This is Google's cache of <http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html>. It is a snapshot of the page as it appeared on Jul 21, 2013 08:06:56 GMT. The [current page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html) could have changed in the meantime. [Learn more](http://support.google.com/websearch/bin/answer.py?hl=en&p=cached&answer=1687222)  
Tip: To quickly find your search term on this page, press **Ctrl+F** or **⌘-F** (Mac) and use the find bar.

[Text-only version](http://webcache.googleusercontent.com/search?q=cache:HMjrb-u6-1gJ:www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html&client=firefox-a&hs=5RK&hl=en&gl=us&strip=1)

|  |  |
| --- | --- |
| **External Interfaces** | [Supported Client/Server Configurations](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/bq9yai0-1.html)   [Additional COM Client Information](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27433.html) |

|  |  |
| --- | --- |
|  | [Provide feedback about this page](http://www.mathworks.com/survey/support/docfeedback.html?http://webcache.googleusercontent.com/search?q=cache:HMjrb-u6-1gJ:www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html+&cd=13&hl=en&ct=clnk&gl=us&client=firefox-a) |

**MATLAB COM Client Support**

[www.kxcad.net Home](http://www.kxcad.net) > [CAE Software Index](http://www.kxcad.net/CAE.html) > [MATLAB Index](http://www.kxcad.net/cae_MATLAB/index.htm) >

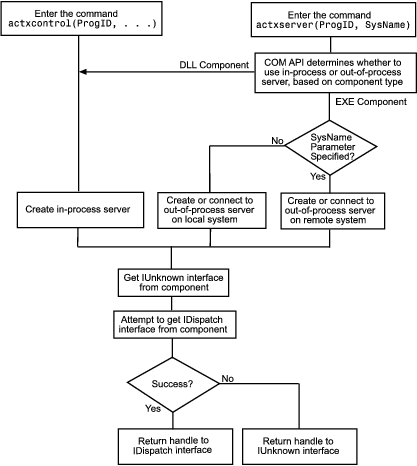
|  |
| --- |
| **On this page…** |
| [Creating the Server Process — An Overview](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f106728)  [Creating an ActiveX Control](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f106314)  [Deploying ActiveX Controls Requiring Run-Time Licenses](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bqo22w0)  [Instantiating a DLL Component](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f104630)  [Instantiating an EXE Component](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f104578)  [Getting Interfaces to the Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f103034)  [Invoking Commands on a COM Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f116344)  [Identifying Objects and Interfaces](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f90970)  [Invoking Methods](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f89715)  [Object Properties](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f57262)  [Control and Server Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f57267)  [Writing Event Handlers](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f89992)  [Saving Your Work](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f107685)  [Releasing COM Interfaces and Objects](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f62902)  [Identifying Objects](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f90226)  [Handling COM Data in MATLAB](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4epvg-1)  [Examples of MATLAB as an Automation Client](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f62644)  [MATLAB COM Client Demo](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bra56xr-1) |

**Creating the Server Process — An Overview**

MATLAB provides two functions to create a COM object:

* [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) — Creates an ActiveX control in a MATLAB figure window.
* [actxserver](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxserver.html) — Creates an in-process server for a dynamic link library (DLL) component or an out-of-process server for an executable (EXE) component.

The diagram below shows the basic steps in creating the server process. For more information on how MATLAB establishes interfaces to the resultant COM object, see [Getting Interfaces to the Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f103034).



[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Creating an ActiveX Control**

You can create an ActiveX control from the MATLAB client using either a graphical user interface or the [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) function directly from the command line. Either of these methods creates an instance of the control in the MATLAB client process and returns a handle to the primary interface to the COM object. Through this interface, you can access the object's public properties or methods . You can also establish additional interfaces to the object, including interfaces that use IDispatch, and any custom interfaces that may exist.

This section describes how to create the control and how to position its physical representation in the MATLAB figure window:

* [Finding Out What Controls Are Installed](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f106333)
* [Finding a Particular Control](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bqnm0md)
* [Creating Control Objects Using a Graphical Interface](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f106354)
* [Creating Control Objects from the Command Line](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f106386)
* [Repositioning the Control in a Figure Window](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f106401)
* [Using Microsoft Forms 2.0 Controls](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f132190)

**Finding Out What Controls Are Installed**

The [actxcontrollist](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrollist.html) function shows you what COM controls are currently installed on your system. Type

list = actxcontrollist

and MATLAB returns a cell array listing each control, including its name, programmatic identifier (ProgID), and filename.

This example shows information that might be returned for several controls (your results might be different):

list = actxcontrollist;

s=sprintf(' Name = %s\n ProgID = %s\n File = %s\n', list{114:115,:})

MATLAB displays:

s =

Name = OleInstall Class

ProgID = Outlook Express Mime Editor

File = OlePrn.OleInstall.1

Name = OutlookExpress.MimeEdit.1

ProgID = C:\WINNT\System32\oleprn.dll

File = C:\WINNT\System32\inetcomm.dll

**Finding a Particular Control**

If you know the name of a control, you can find it in the list and display its ProgID and the path of the directory containing it using a few MATLAB commands. For example the Mwsamp2 control is used in some of the examples in this manual. You can find it with the following code:

list = actxcontrollist;

for ii = 1:length(list)

if ~isempty(findstr('Mwsamp2',[list{ii,:}]))

s = sprintf(' Name = %s\n ProgID = %s\n File = %s\n', ...

list{ii,:})

end

end

The formatted output contained in s is displayed:

s =

Name = Mwsamp2 Control

ProgID = MWSAMP.MwsampCtrl.2

File =

D:\Apps\MATLAB\R2006a\toolbox\matlab\winfun\win32\mwsamp2.ocx

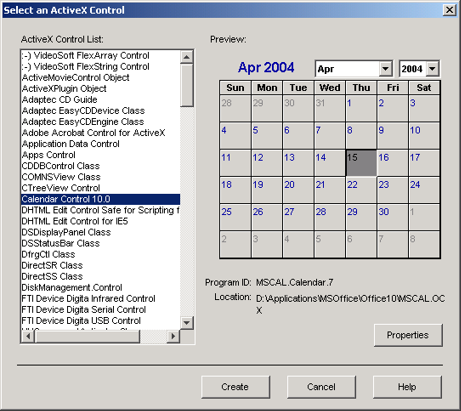
The location of this file might be different on your system.

**Creating Control Objects Using a Graphical Interface**

The simplest way to create a control object is to use the [actxcontrolselect](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrolselect.html) function. This function displays a graphical user interface (GUI) listing all controls installed on your system. When you select an item from the list and click the **Create** button, MATLAB creates the control and returns a handle to it. Type

h = actxcontrolselect

MATLAB displays:



The interface has a selection panel on the left and a preview panel on the right. Click one of the control names in the selection panel to see a preview of the control. (For controls that do not have a preview, the preview panel is blank.) If MATLAB cannot create the control, an error message appears in the preview panel.

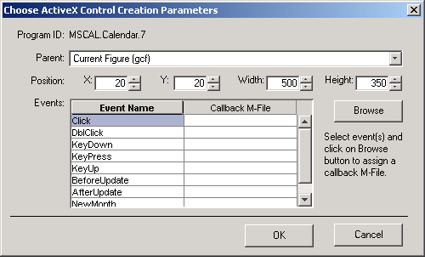
**Setting Properties with actxcontrolselect.** Click the **Properties** button on the actxcontrolselect window to change property values when creating the control. You can select which figure window to put the control in (**Parent** field), where to position it in the window (**X** and **Y** fields), and what size to make the control (**Width** and **Height**).

You can register events you want the control to respond to in this window. (See [Control and Server Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f57267) for an explanation of event registration.) Register an event and the callback routine to handle that event by entering the name of the routine to the right of the event under **Callback M-File**.

You can select callback routines by browsing for their M-files. Click a name in the **Event Names** column and then click the **Browse** button. To assign a callback routine to more than one event, first press the **Ctrl** key and click individual event names, or drag the mouse over consecutive event names, then click **Browse** to select the callback routine.

MATLAB only responds to registered events, so if you do not specify a **Callback M-File**, the event is ignored.

For example, at the actxcontrolselect window, select **Calendar Control 10.0** and click **Properties** to see the window shown below. Enter a **Width** of 500 and a **Height** of 350 to change the default size for the control. Click **OK** in this window, and click **Create** in the actxcontrolselect window to create the Calendar control.



You can also set control parameters using the [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) function. One parameter you can set with actxcontrol but not with actxcontrolselect is the name of an initialization file. When you specify this filename, MATLAB sets the initial state of the control to that of a previously saved control.

**Information Returned by actxcontrolselect.** actxcontrolselect creates an object that is an instance of the MATLAB COM class. The function returns up to two arguments: a handle for the object h, and a 1-by-3 cell array info containing information about the control.

[h, info] = actxcontrolselect;

The cell array info shows the name, ProgID, and filename for the control.

If you select Calendar Control 9.0 and then click **Create**, MATLAB displays:

h =

COM.mscal.calendar.7

info =

[1x20 char] 'MSCAL.Calendar.7' [1x41 char]

Expand the info cell array to show the control name, ProgID, and filename.

info{:}

ans =

Calendar Control 9.0

ans =

MSCAL.Calendar.7

ans =

D:\Applications\MSOffice\Office\MSCAL.OCX

**Creating Control Objects from the Command Line**

If you already know which control you want and you know its ProgID, you can bypass the GUI by using the [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) function to create it.

The only required input when calling the function is the ProgID. However, as with [actxcontrolselect](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrolselect.html), you can supply additional inputs that enable you to select which figure window to put the control in, where to position it in the window, and what size to make it. You can also register any events you want the control to respond to, or set the initial state of the control by reading that state from a file. See the reference page on [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) for a full explanation of its input arguments.

actxcontrol returns a handle to the primary interface to the object. Use this handle to reference the object in other COM function calls. You can also use the handle to obtain additional interfaces to the object. For more information on using interfaces, see [Getting Interfaces to the Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f103034).

This example creates a control to run a Microsoft Calendar application. Position the control in figure window fig3, at a [0 0] x-y offset from the bottom left of the window, and make it 300 by 400 pixels in size:

fig3 = figure('position', [50 50 600 500]);

h = actxcontrol('mscal.calendar', [0 0 300 400], fig3)

MATLAB displays:

h =

COM.mscal.calendar

**Repositioning the Control in a Figure Window**

Once a control has been created, you can change its shape and position in the window with the [move](http://www.kxcad.net/cae_MATLAB/techdoc/ref/move.html) function.

Observe what happens to the object created in the last section when you specify new origin coordinates (70, 120) and new width and height dimensions of 400 and 350:

h.move([70 120 400 350]);

**Using Microsoft Forms 2.0 Controls**

You may encounter problems when creating or using Microsoft Forms 2.0 controls in MATLAB. Forms 2.0 controls are designed for use with applications enabled by Visual Basic for Applications (VBA). Microsoft Office is one such application.

To work around these problems, use the replacement controls listed below, or consult article 236458 in the Microsoft Knowledge Base for further information:

<http://support.microsoft.com/default.aspx?kbid=236458>

**Affected Controls.** You may see this behavior with any of the following Forms 2.0 controls:

* Forms.TextBox.1
* Forms.CheckBox.1
* Forms.CommandButton.1
* Forms.Image.1
* Forms.OptionButton.1
* Forms.ScrollBar.1
* Forms.SpinButton.1
* Forms.TabStrip.1
* Forms.ToggleButton.1

**Replacement Controls.** The following replacements are recommended by Microsoft:

| **Old** | **New** |
| --- | --- |
| Forms.TextBox.1 | RICHTEXT.RichtextCtrl.1 |
| Forms.CheckBox.1 | vidtc3.Checkbox |
| Forms.CommandButton.1 | MSComCtl2.FlatScrollBar.2 |
| Forms.TabStrip.1 | COMCTL.TabStrip.1 |

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Deploying ActiveX Controls Requiring Run-Time Licenses**

When you deploy an ActiveX control that requires a run-time license, you must include a license key, which the control reads at run-time. If the key matches the control's own version of the license key, an instance of the control is created. Use the following procedure to deploy a run-time-licensed control with MATLAB.

**Create an M-File to Build the Control**

First, create an M-file to build the control. This M-file must contain two elements:

* The pragma %#function actxlicense. This pragma causes MATLAB Compiler to embed a function named actxlicense into the stand-alone executable file you build.
* A call to [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) to create the control.

Place this M-file in a directory that is not part of the MATLAB code tree.

Here is an example M-file.

function buildcontrol

%#function actxlicense

h=actxcontrol('MFCCONTROL2.MFCControl2Ctrl.1',[10 10 200 200]);

**Build the Control and the License M-File**

Change to the directory where you placed the M-file you created to build the control. Call the function you defined in the M-file. When it executes this function, MATLAB determines whether the control requires a run-time license. If it does, MATLAB creates another M-file, named actxlicense.m, in the current working directory. The actxlicense function defined in this file provides the license key to MATLAB at run-time.

**Build the Executable**

Next, call MATLAB Compiler to create the stand-alone executable from the file you created to build the control. The executable contains both the function that builds the control and the actxlicense function.

mcc -m buildcontrol

**Deploy the Files**

Finally, distribute buildcontrol.exe, buildcontrol.ctf, and the control (.ocx or .dll).

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Instantiating a DLL Component**

To create a server for a component implemented as a dynamic link library (DLL), use the [actxserver](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxserver.html) function. MATLAB creates an instance of the component in the same process that contains the client application.

The syntax for actxserver, when used with a DLL component, is

actxserver(ProgID)

where ProgID is the programmatic identifier for the component.

actxserver returns a handle to the primary interface to the object. Use this handle to reference the object in other COM function calls. You can also use the handle to obtain additional interfaces to the object. For more information on using interfaces, see [Getting Interfaces to the Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f103034).

Unlike ActiveX controls, any user interface displayed by the server appears in a separate window.

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Instantiating an EXE Component**

You can use the [actxserver](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxserver.html) function to create a server for a component implemented as an executable (EXE). In this case, MATLAB instantiates the component in an out-of-process server.

The syntax for actxserver is

actxserver(ProgID, sysname)

where ProgID is the programmatic identifier for the component, and sysname is an optional argument used in configuring a distributed COM (DCOM) system.

actxserver returns a handle to the primary interface to the COM object. Use this handle to reference the object in other COM function calls. You can also use the handle to obtain additional interfaces to the object. For more information on using interfaces, see [Getting Interfaces to the Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f103034).

Any user interface displayed by the server appears in a separate window.

This example creates a COM server application running Excel. The handle is assigned to h:

h = actxserver('excel.application')

MATLAB displays:

h =

COM.excel.application

MATLAB can programmatically connect to an instance of a COM Automation server application that is already running on your computer. Use the [actxGetRunningServer](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxgetrunningserver.html) function to get a reference to such an application.

The syntax for actxGetRunningServer is

actxGetRunningServer(ProgID)

where ProgID is the programmatic identifier for the component.

This example gets a reference to Excel, which must already be running on your system. The returned handle is assigned to h:

h = actxGetRunningServer('excel.application')

MATLAB displays:

h =

COM.excel.application

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Getting Interfaces to the Object**

The COM component you are working with can provide different types of interfaces for accessing the object's public properties and methods:

* [The IUnknown and IDispatch interfaces](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f103130)
* [One or more custom interfaces](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f103036)

**IUnknown and IDispatch**

When you invoke the [actxserver](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxserver.html) or [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) function, MATLAB creates the server and returns a handle to the server interface as a means of accessing its properties and methods. MATLAB uses the following process to determine which handle to return:

1. MATLAB first gets a handle to the IUnknown interface from the component. All COM components are required to implement this interface.
2. MATLAB then attempts to get the IDispatch interface. If IDispatch is implemented, MATLAB returns a handle to this interface. If IDispatch is not implemented, MATLAB returns the handle to IUnknown.

**Additional Interfaces.** Components often provide additional interfaces, based on IDispatch, that are implemented as properties. Like any other property, you obtain these interfaces using the MATLAB [get](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.get.html) function.

For example, a Microsoft Excel component contains numerous interfaces. To list these interfaces, along with other Excel properties, use the MATLAB get function without any arguments. For example, type:

h = actxserver('excel.application');

h.get

MATLAB displays information similar to:

Application: [1x1 Interface.Microsoft\_Excel\_9.0\_

Object\_Library.\_Application]

Creator: 'xlCreatorCode'

Parent: [1x1 Interface.Microsoft\_Excel\_9.0\_

Object\_Library.\_Application]

ActiveCell: []

ActiveChart: [1x50 char]

.

.

In the following example, h is a handle to a specific interface and Workbooks is the name of the interface. Type

w = h.Workbooks

MATLAB displays:

w =

Interface.Microsoft\_Excel\_9.0\_Object\_Library.Workbooks

The information displayed depends on the version of Excel you have on your system.

**Custom Interfaces**

The following two client/server configurations support a component's custom interface:

* [Introduction](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/bq9yai0-1.html#bq9yam7-1)
* [MATLAB Client and Out-of-Process Server](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/bq9yai0-1.html#bq9yam7-5)

Once you have created the server, you can query the server component to see if any custom interfaces are implemented. Use the [interfaces](http://www.kxcad.net/cae_MATLAB/techdoc/ref/interfaces.html) function to return a list of all available custom interfaces. This list is returned in a cell array of strings.

For example, for a component with the ProgID mytestenv.calculator, type

h = actxserver('mytestenv.calculator')

MATLAB displays:

h =

COM.mytestenv.calculator

Type

customlist = h.interfaces

MATLAB displays:

customlist =

ICalc1

ICalc2

ICalc3

To get a handle to a particular custom interface, use the [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html) function, specifying the handle returned by [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) or [actxserver](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxserver.html), and also the name of the custom interface:

c1 = h.invoke('ICalc1')

c1 =

Interface.Calc\_1.0\_Type\_Library.ICalc\_Interface

Use this handle with the [COM client functions](http://www.kxcad.net/cae_MATLAB/techdoc/ref/f16-6011.html#f16-54223) to access the properties and methods of the object through the selected custom interface.

For example, to list the properties available through the ICalc1 interface, use

c1.get

background: 'Blue'

height: 10

width: 0

To list the methods, use

c1.invoke

Add = double Add(handle, double, double)

Divide = double Divide(handle, double, double)

Multiply = double Multiply(handle, double, double)

Subtract = double Subtract(handle, double, double)

Add and multiply numbers using the Add and Multiply methods of the custom object c1:

sum = c1.Add(4, 7)

sum =

11

prod = c1.Multiply(4, 7)

prod =

28

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Invoking Commands on a COM Object**

This section covers the following topics:

* [Dot Syntax](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq_ysk1-1)
* [An Example of Calling Syntax](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f116689)
* [Specifying Property, Method, and Event Names](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f120091)
* [Implicit Syntax for Calling get, set, and invoke](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f116257)
* [Exceptions to Using Implicit Syntax](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f116268)

**Dot Syntax**

When invoking either MATLAB COM functions or methods belonging to COM objects, the simplest syntax to use is *dot syntax*. Specify the object name, the dot (.), and the name of the function or method you are calling. Enclose any input arguments in parentheses after the function name. Specify output arguments to the left of the equal sign:

outputvalue = object.function(arg1, arg2, ...)

**An Example of Calling Syntax**

To work with the example that follows, first create an ActiveX control called mwsamp. (The mwsamp control is built into MATLAB to enable you to work with the examples shown in the COM documentation. The control displays a circle and text label that you can manipulate from MATLAB.)

Call [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) to create the mwsamp control. This function returns a handle h that you need to work further with the control.

h = actxcontrol('mwsamp.mwsampctrl.2', [200 120 200 200]);

Once you have a handle to an object, you can invoke MATLAB functions on the object by referencing it through the handle.

For example, you can create a custom property, called Position, using the [addproperty](http://www.kxcad.net/cae_MATLAB/techdoc/ref/addproperty.html) function. See [Custom Properties](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f71913).

h.addproperty('Position');

An alternative syntax for the same operation is

addproperty(h, 'Position');

**Specifying Property, Method, and Event Names**

You can specify the names of properties and methods using the simple notation

handle.propertyname

handle.methodname

For example, the mwsamp object has a property called Radius that represents the radius of the circle it draws, and a method called Redraw that redraws the circle. You can get the circle's radius by typing

h.Radius

You can redraw the circle with

h.Redraw

More information is provided on this in the sections [Implicit Syntax for Calling get, set, and invoke](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f116257) and [Exceptions to Using Implicit Syntax](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f116268). Here are a few specific rules regarding how to express property, method, and event names.

**Property Names.** You can abbreviate the names of properties, as long as you include enough characters in the name to distinguish it from another property. Property names are also case insensitive.

These two statements produce the same result:

x = h.Radius

x = h.r

**Method Names.** Method names cannot be abbreviated and are case sensitive.

**Event Names.** Event names are always specified as quoted strings in arguments to a function. Event names cannot be abbreviated and they are not case sensitive.

These statements produce the same result:

h.registerevent({'MouseDown' 'mymoused'});

h.registerevent({'MOUSEdown' 'myMOUSEd'});

**Implicit Syntax for Calling get, set, and invoke**

When calling [get](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.get.html), [set](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.set.html), or [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html) on a COM object, MATLAB provides a simpler syntax that doesn't require an explicit function call. You can use this shortened syntax in all but a few cases (see [Exceptions to Using Implicit Syntax](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f116268)).

Continue with the mwsamp control created in the last section and represented by handle h. To get the value of Radius property and assign it to variable x, use the syntax shown here. MATLAB still makes the call to get, but this shortened syntax is somewhat easier to enter:

x = h.Radius

x =

20

The same shortened syntax applies when calling the set and invoke functions. Compare these two ways of setting a new radius value for the circle and invoking the Redraw method of mwsamp to display the circle in its enlarged size. The commands on the left call set and invoke explicitly. The commands on the right make implicit calls:

h.set('Radius', 40); h.Radius = 40;

h.invoke('Redraw'); h.Redraw;

**Exceptions to Using Implicit Syntax**

There are some conditions under which you must explicitly call get, set, and invoke:

* When the property or method is not public
* When accessing a property that takes arguments
* When operating on a vector of objects

**Nonpublic properties and methods.** If the property or method you want to access is not provided as a public property or method of the object class, or if it is not in the type library for the control or server, you must call [get](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.get.html), [set](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.set.html), or [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html) explicitly. For example, the Visible property of an Internet Explorer application is not public and must be accessed using get and set:

h = actxserver('internetexplorer.application');

% This syntax is invalid because 'Visible' is not public.

v = h.Visible

??? No appropriate method or public field Visible for class

COM.internetexplorer.application.

% You must call the get function explicitly.

v = h.get('Visible')

v =

1

% The same holds true when setting nonpublic properties.

h.set('Visible', 1);

Public properties and methods are those that are listed in response to the following commands on COM object h:

publicproperties = h.get

publicmethods = h.invoke

**Accessing Properties That Take Arguments.** Some COM objects have properties that behave somewhat like methods in that they accept input arguments. This is explained fully in the section [Properties That Take Arguments](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f118333). In order to get or set the value of such a property, you must make an explicit call to the get or set function, as shown here. In this example, A1 and B2 are arguments that specify which Range interface to return on the get operation:

eActivesheetRange = e.Activesheet.get('Range', 'A1', 'B2')

eActivesheetRange =

Interface.Microsoft\_Excel\_5.0\_Object\_Library.Range

**Operating on a Vector of Objects.** If you operate on a vector of objects, (see [Get and Set on a Vector of Objects](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f85093)), you must call get and set explicitly to access properties.

This example creates a vector of handles to two Microsoft Calendar objects. It then modifies the Day property of both objects in one operation by invoking set on the vector. Explicit calls to get and set are required:

h1 = actxcontrol('mscal.calendar', [0 200 250 200]);

h2 = actxcontrol('mscal.calendar', [250 200 250 200]);

H = [h1 h2];

H.set('Day', 23)

H.get('Day')

ans =

[23]

[23]

This applies only to get and set. You cannot invoke a method on more than one COM object at a time, even if you call invoke explicitly.

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Identifying Objects and Interfaces**

You can get some additional information about a control or server using the following functions.

| **Function** | **Description** |
| --- | --- |
| [class](http://www.kxcad.net/cae_MATLAB/techdoc/ref/class.html) | Return the class of an object |
| [isa](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isa.html) | Determine if an object is of a given MATLAB class |
| [iscom](http://www.kxcad.net/cae_MATLAB/techdoc/ref/iscom.html) | Determine if the input is a COM or ActiveX object |
| [isinterface](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isinterface.html) | Determine if the input is a COM interface |

This example creates a COM object in an Automation server running Excel, giving it the handle h, and a Workbooks interface to the object, with handle w.

h = actxserver('excel.application');

w = h.Workbooks;

Use the [iscom](http://www.kxcad.net/cae_MATLAB/techdoc/ref/iscom.html) function to see if variable h is a handle to a COM or ActiveX object:

h.iscom

ans =

1

Use the [isa](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isa.html) function to test variable h against a known class name:

h.isa('COM.excel.application')

ans =

1

Use [isinterface](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isinterface.html) to see if variable w is a handle to a COM interface:

w.isinterface

ans =

1

Use the [class](http://www.kxcad.net/cae_MATLAB/techdoc/ref/class.html) function to find out the class of variable w:

w.class

ans =

Interface.Microsoft\_Excel\_9.0\_Object\_Library.Workbooks

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Invoking Methods**

This section covers the following topics:

* [Functions for Working with Methods](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f89716)
* [Listing the Methods of a Class or Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f89757)
* [Invoking Methods on an Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f91194)
* [Specifying Enumerated Parameters](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f107980)
* [Optional Input Arguments](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f89811)
* [Returning Multiple Output Arguments](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f105307)
* [Argument Callouts in Error Messages](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f89816)

**Functions for Working with Methods**

Use the following MATLAB functions to find out what methods a COM object has and to invoke any of these methods on the object.

| **Function** | **Description** |
| --- | --- |
| [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html) | Invoke a method or display a list of methods and types |
| [ismethod](http://www.kxcad.net/cae_MATLAB/techdoc/ref/ismethod.html) | Determine if an item is a method of a COM object |
| [methods](http://www.kxcad.net/cae_MATLAB/techdoc/ref/methods.html) | List all method names for the control or server |
| [methodsview](http://www.kxcad.net/cae_MATLAB/techdoc/ref/methodsview.html) | GUI interface to list information on all methods and types |

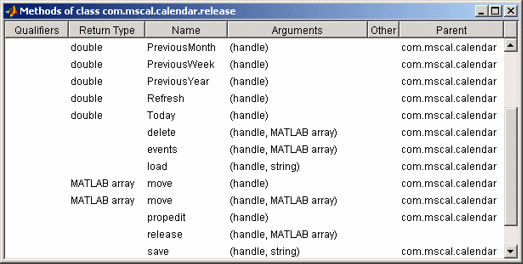
**Listing the Methods of a Class or Object**

You can see what methods are supported by a control or server object either in a graphical display using the methodsview function, or in a returned cell array using the methods function.

**Using methodsview.** The [methodsview](http://www.kxcad.net/cae_MATLAB/techdoc/ref/methodsview.html) function opens a new window with an easy-to-read display of all methods supported by the specified control or server object along with several related fields of information. Type the following to bring up a window such as the one shown below:

cal = actxcontrol('mscal.calendar', [0 0 400 400]);

cal.methodsview



Methods that return void show no **Return Type** in the display.

**Using methods.** The [methods](http://www.kxcad.net/cae_MATLAB/techdoc/ref/methods.html) function returns in a cell array the names of all methods supported by the specified control or server object. This includes MATLAB COM functions that you can use on the object.

When you include the -full switch in the command, MATLAB also specifies the input and output arguments for each method:

cal.methods('-full')

Methods for class COM.mscal.calendar:

release(handle, MATLAB array)

delete(handle, MATLAB array)

MATLAB array events(handle, MATLAB array)

.

.

HRESULT Refresh(handle)

HRESULT Today(handle)

HRESULT AboutBox(handle)

The [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html) function, when called with only a handle argument, returns a similar output.

**Invoking Methods on an Object**

To execute, or *invoke*, any method on an object, use either the MATLAB invoke function, or the somewhat simpler method name syntax.

**Using invoke.** The [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html) function executes the specified method on an object. You can use either of the following syntaxes with invoke:

v = invoke(handle, 'methodname', 'arg1', 'arg2', ...);

v = handle.invoke('methodname', 'arg1', 'arg2', ...);

See the output of [methodsview](http://www.kxcad.net/cae_MATLAB/techdoc/ref/methodsview.html) for a method to see what data types to use for input and output arguments.

The following example reads today's date and then advances it by five years by invoking the NextYear method in a loop.

To get today's date, type

cal = actxcontrol('mscal.calendar', [0 0 400 400]);

cal.Value

ans =

11/5/2001

Call the NextYear method to advance the date, and verify the results:

for k = 1:5

cal.invoke('NextYear');

end

cal.Value

ans =

11/5/2006

**Using the Method Name.** Instead of using invoke, you can just use the name of the method to call it. The syntax for calling by method name is

v = handle.methodname('arg1', 'arg2', ...);

or

v = methodname(handle, 'arg1', 'arg2', ...);

Continuing the example shown in the last section, return to the original data by going back five years.

for k = 1:5

cal.PreviousYear;

end

cal.Value

ans =

11/5/2001

**Specifying Enumerated Parameters**

Enumeration is a way of representing a somewhat cryptic symbolic value by using a more descriptive name that makes it clear what the value stands for. For example, a program that takes atomic numbers of elements as input is easier to work with if the program accepts element names as input rather than requiring you to recall and pass atomic numbers for each element. You can pass the word 'arsenic' as an enumeration for the value 33.

MATLAB supports enumeration for parameters passed to methods under the condition that the type library in use reports the parameter as ENUM, and only as ENUM.

|  |
| --- |
| **Note**    MATLAB does not support enumeration for any parameter that the type library reports as both ENUM and Optional. |

The last line of this example passes an enumerated value ('xlLocationAsObject') to the Location method of a Microsoft Excel Chart object. You have the choice of passing the enumeration or its numeric equivalent:

e = actxserver('Excel.Application');

% Insert a new workbook.

Workbook = e.Workbooks.Add;

e.Visible = 1;

Sheets = e.ActiveWorkBook.Sheets;

% Get a handle to the active sheet.

Activesheet = e.Activesheet;

%Add a Chart

Charts = Workbook.Charts;

Chart = Charts.Add;

% Set chart type to be a line plot.

Chart.ChartType = 'xlXYScatterLines'

C1 = Chart.Location('xlLocationAsObject', 'Sheet1');

When you are dealing with only three numeric values, it is not that difficult to remember the meaning of each. But with programs that require a large number of such values, enumeration becomes more important.

**Optional Input Arguments**

When calling a method that takes optional input arguments, you can skip any optional argument by specifying an empty array ([]) in its place. The syntax for calling a method with second argument (arg2) not specified is as follows:

handle.methodname(arg1, [], arg3);

The example below invokes the Add method of an Excel object. This method adds new sheets to an Excel workbook. The Add method takes up to four optional input arguments:

* Before — The sheet before which to add the new sheet
* After — The sheet after which to add the new sheet
* Count — The total number of sheets to add
* Type — The type of sheet to add

The following code creates a workbook with the default number of worksheets, and inserts an additional sheet after Sheet 1. To do this, you invoke Add specifying only the second argument, After. You can omit the first argument, Before, by using [] in its place. This is done on the last line:

% Open an Excel Server.

e = actxserver('excel.application');

% Insert a new workbook.

e.Workbooks.Add;

e.Visible = 1;

% Get the Active Workbook with three sheets.

eSheets = e.ActiveWorkbook.Sheets;

% Add a new sheet after eSheet1.

eSheet1 = eSheets.Item(1);

eNewSheet = eSheets.Add([], eSheet1);

**Returning Multiple Output Arguments**

If you know that a server function supports multiple outputs, you can return any or all of those outputs to a MATLAB client. Specify the output arguments within brackets on the left side of the equation. This gives the MATLAB client code access to any values returned by the server function.

The syntax shown here shows a server function being called by the MATLAB client. The function's return value is shown as retval. The function's output arguments (out1, out2, ...) follow this:

[retval out1 out2 ...] = handle.functionname(in1, in2, ...);

MATLAB makes use of the pass-by-reference capabilities in COM to implement this feature. Note that pass by reference is a COM feature. It is not available in MATLAB at this time.

**Argument Callouts in Error Messages**

When a MATLAB client sends a command with an invalid argument to a COM server application, the server sends back an error message similar to that shown here, identifying the invalid argument. Be careful when interpreting the argument callout in this type of message:

PutFullMatrix(handle, 'a', 'base', 7, [5 8]);

??? Error: Type mismatch, argument 3.

In the PutFullMatrix command shown above, the fourth argument, 7, is invalid. (It is scalar and not the expected array data type.) However, the error message identifies the failing argument as argument 3.

This is because the COM server receives only the last four of the arguments shown in the MATLAB code. (The handle argument merely identifies the server. It does not get passed to the server). So the server sees 'a' as the first argument, and the invalid argument, 7, as the third.

As another example, submitting the same command with the [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html) function makes the invalid argument fifth in the MATLAB client code. Yet the server still identifies it as argument 3 because neither of the first two arguments is seen by the server:

invoke(handle, 'PutFullMatrix', 'a', 'base', 7, [5 8]);

??? Error: Type mismatch, argument 3.

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Object Properties**

This section covers the following topics describing how to set and get the value of a property, and how to create custom properties:

* [Functions for Working with Object Properties](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f57811)
* [Getting the Value of a Property](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f68519)
* [Setting the Value of a Property](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f68490)
* [Properties That Take Arguments](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f118333)
* [Get and Set on a Vector of Objects](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f85093)
* [Using Enumerated Values for Properties](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f85097)
* [Using the Property Inspector](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f60795)
* [Custom Properties](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f71913)

**Functions for Working with Object Properties**

Use these MATLAB functions to get, set, and modify the properties of a COM object or interface, or to add your own custom properties.

| **Function** | **Description** |
| --- | --- |
| [addproperty](http://www.kxcad.net/cae_MATLAB/techdoc/ref/addproperty.html) | Add a custom property to a COM object |
| [deleteproperty](http://www.kxcad.net/cae_MATLAB/techdoc/ref/deleteproperty.html) | Remove a custom property from a COM object |
| [get](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.get.html) | List one or more properties and their values |
| [inspect](http://www.kxcad.net/cae_MATLAB/techdoc/ref/inspect.html) | Display graphical interface to list and modify property values |
| [isprop](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isprop.html) | Determine if an item is a property of a COM object |
| [propedit](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.propedit.html) | Display the control's built-in property page |
| [set](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.set.html) | Set a property on an interface |

**Getting the Value of a Property**

The [get](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.get.html) function returns information on one or more properties belonging to a COM object or interface. Use get with only the handle argument, and MATLAB returns a list of all properties for the object, and their values:

h = actxserver('excel.application');

h.get

Application: [1x1 Interface.Microsoft\_Excel\_9.0\_

Object\_Library.\_Application]

Creator: 'xlCreatorCode'

Parent: [1x1 Interface.Microsoft\_Excel\_9.0\_

Object\_Library.\_Application]

ActiveCell: []

ActiveChart: [1x50 char]

.

.

To return the value of just one property, specify the object handle and property name using dot syntax:

company = h.OrganizationName

company =

The MathWorks, Inc.

Property names are not case sensitive and may also be abbreviated, as long as you include enough letters in the name to make it unambiguous. You can use 'org' in place of the full 'OrganizationName' property name used above:

company = h.org

company =

The MathWorks, Inc.

You can also use the [get](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.get.html) function, without dot syntax, for this same purpose:

filepath = h.get('DefaultFilePath')

filepath =

H:\Documents

**Getting Multiple Property Values.** To get more than one property with just one command, you must use the get function, specifying the property names in a cell array of strings. This returns a cell array containing a column for each property value:

C = h.get({'prop1', 'prop2', ...});

For example, to get the DefaultFilePath and UserName property values for COM object h, use

h = actxserver('excel.application');

C = h.get({'DefaultFilePath', 'UserName'});

C{:}

ans =

H:\Documents

ans =

C. Coolidge

**Setting the Value of a Property**

The simplest way to set or modify the value of a property is to use an assignment statement like that shown in the second line below. This sets the value of the DefaultFilePath property for object h to 'C:\ExcelWork':

h = actxserver('excel.application');

h.DefaultFilePath = 'C:\ExcelWork';

You can also use the [set](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.set.html) function, without dot syntax, for this same purpose. Specify both the property name and new value as input arguments to set:

h.set('DefaultFilePath', 'C:\ExcelWork');

**Setting Multiple Property Values.** To change more than one property with one command, you must use the set function:

h.set('prop1', 'value1', 'prop2', 'value2', ...);

For example, to set the DefaultFilePath and UserName fields of COM object h, use

h = actxserver('excel.application');

h.set('DefaultFilePath', 'C:\ExcelWork', ...

'UserName', 'H. Hoover');

**Properties That Take Arguments**

Some COM objects have properties that behave somewhat like methods in that they accept input arguments. On a get or set operation, the value they end up getting or setting depends on the arguments you pass in.

The Activesheet interface of a Microsoft Excel application running as a COM server is one example. This interface has a property called Range, which is actually another interface. In order to get the correct Range interface, you must pass in specific range coordinates.

The first line of code shown here (taken from the example in [Using MATLAB as an Automation Client](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f62659)) returns a specific Range interface. Arguments A1 and B2 specify which rectangular region of the spreadsheet to get the interface for:

eActivesheetRange = e.Activesheet.get('Range', 'A1', 'B2')

eActivesheetRange =

Interface.Microsoft\_Excel\_5.0\_Object\_Library.Range

To get or set this type of property, use the get or set function as shown above for the Range property. Enter the input arguments in the parentheses following the property name:

handle.get(propertyname, arg1, arg2, ...);

In some ways, MATLAB handles these properties internally as though they were actually methods. The most important difference is that you need to use [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html), not get, to view the property:

e.Activesheet.invoke

:

Range = handle Range(handle, Variant, Variant(Optional))

:

**Get and Set on a Vector of Objects**

You can use the get and set functions on more than one object at a time by putting the object handles into a vector and then operating on the vector.

This example creates a vector of handles to four Microsoft Calendar objects. It then modifies the Day property of all the objects in one operation by invoking set on the vector:

h1 = actxcontrol('mscal.calendar', [0 200 250 200]);

h2 = actxcontrol('mscal.calendar', [250 200 250 200]);

h3 = actxcontrol('mscal.calendar', [0 0 250 200]);

h4 = actxcontrol('mscal.calendar', [250 0 250 200]);

H = [h1 h2 h3 h4];

H.set('Day', 23)

H.get('Day')

ans =

[23]

[23]

[23]

[23]

|  |
| --- |
| **Note**    To get or set values for multiple objects, you must use the get and set functions explicitly. Syntax like H.Day is only supported for scalar objects. |

**Using Enumerated Values for Properties**

Enumeration makes examining and changing properties easier because each possible value for the property is given a string to represent it. For example, one of the values for the DefaultSaveFormat property in an Excel application is xlUnicodeText. This is easier to remember than a numeric value like 57.

**Finding All Enumerated Properties.** The MATLAB COM get and set functions support enumerated values for properties for those applications that provide them. To see which properties use enumerated types, use the [set](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.set.html) function with the object handle argument:

h = actxserver('excel.application');

h.set

ans =

Creator: {'xlCreatorCode'}

ConstrainNumeric: {}

CopyObjectsWithCells: {}

Cursor: {4x1 cell}

CutCopyMode: {2x1 cell}

.

.

MATLAB displays the properties that accept enumerated types as nonempty cell arrays. Properties for which there is a choice of settings are displayed as a multirow cell array, with one row per setting (see Cursor in the example above). Properties for which there is only one possible setting are displayed as a one row cell array (see Creator, above).

To display the current values of these properties, use [get](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.get.html) with just the object handle argument:

h.get

Creator: 'xlCreatorCode'

ConstrainNumeric: 0

CopyObjectsWithCells: 1

Cursor: 'xlDefault'

CutCopyMode: ''

.

.

**Setting an Enumerated Value.** To list all possible enumerated values for a specific property, use set with the property name argument. The output is a cell array of strings, one string for each possible setting of the specified property:

h.set('Cursor')

ans =

'xlIBeam'

'xlDefault'

'xlNorthwestArrow'

'xlWait'

To set the value of a property, assign the enumerated value to the property name:

handle.property = 'enumeratedvalue';

You can also use the set function with the property name and enumerated value:

handle.set('property', 'enumeratedvalue');

You have a choice of using the enumeration or its equivalent numeric value. You can abbreviate the enumeration string, as in the third line below, as long as you use enough letters in the string to make it unambiguous. Enumeration strings are also case insensitive.

Make the Excel spreadsheet window visible and then change the cursor from the MATLAB client. To see how the cursor has changed, you need to click the spreadsheet window. Either of the following assignments to h.Cursor sets the cursor to the Wait (hourglass) type:

h.Visible = 1;

h.Cursor = 'xlWait'

h.Cursor = 'xlw' % Abbreviated form of xlWait

Read the value of the Cursor property you have just set:

h.Cursor

ans =

xlWait

**Using the Property Inspector**

MATLAB provides a GUI to display and modify properties. Open the Property Inspector using either of these two methods:

* Call the [inspect](http://www.kxcad.net/cae_MATLAB/techdoc/ref/inspect.html) function from the MATLAB command line.
* Double-click the object in the MATLAB Workspace browser.

For example, create a server object running Microsoft Excel, then set the object's DefaultFilePath property to 'C:\ExcelWork':

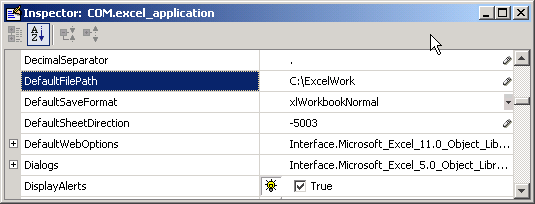
h = actxserver('excel.application');

h.DefaultFilePath = 'C:\ExcelWork';

Next call the inspect function to display a new window showing the server object's properties:

h.inspect

Scroll down until you see the DefaultFilePath property that you just changed. It should read C:\ExcelWork.



Using the Property Inspector, change DefaultFilePath once more, this time to C:\MyWorkDirectory. To do this, select the value at the right and type the new value.

Now go back to the MATLAB Command Window and confirm that the DefaultFilePath property has changed as expected.

h.DefaultFilePath

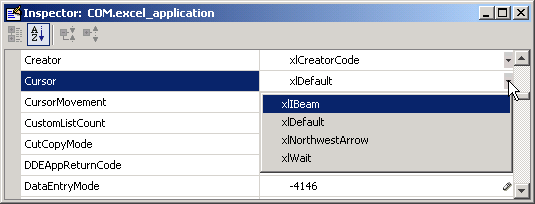
MATLAB displays:

ans =

C:\MyWorkDirectory

|  |
| --- |
| **Note**    If you modify properties at the MATLAB command line, you must refresh the Property Inspector window to see the change reflected there. Refresh the Property Inspector window by reinvoking inspect on the object. |

**Using the Property Inspector on Enumerated Values.** A drop-down list button next to a property value indicates the property accepts enumerated values. Click anywhere in the field on the right to see the values. The following figure displays four enumerated values for the Cursor property. The current value is displayed in the field next to the property name.



To change a property's value using the Property Inspector, use the drop-down list to display the options for that property, and then click the desired value.

**Custom Properties**

You can attach your own properties to a control using the [addproperty](http://www.kxcad.net/cae_MATLAB/techdoc/ref/addproperty.html) function. The syntax shown here creates a custom property for control, h:

h.addproperty('propertyName')

This example creates the mwsamp2 control, adds a new property called Position to it, and assigns the value [200 120] to that property:

h = actxcontrol('mwsamp.mwsampctrl.2', [200 120 200 200]);

h.addproperty('Position');

h.Position = [200 120];

Use [get](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.get.html) to list all properties of control, h. You see that the new Position property has been added:

h.get

ans =

Label: 'Label'

Radius: 20

Position: [200 120]

h.Position

ans =

200 120

To remove custom properties from a control, use [deleteproperty](http://www.kxcad.net/cae_MATLAB/techdoc/ref/deleteproperty.html) with the following syntax:

h.deleteproperty('propertyName')

For example, delete the Position property that you just created, and use get to show that it no longer exists:

h.deleteproperty('Position');

h.get

Label: 'Label'

Radius: 20

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Control and Server Events**

An *event* is typically a user-initiated action that takes place in a server application which often requires a reaction from the client. For example, a user clicking the mouse at a particular location in a server interface window might require the client take some action in response. When an event is *fired*, the server communicates this occurrence to the client. If the client is *listening* for this particular type of event, it responds by executing a routine called an event handler.

The MATLAB COM client can subscribe to and handle the events fired by an ActiveX control or a COM server. Select which events you want the client to listen to by registering each event you want active along with the event handler to be used in responding to the event. When any registered event takes place in the control or server, the client is notified and responds by executing the appropriate handler routine. Event handlers in MATLAB are often implemented using M-files.

This section covers the following topics on registering and responding to events fired from an ActiveX control or a COM server:

* [Functions for Working with Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f108387)
* [Examples of Event Handlers](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bqo0p0z)
* [Responding to Events from a COM Server](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f108692)
* [Responding to Events from an ActiveX Control](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f113200)
* [Responding to Events from an Automation Server](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f112722)
* [Responding to Interface Events from an Automation Server](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4qmoy)

|  |
| --- |
| **Note**   MATLAB does not support interface events from a Custom server. |

**Functions for Working with Events**

Use these MATLAB functions to register and unregister events, to list all events, or to list just registered events for a control or server.

| **Function** | **Description** |
| --- | --- |
| [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) | Create a COM control and optionally register those events you want the client to listen to |
| [eventlisteners](http://www.kxcad.net/cae_MATLAB/techdoc/ref/eventlisteners.html) | Return a list of events attached to listeners |
| [events](http://www.kxcad.net/cae_MATLAB/techdoc/ref/events.html) | List all events, both registered and unregistered, a control or server can generate |
| [isevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isevent.html) | Determine if an item is an event of a COM object |
| [registerevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/registerevent.html) | Register an event handler with a control or server event |
| [unregisterallevents](http://www.kxcad.net/cae_MATLAB/techdoc/ref/unregisterallevents.html) | Unregister all events for a control or server |
| [unregisterevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/unregisterevent.html) | Unregister an event handler with a control or server event |

When using these functions, enter event names and event handler names as strings or in a cell array of strings. These names are case insensitive, but cannot be abbreviated.

**Examples of Event Handlers**

The following examples have implementations of event handlers:

* [Example — Grid ActiveX Control in a Figure](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/bqdwu3j.html#bqd8yq0)
* [Example — Reading Data from Excel](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/bqdwu3j.html#bqd_o32)

**Responding to Events from a COM Server**

This section describes the basic steps you need to take in handling events fired by a COM control or server.

* [Identifying All Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f113530)
* [Registering Those Events You Want to Respond To](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4gepy-1)
* [Identifying Registered Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f111316)
* [Responding to Events As They Occur](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f111372)
* [Unregistering Events You No Longer Want to Listen To](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f111457)

**Identifying All Events.** Use the [events](http://www.kxcad.net/cae_MATLAB/techdoc/ref/events.html) function to list all events the control or server is capable of responding to. This function returns a structure array, where each field of the structure is the name of an event handler and the value of that field contains the signature for the handler routine. To invoke events on an object with handle h, type

S = h.events

**Registering Those Events You Want to Respond To.** Use the [registerevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/registerevent.html) function to register those server events you want the client to respond to. There are two ways you can register events:

* If you have one function to handle all server events, you can register this common event handler using the syntax

h.registerevent('handler');

* If you have a separate event handler function for different events, use the syntax
* h.registerevent({'event1' 'handler1'; 'event2' 'handler2';

...});

For ActiveX controls, you can register events at the time you create the control using the [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) function.

* To register a common event handler function to respond to all events, use

h = actxcontrol('progid', position, figure, 'handler');

* To register a separate function to handle each type of event, use
* h = actxcontrol('progid', position, figure, ...

{'event1' 'handler1'; 'event2' 'handler2'; ...});

The MATLAB client responds only to events you have registered.

**Identifying Registered Events.** The [eventlisteners](http://www.kxcad.net/cae_MATLAB/techdoc/ref/eventlisteners.html) function lists only those events that are currently registered. This function returns a cell array, with each row representing a registered event and the name of its event handler. For example, to invoke eventlisteners on an object with handle h, type

C = h.eventlisteners

**Responding to Events As They Occur.** Whenever a control or server fires an event that the client is listening to, the client responds to the event by invoking one or more event handlers that have been registered for that event. You can implement these routines in M-file programs that you write to handle events. Read more about event handlers in the section on [Writing Event Handlers](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f89992).

**Unregistering Events You No Longer Want to Listen To.** If you have registered events that you now want the client to ignore, you can unregister them at any time using the functions [unregisterevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/unregisterevent.html) and [unregisterallevents](http://www.kxcad.net/cae_MATLAB/techdoc/ref/unregisterallevents.html) as follows:

* For a server with handle h, to unregister all events registered with a common event handling function handler, use

h.unregisterevent('handler');

* To unregister individual events eventN, each registered with its own event handling function handlerN, use

h.unregisterevent({'event1' 'handler1'; 'eventN' 'handlerN'});

* To unregister all events from the server regardless of which event handling function they are registered with, use

h.unregisterallevents;

**Responding to Events from an ActiveX Control**

This section describes how to handle events fired by an ActiveX control. It uses a control called mwsamp2 that ships with MATLAB. The event handler routines for mwsamp2 are defined when you install MATLAB.

Tasks described in this section are

* [Creating a Control and Registering Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f112757)
* [Listing Control Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f112727)
* [Responding to Control Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f112769)
* [Unregistering Control Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f112778)

**Creating a Control and Registering Events.** The [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) function not only creates the control object, but can be used to register specific events as well. The code shown here registers two events (Click and MouseDown) and two respective handler routines (myclick and mymoused) with the mwsamp2 control.

f = figure('position', [100 200 200 200]);

h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], f, ...

{'Click' 'myclick'; 'MouseDown' 'mymoused'});

If, at some later time, you want to register additional events, use the [registerevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/registerevent.html) function:

h.registerevent({'DblClick' 'my2click'});

You can view the event handler code written for the mwsamp2 control in the section [Sample Event Handlers](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f90131).

Unregister the DblClick event before continuing with the example:

h.unregisterevent({'DblClick' 'my2click'});

**Listing Control Events.** At this point, only the Click and MouseDown events should be registered. To see all events that the control can fire, use the events function. This function returns a structure array, where each field of the structure is the name of an event handler and the value of that field contains the signature for the handler routine.

To list all events, whether registered or not, use

S = h.events

S =

Click: 'void Click()'

DblClick: 'void DblClick()'

MouseDown: 'void MouseDown(int16 Button, int16 Shift,

Variant x, Variant y)'

Event\_Args: [1x101 char]

S.Event\_Args

ans =

void Event\_Args(int16 typeshort, int32 typelong,

double typedouble, string typestring, bool typebool)

To list only those events that are currently registered with the control, use the [eventlisteners](http://www.kxcad.net/cae_MATLAB/techdoc/ref/eventlisteners.html) function. This function returns a cell array, with each row representing a registered event and the name of its event handler.

Use eventlisteners to list registered event names and their handler routines:

h.eventlisteners

ans =

'click' 'myclick'

'mousedown' 'mymoused'

**Responding to Control Events.** When MATLAB creates the mwsamp2 control, it also displays a figure window showing a label and circle at the center. If you click on different positions in this window, you see a report in the MATLAB Command Window of the X and Y position of the mouse.

Each time you press the mouse button, the MouseDown event fires, calling the mymoused function. This function prints the position values for that event to the MATLAB Command Window:

The X position is:

ans =

[122]

The Y position is:

ans =

[63]

You also see the following line reported in response to the Click event:

Single click function

Double-clicking the mouse does nothing different, since the DblClick event has been unregistered.

**Unregistering Control Events.** When you unregister an event, the client discontinues listening for occurrences of that event. When the event fires, the client does not respond. If you unregister the MouseDown event, MATLAB no longer reports the X and Y position when you click in the window:

h.unregisterevent({'MouseDown' 'mymoused'});

Now, register the DblClick event, connecting it with handler function my2click:

h.registerevent({'DblClick', 'my2click'});

If you call [eventlisteners](http://www.kxcad.net/cae_MATLAB/techdoc/ref/eventlisteners.html) again, the registered events are now Click and DblClick:

h.eventlisteners

ans =

'click' 'myclick'

'dblclick' 'my2click'

When you double-click the mouse button, MATLAB responds to both the Click and DblClick events by displaying the following in the MATLAB Command Window:

Single click function

Double click function

An easy way to unregister all events for a control is to use the [unregisterallevents](http://www.kxcad.net/cae_MATLAB/techdoc/ref/unregisterallevents.html) function. When there are no events registered, eventlisteners returns an empty cell array:

h.unregisterallevents

h.eventlisteners

ans =

{}

Clicking the mouse in the control window now does nothing since there are no active events.

If you have events that are registered with a common event handling routine, such as sampev.m used in the example below, you can use unregisterevent to unregister all of these events in one operation. The example that follows first registers all events from the server with a common handling routine sampev. MATLAB now handles any type of event from this server by executing sampev:

h.registerevent('sampev');

Verify the registration by listing all event listeners for that server:

h.eventlisteners

ans =

'click' 'sampev'

'dblclick' 'sampev'

'mousedown' 'sampev'

Now unregister all events for the server that use the sampev event handling routine:

h.unregisterevent('sampev');

h.eventlisteners

ans =

{}

**Responding to Events from an Automation Server**

The next section shows how to handle events fired by an Automation server. The example creates a server running Internet Explorer, registers a common handler for all events, and then has you fire events by browsing to Web sites using the Internet Explorer application.

Tasks described in this section are

* [Creating an Event Handler](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4erzj-1)
* [Creating a Server](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4hvpq-1)
* [Listing Server Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4hu8c-1)
* [Registering Server Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4ery1-1)
* [Responding to Server Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f112789)
* [Unregistering Server Events](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f112791)
* [Closing the Application](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f118871)

**Creating an Event Handler.** This example registers all events with the same handler routine, serverevents. Since this example does not ship with MATLAB, you have to create the event handler routine yourself.

Create the file serverevents.m, inserting the following code. Make sure the file is in your current directory.

function serverevents(varargin)

% Display incoming event name

eventname = varargin{end}

% Display incoming event args

eventargs = varargin{end-1}

**Creating a Server.** Next, in your MATLAB session, use the following commands to create your Automation server application.

% Create a server running Internet Explorer.

h = actxserver('internetexplorer.application');

% Make the server application visible.

h.set('Visible', 1);

**Listing Server Events.** Use the [events](http://www.kxcad.net/cae_MATLAB/techdoc/ref/events.html) function to list all events the control or server is capable of responding to, and [eventlisteners](http://www.kxcad.net/cae_MATLAB/techdoc/ref/eventlisteners.html) to list only those events that are currently registered.

h.events

MATLAB displays event information similar to:

:

StatusTextChange = void StatusTextChange(string Text)

ProgressChange = void ProgressChange(int32 Progress,int32 ProgressMax)

CommandStateChange = void CommandStateChange(int32 Command,bool Enable)

:

No events are registered at this time, so [eventlisteners](http://www.kxcad.net/cae_MATLAB/techdoc/ref/eventlisteners.html) returns an empty cell array.

h.eventlisteners

MATLAB displays:

ans =

{}

**Registering Server Events.** Now use your event handler serverevents.

h.registerevent('serverevents');

h.eventlisteners

MATLAB displays:

ans =

: :

'statustextchange' 'serverevents'

'progresschange' 'serverevents'

'commandstatechange' 'serverevents'

: :

**Responding to Server Events.** At this point, all events have been registered. If any event fires, the common handler routine defined in serverevents.m executes to handle that event. Use the Internet Explorer application to browse your favorite Web site, or enter the following command in the MATLAB Command Window:

h.Navigate2('http://www.mathworks.com');

You should see a long series of events displayed in your client window.

**Unregistering Server Events.** Use the [unregisterevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/unregisterevent.html) function to remove specific events from the registry. If the events were registered with a common handler, as in this example, specify the name of the common routine with each event that you want removed from the event registry for that object:

h.unregisterevent({'event1', 'commonhandler'; ...

'event2', 'commonhandler', ...});

Continuing with this example, unregister the progresschange and commandstatechange events:

h.unregisterevent({'progresschange', 'serverevents'; ...

'commandstatechange', 'serverevents'});

To unregister all events for an object, use [unregisterallevents](http://www.kxcad.net/cae_MATLAB/techdoc/ref/unregisterallevents.html). The following commands unregister all the events that had been registered for the Internet Explorer application and then register a single event:

h.unregisterallevents;

h.registerevent({'TitleChange', 'serverevents'});

If you now browse with Internet Explorer, MATLAB only responds to the TitleChange event.

**Closing the Application.** It is always advisable to close a server application when you no longer intend to use it. To unregister all events and close the application, type:

h.unregisterallevents;

h.Quit;

h.delete;

**Responding to Interface Events from an Automation Server**

This example, demonstrating how to handle a COM interface event, shows how to set up an event in a Microsoft Excel's Workbook object and how to handle its BeforeClose event.

To create the event handler OnBeforeCloseWorkbook, create the file OnBeforeCloseWorkbook.m, inserting the following code. Make sure the file is in your current directory.

% Event handler for Excel workbook BeforeClose event

function OnBeforeCloseWorkbook(varargin)

disp('BeforeClose event occured');

When you run the following commands

% Create Excel automation server instance

xl = actxserver('Excel.Application');

% Make it visible

xl.Visible = 1;

% Get collection of workbooks and add a new workbook

hWbks = xl.Workbooks;

hWorkbook = hWbks.Add;

% Register OnClose event

hWorkbook.registerevent({'BeforeClose' @OnBeforeCloseWorkbook});

%% Close the workbook. This will fire Close event and call OnClose handler

hWorkbook.Close

MATLAB displays:

BeforeClose event occured

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Writing Event Handlers**

This section covers the following topics on writing handler routines to respond to events fired from an ActiveX control or Automation server:

* [Overview of Event Handling](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f110799)
* [Arguments Passed to Event Handlers](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f90010)
* [Event Structure](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f90070)
* [Sample Event Handlers](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f90131)
* [Writing Event Handlers Using M-File Subfunctions](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f107572)

**Overview of Event Handling**

An event is fired when a control or server wants to notify its client that something of interest has occurred. For example, many controls trigger an event when the user clicks somewhere in the interface window of a control. In MATLAB, you can create and register your own M-file functions to respond to events when they occur. These functions serve as event handlers. You can create one handler function to handle all events or a separate handler for each type of event.

For controls, you can register handler functions either at the time you create the control (using [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html)), or at any time afterwards (using [registerevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/registerevent.html)).

Both [actxcontrol](http://www.kxcad.net/cae_MATLAB/techdoc/ref/actxcontrol.html) and [registerevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/registerevent.html) use an event handler argument. The event handler argument can be either the name of a single callback routine or a cell array that associates specific events with their respective event handlers. Strings used in the event handler argument are not case sensitive.

For servers, you must use [registerevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/registerevent.html) to register those events you want the client to listen to. For example, to register the Click and DblClick events, use

h.registerevent({'click' 'myclick'; 'dblclick' 'my2click'});

Use [events](http://www.kxcad.net/cae_MATLAB/techdoc/ref/events.html) to list all the events a COM object recognizes. For example, to list all events for the mwsamp2 control, use

f = figure ('position', [100 200 200 200]);

h = actxcontrol ('mwsamp.mwsampctrl.2', [0 0 200 200], f);

h.events

Click = void Click()

DblClick = void DblClick()

MouseDown = void MouseDown(int16 Button, int16 Shift,

Variant x, Variant y)

**Arguments Passed to Event Handlers**

When a registered event is triggered, MATLAB passes information from the event to its handler function, as shown in this table.

**Arguments Passed by MATLAB**

| **Arg. No.** | **Contents** | **Format** |
| --- | --- | --- |
| 1 | Object name | MATLAB COM class |
| 2 | Event ID | double |
| 3 | Start of Event Argument List | As passed by the control |
| end-2 | End of Event Argument List (Argument N) | As passed by the control |
| end-1 | Event Structure | structure |
| end | Event Name | char array |

When writing an event handler function, use the Event Name argument to identify the source of the event. Get the arguments passed by the control from the Event Argument List (arguments 3 through end-2). All event handlers must accept a variable number of arguments:

function event (varargin)

if (varargin{end}) == 'MouseDown') % Check the event name

x\_pos = varargin{5}; % Read 5th Event Argument

y\_pos = varargin{6}; % Read 6th Event Argument

end

|  |
| --- |
| **Note**    The values passed vary with the particular event and control being used. |

**Event Structure**

The second to last argument passed by MATLAB is the Event Structure, which has the following fields.

**Fields of the Event Structure**

| **Field Name** | **Description** | **Format** |
| --- | --- | --- |
| Type | Event Name | char array |
| Source | Control Name | MATLAB COM class |
| EventID | Event Identifier | double |
| Event Arg Name 1 | Event Arg Value 1 | As passed by the control |
| Event Arg Name 2 | Event Arg Value 2 | As passed by the control |
| etc. | Event Arg N | As passed by the control |

For example, when the MouseDown event of the mwsamp2 control is triggered, MATLAB passes this Event Structure to the registered event handler:

Type: 'MouseDown'

Source: [1x1 COM.mwsamp.mwsampctrl.2]

EventID: -605

Button: 1

Shift: 0

x: 27

y: 24

**Sample Event Handlers**

Specify a single callback, sampev:

f = figure('position', [100 200 200 200]);

h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], ...

gcf, 'sampev')

h =

COM.mwsamp.mwsampctrl.2

Or specify several events using the cell array format:

h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], f, ...

{'Click' 'myclick'; 'DblClick' 'my2click'; ...

'MouseDown' 'mymoused'});

The event handlers, myclick.m, my2click.m, and mymoused.m, are

function myclick(varargin)

disp('Single click function')

function my2click(varargin)

disp('Double click function')

function mymoused(varargin)

disp('You have reached the mouse down function')

disp('The X position is: ')

double(varargin{5})

disp('The Y position is: ')

double(varargin{6})

Alternatively, you can use the same event handler for all the events you want to monitor using the cell array pairs. Response time is better than using the callback style.

For example:

f = figure('position', [100 200 200 200]);

h = actxcontrol('mwsamp.mwsampctrl.2', ...

[0 0 200 200], f, {'Click' 'allevents'; ...

'DblClick' 'allevents'; 'MouseDown' 'allevents'})

where allevents.m is

function allevents(varargin)

if (strcmp(varargin{end-1}.Type, 'Click'))

disp ('Single Click Event Fired')

elseif (strcmp(varargin{end-1}.Type, 'DblClick'))

disp ('Double Click Event Fired')

elseif (strcmp(varargin{end-1}.Type, 'MouseDown'))

disp ('Mousedown Event Fired')

end

**Writing Event Handlers Using M-File Subfunctions**

Instead of having to maintain a separate M-file for every event handler routine you write, you can consolidate some or all of these routines into a single M-file using M-file subfunctions.

This example shows three event handler routines, (myclick, my2click, and mymoused) implemented as subfunctions in the file mycallbacks.m. The call to [str2func](http://www.kxcad.net/cae_MATLAB/techdoc/ref/str2func.html) converts the input string to a function handle:

function a = mycallbacks(str)

a = str2func(str);

function myclick(varargin)

disp('Single click function')

function my2click(varargin)

disp('Double click function')

function mymoused(varargin)

disp('You have reached the mouse down function')

disp('The X position is: ')

double(varargin{5})

disp('The Y position is: ')

double(varargin{6})

To register one of these events, call mycallbacks, passing the name of the event handler:

h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], ...

gcf, 'sampev')

h.registerevent({'Click', mycallbacks('myclick')});

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Saving Your Work**

Use these MATLAB functions to save and restore the state of a COM control object.

| **Function** | **Description** |
| --- | --- |
| [load](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.load.html) | Load and initialize a COM control object from a file |
| [save](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.save.html) | Write and serialize a COM control object to a file |

Save the current state of a COM control to a file using the save function. The following example creates an mwsamp2 control and saves its original state to the file mwsample:

f = figure('position', [100 200 200 200]);

h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], f);

h.save('mwsample')

Now, alter the figure by changing its label and the radius of the circle:

h.Label = 'Circle';

h.Radius = 50;

h.Redraw;

Using the load function, you can restore the control to its original state:

h.load('mwsample');

h.get

ans =

Label: 'Label'

Radius: 20

|  |
| --- |
| **Note**    The COM save and load functions are only supported for controls at this time. |

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Releasing COM Interfaces and Objects**

Use these MATLAB functions to release or delete a COM object or interface.

| **Function** | **Description** |
| --- | --- |
| [delete](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.delete.html) | Delete a COM object or interface |
| [release](http://www.kxcad.net/cae_MATLAB/techdoc/ref/release.html) | Release a COM object or interface |

When each interface is no longer needed, use the [release](http://www.kxcad.net/cae_MATLAB/techdoc/ref/release.html) function to release the interface and reclaim the memory used by it. When the entire control or server is no longer needed, use the [delete](http://www.kxcad.net/cae_MATLAB/techdoc/ref/com.delete.html) function to delete it. Alternatively, you can use the delete function on any valid interface. All interfaces for that object are automatically released and the server or control itself is deleted.

|  |
| --- |
| **Note**    In versions of MATLAB earlier than 6.5, failure to explicitly release interface handles or delete the control or server often results in a memory leak. This is true even if the variable representing the interface or COM object has been reassigned. In MATLAB 6.5 and later, the control or server, along with all interfaces to it, is destroyed on reassignment of the variable or when the variable representing a COM object or interface goes out of scope. |

MATLAB automatically releases all interfaces for a control when the figure window that contains that control is deleted or closed. MATLAB also automatically releases all handles for an Automation server when MATLAB is shut down.

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Identifying Objects**

Use these MATLAB functions to get information about a COM object.

| **Function** | **Description** |
| --- | --- |
| [class](http://www.kxcad.net/cae_MATLAB/techdoc/ref/class.html) | Return the class of a COM object |
| [isa](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isa.html) | Detect a COM object of a given class |
| [isevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isevent.html) | Determine if an item is an event of a COM object |
| [ismethod](http://www.kxcad.net/cae_MATLAB/techdoc/ref/ismethod.html) | Determine if an item is a method of a COM object |
| [isprop](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isprop.html) | Determine if an item is a property of a COM object |

Create a COM object, h, in an Automation server running Excel, and also a Workbooks interface, w, to the object:

h = actxserver('excel.application');

w = h.Workbooks;

To find out the class of variable w, use the [class](http://www.kxcad.net/cae_MATLAB/techdoc/ref/class.html) function:

w.class

ans =

Interface.Microsoft\_Excel\_9.0\_Object\_Library.Workbooks

To test a variable against a known class name, use [isa](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isa.html):

h.isa('COM.excel.application')

ans =

1

To see if UsableWidth is a property of object h, use [isprop](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isprop.html):

h.isprop('UsableWidth')

ans =

1

To see if SaveWorkspace is a method of object h, use [ismethod](http://www.kxcad.net/cae_MATLAB/techdoc/ref/ismethod.html). Method names are case sensitive and cannot be abbreviated:

h.ismethod('SaveWorkspace')

ans =

1

Create the sample mwsamp2 control that comes with MATLAB, and use [isevent](http://www.kxcad.net/cae_MATLAB/techdoc/ref/isevent.html) to see if DblClick is one of the events that this control recognizes:

f = figure ('position', [100 200 200 200]);

h = actxcontrol ('mwsamp.mwsampctrl.2', [0 0 200 200], f);

h.isevent('DblClick')

ans =

1

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Handling COM Data in MATLAB**

This section covers the following topics:

* [Passing Data to a COM Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4jkuh)
* [Handling Data from a COM Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4sbv3-1)
* [Unsupported Data Types](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4epvg-5)
* [Passing Data from MATLAB to ActiveX Objects](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq4hokx-1)
* [Passing SAFEARRAY from MATLAB to COM Object](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq552_r)
* [Reading SAFEARRAY from a COM Object in MATLAB](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq552_v)
* [Displaying MATLAB Syntax for COM Objects](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq9byb0-1)

**Passing Data to a COM Object**

When you call a COM object from MATLAB, the MATLAB data types you pass in the call are converted to data types native to the COM object. MATLAB performs this conversion on each argument that is passed. This section describes the conversion.

MATLAB arguments are converted by MATLAB into data types that best represent the data to the COM object. The following table shows all of the MATLAB base types for passed arguments and the COM types defined for input arguments. Each row shows a MATLAB type followed by the possible COM argument matches.

| **MATLAB Argument** | **Closest Type** | **Allowed Types** |
| --- | --- | --- |
| handle | VT\_DISPATCH VT\_UNKNOWN | VT\_DISPATCH VT\_UNKNOWN |
| string | VT\_BSTR | VT\_LPWSTR VT\_LPSTR VT\_BSTR VT\_FILETIME VT\_ERROR VT\_DECIMAL VT\_CLSIDVT\_DATE |
| int16 | VT\_I2 | VT\_UINT VT\_I2 VT\_UI2 |
| int32 | VT\_I4 | VT\_I4 VT\_UI4 VT\_INT |
| single | VT\_R4 | VT\_R4 |
| double | VT\_R8 | VT\_R8 VT\_CY (currency) |
| bool | VT\_BOOL | VT\_BOOL |
| char | VT\_I1 | VT\_I1 VT\_UI1 |

**Variant Data.** variant is any data type except a structure or a sparse array. (Refer to the [Data Type Summary](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_prog/f2-47534.html#bqt_gwr) table in the MATLAB Programming documentation.)

When used as an input argument, MATLAB treats variant and variant(pointer) the same way.

| **MATLAB Argument** | **Closest Type** | **Allowed Types** |
| --- | --- | --- |
| variant | VT\_VARIANT | VT\_VARIANT VT\_USERDEFINED VT\_ARRAY |
| variant(pointer) | VT\_VARIANT | VT\_VARIANT | VT\_BYREF |

**SAFEARRAY Data.** When a COM method identifies a SAFEARRAY or SAFEARRAY(pointer), the MATLAB equivalent is a matrix.

| **MATLAB Argument** | **Closest Type** | **Allowed Types** |
| --- | --- | --- |
| SAFEARRAY | VT\_SAFEARRAY | VT\_SAFEARRAY |
| SAFEARRAY(pointer) | VT\_SAFEARRAY | VT\_SAFEARRAY | VT\_BYREF |

**Handling Data from a COM Object**

Data returned from a COM object is often incompatible with MATLAB data types. When this occurs, MATLAB converts the returned value to a data type native to the MATLAB language. This section describes the conversion performed on the various data types that can be returned from COM objects.

This table shows how MATLAB converts data from a COM object into MATLAB variables.

| **COM Return Type** | **MATLAB Representation** |
| --- | --- |
| VT\_DISPATCH VT\_UNKNOWN | handle |
| VT\_LPWSTR VT\_LPSTR VT\_BSTR VT\_FILETIME VT\_ERROR VT\_DECIMAL VT\_CLSIDVT\_DATE | string |
| VT\_UINT VT\_I2 VT\_UI2 | int16 |
| VT\_I4 VT\_UI4 VT\_INT | int32 |
| VT\_R4 | single |
| VT\_R8 VT\_CY (currency) | double |
| VT\_BOOL | bool |
| VT\_I1 VT\_UI1 | char |
| VT\_VARIANT VT\_USERDEFINED VT\_ARRAY | variant |
| VT\_VARIANT | VT\_BYREF | variant(pointer) |
| VT\_SAFEARRAY | SAFEARRAY |
| VT\_SAFEARRAY | VT\_BYREF | SAFEARRAY(pointer) |

**Unsupported Data Types**

The following data types are not supported in the MATLAB COM interface:

* VT\_I8
* VT\_UI8
* Structure
* Sparse array
* Unsigned integer
* Multidimensional SAFEARRAYs
* Write-only properties
* Enumerated types

**Passing Data from MATLAB to ActiveX Objects**

The tables also show the mapping of MATLAB data types to COM data types that you must use to pass data from MATLAB to an ActiveX object. Note that all other types result in the following warning:

"ActiveX - invalid argument type or value".

**Passing SAFEARRAY from MATLAB to COM Object**

The SAFEARRAY data type is a standard way to pass arrays between COM objects. This section explains how MATLAB passes SAFEARRAY data to a COM object.

* [Default Behavior in MATLAB](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq553k4)
* [Examples](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq553k_)
* [How to Pass a Single-Dimension SAFEARRAY](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#bq553lb)
* [Passing SAFEARRAY By Reference](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#brc5jeu-1)

**Default Behavior in MATLAB.** MATLAB represents an m-by-n matrix as a two-dimensional SAFEARRAY, where the first dimension has m elements and the second dimension has n elements. MATLAB passes the SAFEARRAY by value.

**Examples.** The following examples use a COM object that expects a SAFEARRAY input parameter.

When MATLAB passes a 1-by-3 array

B = [2 3 4]

B =

2 3 4

the object reads

No. of dimensions: 2

Dim: 1, No. of elements: 1

Dim: 2, No. of elements: 3

Elements:

2.0

3.0

4.0

When MATLAB passes a 3-by-1 array

C = [1;2;3]

C =

1

2

3

the object reads

No. of dimensions: 2

Dim: 1, No. of elements: 3

Dim: 2, No. of elements: 1

Elements:

1.0

2.0

3.0

When MATLAB passes a 2-by-4 array

D = [2 3 4 5;5 6 7 8]

D =

2 3 4 5

5 6 7 8

the object reads

No. of dimensions: 2

Dim: 1, No. of elements: 2

Dim: 2, No. of elements: 4

Elements:

2.0

3.0

4.0

5.0

5.0

6.0

7.0

8.0

**How to Pass a Single-Dimension SAFEARRAY.** For information about passing arguments as one-dimensional arrays to a COM object, see the Technical Support solution 1-SKYP9 at <http://www.mathworks.com/support/solutions/data/1-SKYP9.html?solution=1-SKYP9>.

**Passing SAFEARRAY By Reference.** For information about passing arguments by reference to a COM object, see the Technical Support solution 1-SKYPY at <http://www.mathworks.com/support/solutions/data/1-SKYPY.html?solution=1-SKYPY>.

**Reading SAFEARRAY from a COM Object in MATLAB**

This section explains how MATLAB reads SAFEARRAY data from a COM object.

A one-dimensional SAFEARRAY with n elements from a COM object is rendered as a 1-by-n matrix. For example, using methods from the MATLAB sample control mwsamp,

h=actxcontrol('mwsamp.mwsampctrl.1')

a = h.GetI4Vector

MATLAB displays:

a =

1 2 3

A two-dimensional SAFEARRAY with n elements is rendered by MATLAB as a 2-by-n matrix, for example:

a = h.GetR8Array

MATLAB displays:

a =

1 2 3

4 5 6

A three-dimensional SAFEARRAY with 2 elements is rendered as a 2-by-2-by-2 cell array, for example:

a = h.GetBSTRArray

MATLAB displays:

a(:,:,1) =

'1 1 1' '1 2 1'

'2 1 1' '2 2 1'

a(:,:,2) =

'1 1 2' '1 2 2'

'2 1 2' '2 2 2'

**Displaying MATLAB Syntax for COM Objects**

To determine which MATLAB data types to use when passing arguments to COM objects, use the [invoke](http://www.kxcad.net/cae_MATLAB/techdoc/ref/invoke.html) or [methodsview](http://www.kxcad.net/cae_MATLAB/techdoc/ref/methodsview.html) functions. These functions list all of the methods found in an object, along with a specification of the data types required for each argument.

In the following example, a server called MyApp has a method TestMeth1 with the following syntax:

HRESULT TestMeth1 ([out, retval] double\* dret);

This method has no input argument, and it returns a variable of type double. To display the MATLAB syntax for calling the method, type

h = actxserver('MyApp');

h.invoke

MATLAB displays:

ans =

TestMeth1 = double TestMeth1 (handle)

The signature of TestMeth1 is

double TestMeth1(handle)

MATLAB requires you to use an object handle as an input argument for every method, in addition to any input arguments required by the method itself.

Using the variable var, which is of type double, type

var = h.TestMeth1;

or

var = TestMeth1(h);

While the following syntax is correct, its use is discouraged:

var = invoke(h,'TestMeth1');

Now consider the server called MyApp1 with the following methods:

HRESULT TestMeth1 ([out, retval] double\* dret);

HRESULT TestMeth2 ([in] double\* d, [out, retval] double\* dret);

HRESULT TestMeth3 ([out] BSTR\* sout,

[in, out] double\* dinout,

[in, out] BSTR\* sinout,

[in] short sh,

[out] long\* ln,

[in, out] float\* b1,

[out, retval] double\* dret);

Type the commands:

h = actxserver('MyApp1');

h.invoke

MATLAB displays the list of methods

ans =

TestMeth1 = double TestMeth1 (handle)

TestMeth2 = double TestMeth1 (handle, double)

TestMeth3 = [double, string, double, string, int32, single] ...

TestMeth3(handle, double, string, int16, single)

TestMeth2 requires an input argument d of type double, as well as returning a variable dret of type double. Some examples of calling TestMeth2 are

var = h.TestMeth2(5);

or

var = TestMeth2(h, 5);

TestMeth3 requires multiple input arguments, as indicated within the parentheses on the right side of the equals sign, and returns multiple output arguments, as indicated within the brackets on the left side of the equals sign.

[double, string, double, string, int32, single] %output arguments

TestMeth3(handle, double, string, int16, single) %input arguments

The first input argument is the required handle, followed by four input arguments.

TestMeth3(handle, in1, in2, in3, in4)

The first output argument is the return value retval, followed by five output arguments.

[retval, out1, out2, out3, out4, out5]

This is how the arguments map into a MATLAB command:

[dret, sout, dinout, sinout, ln, b1] = TestMeth3(handle, ...

dinout, sinout, sh, b1)

where dret is double, sout is string, dinout is double and is both an input and an output argument, sinout is string (input and output argument), ln is int32, b1 is single (input and output argument), handle is the handle to the object, and sh is int16.

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**Examples of MATLAB as an Automation Client**

This section provides examples of using MATLAB as an Automation client with controls and servers:

* [MATLAB Sample Control](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f62648)
* [Using MATLAB as an Automation Client](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f62659)
* [Connecting to an Existing Excel Application](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f134682)
* [Running a Macro in an Excel Server Application](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f133702)

**MATLAB Sample Control**

MATLAB ships with a simple example COM control that draws a circle on the screen, displays some text, and fires events when the user single- or double-clicks on the control. Create the control by running the mwsamp.m file in the directory, winfun\comcli, or type

h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 300 300]);

This control is stored in the MATLAB bin, or executable, directory along with the control's *type library*. The type library is a binary file used by COM tools to decipher the control's capabilities. See the section [Writing Event Handlers](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#f89992) for other examples that use the mwsamp2 control.

**Using MATLAB as an Automation Client**

This example uses MATLAB as an Automation client and Microsoft Excel as the server. It provides a good overview of typical functions. In addition, it is a good example of using the Automation interface of another application:

% MATLAB Automation client example

%

% Open Excel, add workbook, change active worksheet,

% get/put array, save.

% First, open an Excel Server.

e = actxserver('excel.application');

% Insert a new workbook.

eWorkbook = e.Workbooks.Add;

e.Visible = 1;

% Make the first sheet active.

eSheets = e.ActiveWorkbook.Sheets;

eSheet1 = eSheets.get('Item', 1);

eSheet1.Activate;

% Put a MATLAB array into Excel.

A = [1 2; 3 4];

eActivesheetRange = e.Activesheet.get('Range', 'A1:B2');

eActivesheetRange.Value = A;

% Get back a range.

% It will be a cell array, since the cell range

% can contain different types of data.

eRange = e.Activesheet.get('Range', 'A1:B2');

B = eRange.Value;

% Convert to a double matrix. The cell array must contain only

% scalars.

B = reshape([B{:}], size(B));

% Now, save the workbook.

eWorkbook.SaveAs('myfile.xls');

% Avoid saving the workbook and being prompted to do so

eWorkbook.Saved = 1;

eWorkbook.Close;

% Quit Excel and delete the server.

e.Quit;

e.delete;

|  |
| --- |
| **Note**    Make sure that you always close any workbooks that you add in Excel. This can prevent potential memory leaks. |

**Connecting to an Existing Excel Application**

You can give MATLAB access to a file that is open by another application by creating a new COM server from the MATLAB client, and then opening the file through this server. This example shows how to do this for an Excel application that has a file weekly\_log.xls open:

excelapp = actxserver('Excel.Application');

wkbk = excelapp.Workbooks;

wdata = wkbk.Open('d:\weatherlog\weekly\_log.xls');

To see what methods are available, type

wdata.methods

Methods for class Interface.Microsoft\_Excel\_10.0\_

Object\_Library.\_Workbook:

AcceptAllChanges LinkInfo ReloadAs

Activate LinkSources RemoveUser

: : :

: : :

Access data from the spreadsheet by selecting a particular sheet (called 'Week 12' in the example), selecting the range of values (the rectangular area defined by D1 and F6 here), and then reading from this range:

sheets = wdata.Sheets;

sheet12 = sheets.Item('Week 12');

range = sheet12.get('Range', 'D1', 'F6');

range.value

ans =

'Temp.' 'Heat Index' 'Wind Chill'

[78.4200] [ 32] [ 37]

[69.7300] [ 27] [ 30]

[77.6500] [ 17] [ 16]

[74.2500] [ -5] [ 0]

[68.1900] [ 22] [ 35]

wkbk.Close;

excelapp.Quit;

**Running a Macro in an Excel Server Application**

In the example below, MATLAB runs Microsoft Excel in a COM server and invokes a macro that has been defined within the active Excel spreadsheet file. The macro, init\_last, takes no input parameters and is called from the MATLAB client using the statement

handle.ExecuteExcel4Macro('!macroname()');

Start the example by opening the spreadsheet file and recording a macro. The macro used here simply sets all items in the last column to zero. Save your changes to the spreadsheet.

Next, in MATLAB, create a COM server running an Excel application, and open the spreadsheet:

h = actxserver('Excel.Application');

wkbk = h.Workbooks;

file = wkbk.Open('d:\weatherlog\weekly.xls');

Open the sheet that you want to change, and retrieve the current values in the range of interest:

sheets = file.Sheets;

sheet12 = sheets.Item('Week 12');

range = sheet12.get('Range', 'D1', 'F5');

range.Value

ans =

[ 78] [ 32] [ 37]

[ 69] [ 27] [ 30]

[ 77] [ 17] [ 16]

[ 74] [ -5] [ -1]

[ 68] [ 22] [ 35]

Now execute the macro, and verify that the values have changed as expected:

h.ExecuteExcel4Macro('!init\_last()');

range.Value

ans =

[ 78] [ 32] [ 0]

[ 69] [ 27] [ 0]

[ 77] [ 17] [ 0]

[ 74] [ -5] [ 0]

[ 68] [ 22] [ 0]

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

**MATLAB COM Client Demo**

MATLAB includes a demo illustrating the use of the COM Client with MATLAB. To run the demo, click the **Demos** tab in the MATLAB Help browser. Click to expand the folder called External Interfaces and select Programming with COM.

[[Back to Top of Page](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html" \l "top_of_page)[Back to Top](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html#top_of_page)

|  |  |
| --- | --- |
|  | [Provide feedback about this page](http://www.mathworks.com/survey/support/docfeedback.html?http://webcache.googleusercontent.com/search?q=cache:HMjrb-u6-1gJ:www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27178.html+&cd=13&hl=en&ct=clnk&gl=us&client=firefox-a) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [Supported Client/Server Configurations](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/bq9yai0-1.html) | Supported Client/Server Configurations |  | Additional COM Client Information | [Additional COM Client Information](http://www.kxcad.net/cae_MATLAB/techdoc/matlab_external/f27433.html) |

© 1984-2007 The MathWorks, Inc. • [Terms of Use](http://www.kxcad.net/cae_MATLAB/license.txt) • [Patents](http://www.kxcad.net/cae_MATLAB/patents.txt) • [Trademarks](http://www.kxcad.net/cae_MATLAB/trademarks.txt) • [Acknowledgments](http://www.kxcad.net/cae_MATLAB/acknowledgments.html)