# Saving data in LiveCode standalones

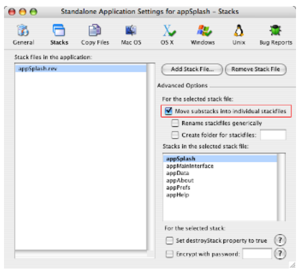
*by Sarah Reichelt*
*5 May 2005*

OK, so you've made your application, it all works perfectly inside LiveCode and now you want to make it into a standalone, so others can use it. Now you find out that standalones cannot save to themselves!
So how can you save the data inside your standalone application. Well it turns out that it isn't so difficult after all, so don't panic :-)
I am going to discuss 4 different methods of saving data. There are undoubtedly more, but these four should give you enough ideas to get started, and then you can develop your own procedures.

1. Saving sub-stacks
2. Saving template stacks
3. Saving preference files
4. Saving data files

**1. Saving sub-stacks**

By far the easiest method is to make sure that you can save all the sub-stacks in your project. This is frequently called the splash screen method because the best way to do it is to have the mainStack of your application be nothing more than a decorative splash screen which takes you to the sub stacks where all the action occurs and where any data changes can be saved as part of the stack.
From the File menu, choose "Standalone Application Settings...". Click on the "Stacks" button in the toolbar and you will see you stack file and all its sub-stacks listed. Tick the checkbox that "Move substacks into individual stackfiles".

| | |
|---|---|
|   Standalone Application Settings for appSplash – Stacks<br>General<br>Stacks<br>Copy Files<br>Mac OS<br>OS X<br>Windows<br>Unix<br>Bug Reports<br>Add Stack File...<br>Remove Stack File<br>Advanced Options<br>Stack files in the application:<br>appSplash.rev<br>For the selected stack file:<br>☑ Move substacks into individual stackfiles<br>☐ Rename stackfiles generically<br>☐ Create folder for stackfiles:<br>Stacks in the selected stack file:<br>appSplash<br>appMainInterface<br>appData<br>appAbout<br>appPrefs<br>appHelp<br>For the selected stack:<br>☐ Set destroyStack property to true<br>☐ Encrypt with password: | *Stack naming:*<br>*You will see from the picture that I use a certain style for naming all my stacks. This is because all applications tends to have some of the same stacks e.g. Prefs, Help, About*<br>*LiveCode hates opening more than one stack with the same name, so I choose a prefix for each project and name all the stacks in that project using that prefix. For this example, I used "app".* |

Close the settings window, save your stack file and choose "Save as Standalone Application..." from the File menu. Your application will be created but the sub-stacks will not be part of the application file, so they can be saved at any time.
In the screen shot above, you can see that I have a single stack file called "appSplash.rev". This contains 6 stacks: the appSplash stack and 5 other sub-stacks. After building the standalone as described, the appSplash stack will not be writable but the other 5 stacks can all be changed and saved as you need.
I built this test application for Windows and Mac OSX and as they produce different file structures it is worth explaining where each standalone keeps its sub-stacks:

- Windows creates a single folder with the exe and the other .rev files in the same folder
- Mac OS X creates an application bundle which looks like a single file but is actually a folder. To see what is inside this folder, control-click on the application and choose "Show Package Contents" from the popup menu. Look in the Contents folder and then in the MacOS folder - you will see your application and the sub-stacks.

You don't have to worry about where the system has put the sub-stacks, the LiveCode engine in the standalone knows where to find everything if you leave it where it was.
**2. Saving template stacks**
Method one makes all the sub-stacks saveable, which may be what you need but can cause problems if the user does not have write access to the folder where the application is installed. If this is likely to be a problem, and you only need a single stack to be saved, you may like to consider the template stack method instead.
Build your standalone into a single application file so that none of the stacks are saveable, but have one of the sub-stacks be a template for the stack you need to save. To save the data, clone this stack and save the copy (with all the data) into an appropriate location e.g. the current user's documents folder.
When the program first starts, look for this data stack and if it doesn't exist, create it from the template with the default settings or data already in place. When shutting down the application, or after changing any data, save this stack again.
To do this, you will need a script like the following:

```
on createDataStack
   clone stack "dataTemplate"
   set the name of stack "Copy of dataTemplate" to
"MyAppData"
   set the fileName of stack "MyAppData"to \
      pathToUsersDocumentsFolder &"/MyAppData.rev"
      save stack "MyAppData"
end createDataStack
```

**3. Saving preference files**

Some applications need very little data saved, maybe a few preferences to deal with window placement or a couple of data fields, or maybe the state of some checkboxes or radio-buttons. In this case saving a preference file is the best way to go. Remember that a user may re-install their system at any time and lose all preference files. No vital data should ever be stored as a preference and your application needs to be able to work without any settings, either by using a default setup or by stopping and asking the user for more information.

LiveCode gives you an easy method to find out where the preference file should be stored, in both Windows & Mac systems. (For Linux / Unix systems, use the user's home folder to store such files.)

Here is my function (Mac or Windows only) for getting the path to the correct folder to store preferences:

```
function getPrefsFolderPath
     if the platform = "MacOS" then
         put specialFolderPath("Preferences") into
prefsFolder
         else if the platform = "Win32" then
             put specialFolderPath(26) into prefsFolder
         end if
         if last char of prefsFolder is not "/" then put
\
        "/" after prefsFolder
         return prefsFolder
 end getPrefsFolderPath
```

For Macs, this gives you the current user's Preferences folder. For Windows, it gives the user's Application Data folder. Note: I always like to check to make sure that a folder path always ends in a slash. The script above handles that.

Having worked out the path to your preference file, you need to have routines to read and write the file and store it's data in your application. My preferred method is to have a preferences stack to allow the user to

edit settings. When the application starts, it reads the preference file and sets the various objects in the preferences stack. Then when I need to know a setting, I just have to check the relevant object on the preferences stack.
When the user edits the preferences stack and wants to save them, I write the new settings to the preferences file. If they edit and then cancel, I can read the original data back from the prefs file.

**4. Saving data files**

If your data is bigger or more important than is suitable for a preference file, then you need to store it somewhere else. Where you store it is basically up to you and your users. It is very convenient to store any data in a sub-folder of your application folder. As with using method 1, this may cause problems if the application is stored in the Applications or Program Files folder and the user does not have write access to that folder. You may wish to locate the user's Documents folder and save in a folder there, or you may prefer to ask the user where they would like the files stored. Some systems allow you to install the complete application in the user's own folder, which means you can be sure of having write access for all the files.
What ever you choose, you need to allow for the user not being able to write - if writing your file gives an error, tell them and let them select another location. If the user can select a location, you will need to store this as a preference so that you don't have to ask every time.
If you use a sub-folder of your stack, make sure you use relative addressing so that the user can move to a new hard drive, or rename their drive without breaking the links. The easiest way to find a relative address is to use the filename of the stack.
If I get the filename of a stack I am working on at the moment, I get this:

```
/Users/sarah/Documents/LiveCode/DateTime.rev
```

(I use Mac OS X; if you have a different operating system, you will get a different structure, but the scripts below will still work.)
Now suppose I wanted to save data in a sub-folder of this stack:

```
put the filename of this stack into tPath
set the itemDelimiter to "/"
put "DateTime folder/" into last item of tPath
if there is not a folder tPath then create folder tPath
```

This gives me a folder called "/Users/sarah/Documents/LiveCode/ DateTime folder/". I have the full path to the folder but the only section hard-wired into the script is the actual folder name. It doesn't matter

where the application goes, this script can still find where to store the data folder.
To write data to the data folder, you can write either binary files or text files. If your data is plain text, it doesn't make any difference, however if you want to save pictures, sounds or any other non-text data, you will need to save in binary format. The command is the same in either case, with a single word change:

```
put tTextData into URL ("file:" & tPath & "TextData")
```

or

```
put tBinData into URL ("binfile:" & tPath &
"BinaryData")
```

The URL command is a great convenience for those of us accustomed to the open file - read or write - close file cycle needed in other languages. All you need to do is decide whether you need to use binfile or file.
To summarize, while a standalone cannot save directly to itself, it can save to separate sub-stacks, it can save cloned stacks and it can save external data files. so there is no need to worry about whether your application can save, it's just a matter of working out which method is most suitable for your particular application.
LiveCode experts will point immediately to the methods that I have not mentioned here of which the most common would be using an external database such as MySQL or writing preferences to the Windows registry, however this is intended as a guide for getting started rather than a complete list of all possibilities. Hopefully it will prove useful.