folder/More Extensions". These are not available unless you activate them.

If you place in "Igor Pro 7 Folder/Igor Extensions" an alias (Macintosh) or shortcut (Windows) for an XOP file or a folder containing XOP files, Igor loads those files also. However, this is not the recommended way to activate an XOP.

Your operating system may not allow you to make changes to the contents of your Igor Pro 7 Folder, for example, if you do not have permission to write to that folder. For that reason, Igor also loads extensions from another folder - "Igor Pro User Files/Igor Extensions" (see **Igor Pro User Files** on page II-31 for details). You can locate this folder by selecting Help→Show Igor Pro User Files.

If you press the shift key while clicking the Help menu, you can choose Help→Show Igor Pro 7 Folder andUser Files. Igor then opens the Igor Pro 7 Folder and the Igor Pro User Files folder on the desktop, making it easy to drag aliases/shortcuts into "Igor Pro User Files/Igor Extensions". This is the recommended way to activate additional WaveMetrics XOPs for a given user.

If you want to activate an extension for all users on a given machine, you can put the alias or shortcut in "Igor Pro 7 Folder/Igor Extensions" folder. You may need to run as administrator to do this.

Changes that you make to either Igor Extensions folder take effect the next time Igor is launched.

See **Creating Igor Extensions** on page IV-195 if you are a programmer interested in writing your own XOPs.

Memory Management

For many years, virtually all new machines have shipped with 64-bit operating systems. Consequently, in this section, we assume that you are running a 64-bit OS.

Igor7 comes in 32-bit and 64-bit versions called IGOR32 and IGOR64 respectively. The only reason to run IGOR32 is if you depend on 32-bit XOPs that have not been ported to 64 bits.

IGOR32 can theoretically address 4GB of virtual address space. For the vast majority of Igor applications, this is more than sufficient. If you must load gigabytes of data into memory at one time, you may run out of memory. This may happen long before you load 4GB of data into memory, because, to allocate a wave for example, you not only need free memory, but the free memory must also be continguous, and memory becomes fragmented over time.

IGOR64 can theoretically address about a billion gigabytes. However, actual operating systems impose far lower limits. On Windows 10, 64-bit programs can address between 128 GB (home edition) and 512 GB (professional edition). On Mac OS X, 64-bit programs can theoretically address the full 64-bit address space.

If you load more data than fits in physical memory, the system starts using "virtual memory", meaning that it swaps data between physical memory and disk, as needed. This is very slow. Consequently, you should avoid loading more data into memory than can fit in physical memory.

Even if your data fits in physical memory, graphing and manipulating very large waves, such as 10 million, 100 million, or 1 billion points, will be slow.

All of this boils down to the following rules:

1. If you don't need to load gigabytes of data into memory at one time then you don't need to worry about memory management.
2. Run IGOR64 unless you rely on 32-bit XOPs that can not be ported to 64 bits.
3. Install enough physical memory to avoid the need for virtual memory swapping.

For further information about very large waves, see **IGOR64 Experiment Files** on page II-35.

Macintosh System Requirements

Igor Pro requires Mac OS X 10.9.0 or later.

Windows System Requirements

Igor Pro requires Windows 7 or later.