How to read in data from an XML file

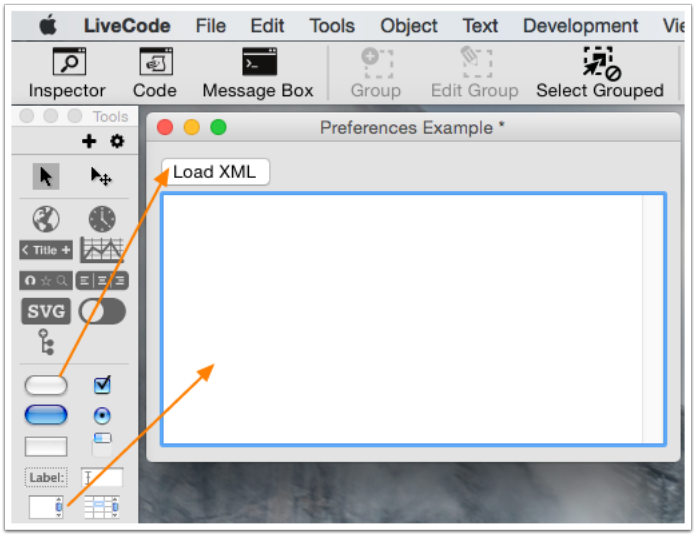
loadPreferences.zip

You can download the sample stack from this url: https://tinyurl.com/y7z4aa7v (https://tinyurl.com/y7z4aa7v)

This lesson will show you how to load an XML file and access the data for use in your application. XML files are a very useful for things like storing preference settings, working with the web and for situations where you need to share data with other programs.

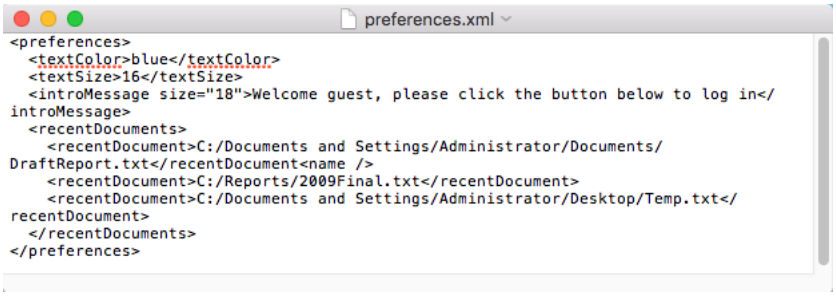
LiveCode provides a well-featured library for dealing with XML files, which can take a little getting used to but is quite straight forward to use.

Create a stack with a button and a field



We start by creating a stack and dragging a button and a field onto it. The button will contain our LiveCode code for reading the XML file. The field will contain the resulting data that is read from the file, set the **name** of the field to "information". Save this stack on your desktop.

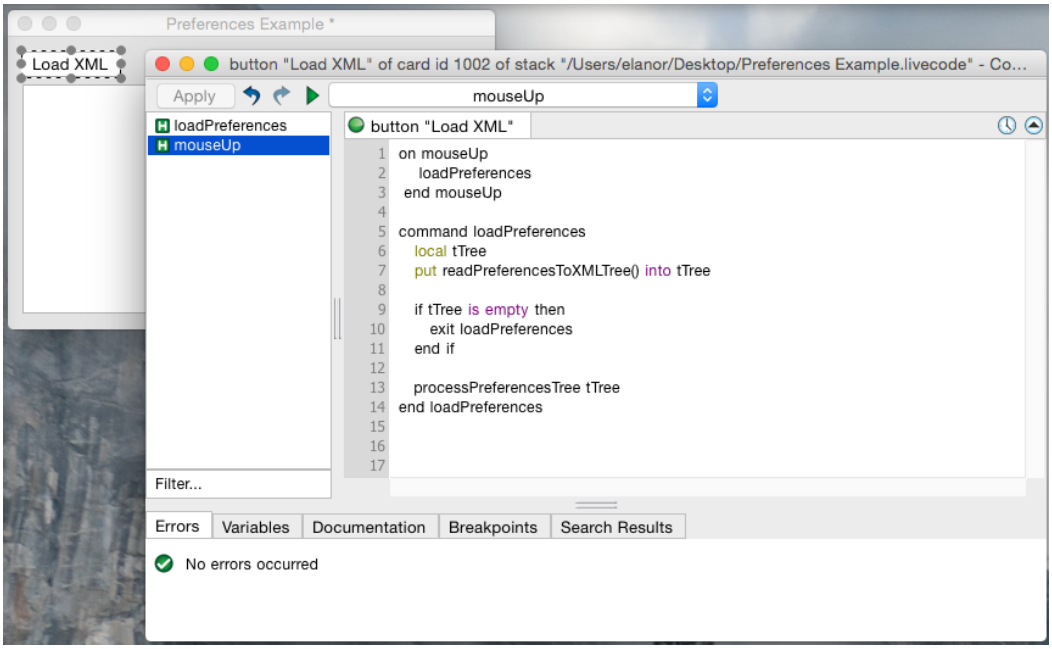
Create an XML file to use as an example



Use Notepad on Windows or TextEdit on Mac OS X to create the example preferences XML file. The file should be plain text. Save the file on your desktop as "Preferences.xml". The XML data looks like this:

```
<preferences>
  <textColor>blue</textColor>
  <textSize>16</textSize>
  <introMessage size="18">Welcome guest, please click the button below to log in</introMessage>
  <recentDocuments>
    <recentDocument>C:/Documents and Settings/Administrator/Documents/DraftReport.txt</recentDocument>
    <recentDocument>C:/Reports/2009Final.txt</recentDocument>
    <recentDocument>C:/Documents and Settings/Administrator/Desktop/Temp.txt</recentDocument>
  </recentDocuments>
</preferences>
```

Tell the button to load the file when the user clicks



(7011/show_image?image_id=495155)

Edit the script of the button by selecting it, then clicking on the "Script" button in the main menu bar. For this example we are going to load an XML file that contains preferences for our application.

Begin the script with the following code:

1. When the button is clicked, load up the preferences and put them into the field

```
on mouseUp
   loadPreferences
end mouseUp
```

2. Load the preferences file

There are two parts to loading the preferences file. The first part is reading the file into memory and creating an XML "tree". The second part is to process the tree and extract the data from it.

This function reads the XML file, and returns the tree. The tree is represented as a number, the actual tree structure and data is managed by LiveCode and so we don't need to worry about it.

```
command loadPreferences
   local tTree
   put readPreferencesToXMLTree() into tTree
   if tTree is empty then
      exit loadPreferences
   end if

   # Read the preferences we require from the tree and display them.
   processPreferencesTree tTree

   # Close the XML tree.
   # This will free up the memory that the tree was using and prevent our application
   # using more memory than it needs or "leaking" memory by creating multiple trees
   # without closing any of them.
   revXMLDeleteTree tTree
end loadPreferences
```

Note that this code doesn't do anything just yet, because we haven't yet implemented the function `readPreferencesToXMLTree` and the command `processPreferencesTree`.

Read the XML file from disk

Next, we implement a function to read the XML. This is done in two steps, first the file is read into a variable like any other text file would be, secondly, an XML "tree" is created from the file. This tree allows us to manipulate the XML data easily.

Add the code to read the XML file as below.

```
private function readPreferencesToXMLTree
   # Find the XML file on disk.
   # This is for now assumed to be in the same location as the stack / application.
   # Note that we restore the itemDelimiter to comma (its default value) afterwards.
   # This is not essential but its good practice to avoid tricky bugs
   # that can arise due to unexpected delimiter values.
   set the itemDelimiter to slash
   local tPreferencesFile
   put item 1 to -2 of the effective filename of this stack & "/Preferences.xml" into tPreferencesFile
   set the itemDelimiter to comma

   # Read the preferences data from the file into a variable.
   # Always check for the result when reading files
   # as its possible that the file may have been deleted or moved.
   local tPreferencesData, tResult
   put url ("file:" & tPreferencesFile) into tPreferencesData
   put the result into tResult
   if tResult is not empty then
      answer error "Failed to read preferences file at location: " & tPreferencesFile
      return empty
   end if

   # Create the XML "tree" from the data,
   # checking to make sure that the file has loaded properly.
   # The revCreateXMLTree function will return a number
   # (the tree's "handle" or "id") if it succeeds,
   # otherwise it will return a message saying why it failed.
   local tTree
   put revXMLCreateTree(tPreferencesData, false, true, false) into tTree
   if tTree is not an integer then
      answer error "Failed to process preferences file with error: " & tTree
      return empty
   end if

   return tTree
end readPreferencesToXMLTree
```

Extract the information out of the XML file and display it in a field

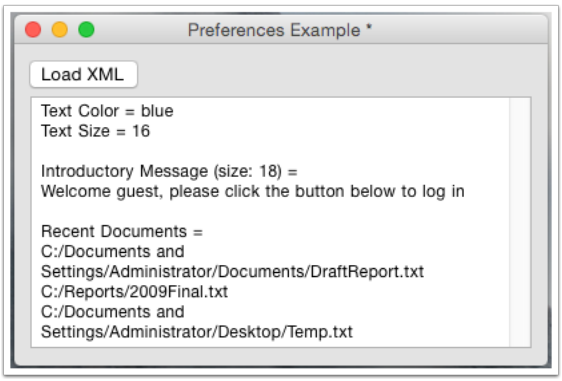
Once we have the XML tree, the final step is to use LiveCode's XML library to get the required information out of it. We use a series of calls to the XML library to extract each piece of information from the tree.

```
private command processPreferencesTree pTree
   # Extract the text color and text size preferences.
   # These are simple nodes in the XML file,
   # we can get what is inside them using the revXMLNodeContents function
   # This function will return a string beginning with "xmlerr,"
   # if it fails, but we don't check this
   # here as we created the file and we know it won't fail.
   local tTextColor
   put revXMLNodeContents(pTree, "preferences/textColor") into tTextColor

   local tTextSize
   put revXMLNodeContents(pTree, "preferences/textSize") into tTextSize

   # Extract the introductory message preference.
   # This node has an attribute. We extract the contents and the
   # attribute in two separate calls.
   # The function revXMLAttribute allows us to read attributes from XML files,
   # its exactly the same as revXMLNodeContents,
```

```
    # except that you also need to tell it which attribute you want.
    local tIntroMessage
    put revXMLNodeContents(pTree, "preferences/introMessage") into tIntroMessage

    local tIntroMessageSize
    put revXMLAttribute(pTree, "preferences/introMessage", "size") into tIntroMessageSize

    # Extract the recent documents list.
    # This is a nested list of nodes, which could have any number of items.
    # First, we get a list of the recent documents, then we can loop
    # through them and get each one in turn.
    # The revXMLChildNames function is useful for returning a list of nodes like this.
    # The last parameter is important as it tells the function to return a unique
    # specifier for each node, allowing us to access them correctly. This will
    # look something like:
    #    recentDocument[1]
    #    recentDocument[2]
    #    recentDocument[3]
    local tRecentDocuments
    put revXMLChildNames(pTree, "preferences/recentDocuments", return, "recentDocument", true) into tRecentDocuments

    # To get each document, we just use revXMLNodeContents again.
    # However here we concatentate the name of each node
    # with the path that all recent document nodes have in common,
    # to get the complete path.
    local tListOfRecentDocuments
    repeat for each line tRecentDocument in tRecentDocuments
       put revXMLNodeContents(pTree, "preferences/recentDocuments/" & tRecentDocument) & return after
tListOfRecentDocuments
    end repeat
    delete the last char of tListOfRecentDocuments

    # Now we output what we read from the file to see if it worked.
    local tOutput
    put "Text Color = " & tTextColor & return after tOutput
    put "Text Size = " & tTextSize & return after tOutput
    put return after tOutput
    put "Introductory Message (size: " & tIntroMessageSize & ") = " & return after tOutput
    put tIntroMessage & return & return after tOutput
    put "Recent Documents = " & return after tOutput
    put tListOfRecentDocuments after tOutput
    set the text of field "Information" to tOutput
end processPreferencesTree
```

Testing

To test the stack switch to **Run** mode and click the button.



9 Comments

Jay Anthony Nunez Thursday Nov 10 2011 at 03:05 AM

Well done demonstration. :-)

Glenn Wednesday Aug 01 2012 at 06:57 PM

Well I did it, but there was no change to the font size nor color.

Hanson Schmidt-Cornelius Friday Aug 03 2012 at 11:16 AM

Hi Glenn,

this lesson describes how to load the content of an XML file and access the data. The data loaded is displayed in the "Information" field but is not applied in this lesson.

If you wish to apply the values loaded, then you would have to implement this as a next step.

Kind Regards,

Hanson

gary Friday Aug 02 2013 at 10:28 PM

What about actual XML files that commence with eg

How do you recommend going about deconstructing these?

Thanks in advance.

Hanson Schmidt-Cornelius Tuesday Aug 06 2013 at 11:57 AM

Hi Gary,

if there is content in the XML file that you think should not be in there when you process it, then you can remove that data first. Use the search and replace functionality within LiveCode to remove any characters or text that you think should not be present. You can do this because XML files are text files.
You can then process the XML data as you require.

Kind Regards,

Hanson

---

gary Tuesday Aug 06 2013 at 01:47 PM

Hi Hanson,

Thanks for your reply. That is what I have been doing, but wasn't sure if it was the correct way to go about it. I don't know why the code I copied over does not appear above but it was tags about xml and soap, and it is these that I have removed and replaced (at both ends of the document) with my own root node.

Gary

Gilar Wednesday Feb 15 2017 at 07:09 AM

i have succes to create an XML file, but it's an ANSI when i check in notepad
Now I need to create a unicode XML file
Could you please to explain how to do this

Thanks

Daniel Sunday Apr 23 2017 at 11:14 AM

In LC 7 and following, the name of the function is not revCreateXMLTree but revXMLCreateTree; the parameters remain the same

---

Elanor Buchanan Wednesday Apr 26 2017 at 05:29 PM

Hi Daniel. Thanks for pointing that out, I have updated the lesson with the correction. Elanor