Jeff's Excruciatingly Simple Extension Tutorial for Fantasy Grounds (Part 2)

Jeffery W. Redding

7/19/2015

version 1.0

Table of Contents

Introduction

Fantasy Grounds is a very powerful and full featured platform for building and running desktop RPG games. It also provides a lot of room for developing custom rulesets, extensions and so forth, but at a high price of complexity and lack of really good introductory material.

Given that, I decided to jump in and write an extension.

Trying to build that extension took a lot of fits and starts, so I decided I would also create this tutorial in hopes that others would find this an easy way to get their feet wet in this realm. If you haven't read Part 1, you may want to go back and check it out. In that part, I set up a basic window and get it displayable in FG. It also has the code that we're going to build on (also in Appendix A).

In this part, I'm going to start adding some functionality to the window. In particular, we're going to make it into a simple extension for creating and saving common chat phrases, with their appropriate emotes. In other word, if you're fond of typing "/ooc scratches his head and wonders aloud when dinner is", then rather than take up a command slot for that, you'll be able to type it in once, select "ooc", and then save it. From that point forward, a single click will send your message to the chat console.

Getting Rolling

In part 1, we ended up with an "unbound" window. This means that it wasn't associated with any database, and we couldn't launch it without doing a chat window command (/openwindow). That's probably got to be the first thing we take care of, so we can build our data, while we build our window.

1 Let's Talk Data

Fantasy grounds is a little different than a lot of other applications. It doesn't us a RDBMS (Relational Database Management System) like a lot of apps. No Oracle, MySql, or even Access database here. Instead, everything is kept in an XML file – Just like our window definition.

When you create a campaign, one of the first things FG does is create a database for the campaign called db.xml. This is created in memory and Fantasy Grounds only writes it out to disk every 5 minutes or so, or when you leave the campaign. This is important to understand so when we look at the db.xml file, it might not match what we expect (unless we log out of the campaign).

Our db.xml file starts out nice and small and fairly easy to digest. Once you start building a campaign, adding characters, encounters, NPCs, items, etc., it gets really ugly really fast.

Luckily for us, since we haven't done any of that in our test campaign, we can look at the whole thing and its really pretty manageable. You should be able to use the same editor you were using before to do this. Go ahead. I'll wait. (If you don't see the file in the directory for this campaign, you may need to exit back to the launcher so that it will get written, remember?)

If you haven't played around in the test campaign, the db.xml file is barely a dozen lines. If you have done some playing, it may be several hundred! If you do this with a real campaign, it will be 10's of thousands of lines!

You'll notice, however, there isn't anything here that is associated with our little window. Again, that's because we were "unbound". Now, let's bind it.

2 One Func to Bind Them All

2.1 The lua bits

The best way to do this is to start in our ExtensionTutorial.lua file. We're going to add an onInit function, and have it register a new icon in the "stack". The "stack" is the set of small icons in the upper right part of the FG window. The "dock" is the big icons on the right.

Here's the function:

```
--------------------------------------------------------------------------
-- onInit - Initialization function for this package
-- --------------------------------------------------------------------
function onInit()
    if(User.isHost()) then
        DesktopManager.registerStackShortcut2("button_ExtTut",
                  "button_ExtTut_down", "sidebar_tooltip_ExtTut",
                  "TutorialWindow", "ExtTutorial");
    end
end
```

Since we did some lua functions in the last tutorial, you should find this somewhat familiar. The first couple of lines are lua comments, followed by the the function declaration. The last line is the function end.

We used an 'if' statement before, so that may make some sense, but this one's a little different. In our earlier use, we had something like "if(rset == "5E") then" and we knew we were looking to see if rset was equal to the string "5E". Here it's a little less clear. First of all, we're calling the function "isHost()", and it's in the package "User". That's not too hard, but what's it comparing it too? If we look at the FG reference pages, it shows that isHost() returns a boolean. In other words, isHost() returns true or false. In essence what we're doing with this if statement is comparing the results of isHost to true. We could have written "if(User.isHost() == true) then", but this is a common shorthand way of saying the same thing.

The next several lines are calling a function in the DesktopManager class that is part of CoreRPG. To use this function, we have to pass in several arguments:

- **button_ExtTut**: This is the icon we want to use in the stack.
- **button_ExtTut_down**: This is the icon we want to use when our button is "down"
- **sidebar_tooltip_ExtTut**: This is a string resource that we want for our tooltip.
- **TutorialWindow**: This is the name of our windowclass to open
- **ExtTutorial**: This is the name we want to give our database node.

2.2 The Extension Bits

As the experienced FG coders we now all are, we notice that only one of those four arguments is currently defined – our windowclass name! That means we're going to have to do some work to register the rest.

Let's do the first three in our extension.xml file. We already know how to register icons, so that's easy enough, and registering a string resource isn't any harder:

```
<base>
    <icon name="ExtTutorialLogo"     file="ExtTutorialLogo.png" />
    <icon name="TutorialTitle5E"     file="graphics/TutorialTitle5E.png" />
    <icon name="TutorialTitle"       file="graphics/TutorialTitle.png" />
    <icon name="button_ExtTut"      file="graphics/button_ExtTut.png" />
    <icon name="button_ExtTut_down" file="graphics/button_ExtTut_down.png" />
    <script name="ExtTutorial"       file="lua/ExtensionTutorial.lua" />
    <includefile source="xml/TutorialWindow.xml" />

    <string name="sidebar_tooltip_ExtTut">Extension Tutorial</string>
</base>
```

As before, I'm not doing a graphics tutorial, so you're on your own for creating the png files. As for the string resource, you can see it as the last line added.

The last argument, “ExtTutorial” isn't one we have to define. The registerStackShortcut2() function will bind our window such that as soon as we open it, we will get a new node created called “ExtTutorial”.

Fire up FG, start the console, and look for our button in the stack. If it's not there, check the console for errors. If it is, click it and you should see our tutorial window. After that, exit FG back to the launcher and then open the db.xml file. You should see our database node near the “root” tag.

Sweet!

2.3 String Theory

This is probably a good enough time to talk about string vs. string resource. From a high-level view, they are the same thing. Both represent some letters in consecutive order. The difference is in how FG handles them. When you use a string (which we've done quite a bit of) the memory allocation is done for that string ***every time you reference it***. This can take up a lot of memory in some cases. String resources, on the other hand, only have 1 copy of the string in memory, and everytime the resource gets used, it points back to that single copy.

Most of the time, that isn't a big deal. There are times, however, when FG makes you treat them very differently. We're not going to do much with that for this tutorial, but it's something to look for as your skills progress.

3 Planning Our Layout

We're to the point now where we've created our window, and we've bound it to a node in db.xml, and we've made it have a clickable button over in the stack. Now we have to do a bit of planning to figure out what is actually in our window.

First off, based on how other windows in FG work, we need a little button in the lower right corner that allows us to enter “edit” mode. When we click that button, we get an icon to the left of it that allows us to add a new record. That's about it for the basic control needs.

For the actual content, we want to have multiple entries (a list), with each having:

- A button that sends the message
- A button that lets you pick what type of message
- A string field that lets us put in the message
- A hidden delete button that gets turned on when we're in edit mode.

All of this needs to be laid out and anchored correctly for the whole thing to work. That can lead to a

rather messy, difficult to navigate window class for our main window. To simplify that, we're going to introduce templates.

3.1 Temple of the Templates

Templates are basically substitution blocks. We can create a template, and then reference it in our windowclass and override parts or add parts to flesh it out.

We've already seen some templates in use in Part 1: banner_campagin and close_campaignlist are a couple examples. We used those templates as tags in our <sheetdata> within our windowclass and they just worked with no special effort on our part. This is because they were templates in CoreRPG, which is what we started our extension on top of.

Now, however, we're going to make some of our own templates. We could do that right in our TutorialWindow.xml file, but that will make the file longer and maybe not as clean and well organized as we might like. Therefore, we're going to create new file called ExtTutorial_Templates.xml in our xml directory for the extension.

We also need to make sure our extension is including that file so we actually pick up those template extensions. Let's start by adding the include in our extension.xml file:

```
...
        <icon name="button_ExtTut_down" file="graphics/button_ExtTut_down.png" />
        <script name="ExtTutorial"       file="lua/ExtensionTutorial.lua" />

        <includefile source="xml/TutorialWindow.xml" />
        <includefile source="xml/ExtTutorial_Templates.xml" />

        <string name="sidebar_tooltip_chatnom">Chatnomicon</string>
...
```

Now we create our template file:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<root>
</root>
```

Nothing really interesting here. The same first line as our other XML file, and opening and closing tags for our root node. Don't forget the root node! FG expects it to be there, and all our templates have to be inside it or FG won't find them!

4 Get Some Control

4.1 Add Some Edit

As you dig around in FG, you'll find the button templates that are used for the “enter edit” functionality that you've seen in other windows and what we're about to add here. You can play around with making those work for you, but I decided to decompose those and make my own control here. It makes it a lot clearer for the purposes of a tutorial to build it from scratch.

Let's start with adding the template, and then walk through how it works. Next, we'll use the template in our TutorialWindow class and test it out. Here's what we're adding to our new template file:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<root>
    <!-- Template for the button that turns on the add and delete buttons -->
    <template name="button_edit_entries">
        <buttoncontrol>
            <anchored width="20" height="20">
                <position>insidebottomright</position>
                <top anchor="bottom" offset="-90"/>
                <right offset=”-30” />
            </anchored>
            <state icon="button_edit" tooltipres="button_editon" />
            <state icon="button_edit_down" tooltipres="button_editoff" />
        </buttoncontrol>
    </template>
</root>
```

Hopefully, you still remember what XML comments look like. That's the first line we've added. With only one entry in our template file, adding a comment isn't necessary, but will be nice to have once we've added a half dozen more templates.

The next line is the template tag. As you can see, the only thing we do here is set a name for the template. This name is what we'll use in our class file (i.e. <button_edit_entries> will be the tag we'll use later).

Next, you can see we're adding a new thing. A buttoncontrol. A buttoncontrol is a simple control that responds to button clicks. Like with all of these tags, it's a good idea to go out to the Fantasy Grounds reference docs and look at what a button control can have as tags and methods. In this case, all we're doing here is using the 'anchored' tag to set position, and the 'state' tag to define what the button looks like in different states.

Looking at our 'anchored' tag in more depth we see width and height for the icon, and then we see a position tag. For now, we've anchored the buttoncontrol to the inside bottom right corner of the window control, and we've told the top of the button control to anchor to the bottom of the window control. Because of that, we also had to add some offsets so that the button actually shows up inside the window. Long term, we'll anchor this to the bottom of the list we put in, but for now at least, this works.

The state tags are used by the button control to set the icon when the button is clicked. It will start with the first state, and then switch to the second when clicked. It will switch back to the first when the button is clicked again, and so on. I've used standard buttons and standard tooltip resources that I found while digging around in the core.

To actually get the template we created to display, we now add it to the TutorialWindow.xml file:

```
...
            </banner_campaign>
            <button_edit_entries name=”list_edit” />
            <resize_campaignlistwithtabs />
...
```

We'll have to do some work yet to get this to do much of anything, but at this point you can reload the ruleset and click the stack icon and see that the button is there. In fact, when you click the button you should see that the tooltip changes, and so does the icon.

4.2 Add Some Add

4.2.1 First, the Button Part

It's time now to add our add button. Like before, we're going to add a template that has a button

control. This time, however, the button will need to start out invisible since it will only appear when we click the edit icon, and it won't have any states since it will disappear when it gets clicked. First, here is the template we need to add to our template file:

```
...
    <!-- Template for the button that adds a chat message node -->
    <template name="button_add_entry">
        <buttoncontrol>
            <anchored width="20" height="20" to="list_edit" position="left">
                <offset>5,0</offset>
            </anchored>
            <icon normal="button_add" pressed="button_add_down" />
            <tooltip textres="button_add" />
        </buttoncontrol>
    </template>
...
```

Like our edit button template, this template starts with a comment, then a tag to start the template. Also like our edit button, this is a button control. The anchored tags is different, though. Since we want this button to appear to the left of the edit button, we anchor it to the edit button (which we called list_edit in the windowclass), positioned it to the left of list_edit, and then added an offset that gives us a small gap between the two.

We then add an icon and tooltip. Once again, I used icons and tooltip resources that were already defined in the CoreRPG.

And, of course, we have to add this template to the TutorialWindow class like this:

```
...
          </banner_campaign>
          <button_edit_entries name="list_edit" />
          <button_add_entry name="list_add" />
          <resize_campaignlistwithtabs />
...
```

Note: There are many different ways I could place these buttons. For example, I could have started with our window list and attached them to that, and then I could have the add/edit buttons float with the list, rather than stay anchored to the window. When you're creating your own app, you will have to fiddle around a bit with how things get anchored to get them to be the way you like. Luckily, it's not hard and you won't do any serious damage if you get it wrong. One thing you are likely to see is nothing – It is very easy to attach something off the window and it won't show at all!

If you reload at this point, you'll see the plus button next to the edit, which isn't really what we want, but it does make it easier to get our layout the way we want. Let's fix that now.

4.2.2 Disappearing Act

The first part of this is easy. We need to add an "invisible" tag to the add button:

```
...
    <!-- Template for the button that adds a chat message node -->
    <template name="button_add_entry">
        <buttoncontrol>
            <anchored width="20" height="20" to="list_edit" position="left">
                <offset>5,0</offset>
            </anchored>
            <icon normal="button_add" pressed="button_add_down" />
            <tooltip textres="button_add" />
            <invisible />
        </buttoncontrol>
    </template>
...
```

That turns the button invisible for us. So how do we turn it on? Remember we said we want to turn it on when we click the edit button. If we look at the reference document for buttoncontrol, we can see that there is an onValueChanged callback that ought to do the trick. Let's add this script to our list_edit control in our windowclass:

```
...
          </banner_campaign>
          <button_edit_entries name="list_edit">
               <script>
                   function onValueChanged()
                       window.list_add.setVisible(getValue() == 1);
                   end
               </script>
           </button_edit_entries>
          <resize_campaignlistwithtabs />
...
```

We added lua script in-line in Part 1, so that should be somewhat familiar to you. We also wrote some lua functions, so you should be feeling right at home, at this point. Let's digest the one (and only) line in this function. This is tricky if you're not used to programming. Let's start from the inside, and work our way out.

- **getValue()** - This function is getting the current value of the list_edit button. If you recall, we set this buttoncontrol up with two states. FG makes those states change the value of the button. In this case it flops back and forth between 0 and 1.
- **getValue() == 1** – In our if statements, we learned about the equality operator (==) and boolean values (true or false). This statement is taking the value returned by getValue and comparing it to 1. If the value is one, then this expression is true. Otherwise, it's false.
- **window.list_add.setVisible()** - This is calling the setVisible function on the list_add control that is within the window. The list_edit button doesn't know about the list_add button directly, but because they share a common parent, the list_edit button can reference the list_add through that parent.

This line could have been written as several lines, like this, as well:

```
local add_btn = window.list_add;
local val = getValue();
if(val == 1) then
    add_btn.setVisible(true);
else
    add_btn.setVisible(false);
end
```

Once you understand how the single line works, you can see it's a cleaner, more compact way to accomplish what we want, even if it's a little harder to read.

Now when you reload and open the window, you'll just see the edit. If you click on it, the add will show up. Click the edit again, and the add goes away. Clicking the add doesn't do anything yet, but we'll get there!

5 Deeper Into the Woods

5.1 Let's List, Lucky Lou!

The time has finally arrived to start working on that list we talked about earlier. As with the rest of this tutorial, I'm going to start simple and build it up.

Let's start with the windowlist that we're adding. As the FG reference docs indicate, a windowlist is a special control that contains windowlist instances in a list that we can reference. Typically, the windowlist is tied to a DB node, and populates the entries from the db entries. That's exactly what ours will do when we get to that point.

Here is what we need to add to our TutorialWindow window class:

```
        <!-- List of entries. -->
        <windowlist name="list">
            <datasource>.</datasource>
            <class>list_entry</class>
            <anchored>
                <left offset=”50” />
                <top offset=”30” />
                <right offset=”-30” />
                <bottom offset=”-65 />
            </anchored>
            <skipempty />
        </windowlist>
```

This windowlist is bound to the same database node as the very top of the extension. We set that to be “ExtTutorial”. Using the dot notation just means 'here'. We could have done something like ./list_entries which would have created a new tag set underneath ExtTutorial. We didn't do that because we don't need it for this tutorial.

There are a lot of different ways to anchor things, but in this case, we've anchored each of the sides to the sides of the parent (TutorialWindow), and we've set an offset that makes sure we're inside and out of the way of some of the other stuff on this window. This makes our list size along with the parent, and keep things tidy. If we didn't have one of these anchors (say the right anchor), then we may not be able to see various controls that we add in the list – they would be far off to the right of our little window.

Just like with the title bar, these offsets don't work with 5E. Instead, we'd have to write a method that would get called to set the anchors/offsets as needed for 5E. Since we already did that with the titlebar, we'll leave that to either a (much) later tutorial, or just an exercise for the student :-)

5.2 Getting Classy

Since a windowlist is a list of windowinstances, we need to tell it the windowclass that will be used to create those instances. Here, you can see we chose “list_entry”. We'll create that in just a bit.

We anchored this list to the inside-to-left of our window. That's all we have to do. (This doesn't quite work with 5E because of the title bar changes, but we're not going to worry about that for this tutorial.

Next, we're going to create our new window class:

```
<windowclass name="list_entry">
    <margins control="0,0,0,2" />
    <sheetdata>
        <button_chataction name="chat_btn" />
        <button_chatcycler name="chat_cycler" />
        <string_chatstring name="chat_string" />
        <button_delete_entry name="delete_entry" />
    </sheetdata>
</windowclass>
```

Not much here. We've set some margins in the window class that will make sure the multiple instances will have some space between them. We've also gone ahead and added some entries in the sheetdata section that we're going to make templates for. As you can see, these are the four controls we talked about above.

5.3 Fill in Some Templates

Here is the initial template for button_chataction:

```
<!-- Template for the button that sends the chat message -->
<template name="button_chataction">
    <buttoncontrol>
        <anchored width="20" height="20">
            <position>insidetopleft</position>
        </anchored>
        <state icon="charlist_typing" tooltip="Send the message" />
        <tooltip text="Send the chat message" />
    </buttoncontrol>
</template>
```

This is another button control, and it's anchored to the inside top left of the window list, which is it's parent. I used a standard icon, and my own tooltip. Everything else should look pretty familiar.

For the button_chatcycler, we're going to use a genericcontrol. Later, we'll write a bunch of custom code to make this 'cycle' through different values everytime it's pressed. For now, of course, it won't do much of anything:

```
<!-- Template for the button that selects the chat message type -->
<template name="button_chatcycler">
    <genericcontrol>
        <anchored to="chat_btn" position="right" width="50">
            <offset>10,0</offset>
        </anchored>
        <frame name="fielddark" offset="7,5,7,5" />
        <stateframe>
            <hover name="fieldfocus" offset="7,5,7,5" />
        </stateframe>
    </genericcontrol>
</template>
```

You'll notice that this control is anchored to the previous control and positioned to the right.

The text field is the easy one:

```
<!-- Template for the string that holds the chat message -->
<template name="string_chatstring">
    <stringfield>
        <anchored to="chat_cycler" position="right">
            <offset>10,0</offset>
        </anchored>
        <empty>...</empty>
    </stringfield>
</template>
```

This is anchored to the right of the chatcycler. The “empty” tag allows us to show something in the field that won't be put in our DB but provides a good visual indicator for the user that something goes here. We could have put “type your chat message here”, or something similar.

Finally, the delete button template:

```
<!-- Template for the button that deletes a chat message node -->
<template name="button_delete_entry">
    <buttoncontrol>
        <anchored width="20" height="20">
            <top />
            <right offset="-5" />
        </anchored>
        <state icon="button_delete" tooltipres="button_delete" />
        <state icon="button_deleteconfirm" tooltipres="button_deleteconfirm" />
        <tooltip textres="button_delete" />
        <invisible />
    </buttoncontrol>
</template>
```

This button is anchored to the far right side of the list_entry windowclass. This keeps it way off to the right, away from everything else. It also starts out as invisible, and we'll have to write some code later to make it visible when we need it.

If we run this, it doesn't show anything new. That's because our DB is empty, and our add button doesn't add anything. Time to work that out.

5.4 Time to Get Dirty with the DB

We already used the onValueChanged() method on the list_edit button to make it turn the add button on and off. Now, we're going to se the onButtonPress() method to make the add button add a new node. In addition, we're going to set the list_edit button back to zero, which will turn the add button off.

We're going to add some script in-line in our list_add button, and it goes something like this:

```
...
            <button_add_entry name="list_add">
                <script>
                    function onButtonPress()
                        DB.findNode("ExtTutorial").createChild();
                        window.list_edit.setValue(0);
                    end;
                </script>
                <target>list</target>
            </button_add_entry>
...
```

Looking into this you can see that when we press the add button, it will find the node called "ExtTutorial" and create a child node underneath. We then call the list_edit setValue method with a value of zero to turn the add button off.

GOTCHA: This code only works for the GM! FG doesn't allow players to create new nodes in the DB. Instead, the client code has to request to create a node from the GM instance. We're not going to cover that here. Maybe in a later tutorial (if I get enough motivation or feedback of interest).

At this point, we can reload our window and hit add a couple times. You can see the entries show up (though there ain't much to see), and you can even type in some text in the text field.

You'll notice our delete buttons don't show up yet since we haven't told them to. Also, our chat icon doesn't do anything because we haven't added an event handler. Also, our chat cycler doesn't do anything or show anything since we haven't gotten that far.

HOWEVER, if you return to the launcher, and then pull up the db.xml file, you should see something like this:

```
...
        <ExtTutorial>
                <id-00001>
                        <chat_string type="string">Type some data</chat_string>
                </id-00001>
                <id-00002>
                        <chat_string type="string">more typing here</chat_string>
                </id-00002>
                <id-00003>
                        <chat_string type="string">and still more</chat_string>
                </id-00003>
        </ExtTutorial>
...
```

We've added entries to our DB, and saved the strings, and made the windowlist auto-magically populate the whole kit-and-caboodle!

5.5 Cleaning Up After Ourselves

5.5.1 First, Let's Lua

Let's take care of that delete problem now.

We know we want the delete buttons to show up when we click the list_edit button, but how do we do it? From the scope of list_edit, we can't actually see the window instances easily, so we can't really reach into them and make that happen. We can, however, ask the list itself (which is a sibling to us, ie. Same parent) to help us out, and that's what we're going to do.

First, let's create a new file called "ExtTutorial_list.lua" in our lua directory. This will contain functions specific to our windowlist. Then, we're going to add an update() method that we will use to turn on and off the delete buttons:

```
function update()
   local bEditMode = window["list_edit"].getValue() == 1;

    for _,w in pairs(getWindows()) do
        w.delete_entry.setVisibile(bEditMode);
    end
end
```

If you recall, lua files don't have to have the opening line that XML files do, nor does the content have to appear within the "root" tag the way FG requires for XML. Instead, we can just start writing our functions (although you should add comments too!)

This is a bit more complicated than the lua script we've done so far, so we'll have to take a bit of time to dissect it. We'll skip the function line and the end statements because you already know what those do.

- **local bEditMode** – This line is using an index in window to find the list_edit button, and then getting the value of that button and setting bEditMode to the boolean true or false. This is similar to what we did for the list_add button.
- **for _,w in pairs** – This is a for loop. This is a special code construct that "loops" through a set of data, running the code that's inside the loop once for every entry in the set. In this case, the set is the list of windows retrieved by the getWindows() method. The "_" is a lua shorthand that says to ignore the index, while the w will be one of the window instances in the list.
- **w.delete_entry** - This line is simply setting the delete_entry control within the window instance on or off based on the value we assigned to bEditMode previously.

That's a lot of new stuff at once, but hopefully at this point it doesn't feel overwhelming.

Now we can add a couple things in our TutorialWindow.xml file. First, in list_edit, we'll add the script file to the windowlist, and then we'll add a call to that function:

```
...
            <!-- List of entries. -->
            <windowlist name="list">
                <script file="lua/ExtTutorial_list.lua" />
                <datasource>.</datasource>
...
            <button_edit_entries name="list_edit">
                <script>
                    function onValueChanged()
                        window.list_add.setVisible(getValue() == 1);
                        window.list.update();
                    end;
                </script>
            </button_edit_entries>
...
```

If we've done our jobs right, we can now reload our ruleset and see that when we click the edit button, the delete buttons show up. When we click it again, they turn off as well. However, when we click the delete buttons themselves, nothing happens since we have yet to tell the button to do anything. That's what we'll tackle next.

5.5.2 Delete is Easy

As we did with our add button, we're just going to add a simple onButtonPress method and have this method delete this list entry. This is actually really easy:

```
...
            <button_delete_entry name="delete_entry">
                <script>
                    function onButtonPress()
                        window.getDatabaseNode().delete();
                    end
                </script>
            </button_delete_entry>
...
```

Here, we're just getting the database node associated with this list entry, and calling it's delete method. Voila! That's all it takes to delete an entry!

Wrap Part Deaux

At this point, if you reload the ruleset and play around, you see we're to the point where it's all pretty functional. We've accomplished quite a bit to this point, too. In fact, in this tutorial, we've managed to:

- Learned about the FG database, and even managed to add and delete records
- Learned how to add a repeating list of windowinstances mapped to our data
- Learned how to add buttons and make them do stuff
- Made some templates to help keep our code clean.

That's not a bad list! Looking at the overall goal, there are only a couple things missing at this point:

1. Our chataction button doesn't have an onButtonPress event, and we haven't written anything to actually send the chat message.
2. Our chatcycler control doesn't display anything or do anything

Number 1 is actually a lot easier than number two. So, when we get Part 3 done, that's where we're going to start off. Until next time, thanks for reading, and please give me some feedback on whether this is helpful or what.

Final Note: The Final code for this part of the tutorial is in Appendix B.

Appendix A: Our Starting Point

If you followed along in part one, you already have this. If not, then here is what we started with for Part 2:

Text 1: extension.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
    © Copyright Jeffery W Redding 2015+ except where explicitly stated otherwise.
    Fantasy Grounds is Copyright © 2004-2013 SmiteWorks USA LLC.

    Any use of material copyrighted by others is unintentional.  Let me know, and I'll make sure credit is
    given appropriately.

    This material is provided freely, without warranty of any kind.  This material is owned by me, so any use,
    modification, or publication, whether electronic, printed, verbal or anything else must provide appropriate
    attribution.
-->

<root release="3.0" version="3">
    <properties>
        <name>Extension Tutorial - Jeffery W. Redding</name>
        <version>0.0.1</version>
        <author>Jeffery W. Redding</author>
        <description>
            A really simple  extension file for
            creating a really simple extension
        </description>
    </properties>
    <base>
        <icon name="ExtTutorialLogo" file="ExtTutorialLogo.png" />
        <icon file="graphics/TutorialTitle5E.png" name="TutorialTitle5E" />
        <icon file="graphics/TutorialTitle.png" name="TutorialTitle" />
        <script name="ExtTutorial" file="lua/ExtensionTutorial.lua" />
        <includefile source="xml/TutorialWindow.xml" />
    </base>
</root>
```

Text 2: ExtensionTutorial.lua

```
-- ---------------------------------------------------------------------------------------------------------------------
-- This is the main file for the Lua Scripts for ExtensionTutorial.
-- Copyright Jeffery W. Redding 2015+ except where explicitly stated otherwise.
-- Fantasy Grounds is Copyright © 2004-2015 SmiteWorks USA LLC.

-- Any use of material copyrighted by others is unintentional.  Let me know, and I'll make
-- sure credit is given appropriately.

-- This material is provided freely, without warranty of any kind.  This material is owned by me, so any use,
-- modification, or publication, whether electronic, printed, verbal or anything else must provide appropriate
-- attribution.
-- ---------------------------------------------------------------------------------------------------------------------

-- ---------------------------------------------------------------------------------------------------------------------
-- setIconByRuleset - This function takes a windowcontrol for an argument and, if that windowcontrol has an icon
--                    it sets that icon to be one of two icons, based on the ruleset.
-- ---------------------------------------------------------------------------------------------------------------------
function setIconByRuleset(wc)
    if(wc.hasIcon() == false) then
        Debug.console("setIcon - no icon", wc)
        return
    end

    local rset = User.getRulesetName()
    Debug.console("setIconbyRuleset rulsetName", rset)
    if(rset == "5E") then
        wc.setIcon("TutorialTitle5E", true)
    else
        wc.setIcon("TutorialTitle", true)
    end
end
```

Text 3: TutorialWindow.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
    © Copyright Jeffery W Redding 2015+ except where explicitly stated otherwise.
    Fantasy Grounds is Copyright © 2004-2013 SmiteWorks USA LLC.

    Any use of material copyrighted by others is unintentional.  Let me know, and I'll make sure credit is
    given appropriately.

    This material is provided freely, without warranty of any kind.  This material is owned by me, so any use,
    modification, or publication, whether electronic, printed, verbal or anything else must provide appropriate
    attribution.
-->

<root>
    <!-- This is the main window for the tutorial.  I'm using the framedef from RPGCore -->
    <windowclass name="TutorialWindow">
        <frame>campaignlistwithtabs</frame>
        <script>
            function onInit()
                --[[ Self points to a windowinstance, in this case. ]]
                Debug.console("self", self)
                Debug.console("windowclass", self.getClass())
                Debug.console("node", self.getDatabaseNode())
                Debug.console("frame", self.getFrame())
            end
        </script>
        <sizelimits>
            <dynamic />
        </sizelimits>
        <sheetdata>
            <banner_campaign>
                <script>
                    function onInit()
                        --[[ Self points to a windowcontrol, in this case. ]]
                        Debug.console("self", self);
                        ExtTutorial.setIconByRuleset(self);
                    end;
                </script>
                <icon>TutorialTitle</icon>
            </banner_campaign>
            <resize_campaignlistwithtabs />
            <close_campaignlist />
        </sheetdata>
    </windowclass>
</root>
```

Appendix B: Our Ending Point

Text 1: extension.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
    © Copyright Jeffery W Redding 2015+ except where explicitly stated otherwise.
    Fantasy Grounds is Copyright © 2004-2013 SmiteWorks USA LLC.

    Any use of material copyrighted by others is unintentional.  Let me know, and I'll make sure credit is
    given appropriately.

    This material is provided freely, without warranty of any kind.  This material is owned by me, so any use,
    modification, or publication, whether electronic, printed, verbal or anything else must provide appropriate
    attribution.
-->

<root release="3.0" version="3">
    <properties>
        <name>Extension Tutorial - Jeffery W. Redding</name>
        <version>0.0.2</version>
        <author>Jeffery W. Redding</author>
        <description>
            A really simple  extension file for creating a really simple extension
        </description>
    </properties>
    <base>
        <icon name="ExtTutorialLogo"     file="ExtTutorialLogo.png" />
        <icon name="TutorialTitle5E"     file="graphics/TutorialTitle5E.png" />
        <icon name="TutorialTitle"       file="graphics/TutorialTitle.png" />
        <icon name="button_ExtTut"      file="graphics/button_ExtTut.png" />
        <icon name="button_ExtTut_down" file="graphics/button_ExtTut_down.png" />
        <script name="ExtTutorial"       file="lua/ExtensionTutorial.lua" />

        <includefile source="xml/TutorialWindow.xml" />
        <includefile source="xml/ExtTutorial_Templates.xml" />

        <string name="sidebar_tooltip_chatnom">Chatnomicon</string>
    </base>
</root>
```

Text 2: ExtensionTutorial.lua

```
-- ---------------------------------------------------------------------------------------------------------------------
-- This is the main file for the Lua Scripts for ExtensionTutorial.
-- Copyright Jeffery W. Redding 2015+ except where explicitly stated otherwise.
-- Fantasy Grounds is Copyright © 2004-2015 SmiteWorks USA LLC.

-- Any use of material copyrighted by others is unintentional.  Let me know, and I'll make
-- sure credit is given appropriately.

--    This material is provided freely, without warranty of any kind.  This material is owned by me, so any use,
--    modification, or publication, whether electronic, printed, verbal or anything else must provide appropriate
--    attribution.
-- ---------------------------------------------------------------------------------------------------------------------

-- ---------------------------------------------------------------------------------------------------------------------
-- onInit - Initialization function for this package
-- ---------------------------------------------------------------------------------------------------------------------
function onInit()
    if(User.isHost()) then
        DesktopManager.registerStackShortcut2("button_ExtTut", "button_ExtTut_down",
            "sidebar_tooltip_ExtTut", "TutorialWindow", "ExtTutorial");
    end
end

-- ---------------------------------------------------------------------------------------------------------------------
-- setIconByRuleset - This function takes a windowcontrol for an argument and, if that windowcontrol has an icon
--                    it sets that icon to be one of two icons, based on the ruleset.
-- ---------------------------------------------------------------------------------------------------------------------
function setIconByRuleset(wc)
    if(wc.hasIcon() == false) then
        return
    end

    local rset = User.getRulesetName()
    if(rset == "5E") then
        wc.setIcon("TutorialTitle5E", true)
    else
        wc.setIcon("TutorialTitle", true)
    end
end
```

Text 3: ExtensionTutorial_list.lua

```
-- ---------------------------------------------------------------------------------------------------------------------
-- This is a local lua script file for the chat list in TutorialWindow.
-- Copyright Jeffery W. Redding 2015+ except where explicitly stated otherwise.
-- Fantasy Grounds is Copyright © 2004-2015 SmiteWorks USA LLC.

-- Any use of material copyrighted by others is unintentional.  Let me know, and I'll make
-- sure credit is given appropriately.

-- This material is provided freely, without warranty of any kind.  This material is owned by me, so any use,
-- modification, or publication, whether electronic, printed, verbal or anything else must provide appropriate
-- attribution.
-- ---------------------------------------------------------------------------------------------------------------------
-- ---------------------------------------------------------------------------------------------------------------------
-- This is a local script package.  This is intended to be included within the "list" window control and will
-- most likely not work anywhere else.
-- ---------------------------------------------------------------------------------------------------------------------

-- ---------------------------------------------------------------------------------------------------------------------
-- update - When called, this checks the state of the list_edit button, and if it's set turns the edit button
--          for a list entry on.  The list_edit button should REALLY be called something like list_edit_enable, to
--          more accurately reflect what it really does.  I left it like this because it's consistent with the core.
-- ---------------------------------------------------------------------------------------------------------------------
function update()
   local bEditMode = window["list_edit"].getValue() == 1;
    for _,w in pairs(getWindows()) do
        w.delete_entry.setVisible(bEditMode);
    end
end
```

Text 4: ExtTutorial_Templates.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
    
-->
<root>
    <!-- Template for the button that turns on the add and delete buttons -->
    <template name="button_edit_entries">
        <buttoncontrol>
            <anchored width="20">
                <position>insidebottomright</position>
                <top anchor="bottom" offset="-90"/>
                <right offset="-30" />
            </anchored>
            <state icon="button_edit" tooltipres="button_editon" />
            <state icon="button_edit_down" tooltipres="button_editoff" />
        </buttoncontrol>
    </template>
    <!-- Template for the button that adds a chat message node -->
    <template name="button_add_entry">
        <buttoncontrol>
            <anchored width="20" to="list_edit" position="left">
                <offset>5,0</offset>
            </anchored>
            <icon normal="button_add" pressed="button_add_down" />
            <tooltip textres="button_add" />
            <invisible />
        </buttoncontrol>
    </template>
    <!-- Template for the button that sends the chat message -->
    <template name="button_chataction">
        <buttoncontrol>
            <anchored width="20" height="20">
                <position>insidetopleft</position>
            </anchored>
            <state icon="charlist_typing" tooltip="Send the message" />
            <tooltip text="Send the chat message" />
        </buttoncontrol>
    </template>
    <!-- Template for the button that selects the chat message type -->
    <template name="button_chatcycler">
        <genericcontrol>
            <anchored to="chat_btn" position="right" width="50">
                <offset>10,0</offset>
            </anchored>
            <!-- frame name="fielddark" offset="7,5,7,5" />
            <stateframe>
                <hover name="fieldfocus" offset="7,5,7,5" />
            </stateframe -->
        </genericcontrol>
    </template>
    <!-- Template for the string that holds the chat message -->
    <template name="string_chatstring">
        <stringfield>
            <anchored to="chat_cycler" position="right">
                <offset>10,0</offset>
            </anchored>
            <empty>...</empty>
        </stringfield>
    </template>
    <!-- Template for the button that deletes a chat message node -->
    <template name="button_delete_entry">
        <buttoncontrol>
            <anchored width="20" height="20">
                <top />
                <right offset="-5"/>
            </anchored>
            <state icon="button_delete" tooltipres="button_delete" />
            <state icon="button_deleteconfirm" tooltipres="button_deleteconfirm" />
            <tooltip textres="button_delete" />
            <invisible />
        </buttoncontrol>
    </template>
</root>
```

Text 5: TutorialWindow.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
    © Copyright Jeffery W Redding 2015+ except where explicitly stated otherwise.
    Fantasy Grounds is Copyright © 2004-2013 SmiteWorks USA LLC.

    Any use of material copyrighted by others is unintentional.  Let me know, and I'll make sure credit is
    given appropriately.

    This material is provided freely, without warranty of any kind.  This material is owned by me, so any use,
    modification, or publication, whether electronic, printed, verbal or anything else must provide appropriate
    attribution.
-->
<root>
    <!-- This is the main window for the tutorial.  I'm using the framedef from RPGCore -->
    <windowclass name="TutorialWindow">
        <frame>campaignlistwithtabs</frame>
        <sizelimits>
            <dynamic />
        </sizelimits>
        <sheetdata>
            <banner_campaign>
                <script>
                    function onInit()
                        ExtTutorial.setIconByRuleset(self);
                    end;
                </script>
                <icon>TutorialTitle</icon>
            </banner_campaign>
            <!-- List of entries. -->
            <windowlist name="list">
                <script file="lua/ExtTutorial_list.lua" />
                <datasource>.</datasource>
                <class>list_entry</class>
                <anchored>
                    <top offset="30" />
                    <left offset="50" />
                    <right offset="-30" />
                    <bottom offset="-65" />
                </anchored>
                <skipempty />
            </windowlist>
            <button_edit_entries name="list_edit">
                <script>
                    function onValueChanged()
                        window.list_add.setVisible(getValue() == 1);
                        window.list.update();
                    end;
                </script>
            </button_edit_entries>
            <button_add_entry name="list_add">
                <script>
                    function onButtonPress()
                        DB.findNode("ExtTutorial").createChild();
                        window.list_edit.setValue(0);
                    end;
                    </script>
                    <target>list</target>
            </button_add_entry>
            <resize_campaignlistwithtabs />
            <close_campaignlist />
        </sheetdata>
    </windowclass>
    <!-- This is the windowclass that is created within the windowlist for every node in the DB -->
    <windowclass name="list_entry">
        <margins control="0,0,0,2" />
        <sheetdata>
            <button_chataction name="chat_btn" />
            <button_chatcycler name="chat_cycler" />
            <string_chatstring name="chat_string" />
            <button_delete_entry name="delete_entry">
                <script>
                    function onButtonPress()
                        window.getDatabaseNode().delete();
                    end
                </script>
            </button_delete_entry>
        </sheetdata>
    </windowclass>
</root>
```