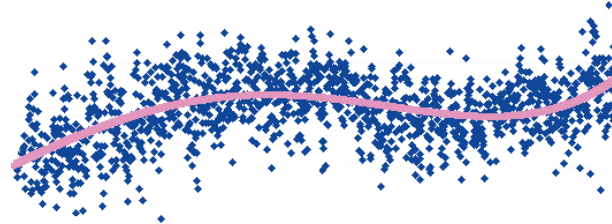


Exploring Global Temperature History: Handling MATLAB® Events in Excel

MATLAB Builder for .NET lets you incorporate your MATLAB programs into desktop applications such as Excel, enhancing familiar application interfaces with the three-dimensional analysis and visualization capabilities of MATLAB.

You can enable Excel to respond to events that occur in your MATLAB based GUI by using MATLAB Builder for .NET to generate a COM component that incorporates an event source. You can freely redistribute generated COM components, making your graphical application available to users who do not have MATLAB installed.



The *lingua franca* for many Microsoft applications, including Excel, is Visual Basic. In an embedded Visual Basic environment, handling the event source of a COM component created in MATLAB Builder for .NET involves four steps:

1. Implement an event source in the MATLAB M-code that you will use to generate your COM component.
2. Create an instance of the COM component in Visual Basic.
3. Create a Visual Basic event sink.
4. Process the event-related data.

In this article we demonstrate these steps using a sample application that analyzes worldwide temperature records from 1880 to 2005 and creates Excel charts from MATLAB data sets. (Because the article focuses on integration with Excel, most of the code is written in Visual Basic; only the “Implementing an Event Source” section contains MATLAB M-code.)

The application has two parts: a COM component created from MATLAB code that reads the data set, displays the navigation GUI, and computes graph data, and an Excel-hosted Visual Basic program that displays charts in response to events in the GUI.

Implementing an Event Source

By default, a deployed COM component exports a function for each compiled M-file. For the COM component to generate an *event source* we must place the `%#event` keyword in the MATLAB callback M-file. For example:

```
function PlotTempData(city, xData, localAnomData,...
                    fitval, latStr, lngStr)
%#event
```

In MATLAB, the `PlotTempData` function graphs the input data, but the `%#event` keyword causes it to behave differently when it's compiled: When we invoke the `PlotTempData` function in the generated component, that component raises a `PlotTempData` event instead of calling the code in the M-file. Event source functions do not need to contain any M-code. If they do, MATLAB will execute the code but the deployed COM component will not.

Using MATLAB Builder for .NET, we generate a COM component from the M-functions `GlobalTemp`, `DisplayLocation` and `PlotTempData` and include the data files `Cities.csv` and `TempAnomData1880.mat`.

```
mcc -g -W com:GlobalTemp,GlobalTempClass,1.0 ...
-T link:lib GlobalTemp DisplayLocation PlotTemp-
Data
-a TempAnomData1880.mat -a Cities.csv
```

The generated component, `GlobalTemp`, exports its API via the class `GlobalTempClass`.

Creating an Instance of a COM Component

To create a COM component instance we declare and initialize a reference variable. To use the `WithEvents` keyword, which indicates that the component raises events, we must declare our reference variable within a Visual Basic class.

In the following declaration, `tempGUI` is an instance variable of the `TempAnomaly` class:

```
Dim WithEvents tempGUI as GlobalTemp.GlobalTempClass
```

We initialize the reference variable in the `Class_Initialize()` function of our class. Visual Basic will call this function each time it creates an instance of our class.

```
Private Sub Class_Initialize()
    Set tempGUI = New GlobalTemp.GlobalTempClass
End Sub
```

Creating an Event Sink

An event sink is an object with a callback function. To listen for an event generated by a MATLAB event source, the event sink's callback function must have the same signature as the MATLAB function to which it corresponds. For example, `tempGUI_PlotTempData`, a method of the `TempAnomaly` class, takes the same number and type of arguments as the MATLAB `PlotTempData` function:

```
Private Sub tempGUI_PlotTempData(ByVal city As Variant, _
    ByVal xData As Variant, ByVal localAnomData As _
    Variant, ByVal fitval As Variant, ByVal latStr _
    As Variant, ByVal lngStr As Variant)
    ' Event handling code goes here
End Sub
```

A note of caution: When the MATLAB Builder for .NET COM component raises an event, it waits for the event-handling routine to complete before continuing its own execution. Callback functions must not invoke functions exported by the component, or infinite deadlock will result.

Processing Event-Related Data

We launch the COM component's GUI from the Excel interface by calling the `GlobalTemp` function in the COM component.

```
Public Sub LaunchGUI()
    tempGUI.GlobalTemp
End Sub
```

Our data set consists of 2,592 longitudinal monthly data series, each corresponding to a 5 x 5 degree area of the Earth, generated by the U.S. National Climatic Data Center. The GUI represents the data on a rotating three-dimensional globe (Figure 1), providing an intuitive way for the end-user to select an area for display. Graphing each data series as a scatter plot with regression line makes it easy to compare results from multiple locations.

Each `PlotTempData` event delivers six pieces of MATLAB data to the Visual Basic event sink. To process the data, the Excel-based Visual Basic code transfers the data to a spreadsheet and then uses the transferred data to create a chart.

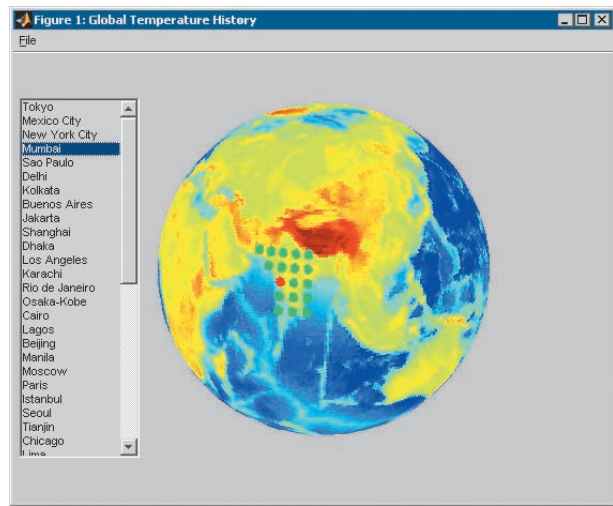


Figure 1. Data navigation interface.

Transferring MATLAB Data to a Spreadsheet

In response to the `PlotTempData` event, the callback function `tempGUI_PlotTempData` transfers the event data from MATLAB variables to the Excel spreadsheet. Because MATLAB passes data to the callback function using 2-D variants, `tempGUI_PlotTempData` extracts the x values from the input `xData` cell array by using variant indexing.

```
Dim xValues As Variant
xValues = xData(1, 1)
```

In Excel, large data series must be entered on a spreadsheet before they are graphed. Because in our example the X and Y value arrays are the same length, a single set of `for`-loops is sufficient to extract all three series and enter them on the spreadsheet. The one-based indexing used by MATLAB data matches the one-based indexing of Excel spreadsheet ranges.

```
For rowCount = 1 To UBound(localAnomData, 1)
    For ColCount = 1 To UBound(localAnomData, 2)
        ActiveSheet.Cells(index, tempCol) = _
            localAnomData(rowCount, _ColCount)
        ActiveSheet.Cells(index, fitCol) = _
            fitval(rowCount, ColCount)
        ActiveSheet.Cells(index, dateCol) = _
            xValues(1, index - startRow + 1)
        index = index + 1
    Next ColCount
Next rowCount
```

Next, the callback function adds a chart to the worksheet by calling a private method, `CreateChart`.

Creating Excel Charts

Since Excel cannot handle dates before January 1, 1900, we must create custom x-axis labels for the time series XY scatter plots (Figure 2).

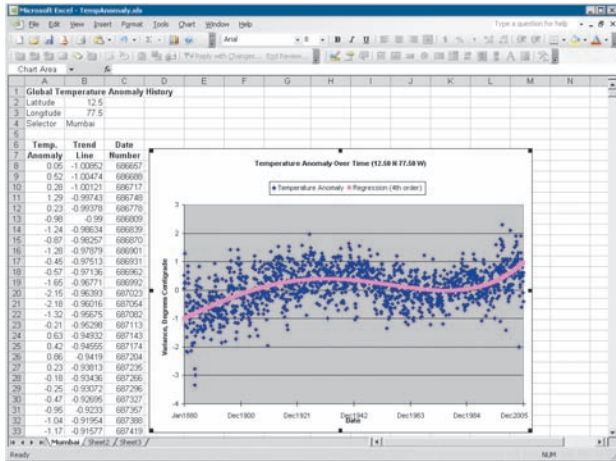


Figure 2. Temperature data and regression line for a single grid box (Microsoft Excel).

The code below adds the first data series to the chart. `tempRange` is the Excel range containing the temperature values and `dateRange` is a range containing the date on which the temperature was recorded.

```
chartObj.chart.SeriesCollection.Add
Source:=tempRange
With chartObj.chart.SeriesCollection(1)
.Name = "Temperature Anomaly"
.xValues = dateRange
End With
```

The COM component precomputes seven x-axis tick mark locations and the x-axis tick labels and sends them to the event sink as a MATLAB matrix. This data must be converted from a two-dimensional (1x1, 1x7) matrix to a one-dimensional (1x7) zero-based array before Excel will graph it.

```
ReDim xLabelData (UBound(xLabels(1, 1), 2) - 1) As Double
For k = LBound(xLabels(1, 1), 2) To UBound(xLabels(1, 1), 2)
    xLabelData(k - 1) = xLabels(1, 1)(1, k)
Next k
```

Excel does not directly support custom tick mark locations and labels, but we can work around this limitation by treating them as another data series. The data points employ "+" shaped markers to form the tick marks. The `for`-loop below assigns each tick mark the appropriate date label and ensures that it displays below the X axis.

```
With chartObj.chart.SeriesCollection(3)
For k = 1 To .Points.Count
.Points(k).HasDataLabel = True
With .Points(k).DataLabel
.Text = " " + xlabelStr(k, 1) + " "
.NumberFormat = "Text"
.Type = xlDataLabelsShowLabel
.Position = xlLabelPositionBelow
End With
Next k
End With
```

The `PlotTempData` event handler creates data series and graphs on the active sheet. We can add multiple graphs to a single Excel workbook simply by opening a new sheet before selecting another grid box in the data navigation GUI.

This application leverages the strengths of MATLAB and Excel: MATLAB three-dimensional graphics enrich the data navigation GUI, while the tabbed worksheet grids in Excel let you rapidly explore and compare data. The resulting combination produces an application that is more efficient, easier to maintain, and easier to use than an application written entirely in one tool. And because components generated with MATLAB Builder for .NET do not require MATLAB to run, your application can be distributed to Excel users anywhere. ◀◀

Event Management Glossary

Component Object Model (COM). An object-oriented programming model that defines how objects interact within a single application or between applications.

Event. An occurrence, such as a mouse click, that triggers an action.

Event source. A producer of events. Event sources raise events.

Event sink. A consumer of events. Event sinks listen for events.

Callback function. A function that notifies a software component that an event has occurred.

Event model. The rules governing event behavior.

RESOURCES

► Overview of the Data Set

www.ncdc.noaa.gov/oa/climate/research/ghcn/ghcngrid.html

► Global Temperature Analysis Code

www.mathworks.com/res/temperature

