LiveCode Lessons (copy)

Last Updated

Apr 04, 2016

Print Article

Comments

Liquid error: internal for this article

How do I add Multiple LiveCode Files in LiveCode Server?

This lesson describes how to use **include** and **require** in order to load multiple LiveCode files into a LiveCode Server application. Conceptual syntax is provided.

Introduction

When developing a larger LiveCode Server application it is often useful to consider the architecture of the system in advance and possibly break down the system into a number of logical files that can be loaded into the application at run time. You can add files to a LiveCode Server application by using the **include** and **require** commands.

Including Script Files

A 'Home' stack is created at start up, which serves as the container for the global script on a LiveCode Server application. This stack sits at the root of the message path and works in the same way as the IDE home stack, which sits after all mainstacks in the message path, but before library stacks and backscripts.

Global script execution begins with either the file that was specified on the command-line (non-CGI mode), or as part of the **PATH_TRANSLATED** environment variable (CGI mode). Further scripts can be added and executed via the **include** or **require** commands. These always affect the global script, regardless of where they are executed from. For example if you include a LiveCode file from a handler in a stack, then it only affects the 'Home' stack script environment.

A script added by **include** or **require** is parsed in full before being executed, with any handler and variable definitions being added to the 'Home' stack environment before any commands placed at global scope are executed. Each command and function present in the added script is executed in order in which it is encountered in the file.

include executes the given script in the context of the global environment, each time it is called. **require** also executes the given script in the context of the global environment, but only if it has not already been included or required. This makes it easy to implement include-once files that can be used for library scripting. **include** and **require** are distinct in the sense that if you first **require** a file and then **include** it, the second include executes the file.

include and **require** only work when running in the server environment, invocation of the commands in other environments throw an error.

If a **scriptExecutionError** message is sent to the 'Home' stack as a result of an uncaught error then the standard engine error stack listing is provided, detailing the errors that occurred at each stage of the stack that is being unwound and a list of all files that have been **included** or **required**.

include

include can fulfill a number of requirements but consider it in the context of global variables and use it as a means of resetting global variables or configurations to default settings. Such default settings may be stored in a *defaults.lc* file. By including this file, any defaults that are specified in that file are reset each time you **include** the *defaults.lc* file. Generally speaking, it would be more appropriate to use a command or function to reset global variables, but this example demonstrates how **include** can be applied in a non-library include context.

The *defaults.lc* file may look something like this:

```
<?lc

put empty into gText

put 1 into gCount

-- more defaults here ....

?>
```

It can contain a number of default settings that have to be set at startup or at a reset instance. You could **include** the file each time there was a need to reset values in a main stack such as *main.lc*:

```
<?lc

-- some processing above here ...

if tResetNeeded then

include "defaults.lc"

end if

-- more processing below here ...

?>
```

require

An example for the use of **require** may be in the context of a generic library that provides a custom database interface layer. This layer may then be accessed via an interface to a customer facing web page and via an administrative interface. If you use **require** then you could add a database interface file *dbInterface.lc* to both the web page interface *webInterface.lc* and administrative interface *adminInterface.lc* files and then include the *webInterface.lc* file and *adminInterface.lc* file in the *main.lc* 'Home' stack. This order would allow you to include both the web page facing interface and the administrative interface and ensure that the customer database layer is only loaded and executed once.

The *dbInterface.lc* could be any kind of database interface library:

```
<?lc

-- provide some database interface in here

?>
```

The web interface file accesses *dbInterface.lc* via the **require** command:

```
<?lc

require "dbInterface.lc"

-- more processing below here ...

?>
```

The administrative interface *adminInterface.lc* accesses the *dbInterface.lc* via the **require** command:

```
<?lc

require "dbInterface.lc"

-- more processing below here ...

?>
```

The *main.lc* stack can include both the *webInterface.cl* and *adminInterface.lc* without needing to know if *dbInterface.lc* is required:

```
<?lc

include "webInterface.lc"

include "adminInterface.lc"

-- more processing below here ...

?>
```

As *dpInterface.lc* is being added with **require**, we do not need to be concerned about whether or not *dpInterface.lc* is being executed more than once, regardless of how many time we may be adding it in a possible file hierarchy.

‹ Prev: How do I choose which LiveCode Server engine to use with On-Rev?
Next: How do I use stacks with LiveCode Server? ›

0 Comments

Add your comment

Name* Email*

Comment*

Subscribe ☐ E-Mail me when someone replies to this comment

Are you human?

I'm not a robot reCAPTCHA Privacy - Terms

Submit Comment