Development Environment Overview

Table of Contents

- Architectural Overview
- User Interface Elements / Object Model
- Display Modes (HD/SD)
- Top-Level Menu
- Entry Points
  - Sub RunUserInterface()
  - Sub RunUserInterface(aa as Object)
  - Sub Main()
  - Sub Main(aa as Object)
    - "Source" parameter
    - "lastExitOrTerminationReason" parameter
  - Sub RunScreenSaver()
  - Sub RunScreenSaverSettings()
- User Interaction/Events
- Customization

Architectural Overview
The diagram above provides a high-level overview of the main system components for the Roku Streaming Player platform. Developer applications are written using the BrightScript programming language. These applications are designed to be standalone entities that can be deployed to a running system with minimal impact.

BrightScript applications are dynamically loaded at runtime and run within a unique context within the BrightScript virtual machine. They are “sand-boxed” and run protected from other areas of the system. Scripts only have access to platform resources that are exposed to the scripting layer as BrightScript components. Developers have a wide selection of built-in elements from the BrightScript programming language, plus additional platform components to build their applications. See the BrightScript Reference and the Component Reference for additional information.

User Interface Elements / Object Model
The Roku SDK uses the BrightScript programming language for development and exposes a set of interfaces to platform services through BrightScript Components. These platform SDK services include capabilities such as networking, video playback, user interface, and data management. The objects in the Roku SDK are divided into two primary areas:

- Core Objects – Fundamental objects that exist on all Roku platforms and are device independent
- Platform Objects – Objects unique to a specific platform, such as the Roku Streaming Player

Developing an application for the Roku Streaming Player consists of writing a BrightScript application, packaging the application and associated resource files and deploying it to the platform. During development packaging consists of a structured ZIP file. For final deployment, tools are provided to create a signed and encrypted application package. At runtime, the player will enumerate the installed applications and display them on the main menu. When the user selects the application, the script(s) are loaded and control is passed to your application. When the user exits, the script is halted and control is returned to the user interface "shell".

User interface functionality available in the SDK includes:

- Top-Level Menu (Launch screen for applications with logo art)
- Poster Screen (Horizontally scrolling list of shows with poster art)
- Springboard (Detail screen with options for displaying individual shows)
- Video Player Screen (Video playback support with progress bar and trick mode support)
- PIN Entry Screen (User entry of PIN for purchase/rental verification)
- Message/Error Dialog (Dialog for display of errors and other user messages)
- Filter Widget (Selection widget for filtering content display by type)
- Rendezvous/Code Registration Screen (Display/validate registration codes)
- Username/Password Registration Screen
- Text Screen (Display formatted text to the user and allow selection of options)
- Search Screen (Keyword based search with progressive disclosure of results)

Detailed information on all these screens can be found in the BrightScript Component Reference.

Display Modes (HD/SD)

The user interface has been designed to support both High Definition (HD) and Standard Definition (SD) displays. By default the streaming player starts in SD mode, but allows the user the option to specify their display preferences. Display preferences are handled globally for all applications by selecting the "display type" option in "settings".

There are three possible modes:

- 4:3 standard-definition (480i/p)
- 16:9 anamorphic widescreen (480i/p)
- HDTV (720p)

The SDK UI objects are SD/HD aware and will automatically display in the correct mode. In some cases, the HD mode will allow the user to see more data on the screen. The SD UI is rendered natively at 480p and the HD UI at 720p. As a developer, no special programming is required to support these display modes. Any artwork used by the application (movie posters, logos, etc.) should be provided in both HD and SD versions and included with the application or downloaded dynamically at runtime. The screen objects will attempt to scale improperly sized artwork, but this could result in a loss of quality or degrade performance. It is strongly recommended that developers provide original artwork in both resolutions.

Top-Level Menu

The top-level menu provides a launch point for all applications. Applications must provide HD and SD versions of the application icon artwork for display on the top-level menu. When an application is selected from the top-level menu, the application is loaded and control is transferred to the application. The user may return to the top level at any time by pressing the Home button. Pressing the Home button will always exit the application and unconditionally return control back to the UI application shell.

Entry Points
There are several reserved function names that serve as entry points to the channel.

**Sub RunUserInterface()**

**Sub RunUserInterface(aa as Object)**

RunUserInterface is the normal entry point which is called when a channel is selected on the Roku Home Screen. It may take a roAssociativeArray parameter that is passed via the External Control APIs.

**Sub Main()**

**Sub Main(aa as Object)**

If there is no RunUserInterface() function in the application, the function Main() will be called as the entry point for the application.

**"Source" parameter**

The Main(aa) and RunUserInterface(aa) associative array contains the "source" named parameter. Its value represents the path the channel was launched from. The values and their meanings are in the table below.

<table>
<thead>
<tr>
<th>Value of &quot;Source&quot;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;homescreen&quot;</td>
<td>launched from the &quot;home&quot; section of the main Roku channel selection page</td>
</tr>
<tr>
<td>&quot;ad&quot;</td>
<td>launched from an ad displayed to the right of the channels on the home page</td>
</tr>
<tr>
<td>&quot;external-control&quot;</td>
<td>launched via the ECP protocol. Typically this is from the Roku mobile app.</td>
</tr>
<tr>
<td>&quot;partner-button&quot;</td>
<td>launched via a partner button on the Roku remote</td>
</tr>
<tr>
<td>&quot;other-channel&quot;</td>
<td>launched from another channel on the Roku</td>
</tr>
<tr>
<td>&quot;auto-run-dev&quot;</td>
<td>launched as a side-loaded developer channel</td>
</tr>
<tr>
<td>&quot;debug-server&quot;</td>
<td>launched via the debug server on port 8080.</td>
</tr>
<tr>
<td>&quot;purchase-dialog&quot;</td>
<td>launched via the purchase dialog in our channel store</td>
</tr>
<tr>
<td>&quot;meta-search&quot;</td>
<td>launched from Roku Search</td>
</tr>
<tr>
<td>&quot;dial&quot;</td>
<td>launched via the DIAL protocol</td>
</tr>
</tbody>
</table>

**"lastExitOrTerminationReason" parameter**

PluginEnums.h defines an ExitCode enum with a set of defined exit code reasons. Every Plugin object has a member to track the last exit code. These member values and their meanings are in the table below.

When running the Plugin, the last exit code is passed to the BrightScript entry point in the parameters associative array via the "lastExitOrTerminationReason" field. The last exit code gets reset to 'unknown' before each run.
<table>
<thead>
<tr>
<th>Value of &quot;lastExitOrTerminationReason&quot;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT_UNKNOWN</td>
<td>Currently used as the default exit reason; either if there was no prior exit (i.e. first launch of the channel after system boot), or if there was no &quot;unusual&quot; exit reason noted (i.e. not due to a BrightScript crash or system resources issue).</td>
</tr>
<tr>
<td>EXIT_POWER_MODE</td>
<td>Used if the channel was exited by the system due to the device being powered off by the user (or by user-scheduled power off).</td>
</tr>
<tr>
<td>EXIT_DIAL_DELETE</td>
<td>Used when a DIAL-controlled channel exits due to a command received from the second screen app to exit the channel.</td>
</tr>
<tr>
<td>EXIT_OUT_OF_MEMORY</td>
<td>Used if a channel exits due to the system being under low memory conditions.</td>
</tr>
</tbody>
</table>
| EXIT_IDLE_AUTO_EXIT                | Used if the system and channel were detected as idle for a prolonged period of time and the system exits the channel per Roku policy.  
**Available since Roku OS 8.1** |
| EXIT_BRIGHTSCRIPT_CRASH            | Used if a BrightScript runtime error caused the channel to exit.  
**Available since Roku OS 8.1** |
| EXIT_BRIGHTSCRIPT_STOP             | Used if a BrightScript 'stop' command was encountered when running the channel in production mode (non-developer side-loaded).  
This is treated by the system the same as a BrightScript runtime error and the channel exits.  
**Available since Roku OS 8.1** |
| EXIT_BRIGHTSCRIPT_TIMEOUT          | Used if a BrightScript execution timeout error occurred and the channel exits.  
(A timeout error occurs indicates that the channel user interface was unresponsive for a prolonged period of time, such as program lock-up).  
**Available since Roku OS 8.1** |

**Sub RunScreenSaver()**

RunScreenSaver is called to launch a screensaver when the Roku has been idle for the configured idle time. The screensaver to launch is selected by the user on the Screensaver settings page. The Roku box may not start the custom screensaver and will use a default screensaver instead if there is insufficient memory to start a screensaver.

**Sub RunScreenSaverSettings()**

RunScreenSaverSettings is called when the user selects “Custom Settings” on the Screensaver settings page.

**User Interaction/Events**

Roku SDK UI objects provide an event oriented model for user interaction. Instead of receiving and directly handling all of the IR events received by the application, the UI elements will handle all navigation commands directly and send higher level events to the script as the focus changes or the
user makes a selection. Scrolling, trick modes and screen-to-screen navigation is handled by the SDK UI objects and the script generally only needs to interact when there is a change in context or the user highlights or selects new data elements. Refer to the Event Loops section or the Class Interface sections for additional information.

Customization

The objects in the user interface framework expose a set of screen types which standardize user interaction and make it easy for developers to quickly write and deploy applications. Screen types enforce a user interaction model and ensure consistency between applications. They may be customized to provide a unique, developer specific look-and-feel. Customization is currently focused on "re-skinning" the application and supports the following types of changes:

- Add an application specific image to the top-level menu
- Change the text to be displayed on the main menu to identify the application
- Change the application logo to be displayed in the header area for the screen
- Change the artwork used on the "overhang" or header area for the screen
- Change the background color for the screen
- Change the colors used for font rendering on text, buttons, and screens

Within the application the developer is free to combine the available screen types and controls as needed to implement their application. The hierarchy of screens is unique for an application and depends on the user experience desired. Some applications may be fairly "flat" while others may have a deeper hierarchy.