Introduction to U3-Aware Application Building in Revolution

Contents

Section 1: Revolution U3 Documentation

Background

Traditional flash drives allow you to store information but they do not allow you to take your *applications* and *settings* with you. U3 "smart" drives are flash drives, but with a difference, you can install your entire user experience – both applications and data, making your computer environment completely portable.

Its technically possible to run a program off any standard flash drive or other removable media. However U3 offers several benefits: a "launch pad" that operates like the familiar standard "Start" menu; a method for applications to bring with it all of its settings such as preferences and registry entries; and a guarantee that once you eject a drive the "host" system is left unchanged. Finally, the U3 platform is backed by major storage vendors. There are millions of U3-smart flash drives being shipped every month, making this rapidly growing platform an exciting new distribution opportunity for application vendors.

In order to run correctly from a U3 drive, an application needs some simple modifications. This document tells you how to produce a *U3 smart application* using Revolution.

Requirements

In order to use a U3 drive you will need the following

1. A U3 compliant flash drive.
2. One of the following operating systems (all Windows)
    1. Windows 2000 with SP 4 and later
    2. Windows XP all versions and Service Packs
    3. Windows server 2003

In order to build U3 applications with Revolution, you will need Revolution version 2.7.3 or later.

Note that to use the sample program and its associated stacks you will need to have QuickTime

installed.

Technology Overview

Each U3 device pairs with a program called the “U3 Launchpad”. The easiest way to think of this is like a Start menu, except that it is attached to the U3 drive, rather than a specific computer. The Launchpad allows the user to run, install, and uninstall applications on the U3 drive.

A U3 application has a life cycle consisting of six phases:

1. Installing the U3 package onto the drive
2. Configuring the host machine (the machine the drive is connected to)
3. Starting the U3 application
4. Stopping the U3 application
5. Cleaning up the host machine
6. Uninstalling the U3 package from the drive.

During a U3 application's life cycle it will be start up with parameters corresponding to these six phases. For example, when the application is installed it could be started with a parameter -u3install . The correspondence between actions and which application gets run with a specific parameter is defined by the manifest file.

When installing the Revolution IDE on a U3 drive, Revolution uses this action to request entering a license key. The installation can only proceed if the correct license key is entered.

The configuration action happens when the U3 application is first run on a host machine (after installation or when the drive is inserted into another machine). This allows the application to configure the host machine before it is started. This message could be used to check the host computer’s permissions for example.

After installation and configuration, an application can be started and stopped as many times as needed. Each time it is launched from a U3 drive the “start” action is performed.

When a user chooses to eject the U3 drive (either by using the eject button in the Launchpad or physically removing the drive), the Launchpad will invoke the *stop* action. At this point the application must quit all running instances. After an application has been stopped, the Launchpad will invoke the *cleanup* action, to ensure the application leaves the computer in the same state it was in before it was configured.

U3 Package Overview

A U3 application package comes in the form of a .u3p file. The user can directly install this file onto the drive using the Launchpad.

A u3p file is essentially a .zip file with a different extension[1] and a number of folders. This zip archive contains the following four folders (or a subset thereof):

1. Manifest
2. Host
3. Device
4. Data

The manifest directory contains all the configuration files for the U3 package. At this point in time[2]

1 Not all ZIP compression types are compatible with the U3 Launchpad. Revolution uses the compatible deflate compression type.

2 This may depend on the version of the Launchpad. Refer to U3 documentation for more details.

it is required that the directory contains at least the following two files: a file manifest.u3i and an icon file for the application.

The manifest.u3i is an XML file that describes the applications properties and actions to perform at each stage of the application's life cycle. For example, it defines the application's *title* and points to its *icon* file (which must be in the manifest directory). In particular, the manifest file defines which *actions* the application much perform in case of *installation*, *configuration*, *start*, *stop*, *cleanup* and *uninstall*.

In Revolution this is all handled automatically by using the *Advanced U3 Settings* in the Standalone Application Settings dialog. For more information about the format of the manifest file, please refer to the U3 documentation[3].

The Data directory contains all the persistent configuration files associated with the U3 application. Such files could include skins, templates, other media files, etc.

The Host directory contains the application executable and related files such as DLLs that are required to execute it. Before the application is started, the Launchpad will extract the U3 application files for the U3 host exec directory to a temporary directory on the host machine. This means that the actual application will run form the host machine! One of the reasons for this is application stability, if, for example, the drive is physically ejected, the application will maintain stability.

The Device directory contains the application files that stay on the drive. This should contain files that do not have to be copied to the host machine to maintain program stability, e.g. infrequently accessed files such as a game level that needs to be loaded only when the player reaches that level.

You can access the locations of these directories through the use of system environment variables. For example, the location of the Host Exec folder on the host machine (after the application was started) can be referenced via U3_HOST_EXEC_PATH. Thus, in Revolution this is available through the use of $U3_HOST_EXEC_PATH. Please be aware that these paths will contain "\" as directory separators instead of Revolutions "/". Thus, you will need to do something like

```
local tU3HostExecPath
put $U3_HOST_EXEC_PATH into tU3HostExecPath
replace "\" with "/" in tU3HostExecPath
```

Alternatively, you can obtain the U3 paths using the following functions in Revolution (by pasting them into your scripts):

```
function u3HostPath
     local tHostPath
     put $U3_APP_DATA_PATH into tHostPath
     replace "\" with "/"  in   tHostPath
     return tHostPath
end u3HostPath
```

Similarly one can define functions such as:

1. u3DataPath (using $U3_APP_DATA_PATH)
2. u3DevicePath (using $U3_APP_DEVICE_PATH)
3. u3DocumentPath (using $U3_DEVICE_DOCUMENT_PATH)

Use the Documents folder on the U3 drive to store user modifiable data, e.g. a text document created by a text editor.

Standalone Settings and U3

Revolution provides the ability to directly and easily build U3 applications. The Standalone Builder

3 Please go to http://www.u3.com for U3 documentation.

now provides the additional option to build you application for U3 drives through a setting on the Windows pane of the Standalone Settings.

Checking the box to build for U3 will build a standard U3 application, that will not handle u3install, u3configure, u3cleanup and u3uninstall messages. The Revolution engine will handle everything that is required.

The advanced settings for U3 allow the user to include individual files or folders in specific folders of the U3 package, namely the Host, Device and Data folders. Furthermore, it allows the user to select actions that will be invoked by the LaunchPad, and thus can be handled by your application through the use of the startup handler and its argument.

Handling U3 actions in Revolution

Revolution provides an elegant and simple way of dealing with U3 actions through the use of arguments to the startup handler. The startup handler is now provided with a single argument being the action the application is to deal with.

The Revolution Engine will start up your application with one of the following arguments passed to its startup handler:

1. u3install
2. u3configure
3. u3cleanup[4]
4. u3uninstall
5. u3launch[5]

Please note that your application will only receive these arguments if the corresponding U3 actions have been selected in the *Advanced U3 Settings* dialog (see above).

After the LaunchPad has invoked an action, it expects the application to quit as soon as it has handled any of the messages above (except for *u3launch*). When exiting your application, the LaunchPad determines if the action was successful through the error code returned. To return error codes, use the *quit* command followed by a number. A 0 should be used to indicate success, any other number will indicate failure. For example:

```
on startUp pMode
      switch pMode
      case "u3install"
            if (user_has_correct_privileges) then
                  quit 0 — the installation can continue
            else
                  quit 1 — the installation cannot continue
            end if
      case "u3cleanup"
            quit 0
      case "u3configure"
            quit 0
      case "u3uninstall"
            quit 0
      case "u3launch"
            -- application started from U3 drive
      break
      case empty
      default
            -- application running normal, not from U3 drive
      break
```

4 Any U3 manifest is required to include the hostCleanUp action. This is because of a bug in an earlier version of the LaunchPad. In order to make this transparent, Revolution handles this automatically through the use of a dummy action if cleanup is not selected in the advanced settings thus making sure that your application will certify.

5 This argument is provided to the startUp handler if the program is starting up from a U3 environment.

```
end switch
end startUp
```

You can determine if the application is running from a U3 drive by setting a flag when receiving the *u3launch* mode in your *startup* handler, or checking environment variables.

For more information on which environment variables are available, please refer to the U3 documentation.

Finally, you should ensure you handle the *u3eject* message.

A user can eject a U3 drive in two ways: *safely* (using the U3 Launchpad's eject button) or *physically* by removing the U3 drive from the U3 connection of the host machine.

In both cases the Revolution engine will send a "*u3eject*" message to the application's main-stack with an argument pType being one of "safe" or "physical", depending on the type of eject.

If pType is "safe" the application can take as much time as it needs to shut down, however, it **must** shut down, otherwise it will violate the U3 specifications. If pType is "physical" the application has no more than 4 seconds to respond, before it is forcefully terminated.

U3 Certification

If you wish to advertise your application is U3 compliant and make your application available on the U3 Software Central portal, it is required that you certify your application as being U3 compliant.

This involves running a few tests on your u3 package and the application itself. Some of these tests should automatically succeed as the package was created by Revolution and thus conforms many of the U3 standards. However you will also need to ensure your application is not hard coded to leave any changes on the users system. For more information on certification please refer to the U3 website[6].

6 http://www.u3.com

Section 2: Sample Revolution program for U3

This section will describe a sample program written in Revolution for a U3 drive. You can access this example stack and its source code in the Resources/Examples folder of your Revolution installation, or download it from our web site.

This example program is a simple music player that allows the user to play music from a U3 drive, and add new music to their collection on the drive by drag dropping music files onto the player interface.

This sample application will demonstrate the life cycle of a U3 application as follows:

1. On *installation* it displays a EULA (End User License Agreement) and the installation is only allowed to proceed the user agrees to the EULA.
2. On *configuration*, the host machine's registry is set up to contain the initial position into the play list.
3. Throughout use of the application the registry is updated to indicate the current position in the play list.
4. When the machine is *cleaned up*, the registry entry is copied back to the U3 device and removed, and thus the registry on the host machine is left unchanged.
5. When the application is *uninstalled*, a dialog will ask if the user wants to keep his or her music files on the drive or remove them.

We started by creating an application that is able to play music files. It has a simple stub function that it calls to find the play list, its position in the play list and the location of the music files. It also has a stub function that saves the current index in the play list.

We implemented the following steps in order.

The EULA

We want our application to display a EULA when it is installed so that the application can only be installed on a U3 drive if this is accepted. We shall also add functionality that disables the program if it is not run from a U3 drive, in this case simply to demonstrate this capability.

First, we need a EULA dialog, this has been included in the Player template application as a sub-stack called "EULA".

To display this dialog on *installation* we put the following code in the *startup* handler of the mainStack:

```
on startup pMode
     switch pMode
     case "u3install"
          hide me
          go to stack "EULA"
     break
     end switch
end startup
```

The Launchpad waits for an application to quit so that it can determine what to do based on the quit code. Thus, the EULA stack implements the following:

```
on agreeToEULA
     quit 0
end agreeToEULA

on disagreeToEULA
     quit 1
```

```
end disagreeToEULA
```

We want to add capability in to make sure that the application only runs from a U3 drive. We only need to add the following to the startup handler (continuing with our handler above):

```
on startup pMode
      switch pMode
      case "u3install"
            hide me
            go to stack "EULA"
      break
      case "u3launch"
            -- allow handler to continue
      break
      case empty
            quit
      break
      end switch
end startup
```

Of course, for optimal user interaction, it may be better to provide a dialog explaining that the application can only be run from a U3 drive, and should not be run otherwise.

Configuration

The first time the application runs, the U3 *configuration* action will be invoked, which in Revolution translates to the argument “u3configure” to the *startup* handler.

We set the play list registry entry to its initial value here:

```
on startup pMode
      switch pMode
      case "u3install"
            hide me
            go to stack "EULA"
      break
      case "u3configure"
            setupRegistry
      break
      case "u3launch"
            -- allow handler to continue
      break
      case empty
            quit
      break
      end switch
end startup
```

The *setupRegistry* handler sets up the registry, and quits the program:

```
on setupRegistry
      get setRegistry("HKEY_CURRENT_USER …
```

(Please refer to the stack script of the Player stack for the complete *setUpRegistry* handler.)

Starting Stopping and Running the Player

We do not need to do any work for U3 with regards to running the program. When the program is

started, a u3launch parameter is provided to the startUp handler.

In order to make it much easier for us to develop this program, we want to make sure that any paths point to local mirrors when the program is in development. Thus, we have provided a getMediaPath() function that returns the folder that contains the music of the player on the drive *only if* the program is running from a U3 drive.

Cleanup

At this stage, all that is required is to reset the registry, thus we augment our *startup* handler, adding a handler to clean the registry:

```
on startup pMode
      switch pMode
      case "u3install"
            hide me
            go to stack "EULA"
      break
      case "u3configure"
            setupRegistry
      break
      case "u3cleanup"
            cleanupRegistry
      break
      case "u3launch"
            -- allow handler to continue
      break
      case empty
            quit
      break
      end switch
end startup
```

(Please refer to the stack script of the Player stack for the complete *cleanupRegistry* handler.)

Uninstall

At this point, we will show a dialog asking the user if he or she wishes to delete the player's music files from the U3 drive.

Once again, we created a sub-stack for this dialog. This dialog stack will *quit 0*, deleting or keeping files depending on the users choice:

```
on startup pMode
switch pMode
      case "u3install"
            hide me
            go to stack "EULA"
      break
      case "u3configure"
            setupRegistry
      break
      case "u3cleanup"
            cleanupRegistry
      break
      case "u3uninstall"
            hide this stack
            go to stack "Dialog" of this stack
      break
      case "u3launch"
            -- allow handler to continue
```

```
      break
      case empty
            quit
      break
end switch
end staruUp
```

Finally...

Take a look at the stack source to see how to copy music files to the device when drag dropped onto the player.

It would be easy to extend this program to include more advanced features. For example, you could include an option to select a specific track, to reorder the play list, to play the list randomly, or to toggle playback looping.

This example U3 application took only a couple of hours to put together. This demonstrates the power of Revolution to quickly and efficiently develop an application that works correctly on the U3 platform.

Enjoy creating your own U3 applications with Revolution!