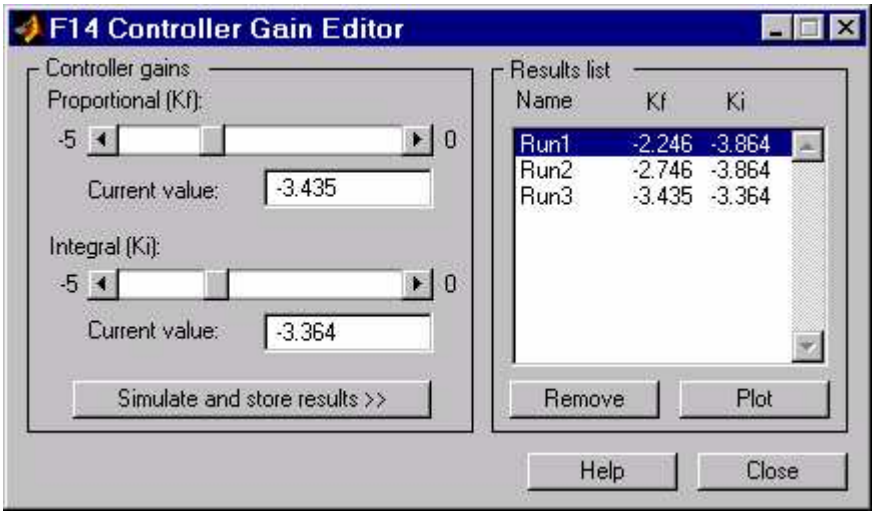
A GUI to Set Simulink Model Parameters

On this page…

About the Simulink Model Parameters Example

This example illustrates how to create a GUI that sets the parameters of a Simulink® model. In addition, the GUI can run the simulation and plot the results in a figure window. The following figure shows the GUI after running three simulations with different values for controller gains.



The example illustrates a number of GUI building techniques:

- Opening and setting parameters on a Simulink model from a GUI.
- Implementing sliders that operate in conjunction with text boxes, which display the current value, as well as accepting user input.
- Enabling and disabling controls, depending on the state of the GUI.
- Managing a variety of shared data using the `handles` structure.
- Directing graphics output to figures with hidden handles.
- Adding a **Help** button that displays `.html` files in the MATLAB Help browser.

Back to Top

View and Run the Simulink Parameters GUI

If you are reading this document in the MATLAB Help browser, you can access the example FIG-file and code file by clicking the following links. If you are reading this on the Web or in PDF form, go to the corresponding section in the MATLAB Help Browser to use the links.

If you intend to modify the layout or code of this GUI example, first save a copy of its code file and FIG-file to your current folder (you need write access to your current folder to do this). Follow these steps to copy the example files to your current folder and then to open them:

1. Click here to copy the files to your current folder.
2. Type `guide f14ex` or click here to open the FIG-file in GUIDE.
3. Type `edit f14ex` or click here to open the code file in the Editor.

You can view the properties of any component by double-clicking it in the Layout Editor to open the Property Inspector for it. You can modify either the figure, the code, or both, and then save the GUI in your current folder using **File** > **Save as** from GUIDE. This saves both files, allowing you to rename them, if you choose.

To just inspect the GUI in GUIDE and run it, follow these steps instead:

1. Click here to add the example files to the MATLAB path (only for the current session).
2. Click here to run the `f14ex` GUI.
3. Click here to display the GUI in the GUIDE Layout Editor (read only).
4. Click here to display the GUI code file in the MATLAB Editor (read only).

> **Note** Do not save GUI files to the `examples` folder where you found them or you will overwrite the original files. If you want to save GUI files, use **File** > **Save as** from GUIDE, which saves both the GUI FIG-file and the GUI code file.

Back to Top

How to Use the Simulink Parameters GUI

> **Note** You must have Simulink installed for this GUI to run. The first time you run the GUI, Simulink opens (if it is not already running) and loads the `f14` demo model. This can take several seconds.

The GUI has a **Help** button. Clicking it opens an HTML file, `f14ex_help.html`, in the Help Browser. This file, which resides in the `examples` folder along with the GUI files, contains the following five sections of help text:

F14 Controller Gain Editor

You can use the F14 Controller Gain Editor to analyze how changing the gains used in the Proportional-Integral Controller affect the aircraft's angle of attack and the amount of G force the pilot feels.

Note that the Simulink diagram `f14.mdl` must be open to run this GUI. If you close the F14 Simulink model, the GUI reopens it whenever it requires the model to execute.

Changing the Controller Gains

You can change gains in two blocks:

1. The Proportional gain (`Kf`) in the Gain block
2. The Integral gain (`Ki`) in the Transfer Function block

You can change either of the gains in one of the two ways:

1. Move the slider associated with that gain.
2. Type a new value into the **Current value** edit field associated with that gain.

The block's values are updated as soon as you enter the new value in the GUI.

Running the Simulation

Once you have set the gain values, you can run the simulation by clicking the **Simulate and store results** button. The simulation time and output vectors are stored in the **Results list**.

Plotting the Results

You can generate a plot of one or more simulation results by selecting the row of results (`Run1`, `Run2`, etc.) in the **Results list** that you want to plot and clicking the **Plot** button. If you select multiple rows, the graph contains a plot of each result.

The graph is displayed in a figure, which is cleared each time you click the **Plot** button. The figure's handle is hidden so that only the GUI can display graphs in this window.

Removing Results

To remove a result from the **Results list**, select the row or rows you want to remove and click the **Remove** button.

Back to Top

Running the GUI

The GUI is nonblocking and nonmodal because it is designed to be used as an analysis tool.

GUI Options Settings

This GUI uses the following GUI option settings:

- Resize behavior: **Non-resizable**
- Command-line accessibility: **Off**
- GUI Options selected:
  - Generate callback function prototypes
  - GUI allows only one instance to run

Opening the Simulink Block Diagrams

This example is designed to work with the `f14` Simulink model. Because the GUI sets parameters and runs the simulation, the `f14` model must be open when the GUI is displayed. When the GUI runs, the `model_open` subfunction executes. The purpose of the subfunction is to:

- Determine if the model is open (`find_system`).
- Open the block diagram for the model and the subsystem where the parameters are being set, if not open already (`open_system`).
- Change the size of the controller Gain block so it can display the gain value (`set_param`).
- Bring the GUI forward so it is displayed on top of the Simulink diagrams (`figure`).
- Set the block parameters to match the current settings in the GUI.

Here is the code for the `model_open` subfunction:

```
function model_open(handles)
if  isempty(find_system('Name','f14')),
        open_system('f14'); open_system('f14/Controller')
        set_param('f14/Controller/Gain','Position',[275 14 340 56])
        figure(handles.F14ControllerEditor)
        set_param('f14/Controller Gain','Gain',...
                get(handles.KfCurrentValue,'String'))
        set_param(...
      'f14/Controller/Proportional plus integral compensator',...
      'Numerator',...
      get(handles.KiCurrentValue,'String'))
end
```

Back to Top

Programming the Slider and Edit Text Components

In the GUI, each slider is coupled to an edit text component so that:

- The edit text displays the current value of the slider.
- The user can enter a value into the edit text box and cause the slider to update to that value.
- Both components update the appropriate model parameters when activated by the user.

Slider Callback

The GUI uses two sliders to specify block gains because these components enable the selection of continuous values within a specified range. When a user changes the slider value, the callback executes the following steps:

- Calls `model_open` to ensure that the Simulink model is open so that simulation parameters can be set.
- Gets the new slider value.
- Sets the value of the **Current value** edit text component to match the slider.
- Sets the appropriate block parameter to the new value (`set_param`).

Here is the callback for the **Proportional (Kf)** slider:

```
function KfValueSlider_Callback(hObject, eventdata, handles)
% Ensure model is open.
model_open(handles)
% Get the new value for the Kf Gain from the slider.
NewVal = get(hObject, 'Value');
% Set the value of the KfCurrentValue to the new value
% set by slider.
set(handles.KfCurrentValue,'String',NewVal)
% Set the Gain parameter of the Kf Gain Block to the new value.
set_param('f14/Controller/Gain','Gain',num2str(NewVal))
```

While a slider returns a number and the edit text requires a string, uicontrols automatically convert the values to the correct type.

The callback for the **Integral (Ki)** slider follows an approach similar to the **Proportional (Kf)** slider's callback.

Current Value Edit Text Callback

The edit text box enables users to enter a value for the respective parameter. When the user clicks another component in the GUI after entering data into the text box, the edit text callback executes the following steps:

- Calls `model_open` to ensure that the Simulink model is open so that it can set simulation parameters.
- Converts the string returned by the edit box `String` property to a double (`str2double`).
- Checks whether the value entered by the user is within the range of the slider:

  If the value is out of range, the edit text `String` property is set to the value of the slider (rejecting the number entered by the user).

  If the value is in range, the slider `Value` property is updated to the new value.
- Sets the appropriate block parameter to the new value (`set_param`).

Here is the callback for the Kf **Current value** text box:

```
function KfCurrentValue_Callback(hObject, eventdata, handles)
% Ensure model is open.
model_open(handles)
% Get the new value for the Kf Gain.
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
% Check that the entered value falls within the allowable range.
if  isempty(NewVal) || (NewVal< -5) || (NewVal>0),
   % Revert to last value, as indicated by KfValueSlider.
   OldVal = get(handles.KfValueSlider,'Value');
   set(hObject, 'String',OldVal)
else % Use new Kf value
   % Set the value of the KfValueSlider to the new value.
   set(handles.KfValueSlider,'Value',NewVal)
   % Set the Gain parameter of the Kf Gain Block
   % to the new value.
   set_param('f14/Controller/Gain','Gain',NewStrVal)
end
```

The callback for the `Ki` **Current value** follows a similar approach.

Back to Top

Running the Simulation from the GUI

The GUI **Simulate and store results** button callback runs the model simulation and stores the results in the `handles` structure. Storing data in the `handles` structure simplifies the process of passing data to other subfunction since this structure can be passed as an argument.

When a user clicks the **Simulate and store results** button, the callback executes the following steps:

- Calls `sim`, which runs the simulation and returns the data that is used for plotting.
- Creates a structure to save the results of the simulation, the current values of the simulation parameters set by the GUI, and the run name and number.
- Stores the structure in the `handles` structure.
- Updates the list box `String` to list the most recent run.

Here is the **Simulate and store results** button callback:

```
function SimulateButton_Callback(hObject, eventdata, handles)
[timeVector,stateVector,outputVector] = sim('f14');
% Retrieve old results data structure
if isfield(handles,'ResultsData') &
~isempty(handles.ResultsData)
        ResultsData = handles.ResultsData;
        % Determine the maximum run number currently used.
        maxNum = ResultsData(length(ResultsData)).RunNumber;
        ResultNum = maxNum+1;
else % Set up the results data structure
        ResultsData = struct('RunName',[],'RunNumber',[],...
                      'KiValue',[],'KfValue',[],'timeVector',[],...
                      'outputVector',[]);
        ResultNum = 1;
end
if isequal(ResultNum,1),
        % Enable the Plot and Remove buttons
        set([handles.RemoveButton,handles.PlotButton],'Enable','on')
end
% Get Ki and Kf values to store with the data and put in the
results list.
Ki = get(handles.KiValueSlider,'Value');
Kf = get(handles.KfValueSlider,'Value');
ResultsData(ResultNum).RunName = ['Run',num2str(ResultNum)];
ResultsData(ResultNum).RunNumber = ResultNum;
ResultsData(ResultNum).KiValue = Ki;
ResultsData(ResultNum).KfValue = Kf;
ResultsData(ResultNum).timeVector = timeVector;
ResultsData(ResultNum).outputVector = outputVector;
% Build the new results list string for the listbox
ResultsStr = get(handles.ResultsList,'String');
if isequal(ResultNum,1)
        ResultsStr = {['Run1',num2str(Kf),' ',num2str(Ki)]};
else
        ResultsStr = [ResultsStr;...
        {['Run',num2str(ResultNum),' ',num2str(Kf),' ', ...
        num2str(Ki)]}];
end
set(handles.ResultsList,'String',ResultsStr);
% Store the new ResultsData
handles.ResultsData = ResultsData;
guidata(hObject, handles)
```

Back to Top

Removing Results from the List Box

The GUI **Remove** button callback deletes any selected item from the **Results list** list box. It also deletes the corresponding run data from the `handles` structure. When a user clicks the **Remove** button, the callback executes the following steps:

- Determines which list box items are selected when a user clicks the **Remove** button and removes those items from the list box `String` property by setting each item to the empty matrix `[]`.

- Removes the deleted data from the `handles` structure.
- Displays the string `<empty>` and disables the **Remove** and **Plot** buttons (using the `Enable` property), if all the items in the list box are removed.
- Save the changes to the `handles` structure (`guidata`).

Here is the **Remove** button callback:

```
function RemoveButton_Callback(hObject, eventdata, handles)
currentVal = get(handles.ResultsList,'Value');
resultsStr = get(handles.ResultsList,'String');
numResults = size(resultsStr,1);
% Remove the data and list entry for the selected value
resultsStr(currentVal) =[];
handles.ResultsData(currentVal)=[];
% If there are no other entries, disable the Remove and Plot
button
% and change the list string to <empty>
if isequal(numResults,length(currentVal)),
        resultsStr = {'<empty>'};
        currentVal = 1;

set([handles.RemoveButton,handles.PlotButton],'Enable','off')
end
% Ensure that list box Value is valid, then reset Value and String
currentVal = min(currentVal,size(resultsStr,1));
set(handles.ResultsList,'Value',currentVal,'String',resultsStr)
% Store the new ResultsData
guidata(hObject, handles)
```

Back to Top

Plotting the Results Data

The GUI **Plot** button callback creates a plot of the run data and adds a legend. The data to plot is passed to the callback in the `handles` structure, which also contains the gain settings used when the simulation ran. When a user clicks the **Plot** button, the callback executes the following steps:

- Collects the data for each run selected in the **Results list**, including two variables (time vector and output vector) and a color for each result run to plot.
- Generates a string for the legend from the stored data.
- Creates the figure and axes for plotting and saves the handles for use by the **Close** button callback.
- Plots the data, adds a legend, and makes the figure visible.

Plotting Into the Hidden Figure

The figure that contains the plot is created as invisible and then made visible after adding the plot and legend. To prevent this figure from becoming the target for plotting commands issued at the command line or by other GUIs, its `HandleVisibility` and `IntegerHandle` properties are set to `off`. This means the figure is also hidden from the `plot` and `legend` commands.

Use the following steps to plot into a hidden figure:

- Save the handle of the figure when you create it.
- Create an axes, set its `Parent` property to the figure handle, and save the axes handle.
- Create the plot (which is one or more line objects), save these line handles, and set their `Parent` properties to the handle of the axes.
- Make the figure visible.

Plot Button Callback Listing

Here is the **Plot** button callback.

```
function PlotButton_Callback(hObject, eventdata, handles)
currentVal = get(handles.ResultsList,'Value');
% Get data to plot and generate command string with color
% specified
```

```
legendStr = cell(length(currentVal),1);
plotColor = {'b','g','r','c','m','y','k'};
for ctVal = 1:length(currentVal);
        PlotData{(ctVal*3)-2} =
handles.ResultsData(currentVal(ctVal)).timeVector;
        PlotData{(ctVal*3)-1} =
handles.ResultsData(currentVal(ctVal)).outputVector;
        numColor = ctVal - 7*( floor((ctVal-1)/7) );
        PlotData{ctVal*3} = plotColor{numColor};
        legendStr{ctVal} = ...
      [handles.ResultsData(currentVal(ctVal)).RunName,'; Kf=',...
      num2str(handles.ResultsData(currentVal(ctVal)).KfValue),...
      ';  Ki=', ...
      num2str(handles.ResultsData(currentVal(ctVal)).KiValue)];
end
% If necessary, create the plot figure and store in handles
% structure
if ~isfield(handles,'PlotFigure') ||...
   ~ishandle(handles.PlotFigure),
        handles.PlotFigure = ...
        figure('Name','F14 Simulation Output',...
        'Visible','off','NumberTitle','off',...
        'HandleVisibility','off','IntegerHandle','off');
        handles.PlotAxes = axes('Parent',handles.PlotFigure);
        guidata(hObject, handles)
end
% Plot data
pHandles = plot(PlotData{:},'Parent',handles.PlotAxes);
% Add a legend, and bring figure to the front
legend(pHandles(1:2:end),legendStr{:})
% Make the figure visible and bring it forward
figure(handles.PlotFigure)
```

Back to Top

The GUI Help Button

The GUI **Help** button callback displays an HTML file in the MATLAB Help browser. It uses two commands:

- The `which` command returns the full path to the file when it is on the MATLAB path
- The `web` command displays the file in the Help browser.

This is the **Help** button callback.

```
function HelpButton_Callback(hObject, eventdata, handles)
HelpPath = which('f14ex_help.html');
web(HelpPath);
```

You can also display the help document in a Web browser or load an external URL. For a description of these options, see the documentation for `web`.

Back to Top

Closing the GUI

The GUI **Close** button callback closes the plot figure, if one exists and then closes the GUI. The handle of the plot figure and the GUI figure are available from the `handles` structure. The callback executes two steps:

- Checks to see if there is a `PlotFigure` field in the `handles` structure and if it contains a valid figure handle (the user could have closed the figure manually).
- Closes the GUI figure.

This is the **Close** button callback:

```
function CloseButton_Callback(hObject, eventdata, handles)
% Close the GUI and any plot window that is open
if isfield(handles,'PlotFigure') && ...
      ishandle(handles.PlotFigure),
```

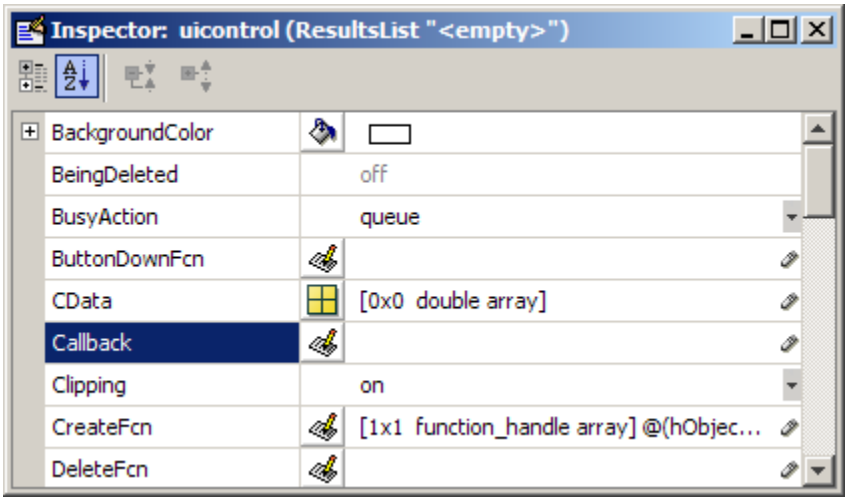
```
    close(handles.PlotFigure);
end
close(handles.F14ControllerEditor);
```

Back to Top

The List Box Callback and Create Function

This GUI does not use the list box callback because the actions performed on list box items are carried out by push buttons (**Simulate and store results**, **Remove**, and **Plot**). GUIDE automatically inserts a callback stub when you add the list box and automatically sets the `Callback` property to execute this subfunction whenever the callback is triggered (which happens when users select an item in the list box).

In this example, there is no need for the list box callback to execute. You can delete it from the GUI code file and at the same time also delete the `Callback` property string in the Property Inspector so that the software does not attempt to execute the callback.



For more information on how to trigger the list box callback, see the description of list box.

Setting the Background to White

The list box create function enables you to determine the background color of the list box. The following code shows the create function for the list box that is tagged `ResultsList`:

```
function ResultsList_CreateFcn(hObject, eventdata, handles)
% Hint: listbox controls usually have a white background, change
%       'usewhitebg' to 0 to use default. See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',...
    get(0,'defaultUicontrolBackgroundColor'));
end
```

Back to Top

Accessing Workspace Variables from a List Box An Address Book Reader