How do I Create an HTML E-Mail?

This lesson describes how to launch a new e-mail compose window from within LiveCode and populate the message body with HTML content. Example code is provided.

Refer to How do I Attach a File to an E-mail? (http://lessons.livecode.com/m/4071/l/13197-how-do-i-create-an-html-e-mail) for information on how to expand this lesson to include attachments.

Note: This lesson assumes that you are using Thunderbird as your default e-mail client. Instructions on setting up Thunderbird are not provided here.

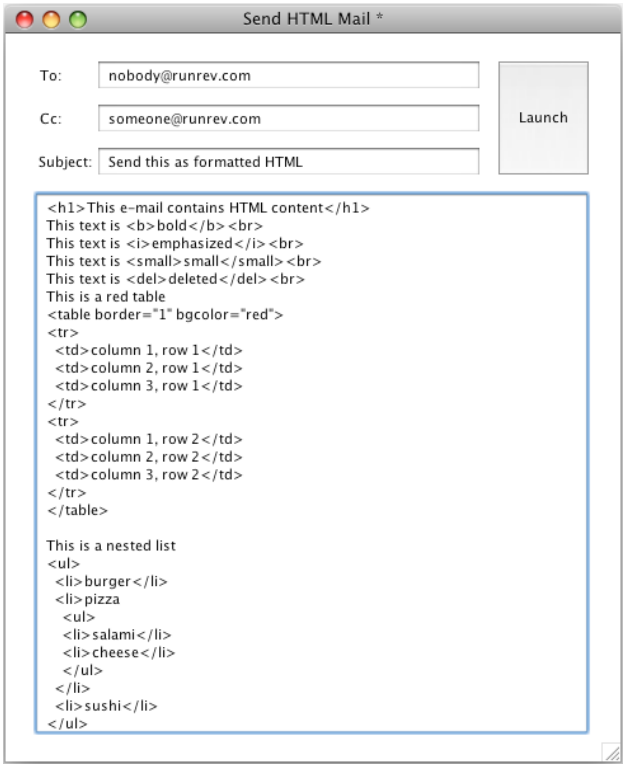
Introduction

LiveCode supports revMail and revMailUnicode to send e-mail via the local e-mail client. This feature is great for sending text based e-mail or data from your online forms. The underlying mailto protocol that is used by LiveCode and is supported by most e-mail clients is limited and does not support attachments or HTML. Some e-mail clients do allow you to pass HTML via the mailto protocol body token, but there is no uniform convention for this.

There are alternatives to using the mailto protocol, and this lesson shows you one such alternative. It uses the Thunderbird command line interface to add the richness of HTML to an e-mail that was created by a LiveCode stack.

Interacting with the e-mail client via the command line interface is very similar to the way revMail and revMailUnicode interact with e-mail clients over the mailto protocol. This allows us to write surprisingly short and powerful code.

Creating the Input Fields



The first step is to set up a user interface through which we can collect the information that is sent to the e-mail client.

We create Text Entry Fields for the **To:** e-mail addresses, the **Cc:** e-mail addresses, the **Subject:** line and the message body.

Give the **To:** field the name *toField*, the **Cc:** field the name *ccField*, the **Subject:** field the name *subjectField* and the body field the name *bodyField*. LiveCode uses the names to look up the content of the fields when creating the string for the command line interface.

We also have to add a button that tells LiveCode to send a message to the command line interface of the e-mail client. Give the button the name Launch.

For demonstration purposes, the fields in the above figure have been populated with values. There is one e-mail address in the **To:** field and one e-mail address in the **Cc:** field. Thunderbird accepts comma delimited e-mail addresses, allowing you to add several recipients to the **To:** and **Cc:** fields. The body contains a number of HTML examples that will later be rendered and formatted by the e-mail client.

Setting up the Thunderbird Launch String

All information included in the e-mail, from the addresses to the message body, is wrapped into one single text string that is then passed, via the Thunderbird command line interface, to the Thunderbird command line interpreter. The text string is the only variable we have to declare for this application.

Add this variable declaration to the *on mouseUp* call of *button Launch*.

 # declare the variable that holds the launch string

```
local lvLaunchString
```

Extracting Data from the Input Fields

The following code extracts the data from the fields of the user interface and calls the command *populateLaunchString* to wrap the data into *lvLaunchString*.

You should also add this code to the *on mouseUp* call of *button Launch*.

```
# populate the launch string with data from the user interface
put empty into lvLaunchString
populateLaunchString lvLaunchString, "to=", the text of field "toField"
populateLaunchString lvLaunchString, "cc=", the text of field "ccField"
populateLaunchString lvLaunchString, "subject=", the text of field "subjectField"
populateLaunchString lvLaunchString, "body=", the text of field "bodyField"
```

Adding the Platform Specific Thunderbird Command Processor

We have to spend some time setting up conditions that allow us to interface with Thunderbird in different environments. Thunderbird is an independent application that does not come with LiveCode and hence has its own installation paths and platform specific configurations.

The following code allows us to set values that are platform specific and apply to MacOS and Windows. The Thunderbird paths shown here are specific to the machines on which the stack was tested, when writing this lesson. It is possible that Thunderbird is installed in a different location on your machine.

This code should also be added to the *on mouseUp* call of *button Launch*.

```
# populate the launch string with platform specific information for thunderbird
if the platform is "MacOS" then # we are on a Mac
        put "/Applications/Thunderbird.app/Contents/MacOS/thunderbird-bin -compose" && lvLaunchString
into lvLaunchString
else if the platform is "Win32" then # we are on Windows
        put "c:\Program Files\Mozilla Thunderbird\thunderbird.exe -compose" && lvLaunchString into
lvLaunchString
else  # we do not support this platform
        exit to top
end if
```

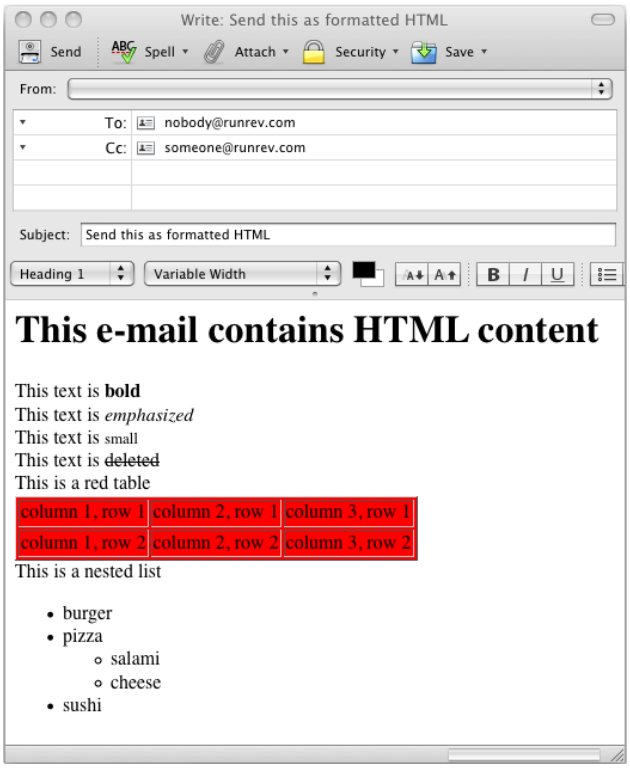
Creating the Launch String

The following code assembles the information from the user interface into *lvLaunchString*. This code is wrapped into a command declaration that should be placed on the *button Launch* card, but outside of the *on mouseUp* call.

```
# assemble tokens and arguments into the launch string
command populateLaunchString @pLaunchString avToken avArgument
        if avArgument is empty then exit populateLaunchString
        if pLaunchString is not empty then put pLaunchString & "," into pLaunchString
        put pLaunchString & avToken & "'" & avArgument & "'" into pLaunchString
end populateLaunchString
```

The command takes three parameters. The first one is a pointer to *lvLaunchString* itself. The second parameter is a token that the Thunderbird command line interpreter uses to identify the content that follows the token. The third argument is the content that the Thunderbird command line interpreter should associate with the token. For example, if we get the token *cc=* and the argument *someone@runrev.com*, then Thunderbird knows that we are sending an e-mail address that should be placed on the *Cc* line of the e-mail.

Launching the Compose Window



The last section of code should be added to the end of the *on mouseUp* call of *button Launch*. This code creates a command process that sends *lvLaunchString* to the shell. We tell LiveCode to hide the console window, this allows us to launch the Thunderbird compose window silently.

```
# launch the thunderbird compose window
set the hideConsoleWindows to true
open process lvLaunchString for update
close process lvLaunchString
```

As you can see from the figure, the information from the LiveCode user interface is now placed in the corresponding fields of the Thunderbird compose window. The HTML tokens have also been rendered and formatted HTML text is displayed.

Note: This lesson has shown you how to create HTML formatted e-mails via the Thunderbird command line interface. It is important to be aware that the examples here do not test the content of the Text Fields for constructs that Thunderbird may interpret as command tokens and argument. Any production application that is designed around this lesson must ensure that Text Field data does not contain values that Thunderbird may interpret as a command.

1 Comments

Christopher Thursday Apr 06 2017 at 04:19 AM

This works for one message but what about generating several emails? The TB command line interface appears to only allow the creation of one message. Call it again and the TB GUI gives back a message saying it can only be opened once. Try to use Write Process ... instead of a second Open command and it is simply ignored.