Building customized scrolling lists on top of the DataView control

Leave a reply

Recently I announced that the source code for the DataView control was now available for LiveCode developers. A developer feeds a DataView rows of data from a variety of data sources. The DataView then displays those rows using templates (groups on a card) specified by a developer.

The DataView is a custom control made up of a group, some child controls of that group, and various behaviors scripts. The custom group is placed on a card wherever a DataView will be used. Out of the box a DataView doesn't do much without some configuration. It has no default UI for rows and it doesn't favor one data source over another. The developer must define the row template(s) to use and provide data for a specific row when the DataView asks for it.

Rather than being a control that is really easy to use but is very limited, the DataView is a control that can be customized for a variety of different needs. It serves as the foundation for any number of customized controls. For example, the DataView Demo application provides a a few examples of data being displayed with a DataView. As of this writing there are five examples:

- List extensions installed in the IDE.
- Create a file browser from a folder of your choosing.
- Browse the stacks open in the IDE.
- List FontAwesome fonts along with icons taken from a SQLite database.
- List movies found in a SQLite database.

In addition to the customized UI that each example uses, a couple of different data sources are used:

- Arrays built from functions, folder listings, and open stacks.
- Database cursors.

In the demo application the ability to assign a database cursor or an array to the DataViews is provided by behaviors that inherit the base DataView behavior and are assigned to the DataView group control. These behaviors control the flow of data from a data source to the DataView behavior. The DataView comes with one behavior that allows a developer to assign a numerically indexed array to the `dvData` property of a DataView (*array_controller_behavior.livecodescript*). The DataView Database Cursor behavior, a separate download, adds a `dvCursor` property. A developer can assign the result of a call to `revQueryDatabase` to the property and the DataView will automatically scroll through the results. There is also the DataView Tree behavior that adds a `dvTree` property for displaying tree structures in a DataView. The File Browser and Stack Browser examples are built on top of it.

While these behaviors that add a `dvData`, `dvCursor`, or `dvTree` property are helpful, they still don't provide templates that dictate what each row will look like. But it is possible to distribute very specialized controls that are built on top of a DataView. These specialized controls can focus on providing a very specific feature to an application. For example, a File Browser control could be distributed which contained the necessary behavior and row templates for browsing the file system. The developer using the control might only need to specify some colors to use for folder and file icons and assign the *dvRootFolder* property in order to have a working file browser in their application.

A similar control could be made for a generic tree browser. For many use cases the ability to specify an icon, label, and colors, along with the necessary event messages to respond to user actions, would be adequate for a developer to add a tree UI to his or her application.

This modular approach to building on top of DataViews is greatly simplified because the DataView is packaged up as a Levure Helper. A Helper in the Levure Framework is a means of adding complex functionality to a Levure application by simply dropping a folder into the Levure application project folder. The folder contains a configuration file (*helper.yml*) that tells Levure how to use the files within the folder. For example, the DataView helper folder contains seven script files and a stack with a sample DataView control. Six of the scripts serve as behaviors while the other script is used as a frontscript.

If we consider the File Browser example within the context of a Levure Helper then distributing the control becomes much simpler. The developer creates a behavior script which extends the DataView behavior along with any row templates (groups on a card) that the File Browser needs. Those files are placed in a single folder along with the *helper.yml* file. The developer who wants to use the control adds the DataView helper folder and the File Browser helper folder to the *./helper* folder of a Levure application. After that all of the code required to make the control work is available.

Going forward I hope to see other developers build specialized controls with beautiful UIs on top of the DataView.

The DataView control has been tested on macOS, Windows, and iOS in LiveCode 8 and 9.

This entry was posted in Uncategorized and tagged LiveCode on January 16, 2019.

← Notes on Setting up Windows to Build LiveCode Community

Creating a DataView Control →

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

Post Comment

SEARCH POSTS

Search

RECENT ARTICLES

The Bakers Assistant IDE Plugin
Creating a DataView Control
Building customized scrolling lists on top of the DataView control
Notes on Setting up Windows to Build LiveCode Community
VMWare Fusion: Mounting a network drive for reliable cross-platform development with LiveCode
Exporting SVG files from Adobe Illustrator for use with drawingSvgCompileFile()
Using the livecode standalone engine to run script only stacks on a server
Accounting for new widget properties in an OnLoad handler

LINKS

LiveCode
LiveCode Conference 2019
ScreenSteps (my company)
This Week in LiveCode Newsletter