UPPSALA
UNIVERSITET

Writing Timestudio plugins

Authors: Nils Olofsson, Pär Nyström
Version: 1.1

Index

1. Introduction

1.1 What is Timestudio?

Timestudio is a Matlab graphical interface, framework and toolbox for research that is in need of time series analysis and or general analysis. The core software is developed at the institute of psychology at Uppsala University with the mindset of making research and custom analyses accessable for all. TimeStudio is open source.

For license and terms of use please see separate license document.

1.2 What this document is

This document contains some information and guidelines useful when writing plugins/extensions for Timestudio as well as some general information about the architecture and structure of Timestudio itself. It also contains a small list of functions used when creating simple graphical user interfaces and an example plugin in the form of a template.

1.3 What this document is not

This document is not a complete user manual for Timestudio and it does not contain any broad information or guidelines on general Matlab-programming, which is to be considered a prerequisite to be reading this.

2. Timestudio program structure and workflow

2.1 The TS-object

The TS-object, which is a custom data object in Matlab, is the main workhorse and data structure in Timestudio. It contains all information about the Timestudio instance that is currently being run by the user. It has several Matlab-structs connected inside that contain all data and various other information. These are visible in figure 1 along with these structs' inner fields. Also note that some of these are in fact arrays of structs, for example the ALLSUBJ field is an array where each entry in the array is an ALLSUBJ-struct that represents one subject in the study. The TS object is a parameter into all plugin-functions so that all data in the current instance of Timestudio can be accessed inside the plugin.
The TS-object is also visible and accessible from the Matlab workspace where it can be explored during runtime.

The most important subfields inside the TS-object when writing a plugin is usually the ALLSUBJ and the ALLINFO fields. ALLSUBJ is an array of all subjects in the current study and inside these subjects various measurement data is stored. In ALLINFO various information on the current state of Timestudio is stored, such as what subjects are marked and what the current subject is. A subfield in the TS-object is accessed by writing for example TS.ALLSUBJ(1).name, which will give the field "name" of the first subject in the TS-object.

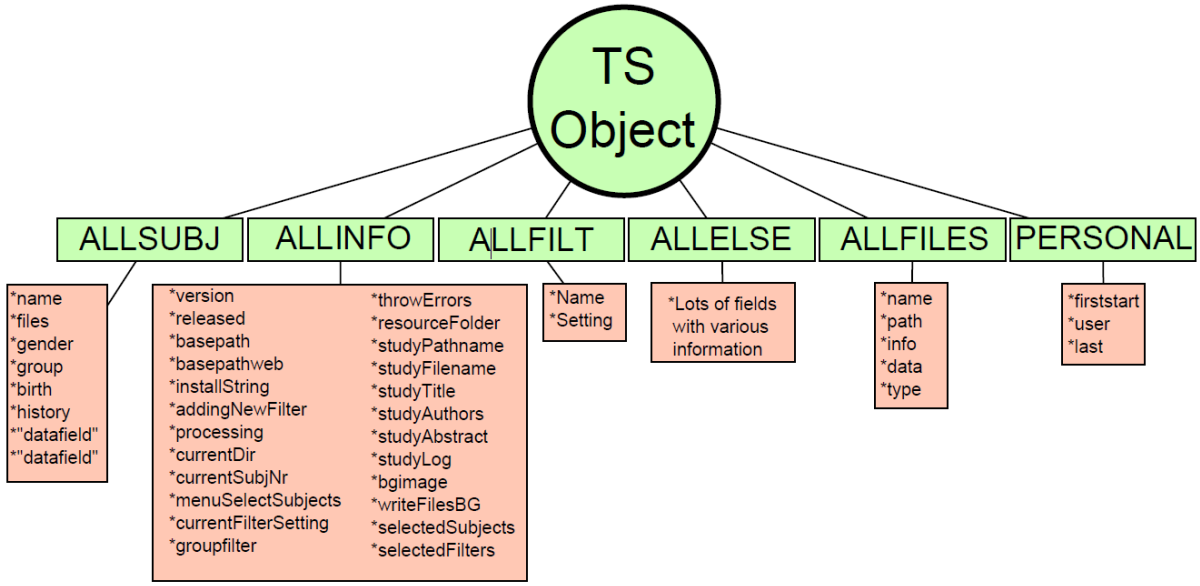

Figure 1. TS-object and its inner structures. Inner structures and their data are accessed by writing for example TS.ALLSUBJ(1).name or TS.ALLINFO.version. Of all these the only ones that typically should be altered from a plugin are the ALLSUBJ-structs.

2.2 Subjects

Subjects are stored in the ALLSUBJ-structure in the TS-object and they represent the different subjects in a study. They are usually created by loading a file that contains some measurement data into Timestudio and they are then named after this file. They can however have more than one measurement file associated with them and their names can be changed by the user. When data is read from these files it is stored inside the subject as a struct named for example "eyetracking" or "skinconductance" in the place where it says "datafield" in ALLSUBJ in figure 1. So to for example access the skinconductance information in subject 1 you would write TS.ALLSUBJ(1).skinconductance. The accessed structure is then likely to contain some subfields of its own such as a start time, an array with time values and some arrays with measurement data.

2.3 Plugins

Pluings in this context are Matlab-functions written to work together with Timestudio. They should take two arguments as input, the TS-object and an argument called varargin which allows the plugin to accept any number of input arguments. The TS-object is used to access the data in the current study. Plugins always have a graphical user interface for changing various parameters that are used during runtime. By default plugins will be given a user interface for loading and saving the plugin settings. For more information about their structure see section 3 or study an actual plugin file or template, for example the one in
Appendix B.

2.4 Folder structure

The Matlab-function files associated with timestudio is placed into subfolders depending on functionality and what type of data it is used on. "plugin_core", "plugin_eyetracking", "plugin_skinconductance", and "subfunctions" are examples. The folder "plugin_core" should contain plugins that are general in the sense that they act on any subfield of the TS data structure. The directory "subfunctions" mainly contains functions used by Timestudio itself such as functions for loading and saving studies, setting up the main GUI etc but also has some smaller functions that are shared between plugins.

2.5 Workflow

A typical workflow in Timestudio may look something like this:

1. One or more subjects with the same kind of data fields are created and selected. This may be done by loading several data files of the same format into Timestudio.
2. A plugin to read that data is selected from the plugin drop down menu and the user selects the settings he or she wish to use. Then the user pushes the "Do selected work" button while the subjects are selected and the file reading plugin processes the data and writes it into the subjects in the study.
3. Then some other plugins are chosen from the drop down menu and added to the study. These plugins may then do some additional processing or analysis of the data that was created in step 2.

3. Plugin creation and guidelines

3.1 General information

Plugins can be created completely from scratch or by using any of the plugin templates that can be found in the "*/timestudio/plugins/plugin_template"*-folder. When using a template some of the code that must be present is already there and some basic UI-functions is already in place. All of the UI-functions may not be needed and can safely be removed. It is important that your plugin is named correctly and placed in the correct folder.

When writing a plugin it is to be considered good practice too avoid dependencies on external toolboxes that may not be included with the main Matlab-runtime environment. This in order to make the plugin as accessible as possible. However it is not a requisite and may be hard to avoid depending on what you are going to implement.

A timestudio plugin contains two main parts (these can be seen in Appendix B):

1. a GUI part that sets up an interface that the user can use to change some settings in the processing the plugin is supposed to do. Timestudio has some predefined gui-functions that are wrappers of Matlabs built in gui-functions that can be used.
2. a processing part which is entered after a "if(TSplugin_processing(TS))" clausal. In that part the program flow is like that of an ordinary Matlab function or script and usually starts by reading in the states of the input fields in the GUI and then maybe use that information to load in some data from the currently selected subjects before it does some processing on that data.

3.2 Some conventions used within Timestudio

If you are writing your own file reader for reading data into Timestudio please use these following conventions to make that data compatible with other data and with all core plugins.

- The time and date the data measurement started is called "starttime" and is given in the format "yyyy-mm-dd hh:mm:ss".
  If finer resolution of starttime such as additional hundreds of a second or ms are given they can be added so that format becomes "yyyy-mm-dd hh:mm:ss:uuu".
- The time vector is called "time" and its unit is in seconds.
- The sample frequency value is called "samplerate" and its unit should be samples per second.
- Event data is stored in arrays of structs called "events" that should have the three fields "name", "oldname" and "time" inside. Event data is stored inside a datafield, for example like "TS.ALLSUBJ(2).skinconductance.events".

3.3 Using and storing resource files

When a study is saved two kinds of data files may be added to and stored together with the study inside the TS-object. Plugin files (m-files) and resource files. A resource file may be some kind of data file that contains raw measurement data of the kind that is read into a subject or it may be some kind of stimuli file such as an image, a movie or a sound. Measurement data files are typically added automatically when the subject is created or added later to the subject from the subject editing interface. Stimuli files may however need to be added from the plugin itself.

To do this they are simply added to the subject or subjects the stimuli is used with "files"-field, see figure 1. When the study is saved they may then be automatically loaded into the study file and labeled as resource files in the ALLFILES struct, given that a correct path and filename has been entered.

There is no technical differentiation between a stimuli file and a measurement data file in Timestudio so they may be treated and added in the same way. Only typical difference to consider is that a measurement file probably is unique to a subject while a stimuli file probably isn't.

3.4 Creating a plugin by using a automatically loaded template

The most straight forward way of creating a new plugin is by using the "Create plugin" button in timestudio as seen in figure 2. This will open a prompt with two textfields where you enter what type of plugin it is supposed to be and what the name of the plugin should be. If the plugin doesn't already exist this will copy a template into a folder named "plugin_type" and rename the file copy to "type_name". Its will also change the name of the function inside the file so that it fits the file name. After this is done the slightly altered template is opened in Matlabs standard editor from which the plugin can be written.

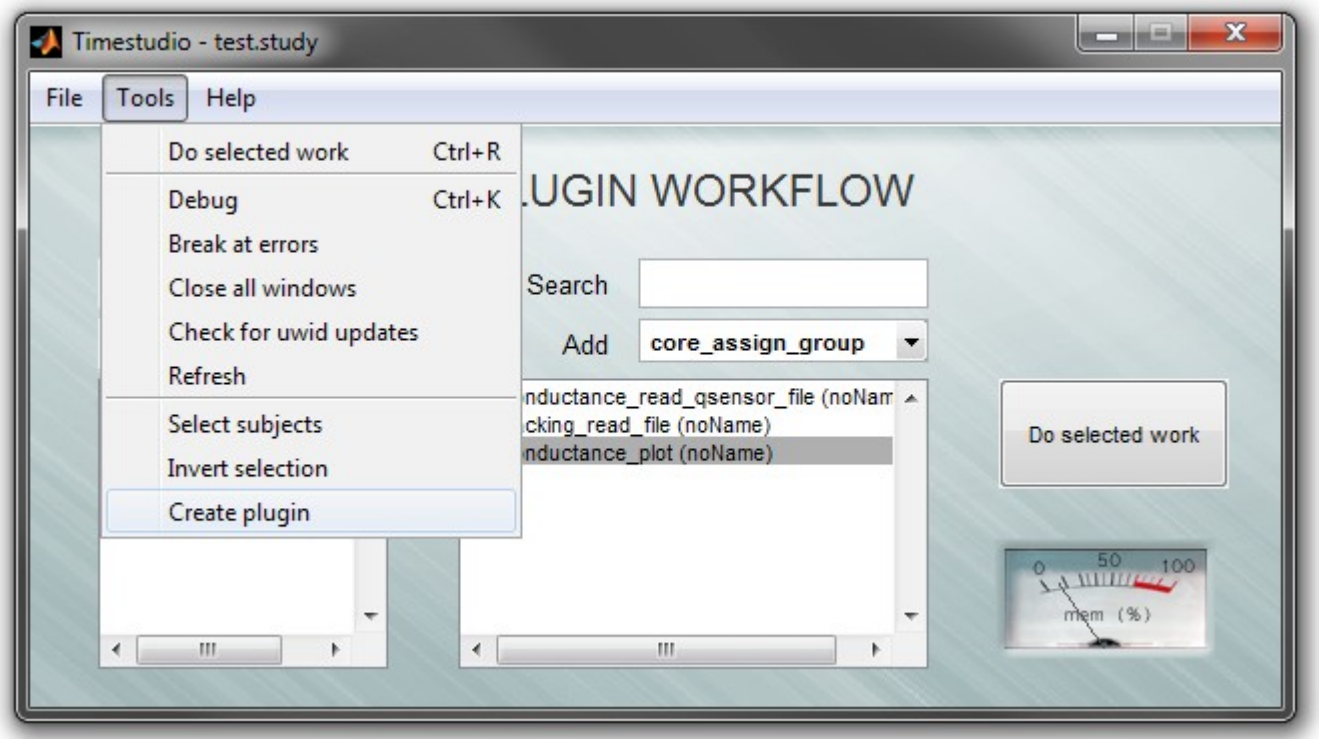

Figure 2: Select "Create plugin" to create a plugin using a standard template. The plugin file is named and placed within the folder structure automatically.

3.5 Creating a plugin from scratch or by using a manually loaded template

You are of course free to start writing from scratch or start by opening a plugin or template with a text editor such as Matlabs standard editor. The same things applies as when opening a template from the Timestudio interface. The only difference is that you have to rename the function inside the file and place the altered file in the correct folder when saving it. If you start from scratch you of course have to write the overall structure with a gui-part and a processing part yourself.

3.6 Using the new plugin from the main GUI

When a new plugin has been created it will appear in the plugin list only after Timestudio has been restarted, refreshed or if the search-box has been used. Once it it available in the plugin list you can use it as any other plugin.

3.7 Troubleshooting

- If the plugin does not appear in the plugin list: make sure that the plugin file is located in the correct directory. Also, make sure that the name of the plugin starts with "*plugin_*".See other plugins for examples.
- If the plugin appears in the plugin list, but no plugin window is opened when the plugin is selected: make sure that you plugin m-file contains the row

```
TSplugin_init(TS, 600, 300);
```

- **If you have other problems or questions: start a thread in the Timestudio forum: http://timestudioproject.com/forum**

Appendix A - List of important functions

TScontrol

[uihandle1 uihandle2] = TScontrol(style, position, description, defaultValue, tooltip, varargin)
Used for adding various types of graphical user interfaces to a plugin.

- style = String value, see below.
- position = Position of control in grid units (grid = 30x30 element layout).
  Example: [1 2 6 3] means x=1, y=2, width=6 and height=3
- description = Text string that shows up on component.
- defaultValue = Default value.
- tooltip = Information that is displayed when the mouse hoovers over the component.
- varargin = Optional parameter pair, such as callbacks for pushbuttons, etc.
  Example: 'Callback', @buttonWasPressed
  For an implemented example of a callback function see the uiSlider in AppendixB.

The type used is specified by the "style" parameter and is supposed to be a string that can have one of the following values:

- 'edit' - Single or multi-line textbox
- 'checkbox' - A standard check box
- 'frame' -
- 'listbox' -
- 'popupmenu' - Sometimes called combobox or dropdown menu
- 'pushbutton' -
- 'radiobutton' -
- 'slider' - A horizontal slider.
- 'togglebutton' -
- 'plot' or 'axis' - Inserts an axis by grid coordinates
- 'knob' -
- 'text' - Text with transparent background

TSplugin_init

pluginFig = TSplugin_init(TS, width, height, varargin)
This function creates a standard plugin window with some predefined components in top: settings, help, comments, output and "Use plugin"-button. This function also implements behaviors for these components. It should be called once in the beginning of your plugin with at lest the three first parameters given. The fourth parameter can be used to set the plugin to be used as a groupfilter**(?)**.

TSplugin_end

TSplugin_end(TS, pluginresult)
Gives a call to a function that updates the layout in Timestudio. Should be called once at the end of the plugin with TS as parameter.

TSshowMessage

[helpFig, helpTxt] = TSshowMessage(message, title, type, positionOffset, videolink)
By giving the first two parameters only it can be used to present formatted text in a figure. The figure will then appear at the center of the screen. Both the message and the title are strings.

Matlab message and dialog boxes

Another way of displaying messages to the user without using the Matlab command window is by using Matlabs standard message or dialog boxes such as 'msgbox' or 'dialog'.
Like TSshowMessage they usually have at least one necessary input which is a message in the form of a string that should be displayed.
For more documentaion please see for example the Matlab help text for these and their associated functions.

TSstrsplit

out = TSstrsplit(string, splitter)
Since the input from a TScontrol of type "edit" is a string it may be necessary to split it into different parts. For example if the user is supposed to give more than one entry in the text field. This can be done with this function, just give the string that should be split and on what characters the splitter should separate the string. The output is a cell array of the string parts the larger string was split into.

TScheckSlashes

path = TScheckSlashes(path)
The slash (/ or \) that is used to traverse folders is different in different operating systems. Windows uses \ and unix based systems such as Mac OS and Linux uses /. In order to make the plugin platform independent in that regard you will have to check so that paths to a certain file or folder given and used are using the correct slash. By entering the path into this function in any format, regarding the slashes, a version of that path that uses the correct slashes will be returned. Path is a string that contains a path, for example 'C:\Users\User1\myDocuments\'. If this is given to this function on a unix system a string of the form 'C:/Users/User1/myDocuments/ will be returned.

Appendix B - Template example

```
% This plugin is a rich template that illustrates some important functionality
% in Timestudio like how to set up a basic graphical user interface and load
% data from and write to timestudio.
%
% NOTE: Are there anything special you maybe should take into consideration when
% using this plugin?

% REVISIONS:
% dd/mm/yyyy - On this date I fixed something that was broken!
% dd/mm/yyyy - On this date I implemented a new feature!
%
% TODO: This should be fixed or this feature should be implemented.
%
% Author: Your Name
% Last revised: dd/mm/yyyy

function template_rich_example(TS, varargin)

%-------------------------------------------------------
% Creates new window 600 pixels wide, 300 pixels high.
%-------------------------------------------------------
TSplugin_init(TS, 600, 300);

%---------------------------------
% Your GUI should be built here
%---------------------------------
uiFieldA= TScontrol('edit',          [1 1 6 1], 'Edit A', 100,  'Enter a number');
uiFieldB= TScontrol('edit',          [1 2 6 1], 'Edit B', 'hi!','Enter a string');
uiSlider= TScontrol('slider',        [7 1 6 1], '', 100, 'Tooltip text', 'Max', 100,
'Min', 0, 'Callback', @sliderChanged);

uiToggle= TScontrol('togglebutton', [7 2 6 1], 'Hello2', 'Text on toggle', 'This is
a button to toggle');
uiCheck = TScontrol('checkbox',      [7 3 1 1], 'Ceck me', 0, 'Check this to
check.');

uiDrop  = TScontrol('popupmenu',     [1 3 6 1], {'Choose'}, {'Value 1', 'Value 2',
'Value 3'}, 'Choose something from the list');
uiList  = TScontrol('listbox',       [1 4 6 4], '', {'Item 1', 'Item 2', 'Item 3'},
'Tooltip text');

%-----------------------------------------------
% Callback function for slider
% uiSlider and uiFieldA are global in this scope
%-----------------------------------------------
function sliderChanged(obj, eventData)
    sliderValue = get(uiSlider, 'Value');
    set(uiFieldA, 'string', num2str(sliderValue));
end
```

```
%------------------
% Process
%------------------
if TSplugin_processing(TS)
    %-------------------------------------------------
    % Read input from the GUI input fields!
    %-------------------------------------------------
    fieldA  = str2double(get(uiFieldA,'string'));
    fieldB  = get(uiFieldB,'string');

    toggle  = get(uiToggle,'value');

    drop    = get(uiDrop,'value');
    list    = get(uiList,'value');

    check   = get(uiCheck, 'value');

    %-------------------------------------------------
    % Display the results of the input!
    %-------------------------------------------------
    fprintf('Value of field A:      %f\n', fieldA);
    fprintf('Value of field B:      %s\n', fieldB);

    fprintf('Value of drop down:    %f\n', drop);
    fprintf('Value of list menu:    %f\n', list);

    if(toggle); fprintf('Toggle button is ON! \n');
    else        fprintf('Toggle button is OFF!\n');
    end

    if(check);  fprintf('Checkbox is checked!    \n');
    else        fprintf('Checkbox is NOT checked!\n');
    end

    %-----------------------------------------------------
    % Display some information about the marked subject
    %-----------------------------------------------------
    subjnr = TS.ALLINFO.currentSubjectNr;
    fprintf('\nAbout current subject:\n')
    disp(TS.ALLSUBJ(subjnr))

    fprintf('\n\n\n\n');

end % of processing

%-- Save the results, do not edit below! --%
TSplugin_end(TS);
end

%-----------------------------------------------------------------
% This is a subfunction only visable to functions in this file.
% It takes an input named "in" and gives an output named "out".
%-----------------------------------------------------------------
function out = change(in)
    out = 2*in;
end
```