Civilization 5 UI Tutorial

Part 2, Dynamic Buttons

Version: 1.0

Status: Draft

Date: 11 May 2012

Author: William Howard

Contents

Overview

The objective of this tutorial is to illustrate how to create Civ 5 User Interface (UI) elements (specifically buttons) from Lua code.

The tutorial takes the practical approach - first we will write some "code", then we will look at what it does. At each step we will have working UI dialogs - which may or may not in themselves be useful - but which can be used as a working starting point for your own UI mods.

This tutorial assumes you have used ModBuddy and FireTuner to create your own simple non-UI mods, and also that you have a basic understanding of XML. Some Lua will be needed, but that will be explained in each step.

All the code in this tutorial can be downloaded from the in-game ModHub as "Test - UI Tutorial - 2 Dynamic Buttons" found under the "Other" category.

So, to start we need a ModBuddy project ...

Create a Button Mod

Using ModBuddy, create a new mod called "Test - Button" (or some such).

To this mod add the two folders "UI" and "XML". In the UI folder create the two files "Button.xml" and "Button.lua" (delete the standard content added to these files). In the XML folder create the file "ButtonText.xml" (you can leave the standard content in this file).

In the mod's properties, on the "Mod Info" tab, uncheck "Affects Saved Games". On the "Actions" tab, add an "On Mod Activated - Update Database" entry for "XML/ButtonText.xml". On the "Content" tab, add an "InGameUIAddin" for "UI/Button.xml".

Save the project.

Button 1 - Review of Tutorial 1, Dialog 5b

In "Tutorial 1 - The Basics", we covered the common components needed to construct a dialog for presenting information to the user and getting their response (in the form of button clicks). Dialog 5b of that tutorial contained three buttons to change the language used to greet the user. This tutorial will build on that code so our first task is to copy it into our new ModBuddy project.

Add the following to the "UI/Button.xml" file

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

  <Grid Size="600,400" Anchor="C,C" Style="Grid9DetailFive140"
ConsumeMouse="1">
```

```
    <Label ID="Message" Anchor="C,C" Font="TwCenMT24"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha"
String="TXT_KEY_TEST_BUTTON_MESSAGE"/>

    <Stack Anchor="C,B" Offset="0,100" StackGrowth="Right"
Padding="10">
      <GridButton ID="LangEN" Size="140,36" Style="BaseButton">
        <Label Anchor="C,C" Offset="0,-2"
String="TXT_KEY_TEST_BUTTON_BUTTON_ENGLISH" Font="TwCenMT24"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha" />
      </GridButton>

      <GridButton ID="LangFR" Size="140,36" Style="BaseButton">
        <Label Anchor="C,C" Offset="0,-2"
String="TXT_KEY_TEST_BUTTON_BUTTON_FRENCH" Font="TwCenMT24"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha" />
      </GridButton>

      <GridButton ID="LangDE" Size="140,36" Style="BaseButton">
        <Label Anchor="C,C" Offset="0,-2"
String="TXT_KEY_TEST_BUTTON_BUTTON_GERMAN" Font="TwCenMT24"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha" />
      </GridButton>
    </Stack>

    <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>
```

Add the following to the "UI/Button.lua" file

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

local leaderName = GameInfo.Leaders[Players[
Game.GetActivePlayer()]:GetLeaderType()].Description
Controls.Message:LocalizeAndSetText(
"TXT_KEY_TEST_BUTTON_HELLO_LEADER", leaderName)

function OnLangEN()
```

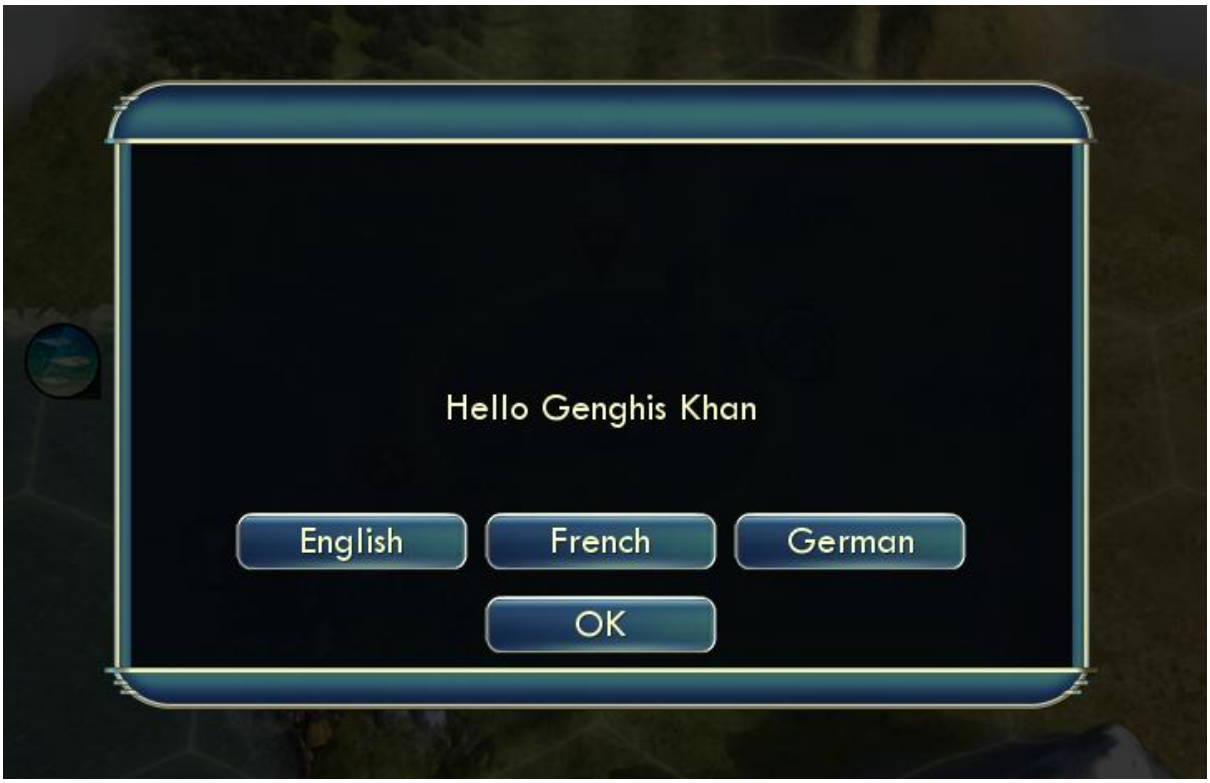
```
  Controls.Message:LocalizeAndSetText(
"TXT_KEY_TEST_BUTTON_HELLO_LEADER",
Players[Game.GetActivePlayer()]:GetName())
end
Controls.LangEN:RegisterCallback(Mouse.eLClick, OnLangEN)

function OnLangFR()
  Controls.Message:LocalizeAndSetText(
"TXT_KEY_TEST_BUTTON_BONJOUR_LEADER",
Players[Game.GetActivePlayer()]:GetName())
end
Controls.LangFR:RegisterCallback(Mouse.eLClick, OnLangFR)

function OnLangDE()
  Controls.Message:LocalizeAndSetText(
"TXT_KEY_TEST_BUTTON_GUTENTAG_LEADER",
Players[Game.GetActivePlayer()]:GetName())
end
Controls.LangDE:RegisterCallback(Mouse.eLClick, OnLangDE)
```

Add the following to the "XML/ButtonText.xml" file

```
<?xml version="1.0" encoding="utf-8"?>
<GameData>
  <Language_en_US>
    <Row Tag="TXT_KEY_TEST_BUTTON_HELLO_LEADER">
      <Text>Hello {1_LeaderName}</Text>
    </Row>
    <Row Tag="TXT_KEY_TEST_BUTTON_BONJOUR_LEADER">
      <Text>Bonjour {1_LeaderName}</Text>
    </Row>
    <Row Tag="TXT_KEY_TEST_BUTTON_GUTENTAG_LEADER">
      <Text>Guten Tag {1_LeaderName}</Text>
    </Row>

    <Row Tag="TXT_KEY_TEST_BUTTON_BUTTON_ENGLISH">
      <Text>English</Text>
    </Row>
    <Row Tag="TXT_KEY_TEST_BUTTON_BUTTON_FRENCH">
      <Text>French</Text>
    </Row>
    <Row Tag="TXT_KEY_TEST_BUTTON_BUTTON_GERMAN">
      <Text>German</Text>
    </Row>

    <Row Tag="TXT_KEY_TEST_BUTTON_BUTTON_OK">
      <Text>OK</Text>
    </Row>
    <Row Tag="TXT_KEY_TEST_BUTTON_BUTTON_OK_TT">
```

```
        <Text>Left click to dismiss the popup</Text>
      </Row>
    </Language_en_US>
  </GameData>
```

Save the files and build the mod. Start Civ 5, enable the mod, and start a new game. In the middle of the screen you should see the "Hello Leader" dialog box



That's good - we haven't broken anything! (I've removed the decoration for clarity.)

Now, what if we want to add another language? As mentioned in the first tutorial we would need to add the XML elements and the Lua code for it. We wouldn't need to rearrange the buttons as they are in a stack, but we would need to edit the size of all of them (as four buttons won't fit across the screen).

There must be an easier way? There is a "better" way, but in the short run it's not easier, but will pay dividends in the longer term.

In technical terms, the English, French and German buttons are "instances" of a general language button. The general language button has an appearance (style, colour, size, mouse-over event, etc) but no specific details (text or greeting displayed). The English instance of the language button takes its appearance from the general language button and supplies the specific English text and greeting to display, likewise the French instance uses the general appearance and provides the specific French text and greeting.

We need the XML and Lua equivalent of this.

Button 1b - Button Instances

Copy the following to the "UI/Button.xml" file

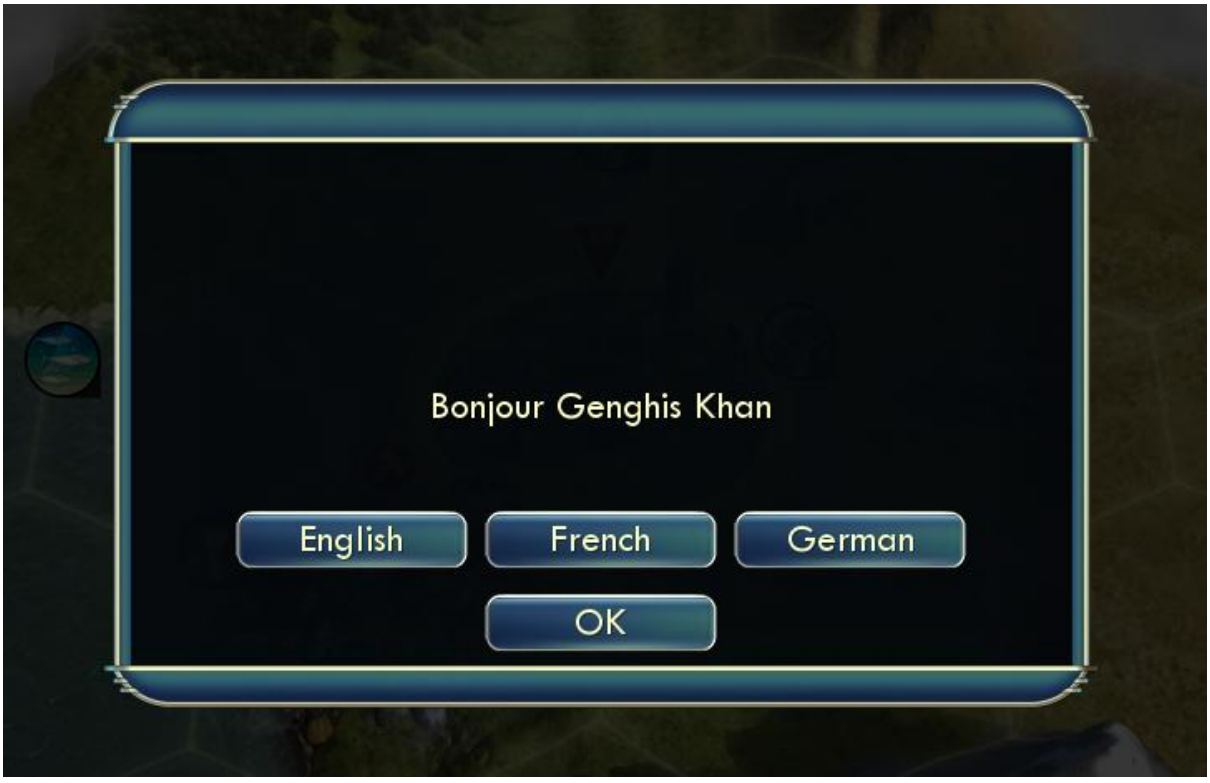
```
<?xml version="1.0" encoding="utf-8" ?>
<Context>
  <Box Style="BGBlock_ClearTopBar" />

  <Instance Name="LangButton">
    <GridButton ID="Button" Size="140,36" Style="BaseButton">
      <Label ID="Label" Anchor="C,C" Offset="0,-2" Font="TwCenMT24"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha" />
    </GridButton>
  </Instance>

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

    <Stack ID="LangStack" Anchor="C,B" Offset="0,100"
StackGrowth="Right" Padding="10"/>

    <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>
```

Copy the following to the "UI/Button.lua" file

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

local m_Languages = {
  {Lang="EN", Text="TXT_KEY_TEST_BUTTON_BUTTON_ENGLISH",
Greeting="TXT_KEY_TEST_BUTTON_HELLO_LEADER"},
  {Lang="FR", Text="TXT_KEY_TEST_BUTTON_BUTTON_FRENCH",
Greeting="TXT_KEY_TEST_BUTTON_BONJOUR_LEADER"},
  {Lang="DE", Text="TXT_KEY_TEST_BUTTON_BUTTON_GERMAN",
Greeting="TXT_KEY_TEST_BUTTON_GUTENTAG_LEADER"},
}

function OnLang(iLang)
```

```
  Controls.Message:LocalizeAndSetText(m_Languages[iLang].Greeting,
Players[Game.GetActivePlayer()]:GetName())
end

function Init()
  for iLang, lang in ipairs(m_Languages) do
       local instance = {}
    ContextPtr:BuildInstanceForControl("LangButton", instance,
Controls.LangStack)

    instance.Label:LocalizeAndSetText(lang.Text)

     instance.Button:RegisterCallback(Mouse.eLClick, OnLang)
     instance.Button:SetVoid1(iLang)
  end

 Controls.LangStack:CalculateSize()
 Controls.LangStack:ReprocessAnchoring()

  OnLang(1)
end

Init()
```

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



Nothing has changed (except I clicked on a different button to get the greeting in a different language before capturing the screen), but how can that be as our <Stack> element is clearly empty

```
<Stack ID="LangStack" Anchor="C,B" Offset="0,100" StackGrowth="Right"
Padding="10"/>
```

The answer, of course, is that the language buttons were added by the Lua code. Before we look at the Lua, we need to understand the <Instance> element in the context.

```
  <Instance Name="LangButton">
    <GridButton ID="Button" Size="140,36" Style="BaseButton">
      <Label ID="Label" Anchor="C,C" Offset="0,-2" Font="TwCenMT24"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha" />
    </GridButton>
  </Instance>
```

The <Instance> element defines a generic instance of something (usually a button) that Lua code can then create specific variants of.

The Instance element must have a Name attribute and contain one main UI component which itself must have an ID attribute. The main UI component can then contain other components that may or may not need to have ID attributes. If you need to place more than one UI component directly within the <Instance> element, don't! Use a <Container> element as the main UI component and place the other components within that.

Our Instance contains a single GridButton that looks like our French, English and German buttons without the String attribute on the enclosed <Label> element. Which is what we would expect - we have defined the general appearance of the button, without any of the specifics.

So how do we add the buttons to the stack? With Lua of course! And as there's quite a lot of it, well break it down into small chunks.

Firstly, we need to define how many buttons we are going to have and what the specifics of each button are - to do this we use a Lua table, where each entry is a set

```
local m_Languages = {
  {Lang="EN", Text="TXT_KEY_TEST_BUTTON_BUTTON_ENGLISH",
Greeting="TXT_KEY_TEST_BUTTON_HELLO_LEADER"},
  {Lang="FR", Text="TXT_KEY_TEST_BUTTON_BUTTON_FRENCH",
Greeting="TXT_KEY_TEST_BUTTON_BONJOUR_LEADER"},
  {Lang="DE", Text="TXT_KEY_TEST_BUTTON_BUTTON_GERMAN",
Greeting="TXT_KEY_TEST_BUTTON_GUTENTAG_LEADER"},
}
```

Don't Panic! All this says is that we have a table (called m_Languages) that has three entries (one each for English, French and German - in that order) and each entry (language) has three pieces of information associated with it - the ISO language code (Lang), the text to put on the button (Text) and the greeting message (Greeting). To get the button text for English (the first language) we can use "m_Languages[1].Text" and the greeting for German is "m_Languages[3].Greeting". (The ISO language code is unused, but makes the table slightly easier to read.)

Right, so we need to read each language entry out of the table in turn, create a specific variant of the generic language button and add it to the stack! Let's break that down a bit.

```
for iLang, lang in ipairs(m_Languages) do
  local instance = {}
  ContextPtr:BuildInstanceForControl("LangButton", instance,
Controls.LangStack)

  instance.Label:LocalizeAndSetText(lang.Text)
end
```

The first line "for iLang, lang in ipairs(m_Languages) do" reads each language out of the table in turn. In "iLang" we get the index (1 for English, 2 for French, etc) and in "lang" we get the details of the language (so lang.Text, lang.Greeting, etc)

The second line "local instance = {}" creates an empty instance, while the third line "ContextPtr:BuildInstanceForControl("LangButton", instance, Controls.LangStack)" fills the instance in with the generic information from the <Instance Name="LangButton"> element in the context and adds it to the "Controls.LangStack" control.

The fourth line "instance.Label:LocalizeAndSetText(lang.Text)" sets the specific language text (lang.Text) on the label of the button.

The fifth line "end" just closes the loop in the first line.

We skipped two lines, but we'll come back to those in a bit.

After we've created each of the buttons and added them to the stack, we have a bit of house-keeping to do. When the game engine first created the context the stack was empty, so it took up no space on the screen. Now we've put something into the stack, we need to tell the game engine and get it to re-layout the components. The following two lines of Lua do this

```
Controls.LangStack:CalculateSize()
Controls.LangStack:ReprocessAnchoring()
```

how, we don't really need to know, you just need to remember to add those two lines anytime you add items to a stack from Lua code.

OK, so we now have our three buttons back, but if we left it like that they wouldn't do anything as we've not registered callbacks for them.

If we needed one callback per button, we wouldn't really be gaining anything from using instances to create dynamic buttons. Fortunately, we can pass a numeric value to a callback, and we can use that value to determine which button was pressed. The obvious value to pass in our case is the language index. Which is what the two skipped lines do

```
instance.Button:RegisterCallback(Mouse.eLClick, OnLang)
instance.Button:SetVoid1(iLang)
```

The first line "instance.Button:RegisterCallback(Mouse.eLClick, OnLang)" registers the "OnLang()" callback to the button - so every button gets the same callback.

The second line "instance.Button:SetVoid1(iLang)" tells the button what value to pass to the callback - in this case iLang, which is 1 for English, 2 for French etc.

The callback now needs to collect the value, and use it to select the appropriate greeting

```
function OnLang(iLang)
  Controls.Message:LocalizeAndSetText(m_Languages[iLang].Greeting,
Players[Game.GetActivePlayer()]:GetName())
end
```

So what was the point? We have reduced the XML a bit, but made both the XML and the Lua far more complex.

Button 1c - Adding Another Language

OK, let's add another language. To do this for Button 1 is a lot of work, for Button 1b it is as trivial as adding a new entry into the m_Languages table.

Change the m_Languages table in the "UI/Button.lua" file to

```
local m_Languages = {
  {Lang="EN", Text="TXT_KEY_TEST_BUTTON_BUTTON_ENGLISH",
Greeting="TXT_KEY_TEST_BUTTON_HELLO_LEADER"},
  {Lang="FR", Text="TXT_KEY_TEST_BUTTON_BUTTON_FRENCH",
Greeting="TXT_KEY_TEST_BUTTON_BONJOUR_LEADER"},
  {Lang="DE", Text="TXT_KEY_TEST_BUTTON_BUTTON_GERMAN",
Greeting="TXT_KEY_TEST_BUTTON_GUTENTAG_LEADER"},
  {Lang="YO", Text="TXT_KEY_TEST_BUTTON_BUTTON_YODA",
Greeting="TXT_KEY_TEST_BUTTON_YODA_LEADER"}
}
```

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

```
<Row Tag="TXT_KEY_TEST_BUTTON_BUTTON_YODA">
  <Text>Yoda</Text>
</Row>
<Row Tag="TXT_KEY_TEST_BUTTON_YODA_LEADER">
  <Text>{1_LeaderName} good day it is</Text>
</Row>
```

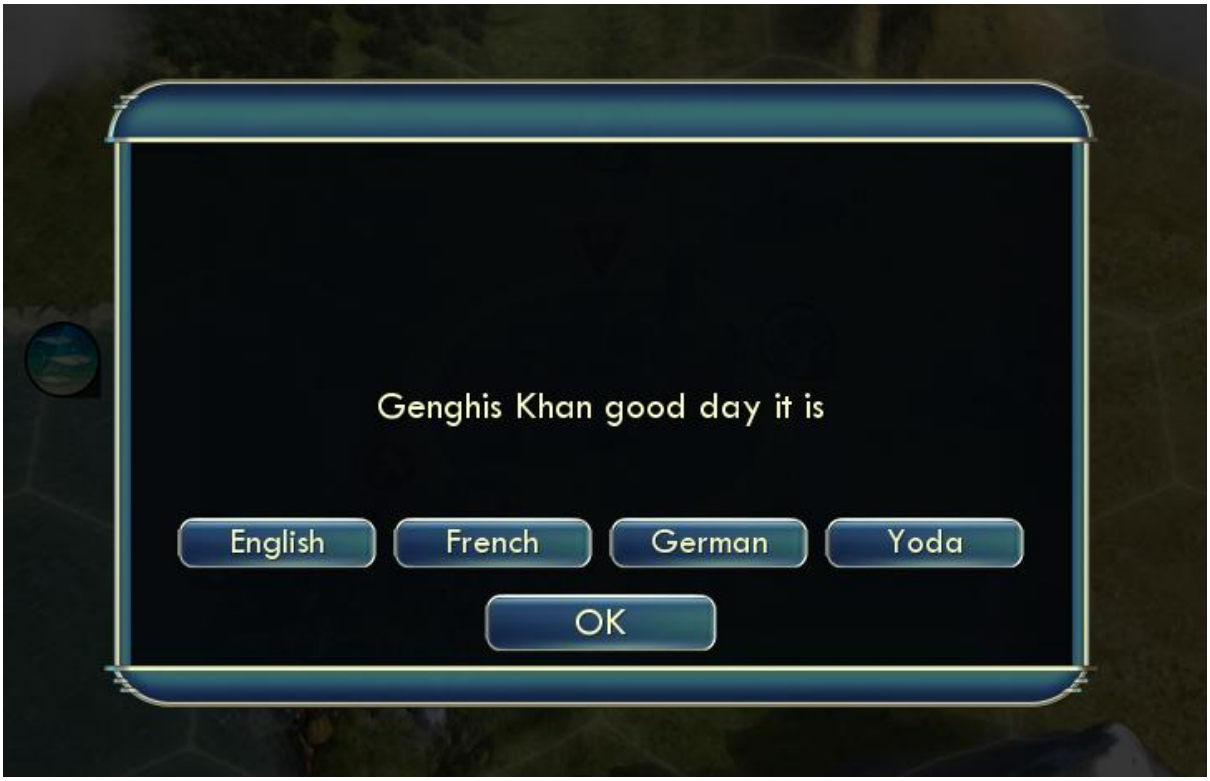
save the changes, rebuild the mod, restart Civ 5, re-enable the mod and start a new game. You'll see the language dialog with four buttons that don't quite fit. While we could make the dialog wider, we'll shrink the buttons instead. If this was Button 1 we'd need to edit four button elements, but as our buttons are created from an Instance, we just need to change the appearance in one place

Make the following changes to "UI/Button.xml"

```
<Instance Name="LangButton">
  <GridButton ID="Button" Size="120,32" Style="BaseButton">
```

```
    <Label ID="Label" Anchor="C,C" Offset="0,-2" Font="TwCenMT22"
FontStyle="Shadow" ColorSet="Beige_Black_Alpha" />
  </GridButton>
</Instance>
```

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



In general, whenever you find yourself copying and pasting chunks of XML and making only a few edits to text strings "Think Instance". It may be more work upfront to create the associated Lua, but it will save a lot of time, effort and annoying layout mistakes later.

Continued in Part 2