**Flags**

| | |
|---|---|
| /W=*winName* | Draws to the named window or subwindow. When omitted, action will affect the active window or subwindow. This must be the first flag specified when used in a Proc or Macro or on the command line.<br>When identifying a subwindow with *winName*, see **Subwindow Syntax** on page III-92 for details on forming the window hierarchy. |

**Details**

The coordinate system as well as the text's font, size, style and other properties are determined by the current drawing environment. The text is drawn in the current draw layer for the window, as determined by SetDrawLayer.

**See Also**

Chapter III-3, **Drawing**.

The **SetDrawEnv**, **SetDrawLayer** and **DrawAction** operations.

## DrawUserShape

`DrawUserShape [/W=winName /MO=options] x0, y0, x1, y1, userFuncName, textString, privateString`

The DrawUserShape operation is similar to built-in drawing operations except the shape is defined by a user-defined function which is executed when the shape is drawn.

DrawUserShape was added in Igor Pro 7.00.

**Parameters**

For rectangular shapes (*x0*,*y0*) defines the top left corner while (*x1*,*y1*) defines the lower right corner in the currently active coordinate system. For line-like shapes, (*x0*,*y0*) specifies the start of the line while(*x1*,*y1*) specifies the end.

*userFuncName* specifies your user-defined function that uses built-in drawing operations to define the shape and provides information about it to Igor.

*textString* specifies text to be drawn over the shape by Igor. Pass "" if you do not need text. Using escape codes you can change the font, size, style, and color of the text. See **Annotation Escape Codes** on page III-53 or details.

*privateString* is text or binary data used by the user-defined function for any purpose. It is typically used to maintain state information and is saved in recreation macros using a method that supports binary. Pass "" if you do not need this.

To support modifying an existing shape, each of the last three parameters above can be `_NoChange_`. The *x0*, *y0*, *x1*, *y1* parameters can be all zero for no change. Using `_NoChange_` when there is no existing shape is an error. To target an existing shape, it must be selected by the drawing tools.

**Flags**

| | |
|---|---|
| /W=*winName* | Draws to the named window or subwindow. When omitted, action will affect the active window or subwindow. This must be the first flag specified when used in a Proc or Macro or on the command line.<br>When identifying a subwindow with *winName*, see **Subwindow Syntax** on page III-92 for details on forming the window hierarchy. |
| /M=*options* | An integer that Igor passes to your user-defined drawing function for use for any purpose. |

**Details**

The user-defined function must have the following form and structure parameter.

```
Function MyUserShape(s) : DrawUserShape // Optional ": DrawUserShape" indicates
                                        // that this function should be added
   STRUCT WMDrawUserShapeStruct &s      // to the menu of shapes in the drawing palette

   Variable returnValue= 1
```

```
    StrSwitch(s.action)
        case "draw":
            Print "Put DrawXXX commands here"
            break
        case "drawSelected":
            Print "Put Draw commands when selected (may be same as normalDraw) here"
            break
        case "hitTest":
            Print "Put code to test if mouse is over a control point here"
            // Set doSetCursor and optionally cursorCode if so"
            // Set returnValue to zero if not over control point or, if in operate
            // mode and you don't want to start a button operation.
            break
        case "mouseDown":
            if( s.operateMode )
                Print "Mouse down in button mode."
            else
                // Code similar or same as hitTest.
                Print "User is starting a drag of a control point."
            endif
            // Set returnValue to zero if if no action or redraw needed
            break
        case "mouseMove":
            if(s.operateMode)
                Print "Roll-over mode and mouse is inside shape."
            else
                Print "Mouse has been captured and user is dragging a control point."
            endif
            // Set returnValue to zero if if no action or redraw needed
            break
        case "mouseUp":
            if(s.operateMode)
                Print "Mouse button released."
            else
                Print "User finished dragging control point."
            endif
            break
        case "getInfo":
            Print "Igor is requesting info about this shape."
            s.textString= "category for shape"
            s.privateString= "shape name"
            s.options= 0// Or set bits 0, 1 and/or 2
            break
    EndSwitch

    return returnValue
End
```

WMDrawUserShapeStruct is a built-in structure with the following members:

**`WMDrawUserShapeStruct` Structure Members**

| Member | Description |
|---|---|
| `char action[32]` | Input: Specifies what action is requested. |
| `Int32 options` | Input: Value from /MO flag. |
| | Output: When action is `getInfo`, set bits as follows: |
| | Set bit 0 if the shape should behave like a simple line. When resizing end-points, you will get live updates. |
| | Set bit 1 if the shape is to act like a button; you will get mouse down in normal operate mode. |
| | Set bit 2 to get roll-over action. You will get `hitTest` action and if 1 is returned, the mouse will be captured. |
| `Int32 operateMode` | Input: If 0, the shape is being edited; if 1, normal operate mode (only if options bit 1 or 2 was set during `getInfo`). |
| `PointF mouseLoc` | Input: The location of the mouse in normalized coordinates. |

**`WMDrawUserShapeStruct` Structure Members**

| Member | Description |
|---|---|
| `Int32 doSetCursor` | Output: If action is `hitTest`, set true to use the following cursor number. Also used for `mouseMoved` in rollover mode. |
| `Int32 cursorCode` | Output: If action is `hitTest` and `doSetCursor` is set, then set this to the desired Igor cursor number. |
| `double x0,y0,x1,y1` | Input: Coordinates of the enclosing rectangle of the shape. |
| `RectF objectR` | Input: Coordinates of the enclosing rectangle of the shape in device units. |
| `char winName[MAX_HostChildSpec+1]` | Full path to host subwindow |
| `// Information about the coordinate system` | |
| `Rect drawRect` | Draw rect in device coordinates |
| `Rect plotRect` | In a graph, this is the plot area |
| `Rect axisRect` | In a graph, this is the plot area including axis standoff |
| `char xcName[MAX_OBJ_NAME+1]` | Name of X coordinate system, may be axis name |
| `char ycName[MAX_OBJ_NAME+1]` | Name of Y coordinate system, may be axis name |
| `double angle` | Input: Rotation angle, use when displaying text |
| `String textString` | Input: Use or ignore; special output for `getInfo` |
| `String privateString` | Input and output: Maintained by Igor but defined by user function; may be binary; special output for `getInfo` |

The constants used to size the char arrays, `MAX_HostChildSpec` and `MAX_OBJ_NAME`, are defined internally in Igor and are subject to change in future versions.

The drawing commands you provide for the `draw` and `drawSelected` actions must use normalized coordinates ranging from (0,0) to (1,1). For example, to draw a rectangle you would use

```
DrawRect 0,0,1,1
```

Generally, you will not set drawing environment parameters such as color or fill mode but will leave that to the user of your shape just as they would for built-in shapes such as a Rectangle.

Drawing commands such as **SetDrawEnv**, **DrawRect**, etc., include a /W flag that directs their action to a particular window or subwindow. However, when called from a DrawUserShape function, the /W flag is ignored and all drawing commands are directed to the window or subwindow containing the shape.

You will get the `drawSelected` action when the user has selected your shape with the arrow draw tool. For simple shapes, this can use the same code as the draw action but if you want the shape to be editable, you can draw control points such as a yellow dot. You would then provide code in the `hitTest` and `mouseDown` actions that determines if the user has moved over or clicked on a control point. For an example of this, see the FatArrows procedure file in the User Draw Shapes demo experiment.

Your function should respond to the `getInfo` action by setting the textString field to a category name and the privateString field to a shape name. A category name might be "Arrows" and a shape name might be "Right Arrow". These names are used to populate the menus you get when right-clicking the user shapes icon in the drawing tools palette. You can also set bits in the options field to enable certain special actions. For examples of these, see the LineCallout and button procedure windows in the User Draw Shapes demo experiment.

Because the user-defined function runs during a drawing operation that cannot be interrupted without crashing Igor, the debugger cannot be invoked while it is running. Consequently breakpoints set in the function are ignored. Use **Debugging With Print Statements** on page IV-212 instead.

To support button-like shapes in graphs and layouts, an additional drawing layer named Overlay was added in Igor Pro 7.00. This layer is drawn over the top of everything else and is not printed or exported. Objects in the Overlay draw layer can update without the need to redraw the entire window as would be