

Continued from Part 1

Button 2 - City States List

An instance can be any UI component and when not buttons are usually text labels in a list. While creating instances is a useful technique when multiple similar items are known in advance, the real power of them comes when creating items that are not known in advance - for example a list of all the City States in the current game.

Change the "UI/Button.xml" file to

```
<?xml version="1.0" encoding="utf-8" ?>
<Context>
    <Box Style="BGBlock_ClearTopBar" />

    <Instance Name="CityState">
        <Label ID="Label" Anchor="L,T" Font="TwCenMT20" FontStyle="Shadow"
ColorSet="Beige_Black_Alpha" />
    </Instance>

    <Grid Size="600,400" Anchor="C,C" Style="Grid9DetailFive140"
ConsumeMouse="1">
        <Label ID="Message" Anchor="C,T" Offset="0,50" Font="TwCenMT24"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha"
String="TXT_KEY_TEST_BUTTON_CS_TITLE"/>

        <ScrollPanel ID="CSPanel" Anchor="C,C" Size="200,200"
Vertical="1">
            <Stack ID="CSStack" Anchor="C,T" StackGrowth="Bottom"
Padding="5"/>

            <ScrollBar Offset="0,18" Anchor="R,T" AnchorSide="O,I"
Style="VertSlider" Length="164" />
            <UpButton Offset="0,0" Anchor="R,T" AnchorSide="O,I"
Style="ScrollBarUp" />
            <DownButton Offset="0,1" Anchor="R,B" AnchorSide="O,I"
Style="ScrollBarDown" />
        </ScrollPanel>

        <GridButton ID="OK" Size="140,36" Anchor="C,B" Offset="0,50"
Style="BaseButton" ToolTip="TXT_KEY_TEST_BUTTON_BUTTON_OK_TT">
            <Label Anchor="C,C" Offset="0,-2"
String="TXT_KEY_TEST_BUTTON_BUTTON_OK" Font="TwCenMT24"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha" />
        </GridButton>
    </Grid>
</Context>
```

and the "UI/Button.lua" file to

```
function OnOK()
    ContextPtr:SetHide(true)
end
Controls.OK:RegisterCallback(Mouse.eLClick, OnOK)

function Init()
    for iCS = GameDefines.MAX_MAJOR_CIVS, GameDefines.MAX_CIV_PLAYERS-1,
1 do
        local pCS = Players[iCS]
        if pCS:IsEverAlive() then
            local instance = {}
            ContextPtr:BuildInstanceForControl("CityState", instance,
Controls.CSSStack)

            instance.Label:SetText(pCS:GetName())
        end
    end

    Controls.CSSStack:CalculateSize()
    Controls.CSSStack:ReprocessAnchoring()
    Controls.CSPanel:CalculateInternalSize()
end

Init()
```

and add the following to the "XML/ButtonText.xml" file

```
<Row Tag="TXT_KEY_TEST_BUTTON_CS_TITLE">
    <Text>City States</Text>
</Row>
```

save the changes, rebuild the mod, restart Civ 5, re-enable the mod and start a new game.



The dialog displays a list of all the City States in the current game. Before looking at how the City States get into the list, we need to look at how the list is defined.

```
<ScrollPanel ID="CSPanel" Anchor="C,C" Size="200,200" Vertical="1">
  <Stack ID="CSStack" Anchor="C,T" StackGrowth="Bottom" Padding="5"/>

  <ScrollBar Offset="0,18" Anchor="R,T" AnchorSide="O,I"
Style="VertSlider" Length="164" />
  <UpButton Offset="0,0" Anchor="R,T" AnchorSide="O,I"
Style="ScrollBarUp" />
  <DownButton Offset="0,1" Anchor="R,B" AnchorSide="O,I"
Style="ScrollBarDown" />
</ScrollPanel>
```

The list itself is just an empty stack, that will be populated by instances of City State names. But as the list will very likely be too long to fit in the dialog, it has been placed in a `<ScrollPanel>` element.

A `ScrollPanel` (like `Grid`, `Box`, etc) delimits an area of the screen that other controls can be placed within. The difference with the `ScrollPanel` is that if the contents don't fit, rather than overflowing the edges, they are clipped to the defined area and a scroll bar added so the user can view different sections of the total area.

Scroll panels will scroll their contents vertically (`Vertical="1"`) or horizontally (`Vertical="0"`) but not both at the same time. The nested `<ScrollBar>`, `<UpButton>` and `<DownButton>` elements define where the scroll bar will appear. The standard for vertically scrolling panels is on the right hand side, so most of the time we don't need to alter the majority of the attributes on these elements. The only one we regularly need to change is the `Length` attribute on the `ScrollBar` element and that should be set to "Height of scroll panel minus two times eighteen" - in our case $200 - 2 \times 18 = 164$

After we have placed items into a stack which is contained within a scroll panel, in addition to the "two lines of house keeping code" for the stack, we need to add an extra line for the scroll panel.

```
Controls.CSStack:CalculateSize()  
Controls.CSStack:ReprocessAnchoring()  
Controls.CSPanel:CalculateInternalSize()
```

The contents of the list is just the City State name, so our <Instance> just contains a <Label> element

```
<Instance Name="CityState">  
  <Label ID="Label" Anchor="L,T" Font="TwCenMT20" FontStyle="Shadow"  
  ColorSet="Beige_Black_Alpha" />  
</Instance>
```

which by now should all be familiar!

The code to fill in the list is

```
for iCS = GameDefines.MAX_MAJOR_CIVS, GameDefines.MAX_CIV_PLAYERS-1, 1  
do  
  local pCS = Players[iCS]  
  if pCS:IsEverAlive() then  
    local instance = {}  
    ContextPtr:BuildInstanceForControl("CityState", instance,  
Controls.CSStack)  
  
    instance.Label:SetText(pCS:GetName())  
  end  
end
```

which basically loops over all possible City State "players" ("for iCS = GameDefines.MAX_MAJOR_CIVS, GameDefines.MAX_CIV_PLAYERS-1, 1 do") works out if they were in play at the start of the game ("local pCS = Players[iCS] if pCS:IsEverAlive() then") and, if they were, creates a new instance and sets the label to the name of the City State ("pCS:GetName()").

Which is great if you want to know all the City States there were at the start of the game, but typically you want a list of all those City States that you have met and have not been captured.

Button 2b - Met City States List

"We want a list of all City States we have met and have not been captured." Sounds so simple!

The problem is that this list (potentially) changes every time we look at it. So far we have only added items to a stack, but the "and have not been captured" requirement means that we will need to remove any that are captured after we have met them. Additionally, we run the risk of creating duplicates - first time we have met City States X and Z, so the list contains "X, Z", second time we look at the list having also met Y, if we just keep adding the list will contain "X, Z, X, Y, Z". Rather than keeping track of which City States we have added, it is simpler to empty the list each time we

look at it and then fill it with all met and not captured City States. So this raises the question "How do we remove items from a stack"?

While there are two Lua methods to remove items from a stack - `ReleaseChild()` and `DestroyAllChildren()` - these methods appear to contain subtle errors and over the course of a game "leak memory" and cause Civ 5 to crash. It would also appear that the developers are aware of this problem as they have provided an alternative mechanism to `BuildInstanceForControl()` / `ReleaseChild()` / `DestroyAllChildren()` which does not suffer from this problem.

There are no changes to "UI/Button.xml"

Change the "UI/Button.lua" file to

```
include("InstanceManager")
local g_CSManager = InstanceManager:new("CityState", "Label",
Controls.CSSStack)

function OnShowHide(bHide, bInit)
    if (not bHide) then
        UpdateCsList()
    end
end
ContextPtr:SetShowHideHandler(OnShowHide)

function OnOK()
    ContextPtr:SetHide(true)
end
Controls.OK:RegisterCallback(Mouse.eLClick, OnOK)

function UpdateCsList()
    local iTeam = Game.GetActiveTeam()

    g_CSManager:ResetInstances()

    for iCS = GameDefines.MAX_MAJOR_CIVS, GameDefines.MAX_CIV_PLAYERS-1,
1 do
        local pCS = Players[iCS]
        if pCS:IsEverAlive() then
            if (pCS:IsAlive() and Teams[pCS:GetTeam()]:IsHasMet(iTeam)) then
                local instance = g_CSManager:GetInstance()

                instance.Label:SetText(pCS:GetName())
            end
        end
    end

    Controls.CSSStack:CalculateSize()
    Controls.CSSStack:ReprocessAnchoring()
    Controls.CSPanel:CalculateInternalSize()
```

```
end
```

Save the changes, rebuild the mod, restart Civ 5, re-enable the mod and start a new game.

The initial dialog will be empty, so dismiss it with the OK button. Found your capital, build a Scout or two and explore until you've found a couple of City States.

Using FireTuner show the dialog again, the list should have entries for those City States. Dismiss the dialog and wander around a bit more until you've met a few City States. Show the dialog again



(Obviously your list will be different and depends on which City States are in your game and which ones you have met!) The important thing is that there are no duplicates, despite having looked at the list twice.

So what's the trick to removing list items? Before we look at that we need to cover a few other things in the Lua file.

```
ContextPtr:SetShowHideHandler(OnShowHide)
```

A "handler" is just another term for a "callback". This Lua code tells the core game engine that we want to be told every time our context is either shown or hidden.

```
function OnShowHide(bHide, bInit)
    if (not bHide) then
        UpdateCsList()
    end
end
```

Our code is only interested when it is being shown (ie "not hidden") as it then needs to update the city state list - this is what causes the contents of the list to be updated every time we look at the dialog.

```
include("InstanceManager")
```

We use the "InstanceManager" provided by Civ 5 to create and destroy the entries in the list (a bit like we used the standard IconSupport for cutting out icons)

The InstanceManager allows us to manage many instances so we need to create a special manager for our City State stack

```
local g_CSManager = InstanceManager:new("CityState", "Label",  
Controls.CSSStack)
```

What you need to know about that line is that "CityState" refers to the name of the <Instance> we want to use from our context, "Label" refers to the main UI component in the <Instance> (remember there can only be one main or outer component in an Instance) and Controls.CSSStack is the control we are going to be adding to.

The UpdateCsList() function is where the hard work happens

```
function UpdateCsList()  
    local iTeam = Game.GetActiveTeam()  
  
    g_CSManager:ResetInstances()  
  
    for iCS = GameDefines.MAX_MAJOR_CIVS, GameDefines.MAX_CIV_PLAYERS-1,  
    1 do  
        local pCS = Players[iCS]  
        if pCS:IsEverAlive() then  
            if (pCS:IsAlive() and Teams[pCS:GetTeam()]:IsHasMet(iTeam)) then  
                local instance = g_CSManager:GetInstance()  
  
                instance.Label:SetText(pCS:GetName())  
            end  
        end  
    end  
  
    Controls.CSSStack:CalculateSize()  
    Controls.CSSStack:ReprocessAnchoring()  
    Controls.CSPanel:CalculateInternalSize()  
end
```

Apart from the extra code to check if we have met the City State and if it is still alive (ie not captured) there are only two differences from the previous code.

The first bold line ("g_CSManager:ResetInstances()") tells our City State specific instance manager to "remove" any existing instances from the list, while the second bold line ("local instance = g_CSManager:GetInstance()") creates an instance for us to fill in.

In general, if you are only adding to a stack "Think BuildInstanceForControl()" but if you need to be able to remove items (or empty it completely) "Think InstanceManager".

Button 2c - Sorting Lists

A very common requirement of lists is to be able to sort them.

Change the "UI/Button.lua" file to (differences are in bold)

```
include("InstanceManager")
local g_CSManager = InstanceManager:new("CityState", "Label",
Controls.CSStack)

local g_SortTable

function OnShowHide(bHide, bInit)
    if (not bHide) then
        UpdateCsList()
    end
end
ContextPtr:SetShowHideHandler(OnShowHide)

function OnOK()
    ContextPtr:SetHide(true)
end
Controls.OK:RegisterCallback(Mouse.eLClick, OnOK)

function SortByName(a, b)
    local sNameA = g_SortTable[tostring(a)].Name
    local sNameB = g_SortTable[tostring(b)].Name

    return sNameA < sNameB
end

function UpdateCsList()
    local iTeam = Game.GetActiveTeam()

    g_CSManager:ResetInstances()
    g_SortTable = {}

    for iCS = GameDefines.MAX_MAJOR_CIVS, GameDefines.MAX_CIV_PLAYERS-1,
1 do
        local pCS = Players[iCS]
        if pCS:IsEverAlive() then
            if (pCS:IsAlive() and Teams[pCS:GetTeam()]:IsHasMet(iTeam)) then
                local sCsName = pCS:GetName()

                local instance = g_CSManager:GetInstance()
                g_SortTable[tostring(instance.Label)] = {Name=sCsName}
            end
        end
    end
end
```


UI Tutorial - Dynamic Buttons

```
        instance.Label:SetText(sCsName)
    end
end
end
```

Controls.CSStack:SortChildren(SortByName)

```
Controls.CSStack:CalculateSize()
Controls.CSStack:ReprocessAnchoring()
Controls.CSPanel:CalculateInternalSize()
end
```

Save the changes, rebuild the mod, restart Civ 5, re-enable the mod and start a new game. Do the "wander around until you've met a few City States" thing, then display the dialog (via FireTuner)



The City State list is in alphabetical order!

In order to be able to sort the list we have to keep a note for each list item of what we want to sort it on - the City State name in this case. We do this by creating a table that associates the list entry with the City State name - called `g_SortTable`

```
local g_SortTable
```

Just after emptying the list, we also empty the sort table

```
g_SortTable = {}
```

then, as we create instances and add them to the stack, we also create an association between the entry and the City State name in the sort table

UI Tutorial - Dynamic Buttons

```
g_SortTable[tostring(instance.Label)] = {Name=sCsName}
```

That "tostring(instance.Label)" looks a little strange! It's a necessity of the way that stacks store their entries and how we need to do the sorting, no need to understand what it's doing, you just need to know that the ".Label" bit **must** be the ID of the main component in the <Instance>

Finally we do the sorting ("Controls.CSSStack:SortChildren(SortByName)") by telling the stack to sort its children using our "SortByName()" function to compare two entries and decided which should appear first in the list.

The SortByName() function

```
function SortByName(a, b)
    local sNameA = g_SortTable[tostring(a)].Name
    local sNameB = g_SortTable[tostring(b)].Name

    return sNameA < sNameB
end
```

receives two values, called "a" and "b" here, which it must decide which one is to appear first in the list. If "a" should appear first, the function must return "true", otherwise "b" should appear first and it must return "false". Unfortunately, we don't receive the names of the City States in the list, but a value that represents the Label control. Which is why we stored those strange "tostring" things in the g_SortTable, as we can now use that table to look up the City State names, which is what these two lines do

```
local sNameA = g_SortTable[tostring(a)].Name
local sNameB = g_SortTable[tostring(b)].Name
```

with the actual City State names in hand we can now make our decision as to which should come first

```
return sNameA < sNameB
```

Summary

This tutorial has covered the Civ 5 User Interface (UI) elements and techniques for creating dynamic buttons and scrollable lists from Lua, and they are

- Dynamic Controls
 - Instance
 - :BuildInstanceForControl()
 - :DestroyAllChildren() and :ReleaseChild() (and why **not** to use them)
 - InstanceManager
- Scrolling
 - ScrollPanel
 - DownButton
 - ScrollBar
 - UpButton
- Sorting
 - :SortChildren()

All of the XML and Lua code for the examples in this tutorial can be downloaded from the Mod Hub as "Test - UI Tutorial - 2 Dynamic Buttons" (in the Other category). Each button step is included separately and can be displayed from the FireTuner Lua Console tab by selecting the Button context and then entering "ShowN()" in the command line (where N is the step to display, eg "Show1()", "Show2c()", etc)

Part 3 of this series of UI Tutorials will cover creating drop-down menus.