SIMPLE TIMING of MATLAB GUIs USED in MS-WINDOWS for DATA ACQUISITION

Pavel Nesládek & Miloš Steinhart
AMTC Dresden & UMCH Praha

Abstract

GUIs are easy-to-build and debug tools. They can be used, among others, for device control, data acquisition and pre-treatment with the advantage of the great math power of MATLAB within an easy reach. This contribution proposes an elegant possibility of timing, which is unfortunately not easy to accomplish in the basic MATLAB. The timer is a small program running in the Windows environment. Its function is to periodically click a pushbutton in a GUI by simulating an action of a mouse through sending an appropriate message into the Windows message queue. The timer can be used in cases when sampling or other action is to be done periodically in time points, which are not necessarily exactly equidistant. These cases are e.g. data acquisition or periodical monitoring of some dangerous status of the controlled instruments. The main advantage of our solution is the simplicity and speed of preparation of the GUI, due to the fact that it can be debugged first independently and only then the timer is used.

Introduction

Although MATLAB is meant to be mainly a math environment, its GUIs can be excellent tools for data acquisition. They can be easily built and debugged. It is for instance relatively easy to build a GUI, which would communicate with a measuring instrument via a serial port. In versions lower than 5 some kind of real time toolbox (RTT) is necessary. From the version 6 serial ports are supported directly but RTT brings improved performance. Pushbuttons of the GUI then perform some action e.g. send a message to read some particular data. With the debugging tools it is easy to have the GUI running quickly. Unfortunately, in the basic MATLAB, the possibilities for periodical data sampling are rather limited, even if the RTT is used, and programmers have to use one of inconvenient tools, such as a loop employing the pause function. The problem could be probably solved using SIMULINK but in many applications this would be an expensive luxury.

Our idea was to build a timer as a separate program running in a Window environment, which would for instance periodically simulate pressing a particular pushbutton in a GUI. To keep our solution as simple as possible, we have not chosen OLE client/server architecture communication but we have been using the window message queue.

In our contribution we describe the principles of functioning of our timer and illustrate some of its possible applications.

Window message processing

The event driven process executing in Windows is controlled by so called message loop. The Windows operating system (WOS), as well as any thread, has its own input queue. The WOS receives messages from input devices, such as the mouse or the keyboard, determines the target object and sends the message to it. The mandatory part of the target program Win_Main receives the message. It may, for instance, translate the hot keys and then it dispatches the message

(possibly modified) back to the WOS, which eventually calls the second part of the program, the Window_Function with the message as a parameter. The Window_Function contains a switch by means of which the particular action is taken. It usually means generating of further messages, which are sent to their parent windows or to the WOS.
The windows message passes through the WOS queue as TMsg record with the length of 18 bytes and it is converted into the 14 byte TMessage record, when forwarded to the application message loop. The message can be put in the (WOS) queue by calling SendMessage or PostMessage in the form of the call, which in Pascal looks as follows:
function SendMessage(Wnd: HWnd; Msg, WParam: Word; lParam: Longint): Longint.
Wnd is a handle. It is a number given to any target object, such as a pushbutton, an edit line, a window or a static text at the time of its initialization. The handle represents a unique address of any object in the system.
Msg specifies the message to be sent and is commonly expressed by a window constant, For example wm_Command or wm_LbuttonDown for command or pressing of the left mouse button respectively. WParam and lParam are additional, message specific parameters.

Let us, for instance, follow the way an executing program is stopped and its instance is disposed from the memory as a response of choosing "Exit" from the its system menu. At first the wm_close message is sent as the order that the application should terminate. The application first closes all its child windows. This is accomplished by sending the same message, addressed to them using the appropriate handles. Only them the application confirms, that the message was successfully processed, by returning a zero to the WOS. This may allow the user to confirm ending of the program or to save an opened document. After confirmation by the application, the system deactivates the non-client and client area and the application itself by sending the wm_ncactivate, wm_activate and wm_activateapp messages respectively. Finally, when the response to all these messages is zero, the application's window is destroyed by calling wm_destroy and wm_ncdestroy. Than the object is removed from the memory. To trigger all this action you need only to send the wm_close or wm_Quit message:
SendMessage(Wnd,wm_close,0,0);
The same way by sending wm_command the methods of the window are called by WOS, when its menu item has been selected.
SendMessage(Wnd, wm_Command, cm_Open, context);
Here, cm_Open is the Id number of a menu item or control and context may contain additional information about the initiator of the message. The low order word identifies the control in the case the message was sent from a control and the high order word provides the notification message of the control or contains 1 for an accelerator (a hot key) or 0 for a menu item.
By clicking of an application's button a wm_Command message is sent to its parent window:
Send Message(Wnd, wm_Command, Id, makelong(ButtonWnd, en_update));
This example demonstrates, that the button notification en_update and its handle are sent as additional information to the parent window.
The following table shows messages corresponding to particular action on standard windows objects like buttons, check boxes, static texts or edit lines:

Action	Message and parameters
push button	wm_Command, ButtonId, MakeLong(ButtonWnd, en_update)
set checkbox state	bm_SetCheck, State, 0
get checkbox state	bm_GetCheck, 0, 0
select menu item	wm_Command, Cmd, 0
set static text or	
set edit line	wm_SetText, 0, @Text
get static text or	
get edit line	wm_GetText, 0, @Text

However, the approach to use the above information to control pushbuttons in a Matlab GUI, will fail. The reason is that in a GUI all pushbuttons have the same one Id number. So do have all other controls of the same type.
This leads to a question, what approach to use to access the particular button or other control. For the answer we have to look in closer detail to what happens in a window message queue when user clicks a pushbutton in a Matlab GUI.

Controlling a Matlab GUI

As mentioned above each member of some group of controls in a GUI has the same Id number. So behavior of pushbuttons and windows as a response to various messages had to be studied.
Most promising was sending of the messages the way they are sent, when the left mouse button is clicked. This generates two messages wm_LButtonDown and wm_LButtonUp and sends them to the button when its parent window is in the focus. Depending from the action a set of other messages may occur, like wm_Paint.
If the window containing the pushbutton is inactive, this causes activating of the window but then the pushbutton receives only the second message the wm_LButtonUp. For this reason, it is necessary to activate window by sending to it the wm_MouseActivate message first. This is one of the most important differences to the approach, using the pushbutton's Id, described above, which allows accessing a program in the background without the need to change the focus. The difference for the user is that, anytime the program should be controlled, the focus must be changed and other work on the system is virtually out of the question.
This fortunately not the case of the Matlab GUIs. There it is not necessary to send the wm_MouseActivate message and the procedure works well, even if its window is iconized or the computer (notebook) is in the standby mode, without disturbing it. This is apparently caused by difference in the handling of the mouse events by Matlab.
The message sequence for such programs is the following:
wm_MouseActivate, Button_Wnd, MakeLong(Buton_Id, wm_LButtonDown));
wm_SetCursor, Button_Wnd, MakeLong(Buton_Id, wm_LButtonDown));
wm_LButtonDown, 0, MakeLong(X_coordinate, Y_coordinate));
wm_LButtonUP, 0, MakeLong(X_coordinate, Y_coordinate));

“Add on” features

The control of software by windows messages has further advantages. You can use deactivated controls in the dialogs and window for example. The reason for this behavior is missing checking in windows operating system. It means the system queue accepts message from deactivated or not visible controls the same way as from active controls.

This way of controlling provides also the possibility of feedback from the software controlled. By using wm_gettext can you obtain any text from the user interface (Window) of any program.

Examples of the Use

The usefulness of timing is obvious. Here we want to illustrate several cases where our timer proved to be reliable for a long time.

Monitoring program, which communicates with a temperature controller Eurotherm 2416 connected to a sample stage using peltier elements: The primary use of the timer is as a watch-dog. It provides periodic checks for a dangerous situation when the current runs in the cooling direction but the element heats instead of cooling. This signalizes some problems with cooling of the peltiers and the cooling current must be reduced to zero as quickly as possible. The watch-dogs window is normally iconized to avoid accidental interference of the user. When needed, second instance of the timer can be run to periodically read the current temperature and log it.

Demonstration of robustness of a temperature control program: To illustrate time development in the system, simulated value is periodically calculated and displayed using one instance of the timer. Then a random disturbance can be introduced with considerably longer period to test the response of the system using second instance of the timer. Of course, even more then two instances can be used, if needed.

The timer can be naturally used also for other programs than MATLAB GUIs. An example is a program controlling a texture sample holder in the x-ray diffractometer. The instrument is controlled by a company-written program, which is deliberately a black-box. It however allows to perform multiple scans and announces the number of the particular scan as well as the fact each has been accomplished in a message box of its control window. In this case a special version of the timer checks this message box and when it finds, that a scan was completed, it turns the sample holder several degrees before the next scan is started. This means a possibility to break-in a closed program without the necessity to know its source code .

Concluding remarks

We have found a simple solution of time sampling in basic MATLAB GUIs without the necessity of purchasing additional toolboxes or SIMULINK. The usefulness and reliability of our solution, which is a small program running in the Windows environment is illustrated.

pavel.nesladek@amtc-dresden.com; stein@imc.cas.cz