This post describes aspects of the Firaxis Live Tuner (FLT). The information herein was discovered by a moderately experienced Civ gamer who was new to FLT. The posting describes building a Live Tuner Panel (LTP) and, to the amount discovered, identifies the location of pertinent FTP files in a Steam-configured Civ VI installation on a Windows 10 machine, which located Civ VI in “C:\Program Files (x86)\Steam\steamapps\common\Sid Meier's Civilization VI”. For the sake of brevity this post will represent this location with a tilde (~).

Just to set expectations, the post is not a step-by-step guide. However, the concepts described should be helpful in developing LTPs. Perhaps the posting will save others the many hours spent learning FLT by ‘black box’ experimentation and testing.

**Opening an LTP**

The first challenge after opening the FLT app was discovering how to open an LTP. It turned out to be straightforward once the location of the panel specifications was known. Click File > Open in the menu bar. On my machine, the ltp files that define panels are in “~\Debug”. Have fun opening and exploring the out-of-the box panels. However, remember FLT interacts with the game that is open.

**Building an LTP**

I’m sure there are as many approaches to building a new LTP as there are FLT users. Maybe more. I’ll describe the approach I used. This time. Perhaps you will find it informative, maybe even inspirational. The example presented here creates a panel to support the user in selecting resources and, when requested, to highlight resources selected on the map. The desire to build the panel arose from my frustration in finding late-game strategic resources, such as oil, on a mostly explored map. Limited by my knowledge of the Civ VI UI, the search required visual scans of the entire map to find the desired resource among many displayed – a needle-in-the-haystack search. Wouldn’t it be nice to automatically highlight resources of interest? As it turns out, a panel with these capabilities was a challenge in that it required building a panel that both loaded Lua code and had access to “InGame” variables. More about this later.

The approach I took (perhaps readers can suggest simpler approaches?) requires three new files and a change to one distribution file (perhaps not for the faint of heart). The new files include an ltp file that defines the panel and a pair of files that define, initialize and load needed variables, including those containing a Lua function (https://www.lua.org/pil/6.html is a useful description of Lua functions).

**Understanding the ltp FIle**

An ltp file defines a panel. It can be created using the \*New Panel\* tab in FLT and modified using a text editor. I used the \*New Panel\* tab to create a template of xml tags and Sublime Text 3 to build a panel called Resource Locator, which is specified in the attached ltp file. BTW, I locate ltp files defining my panels with the ones in the FLT distribution. To differentiate my files from those in the FLT distribution, I begin my files with an underscore (\_) or two.

An ltp is essentially an xml file in which data is tagged. There are two introductory tags - <Name> and <App>. They give the panel a handle and identify Civ6 as the application related to the panel. Next there are a pair of tags, <EnterAction> and <ExitAction>, which can contain Lua code. <EnterAction> specifies behavior when the panel receives focus. Similarly, <ExitAction> specifies behavior when the panel loses focus. The next pair of tags, <CompatibleStates> and <LoadStates >, control the Lua variables to which the panel has access. More about these tags later. The remaining tags specify contents of the panel. They can contain any number of sub elements. Each sub element has tags that specify the name, location and interactive behavior of an LTP item, such as an action, control, view, tracker or list. To be honest, I have only explored the actions, controls, and lists. Perhaps other users with more FLT experience can describe views and trackers. Actions define buttons on the LTP. Controls support display and entry of strings, integers, floating point numbers and Boolean values. Lists display tabular data and support the user in selecting an item among those displayed.

Two tags, whose function is less than obvious, specify the access the panel will have to Lua variables. The Civ app creates what I call *contexts* to control the scope of variables (and thereby access). The <CompatibleStates> tag identifies the contexts to which the panel has access. The <LoadStates> tag loads Lua code that can be used in defining the LTP. However, from my experimentation, code loaded via <LoadStates> is in a context with limited access to Lua variables describing game data, such as those available in the InGame Compatible State (CS). Clearly, setting the values of these tags would be critical to building the Resource Locator panel.

*<CompatibleStates>*

Value(s) of the <String> tag in <CompatibleStates> can be selected by checking boxes on \*New Panel\*. They can also be added to the ltp code in a text editor. Note, once the panel is opened, FLT allows selection among the <String> items specified in <CompatibleStates>. Each item appears in the left most selection box of the panel’s menu bar. The item selected specifies the context, and thereby the variables, to which the panel has access. Note that the panel may behave differently depending on the CS selected.

So far in my FLT exploration, I have identified two ways to define new <CompatibleStates>. By far the simplest is to create one by using <LoadStates> to load a Lua file. This method is described in the following paragraph. The second way involves loading Lua code during the creation of a game. More capable and more complicated. It’s described in the next section.

*<LoadStates>*

Value(s) of the <String> tag in <LoadStates> load Lua files, whose code can prepare the environment for the panel by doing such things as loading event handlers or initializing variables. Each <LoadStates> value identifies the name of a pair of files, one of type “.xml” and one of type “.lua”, which, if located in “~\Base\Assets\UI”, is loaded when the panel opens. <LoadStates> values create Compatible States. As an experiment, use <LoadStates> to load an xml/lua pair and go to the \*New Panel\* tab. The name of the Load State (LS) will appear in the checkboxes.

The xml file can be very simple (see the one used for the Resource Locator panel), but I believe it is needed to load the Lua file. The Lua file can define variables that can be used in ltp panels, but only if the LS is identified in <CompatibleStates> and the CS is selected in the in the left most selection box of the panel’s menu bar. Good news. We can use <LoadStates> to create Lua variables for a new panel.

However, there’s a catch. LS variables are only available to the panel when their CS is selected in the FLT. So, if your panel needs to access variables available in “InGame” and you need to define other variables in a Lua file you would like to load with the panel, you’re stuck. One CS supports the panel’s access to the InGame variables, but not your variables. The other CS supports the panel’s access to your variables but not the InGame variables.

The challenge I faced in developing the Resource Locator panel was to define variables in Lua code that support the ltp file and have those variables accessible to the ltp file along with the variables provided by the “InGame” CS. The next section discusses my solution.

**Building a General-Purpose Context**

Before describing an approach to defining a CS that provides access to both InGame and special-purpose variables, I would like to present the results of an examination of Civ VI distribution files. The following observations have been, at least in part, verified by experimentation:

* Civ VI defines contexts and related CSs during the initialization of a game.
* Civ VI defines all CSs that are initially in the LTP panel’s dropdown in three pyramids.
* InGame.xml, Civ6\_DebugContext.xml and TutorialUIRoot.xml are each at the top of a pyramid.
  + Each of these creates a CS and identifies “child” CSs.
  + Lua files of the same name, but of type lua (e.g., InGame.lua), specify the context.
* <LuaContext> items in an xml file of one of the three pyramids specify “child” CSs.
  + The ID parameter is the name of the CS.
  + The FileName parameter identifies:
    - The lua file that specifies context variables.
    - The accompanying xml file, which can identify “child” CSs.
    - There must be both an xml and a lua file named FileName.
* A context “inherits” variable accesses from their “parent”.
* Civ VI monitors the files that specify a pyramid and reloads changed files.
* There are other xml/lua file pairs in the distribution that do not create CSs. These are loaded via a <LuaContext> item in one or more of the following:
  + MultiplayerSelect.xml
  + FrontEnd.xml
  + InGameGameSettingsMenu.xml
  + StartupScreen.xml
  + WorldBuilder.xml.

Based on these observations, the Resource Locator panel was built by adding a new CS to the InGame pyramid. This required a one line addition to the InGame.xml distribution file. Without understanding the <Container> parameter, I mimicked the <LuaContext> entry for “AI\_Diplomacy” and added a new <LuaContext>, EFN\_Tuner, to the “Core” container. Additionally, I created EFN\_Tuner.xml and EFN\_Tuner.lua and placed them in ~\Base\Assets\UI. EFN\_Tuner.xml is very simple. It only contains an <xml> and a <Context> tag. EFN\_Tuner.lua initializes variables needed by the LTP, including a function that determines if a player can “see” a resource on the map (e.g., whether the player has researched Bronze Working and can therefore see Iron on the map). Finally, I set <CompatibleStates> to EFN\_Tuner in \_\_Resource Locator.ltp. When a game loads, EFN\_Tuner creates a CS that has access to InGame variables and the new variables specified in EFN\_Tuner.lua.

Works good. Try it out!

**Additional Observations**

Civ VI creates log files that are very useful in understanding behavior and debugging. On my installation, they are located in “C:\Users\genen\Documents\My Games\Sid Meier's Civilization VI\Logs”. Of particular use is Lua.log. It’s associated with io.stdout and, as such, it captures all print output.

On a final note, during my examination of the Civ VI distribution files, I came across irregularities. I have corrected them in my Civ VI files and have experienced no unusual behavior. I’m guessing the xml parser Civ VI uses is insensitive to the irregularities. However, the parser I used in my exploration was sensitive. I tired of the warnings it generated and made the following changes:

1. ~\Base\Assets\UI\Civ6\_Styles.xml – Repeated |/>| removed from the end of lines 2034 and 2014.
2. ~\Base\Assets\UI\Choosers|TradeRouteChooser.xml - Repeated |/>| removed from the end of line 48.
3. ~\Base\Assets\UI\EndGame\EndGameMenu.xml - |/>| removed before |Anchor=”R,C” />| in line 210.
4. ~\Base\Assets\UI\Panels\CityPanel.xml - Repeated |>| removed from the end of line 172 .
5. ~\Base\Assets\UI\Popups\EspionagePopup.xml - Repeated |/>| removed from the end of lines 112 and 117.
6. ~\DLC\AztecMontezuma\Data\Aztec\_Montezuma\_Data.xml – changed tag end name </AiFavoredITems> to </AiFavoredItems> to match the start tag in line 185. “Outdented” lines 185 through 213.

**Posting Notes:**

1. I would have posted in the Civ6 - Modding Tutorials & Reference page, but I have insufficient privileges. Perhaps someone with appropriate privileges can determine if the post should be moved?
2. I included the word document in which I wrote the post. It better preserves formatting.
3. I added “.txt” to the file names of the files need to create the Resource Locator panel. This allowed me to upload them. Perhaps there is another more appropriate way to share files?