

Creating a Video Library Application

1 Playing Video in a Revolution Application

1.1	What You Will Create in This Chapter	4
1.2	Create the Window Where Your Video Will be Displayed	5
1.3	Add the Video Player to the Window	11
1.4	Organizing Your Video Files	21
1.5	Add the Video Files Menu	24
1.6	Populating the Video Files Menu	31
1.7	The Card Script In Detail	39
1.8	Loading Video Files When a User Selects a Video From the Menu	48
1.9	Adding Code to Setup the Application Once it is Launched	55
1.10	Creating Your Executable Application for Mac and Windows	58
1.11	Sharing Your Application	65

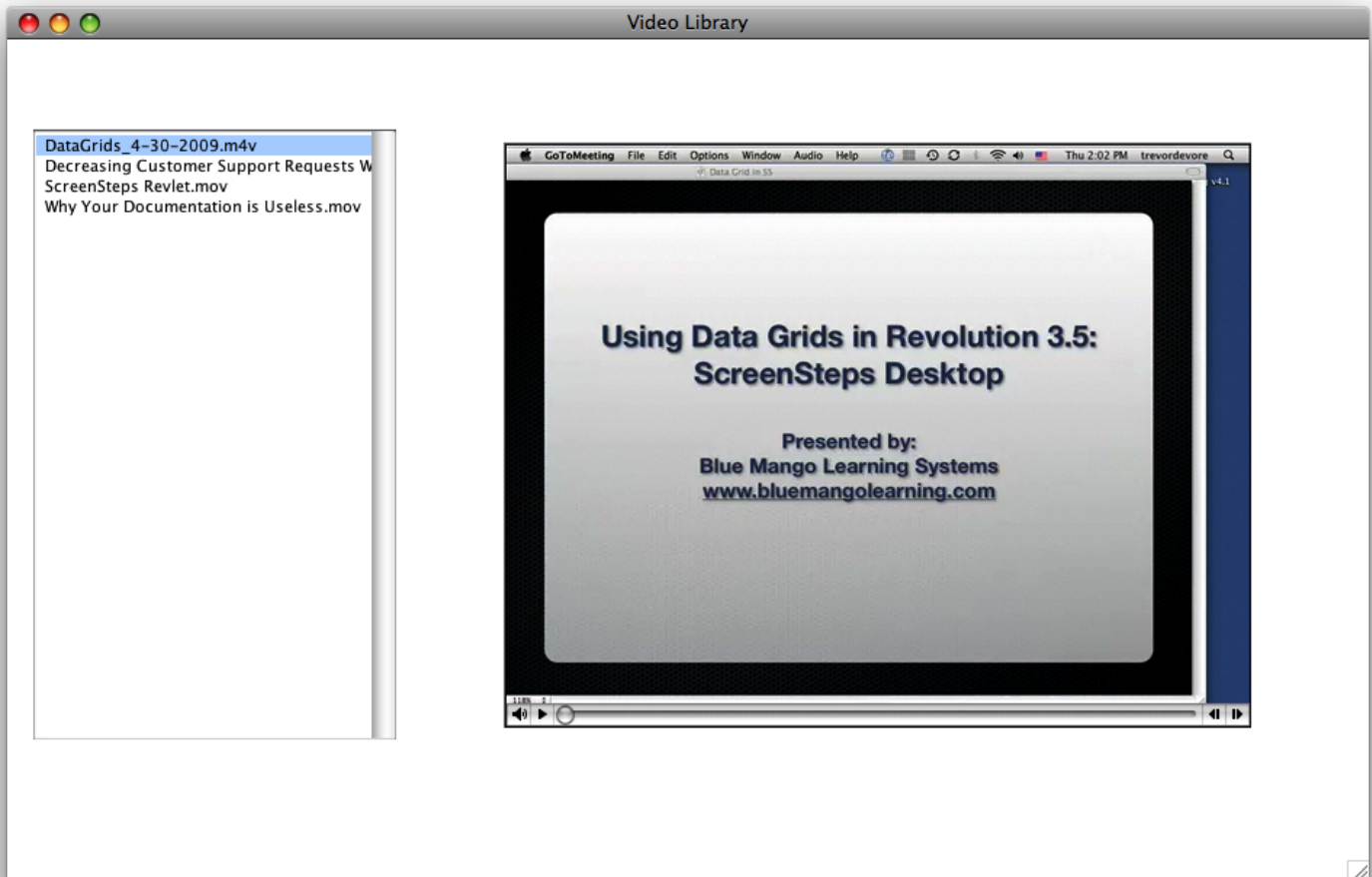
Playing Video in a Revolution Application

What You Will Create in This Chapter

In this chapter you will learn how to create a basic video player application. The application will:

- 1) List the videos files in a folder in a menu.
- 2) Allow you to choose a video file from the menu and play it.

End Product

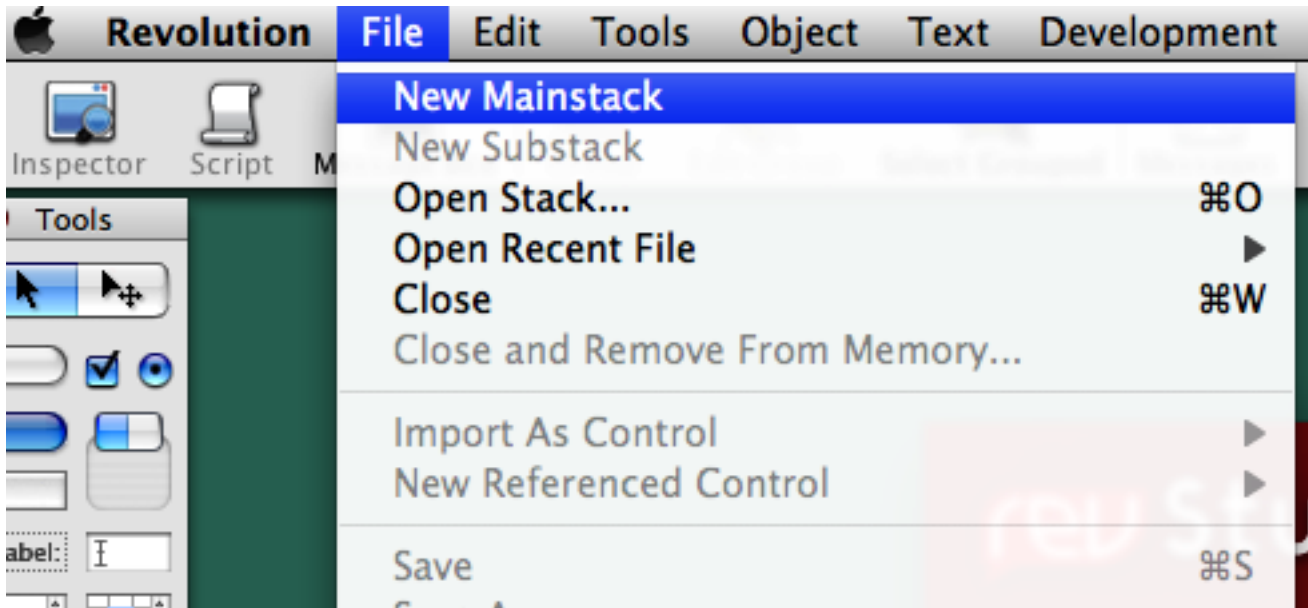


The end product will resemble this.

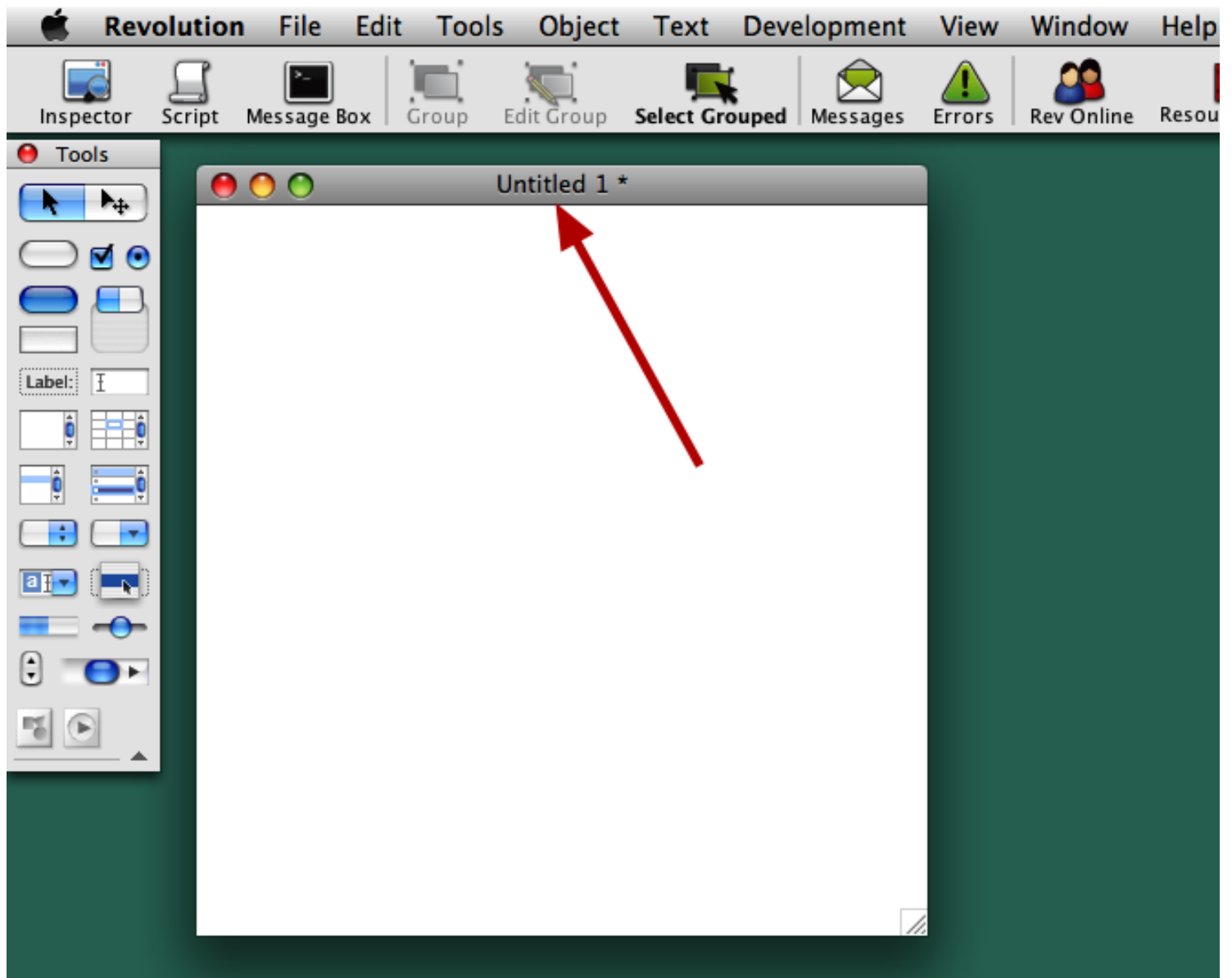
Create the Window Where Your Video Will be Displayed

To begin, you need to create the window that will display your menu and video player. In Revolution a window is referred to as a **stack**. Let's look at how to create a stack.

File > New Mainstack

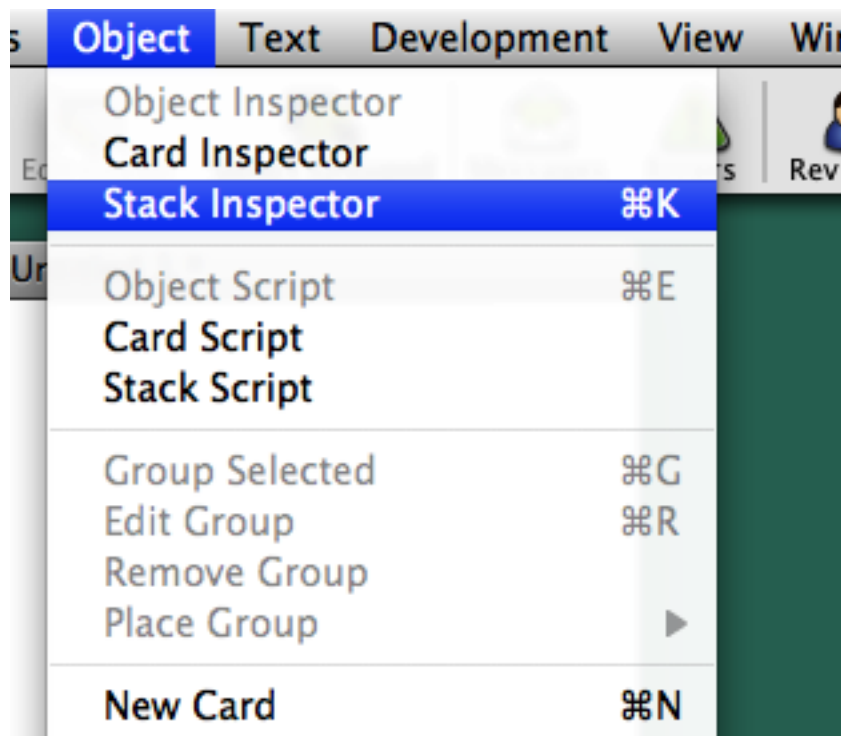


To create a new stack (or window), select **File > New Mainstack**.



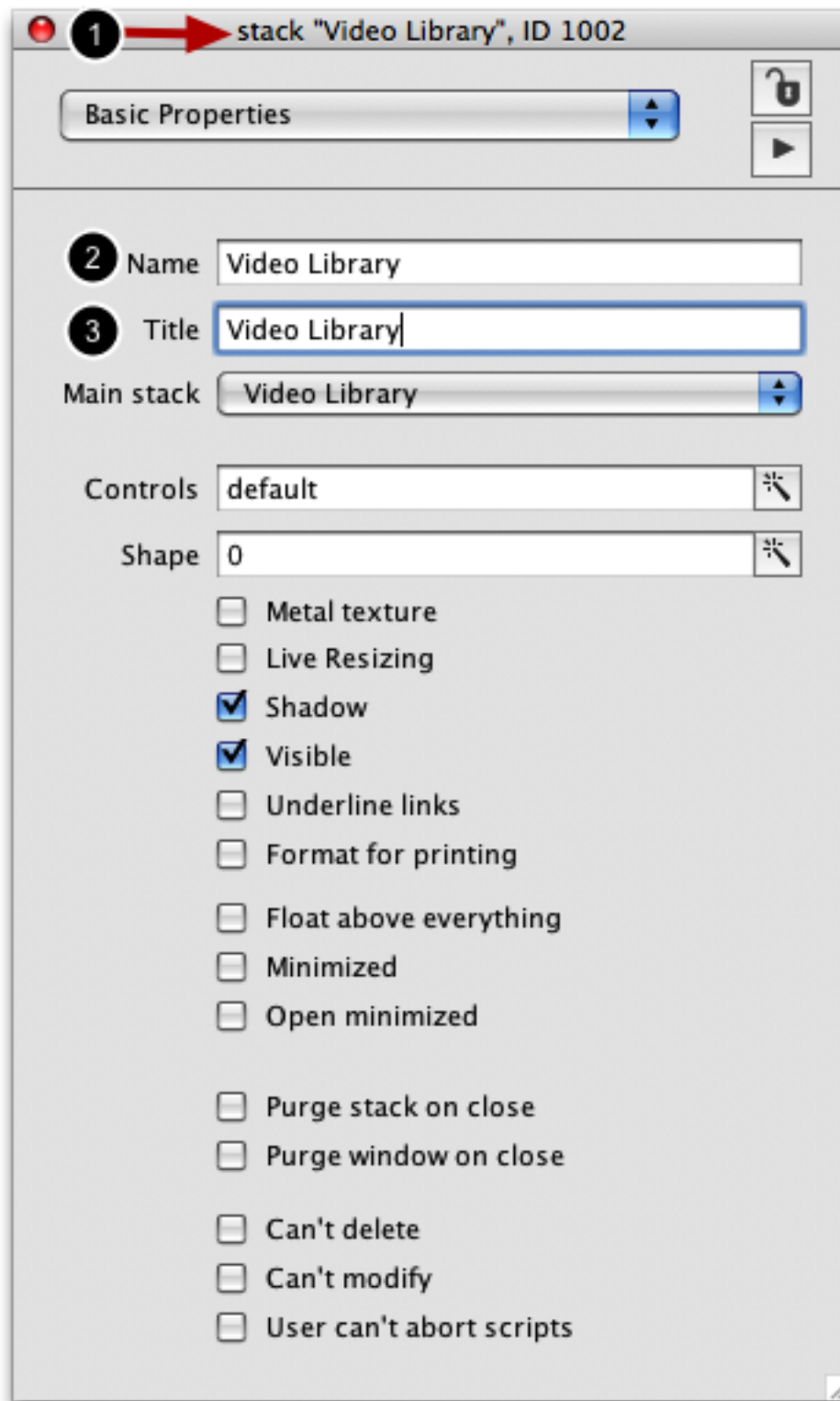
After selecting **File > New Mainstack** a new stack window should appear that says **Untitled 1** in the window title bar.

Give the Window a Name



Next you will give the stack window a name and label. You can do this using the **Object Inspector**. To open the Object Inspector for a stack choose **Object > Stack Inspector**.

The Object Inspector

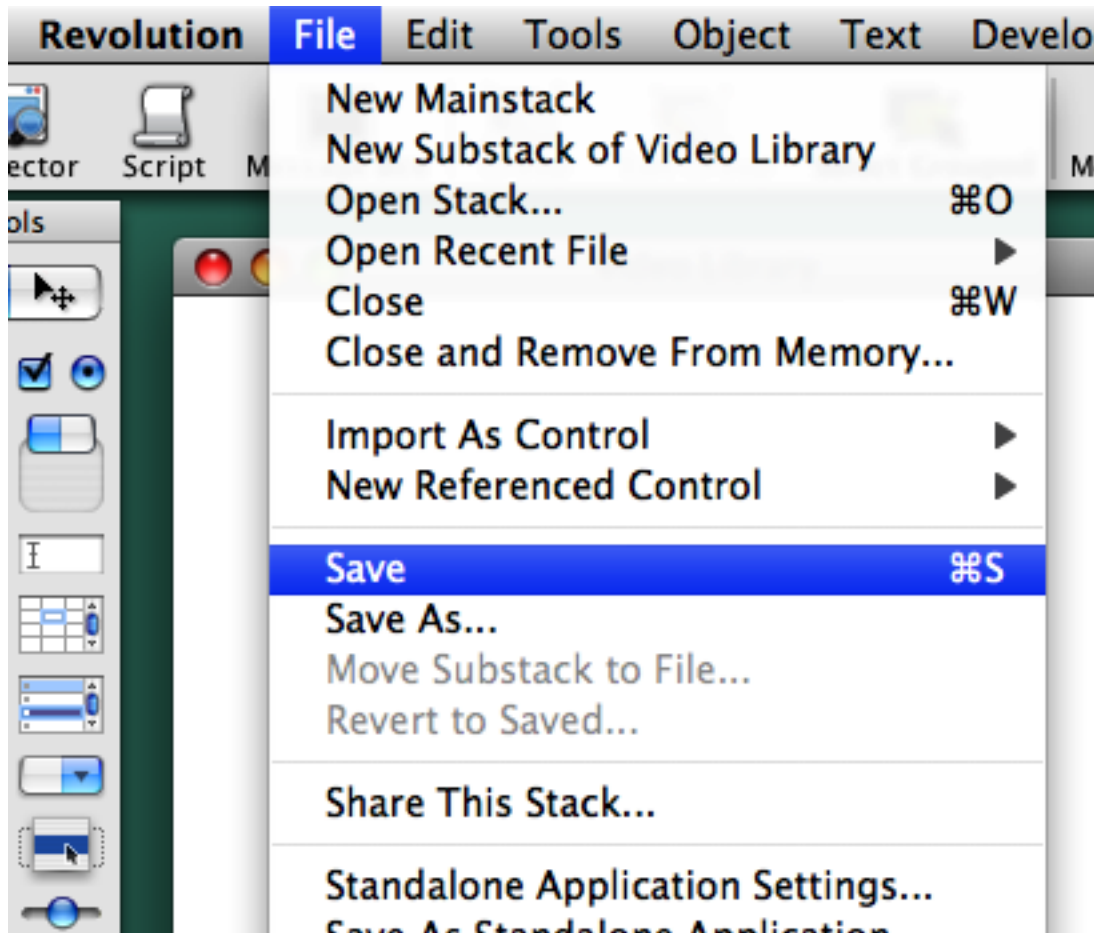


When the Object Inspector opens you should see the word "stack" in the title bar (1). Give the stack a name (2) and title (3).

The **name** of a stack becomes important when you start writing revTalk later on and you need to refer to the stack.

The **title** of a stack is what is displayed in the title bar of the stack window. It is okay if the name and title are the same right now but you might update the title property later on if you were to localize your application for other languages.

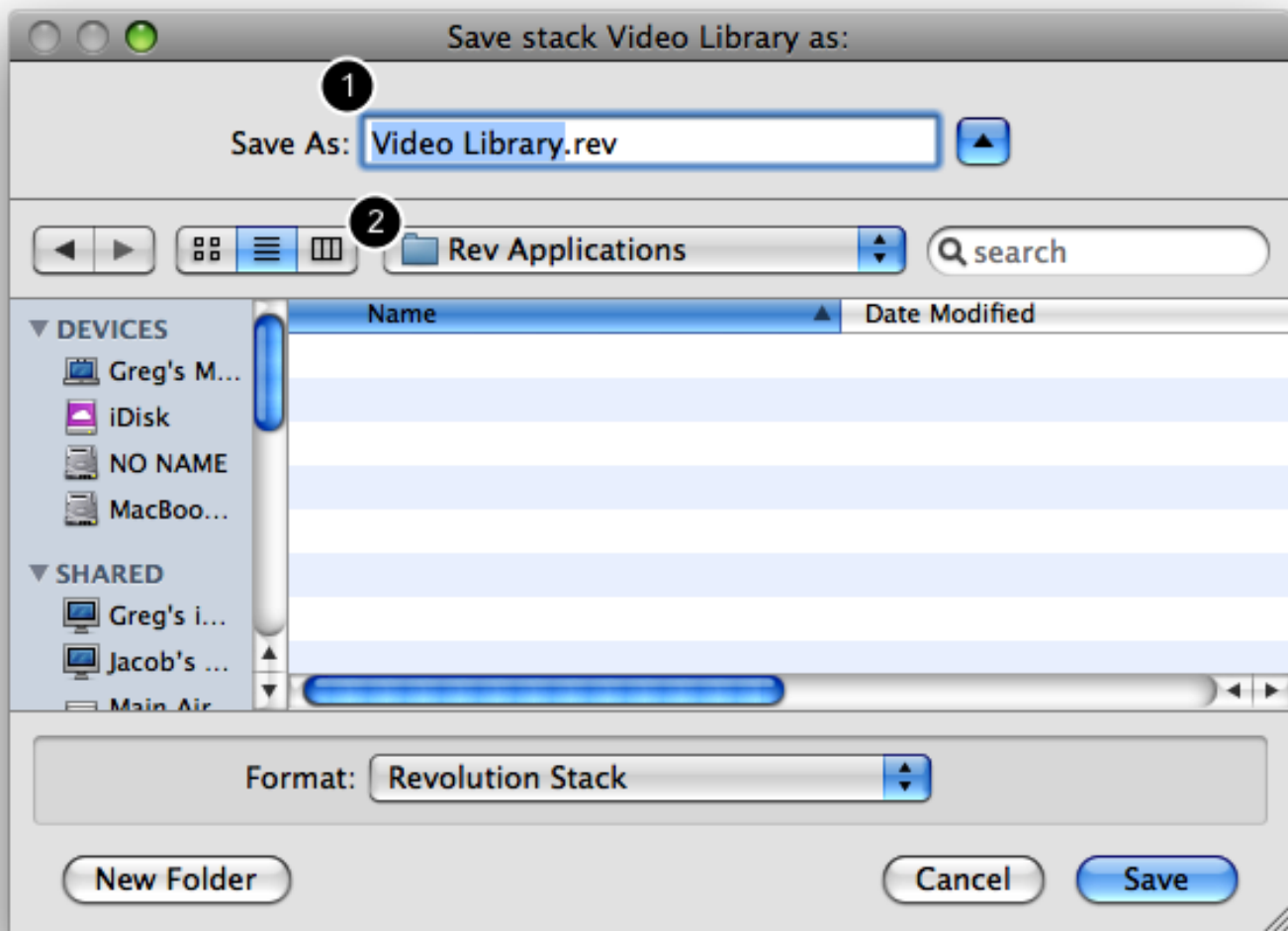
Save Your Work



You should now save your work. So far you have created a stack window in memory. That means that if you were to close Revolution the stack you created would be gone forever. What you need to do is save your stack window to a stack file. A stack file is how Revolution stores all your work on disk.

Before continuing and saving the stack file you should create a folder especially for the files related to this application.

Choose **File > Save** to display the save file dialog.

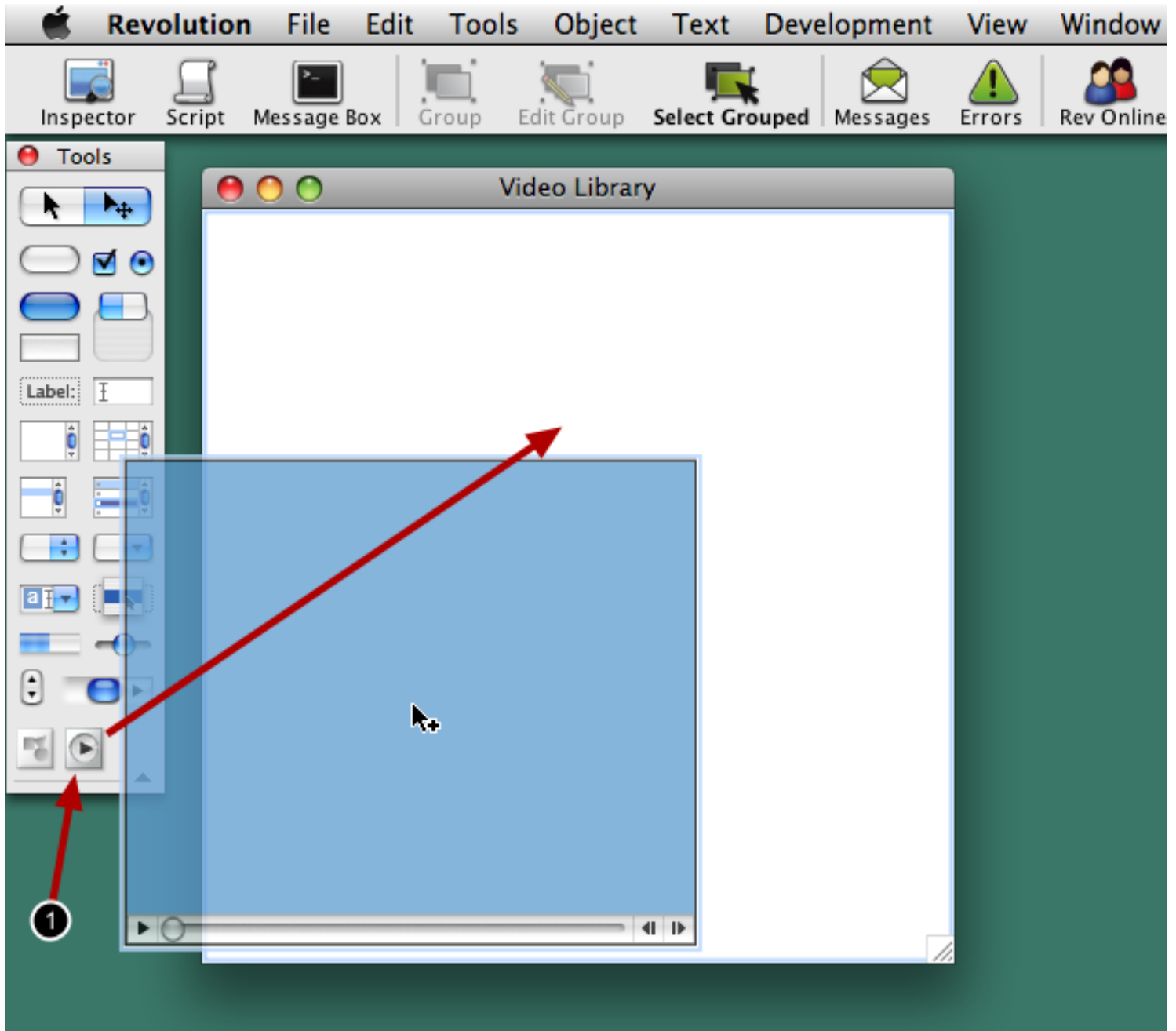


Name your stack file **Video Library.rev** (1) and save it in a folder that you have created especially for this application. The folder I'm using is **Rev Applications** (2).

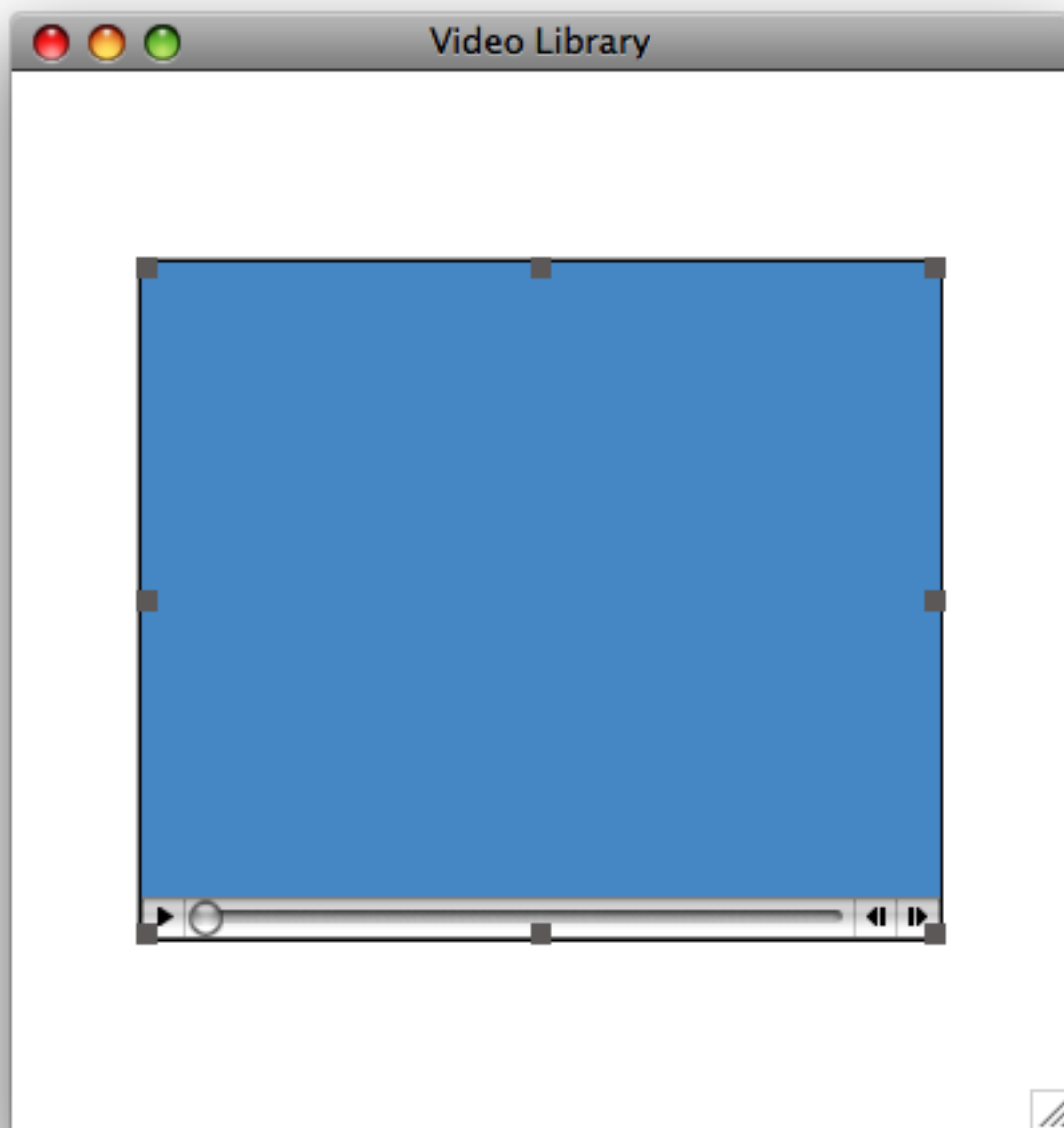
Add the Video Player to the Window

Now that you have created a stack window and saved your work it is time to start adding objects to your window. Let's start with adding an object that can play QuickTime videos. In Revolution the Player object can do this.

Add Player Object To Stack Window

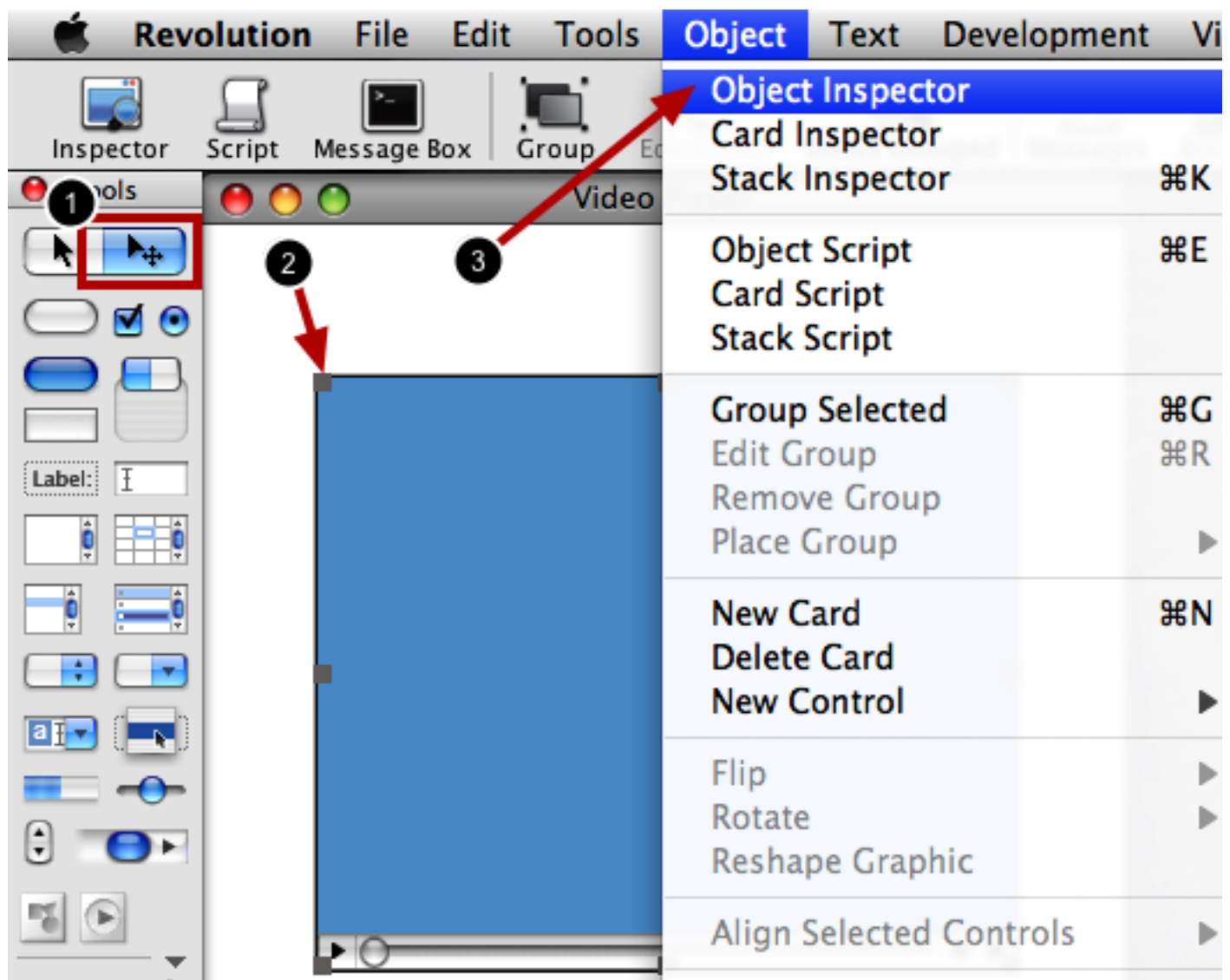


Locate the **Player** object icon in the **Tools** palette (1). Click on drag the Player object from the Tools palette onto your stack window.



After you drop the **Player** object onto your stack window your window should look something like this.

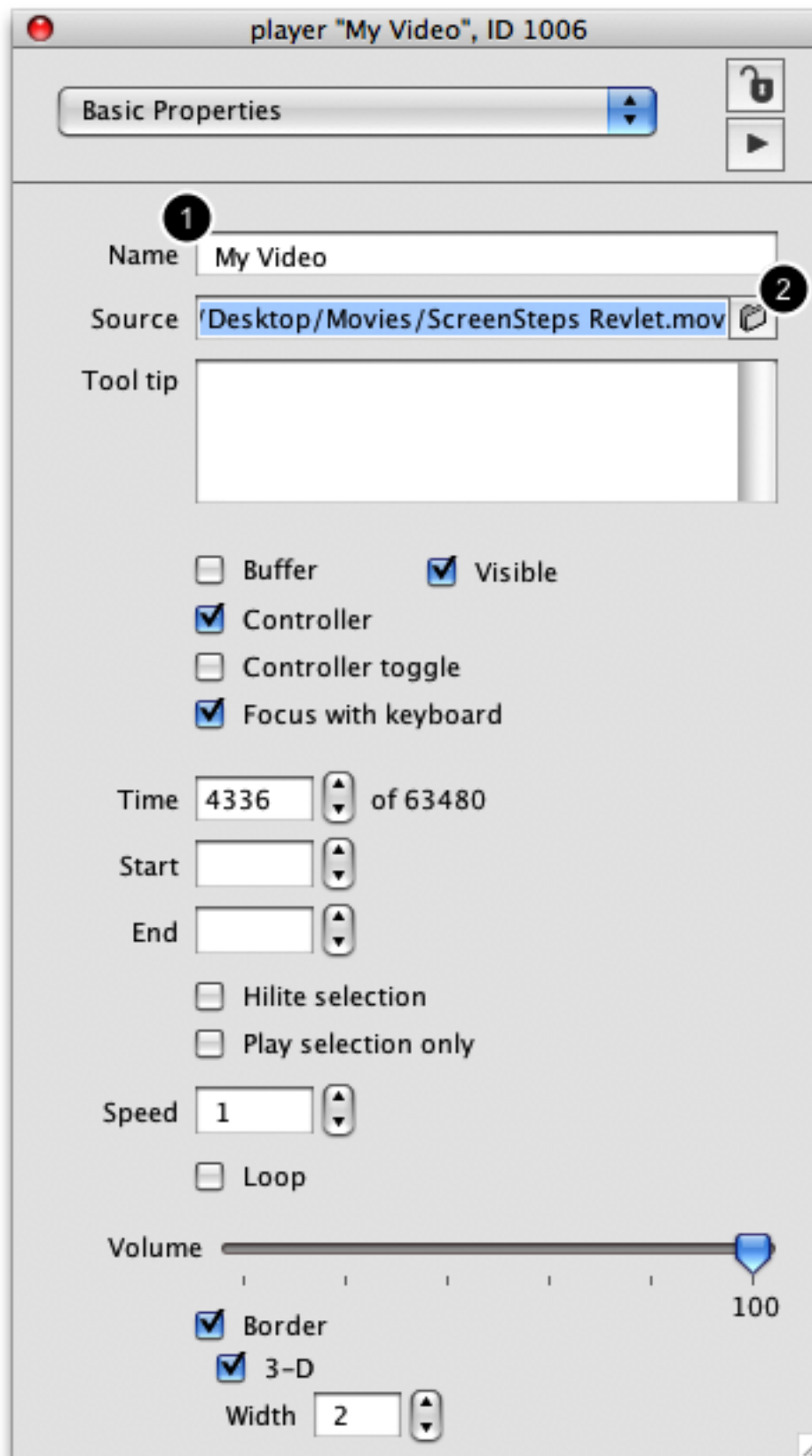
Edit Player Object Properties



Now you need to give the Player object a name and load a video to play. You can perform both of these actions from within the Object Inspector. To open the Object Inspector for the Player:

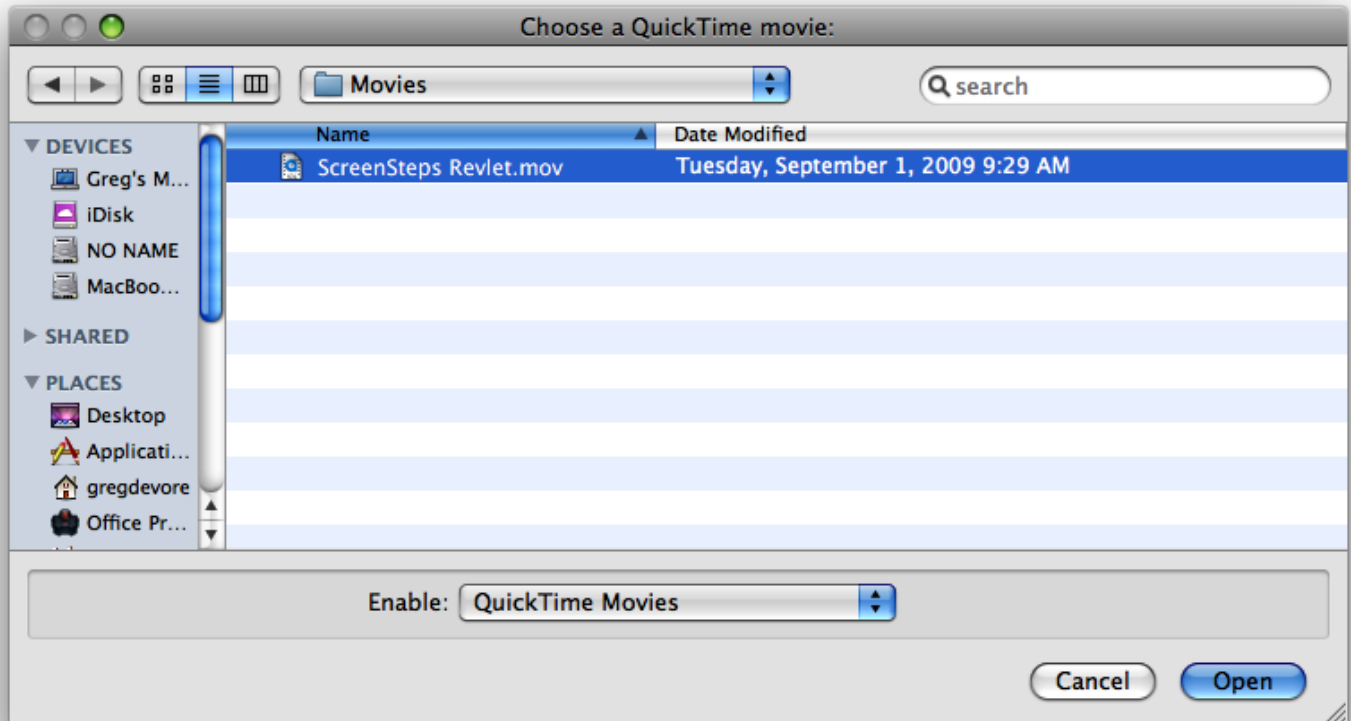
- 1) Make sure the **Edit** tool is selected in the **Tools** palette.
- 2) Select the Player (selection handles will appear around the Player)
- 3) Choose **Object > Inspector**

Set Name and Source



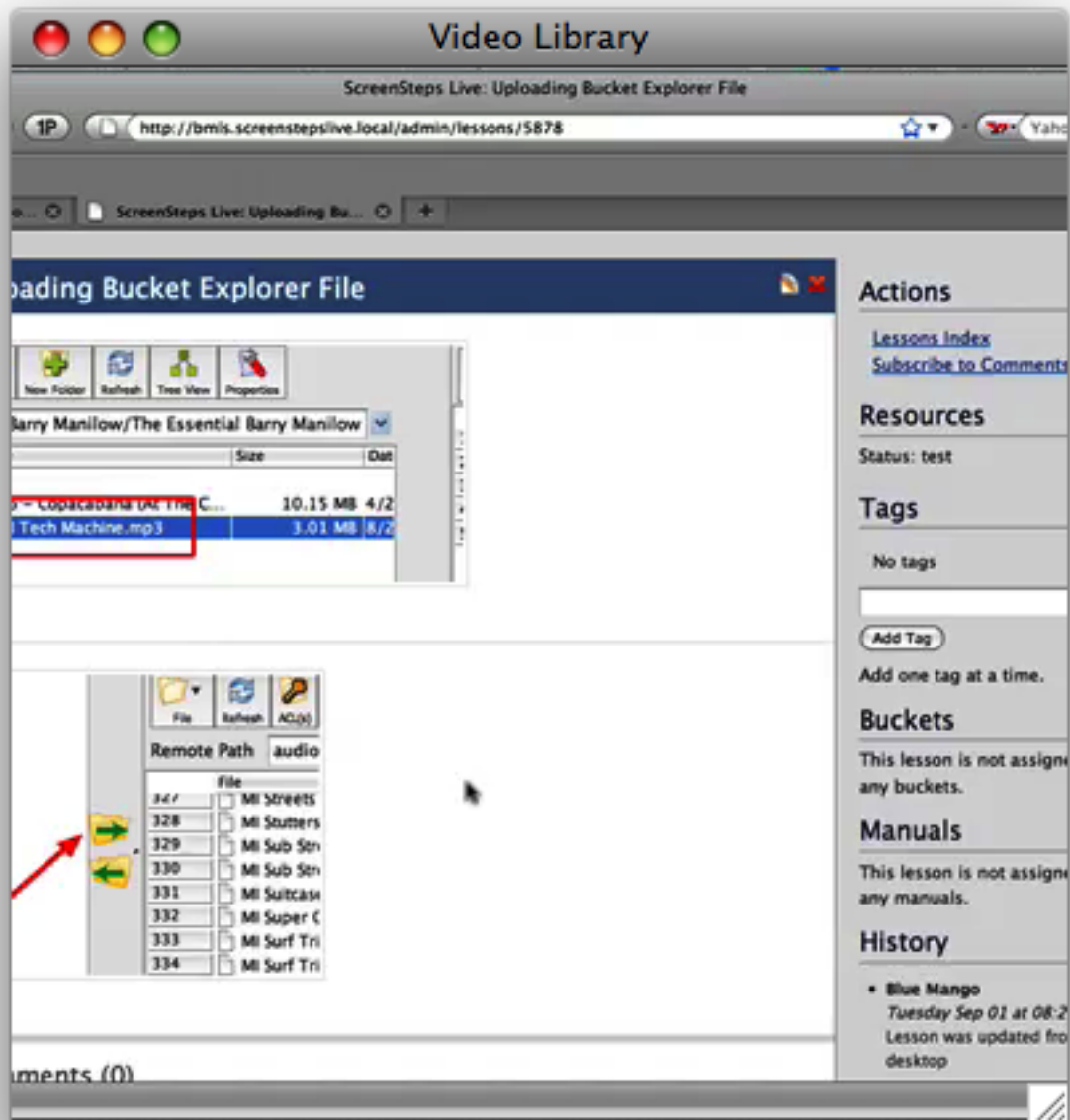
With the Object Inspector open name the Player object **My Video** (1). Next, click on the folder icon next to the **Source** field (2). This will display a file selection dialog allowing you to select a movie file.

Select Movie File



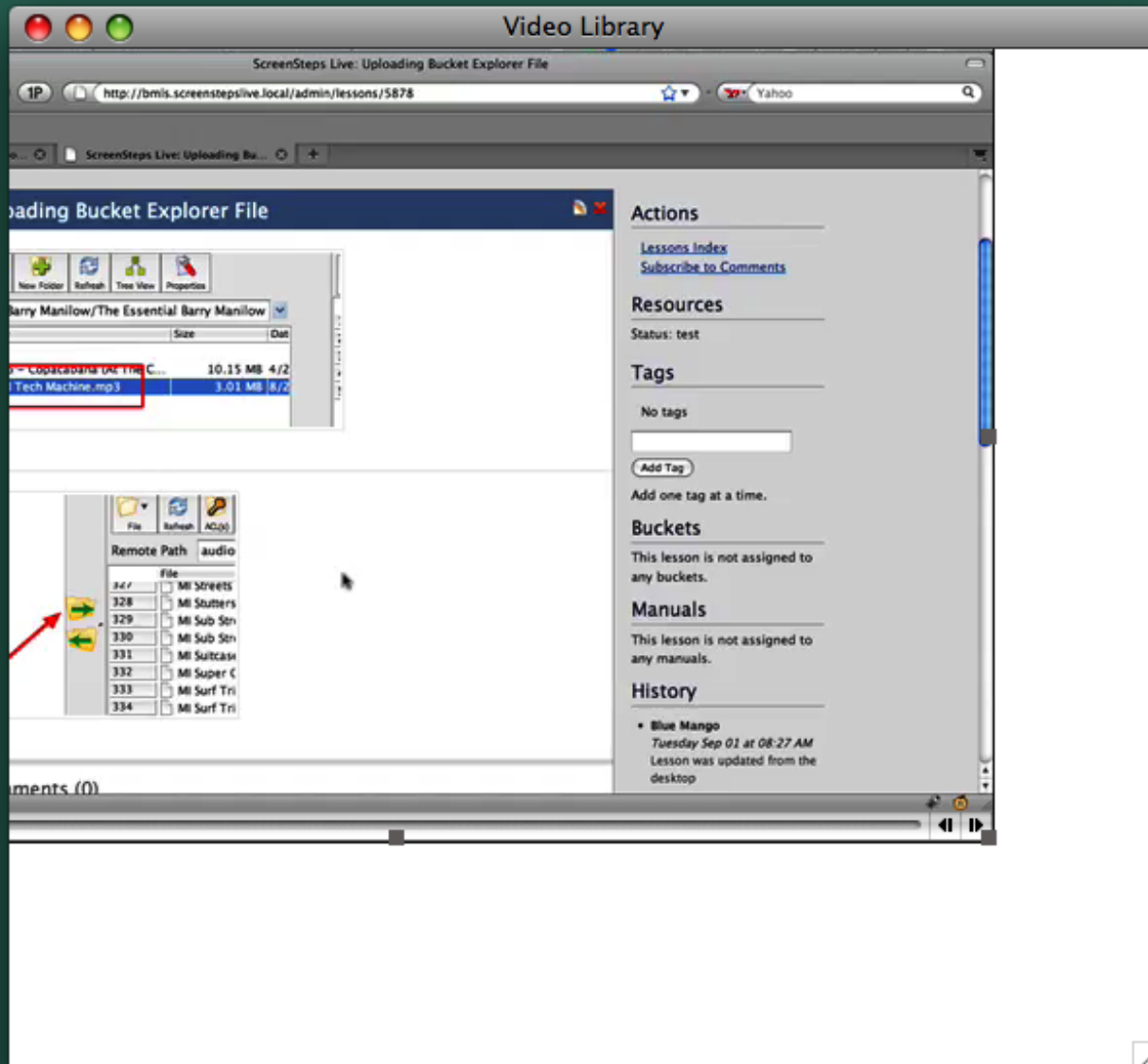
Locate a QuickTime compatible movie file on your computer and click **Open** to load the movie.

The Result



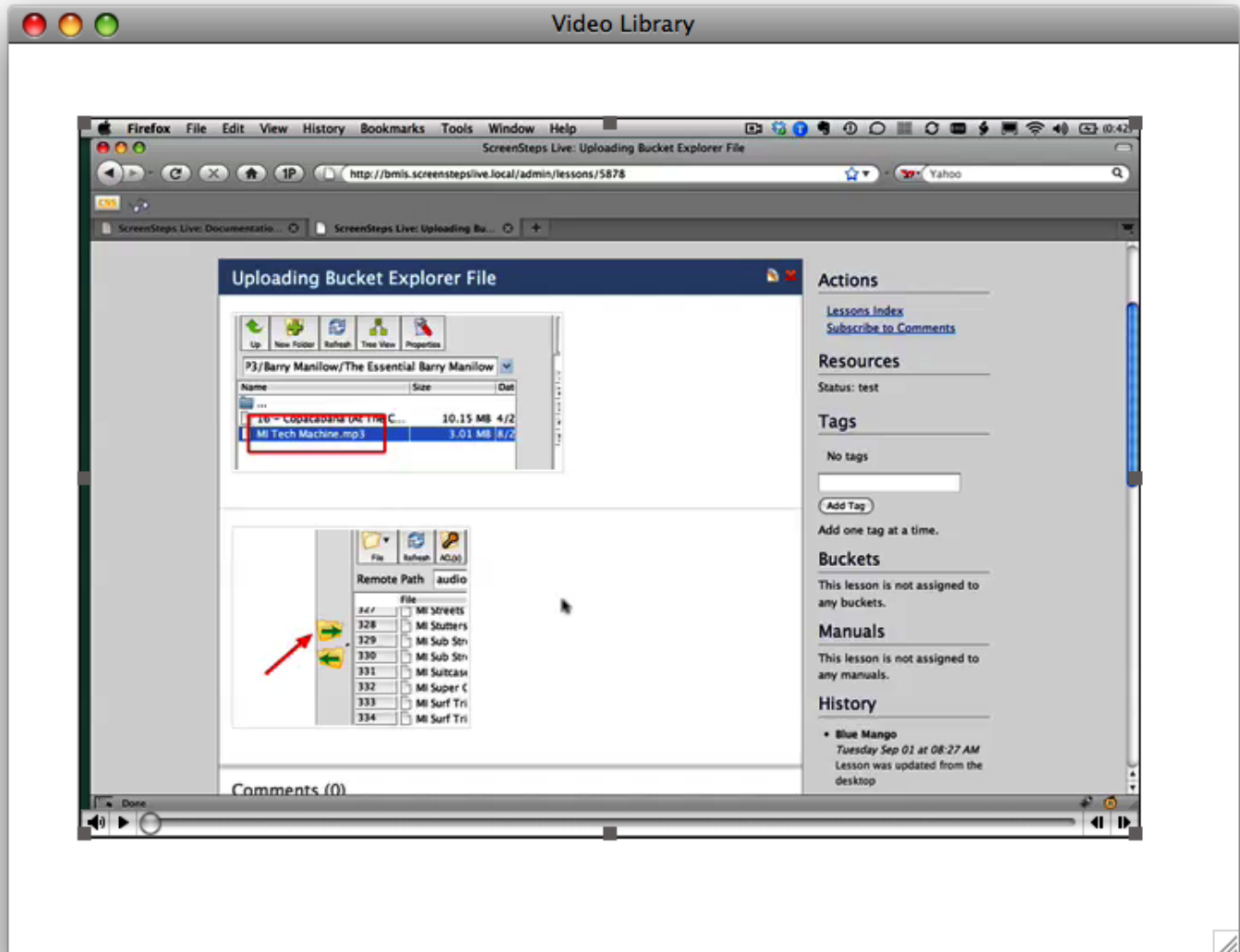
The movie file should load into the Player object. In this example the movie dimensions are larger than the stack window. By default Revolution resizes the Player object to the dimensions of the movie file automatically. This has caused the Player object to be larger than the stack window.

Resize Stack Window



Resize the stack window so that the Player object fits within it. You can resize the stack window like you would any other window on your operating system.

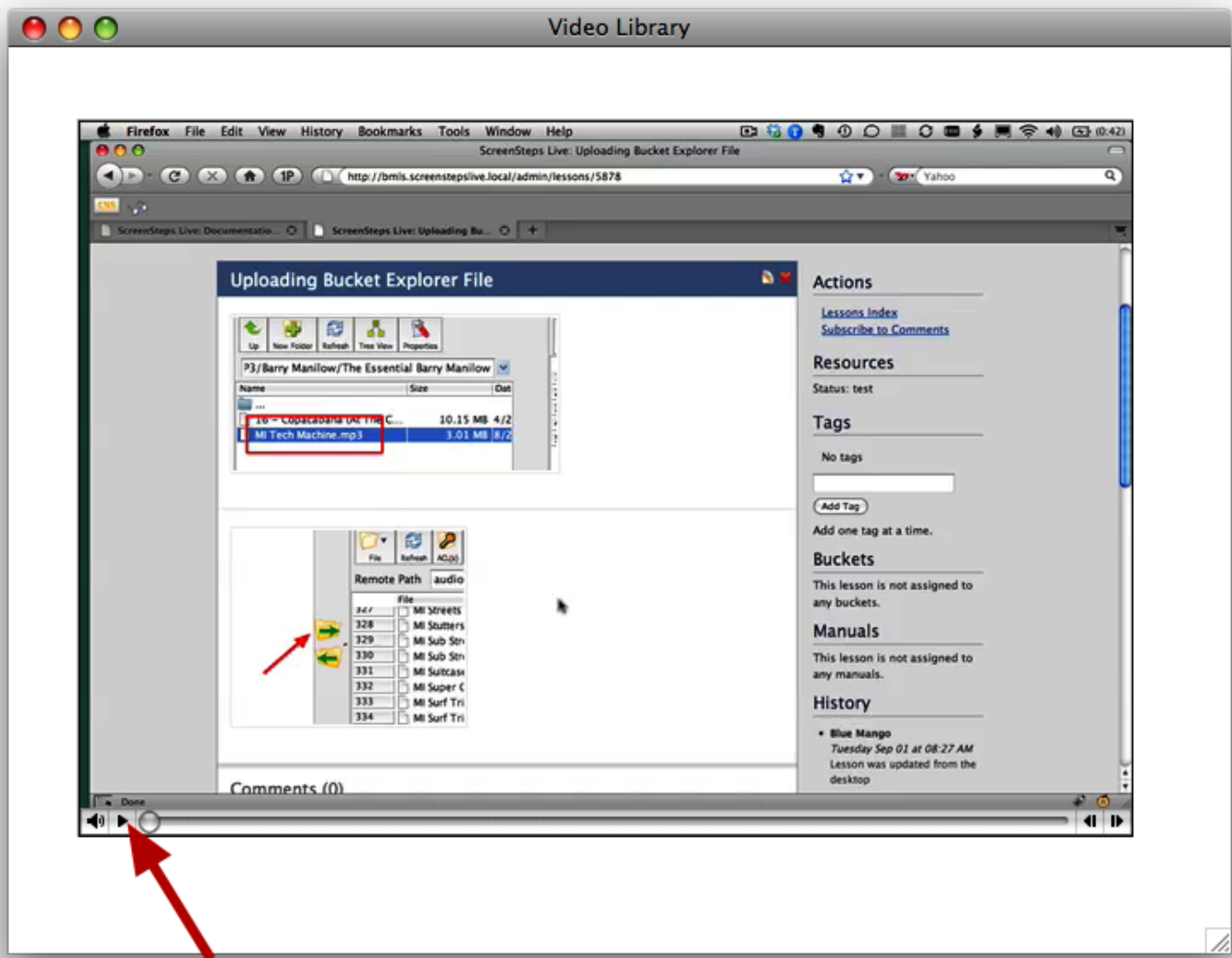
Center The Player Object



Now click on the Player object and drag it so that it appears centered within the stack window.



Now you can preview the movie. To play the movie select the **Browse** tool in the **Tools** palette.



After selecting the Browse tool you can click on the play button to play the video.

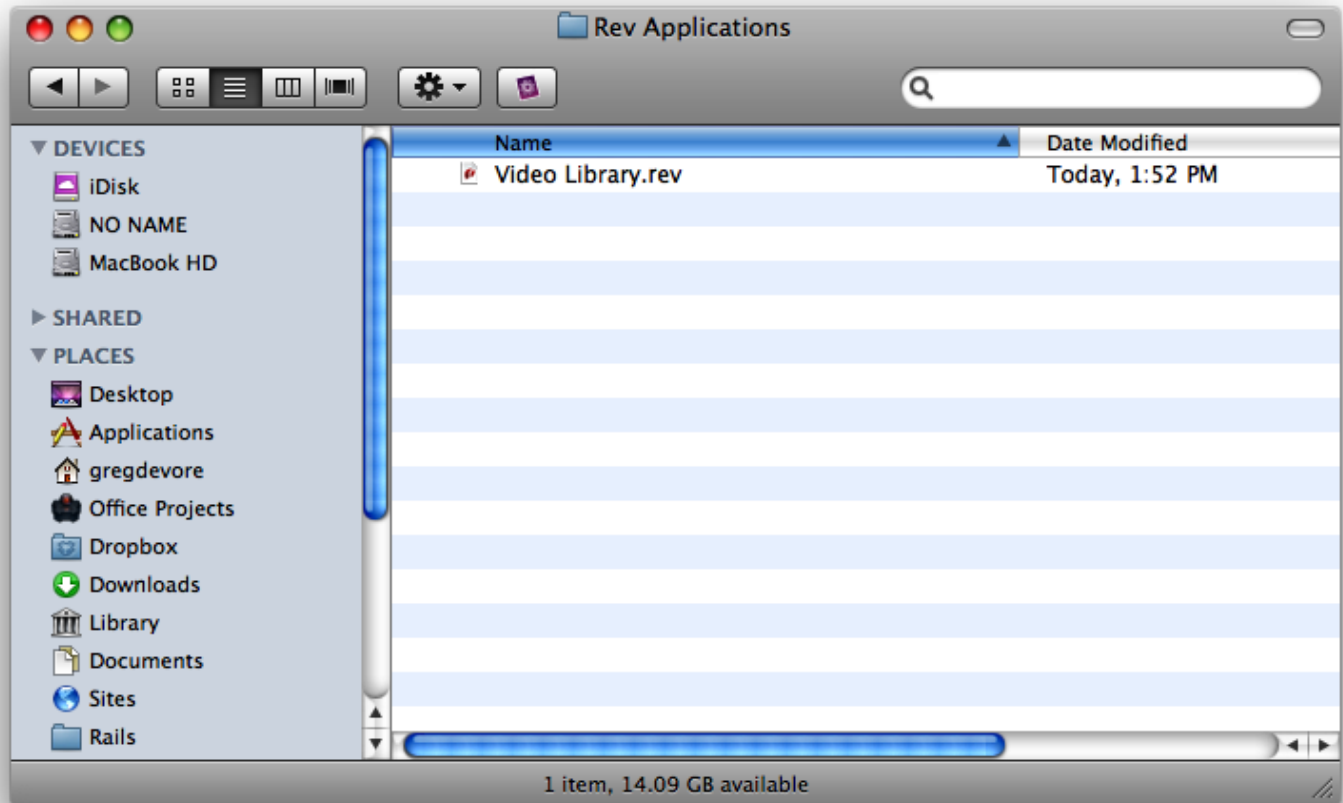
Next Steps

Now that you have created a Player object we will look at how to create a menu and dynamically load a movie based on the menu selection.

Organizing Your Video Files

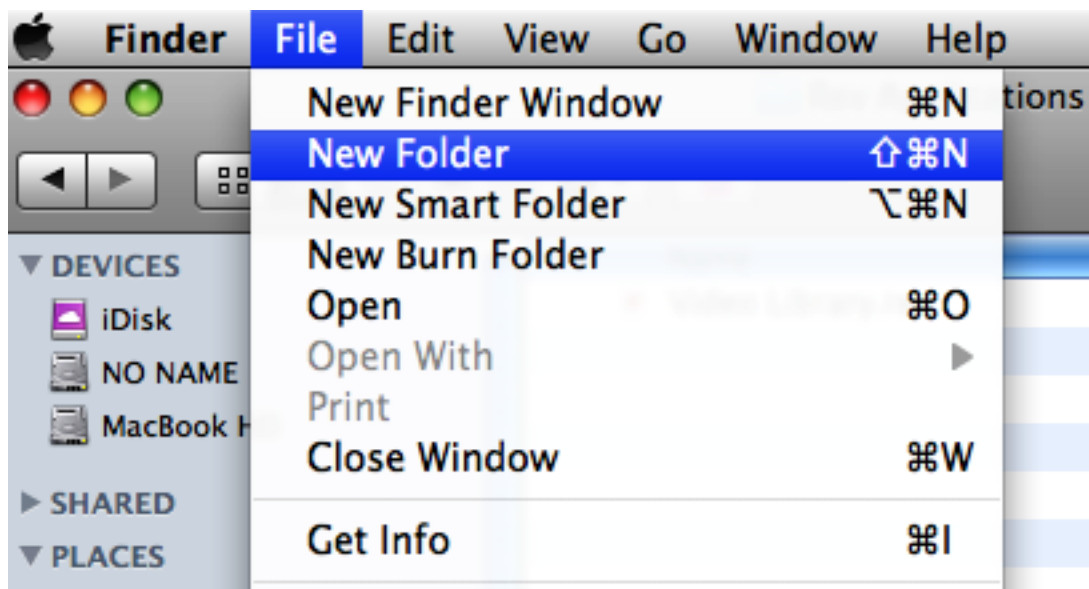
Before we build the menu you will need to organize your video files into a single folder that resides alongside your stack file.

Locate Stack File

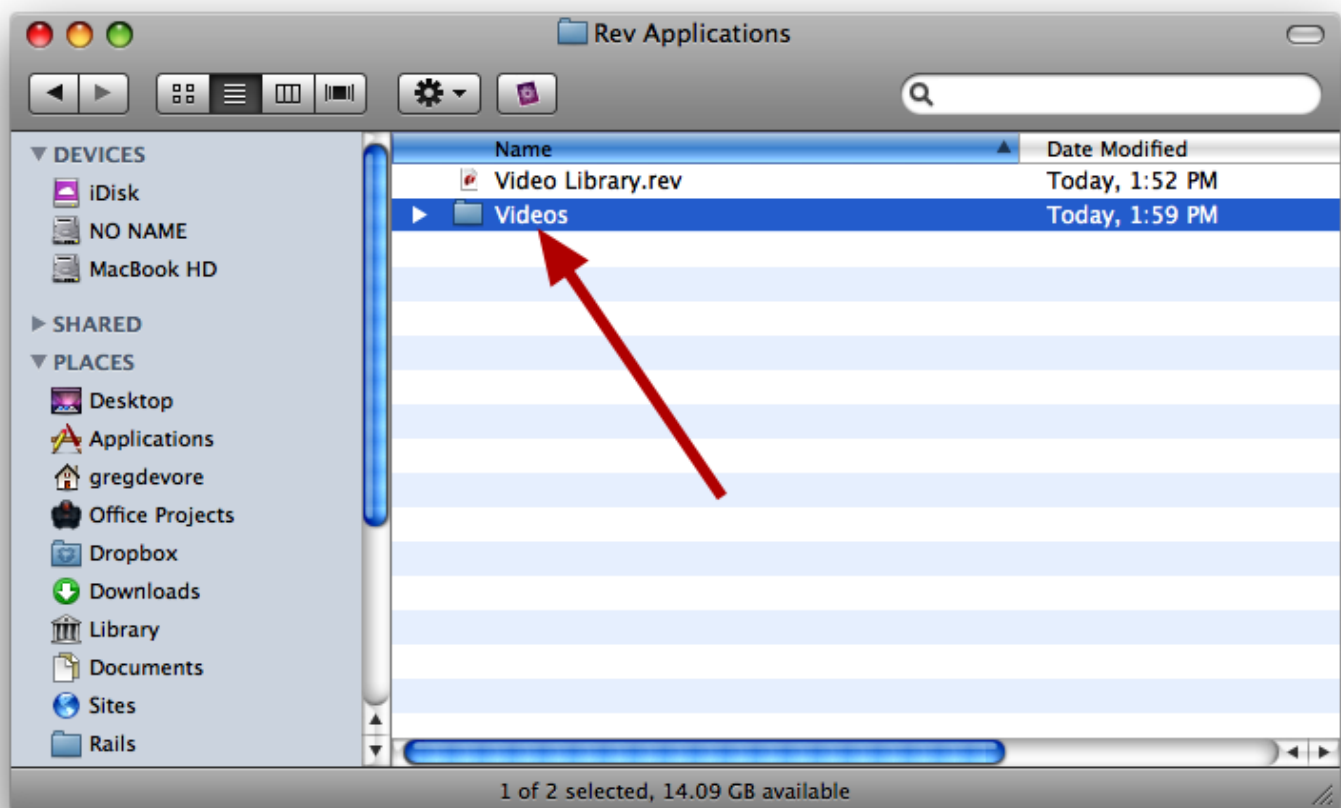


Locate the stack file that you saved early on.

Create New Folder

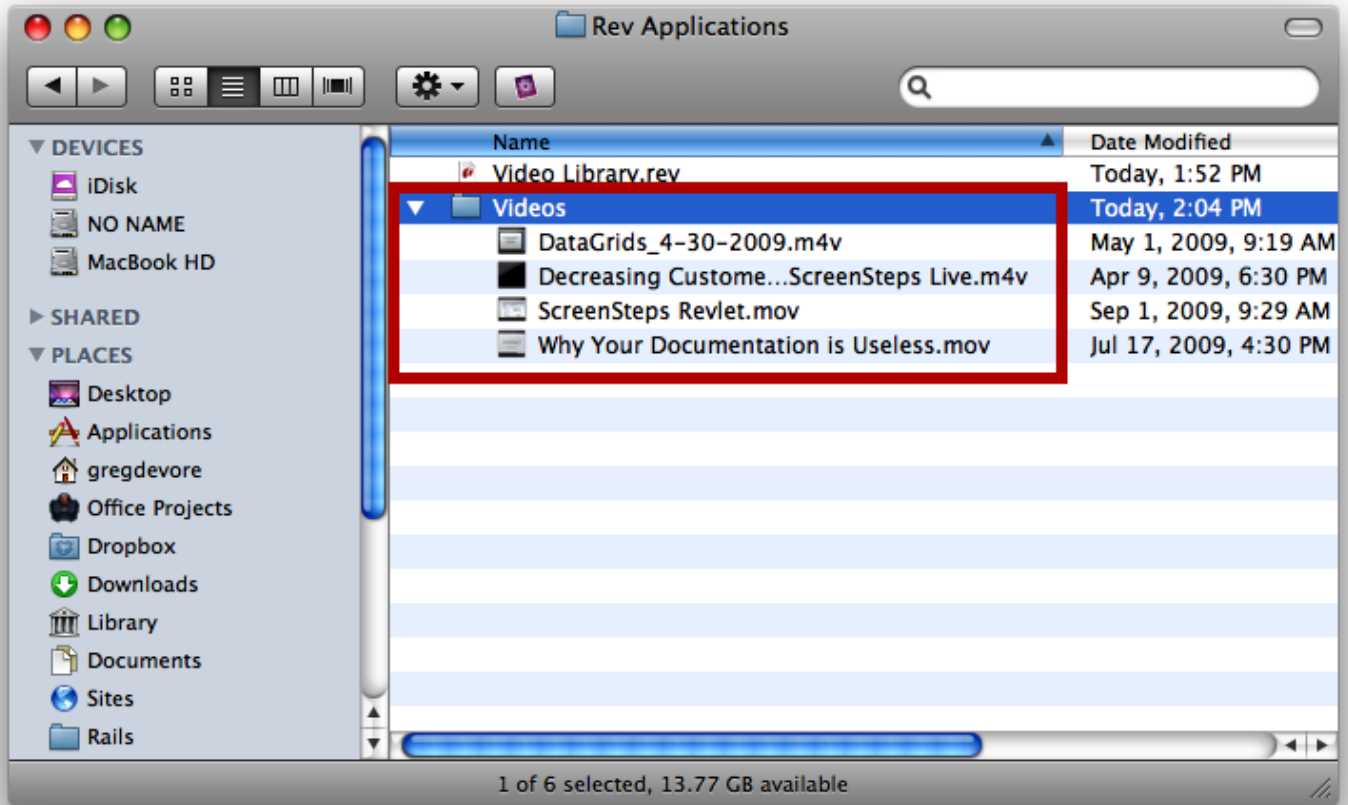


Create a new folder alongside the stack file.



Name the folder **Videos**.

Place Videos in Folder



Place any videos you want to make available in your application inside this folder.

Add the Video Files Menu

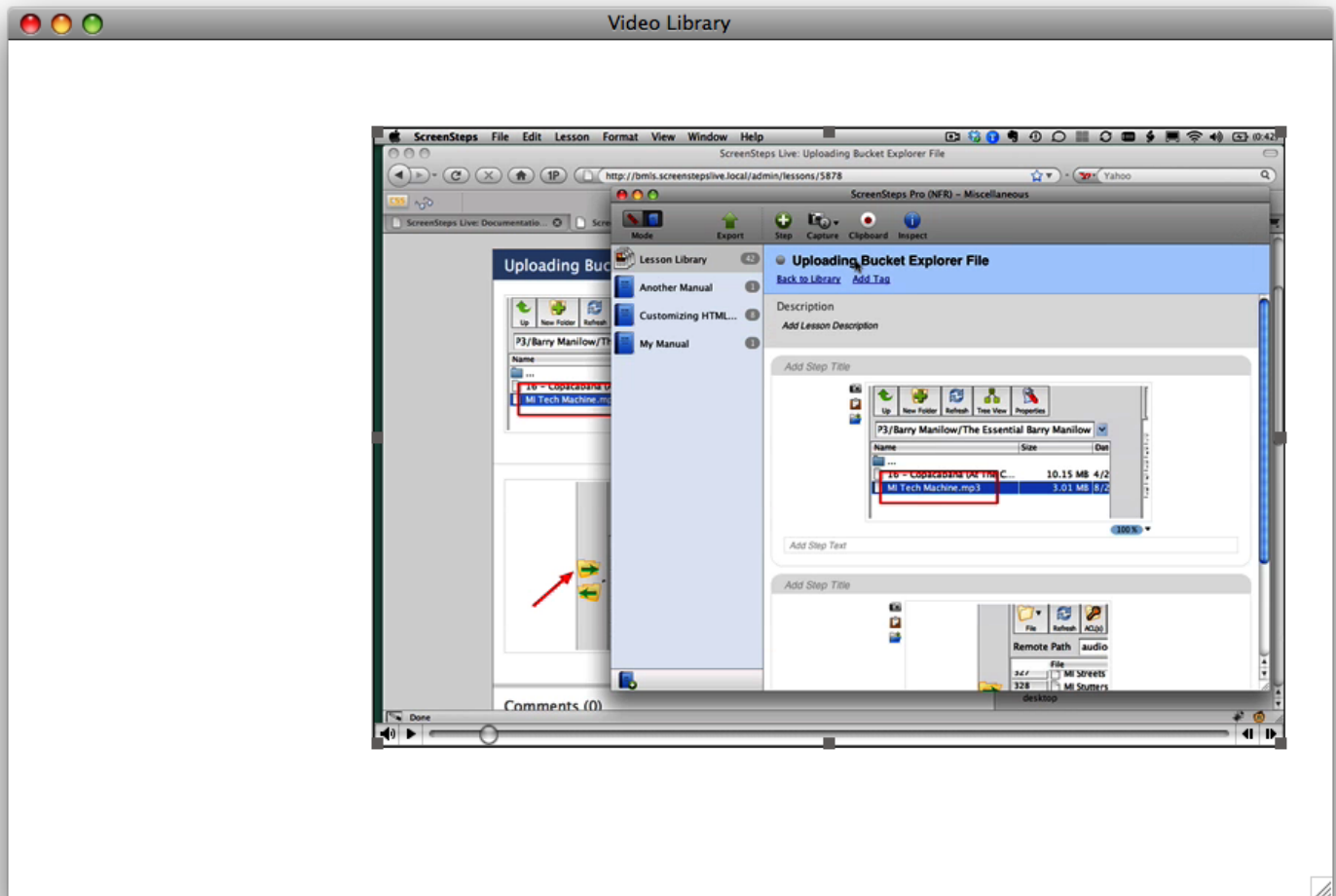
Now we will add a menu to the application that allows you to select the video file to play.

Activate Edit Tool



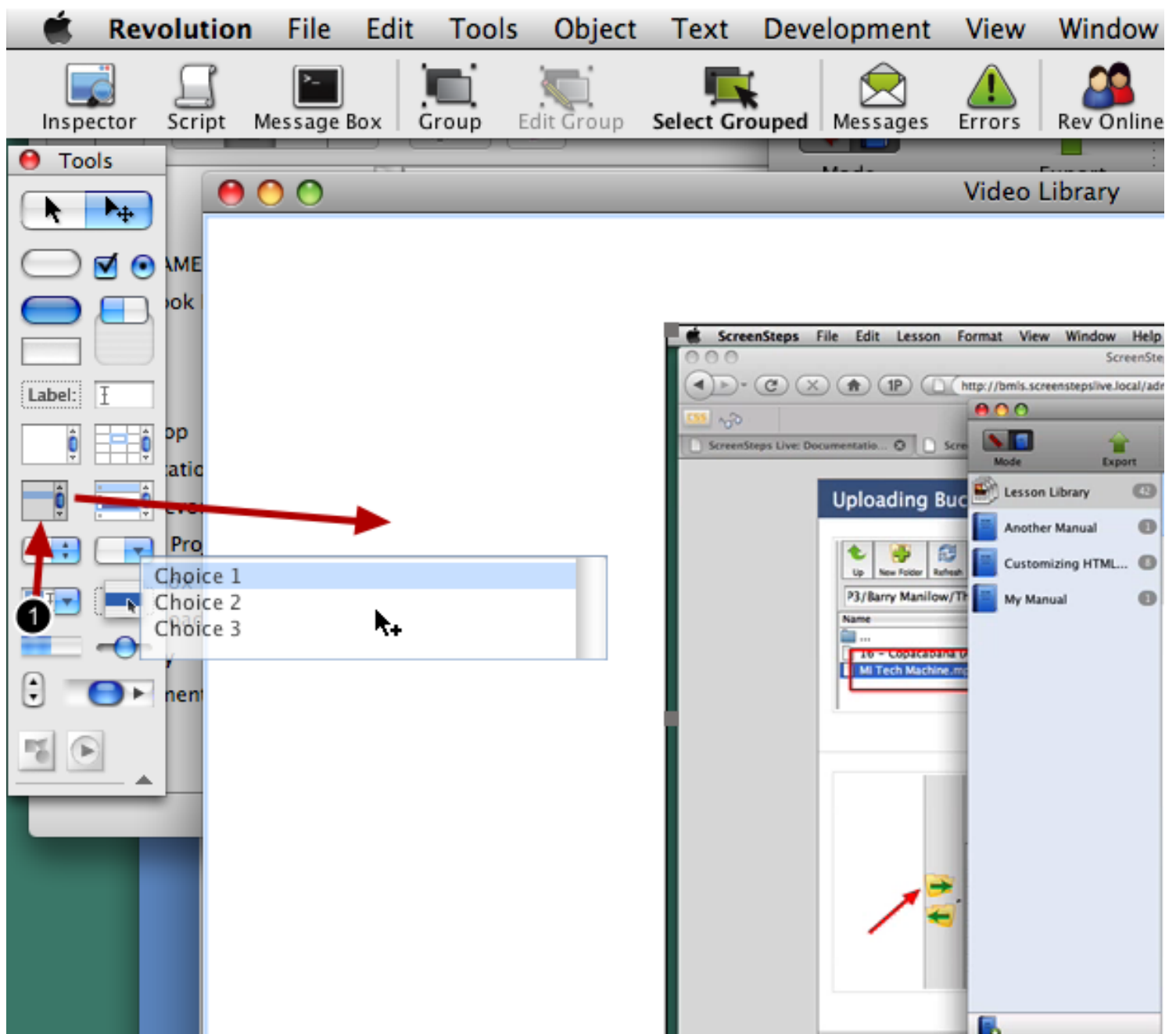
Click on the **Edit** tool in the **Tools** palette.

Reposition Player Object



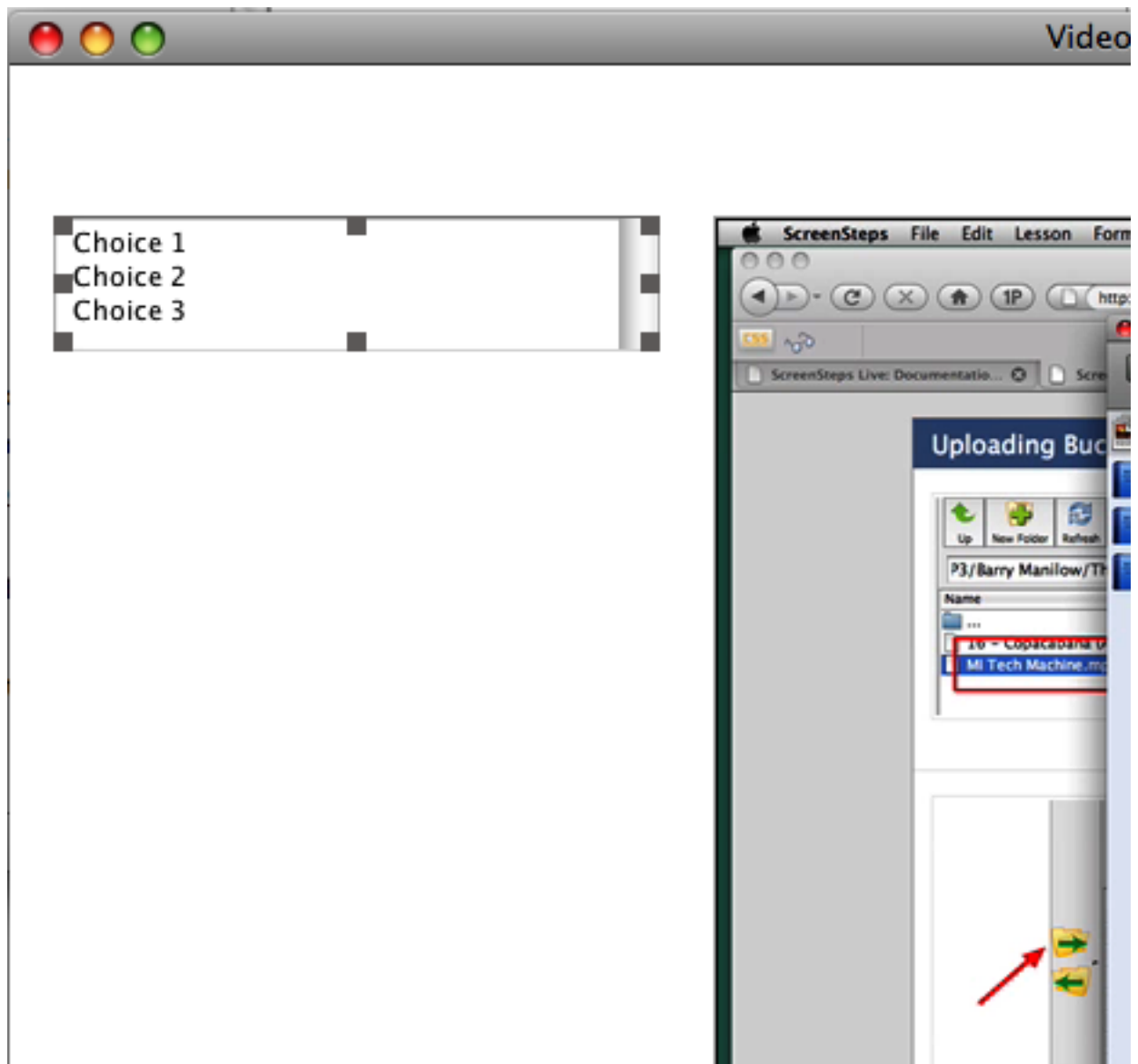
Click on the Player object and reposition it so that there is room for a menu on the left side of the window. Make the window larger if you need to.

Add List Field To Window



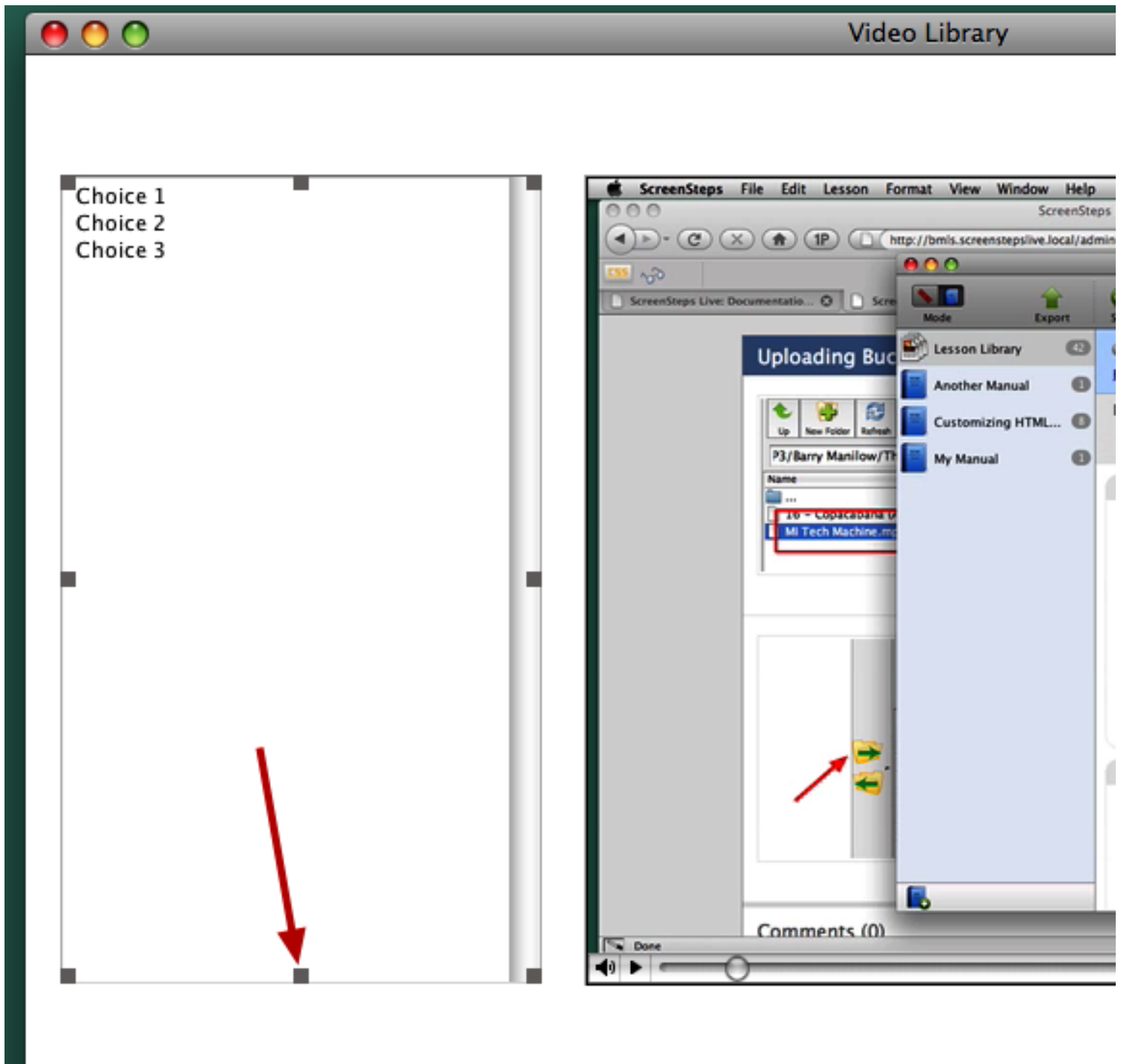
To make our menu we will use a **List Field**. A List Field displays multiple lines of text and allows the user to click on individual lines.

Locate the List Field in the **Tools** palette (1) and drag it onto the window.



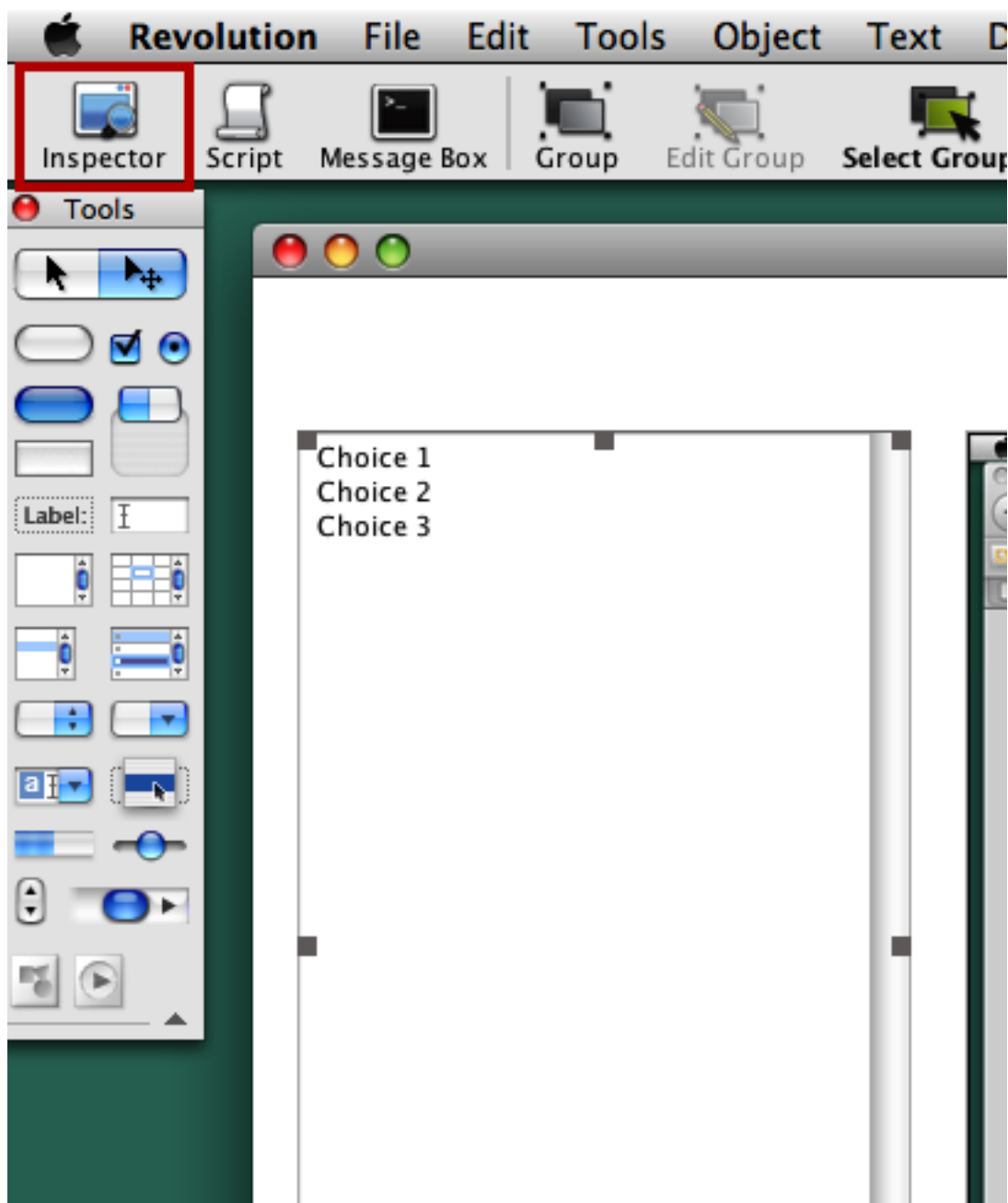
A new List Field will appear in the window.

Resize the List Field



Use the selection handles to resize the List Field.

Edit List Field Properties



With the **List Field** selected click on the **Inspector** button to open the Object Inspector.

Name the List Field

field "Scrolling List Field", ID 1008

Basic Properties

Name

Tooltip

☐ Disabled ☒ Focusable

☒ Visible ☒ Focus border

☒ Share text ☐ Tab on Return

☒ Don't wrap ☒ Three dimensional

☒ Lock text ☒ Show border

☒ Opaque Border width

Scrollbars: ☐ H ☒ V

Hilites: ☒ autoHilite

☒ List behavior

☐ Multi-line

☐ Non-contiguous

☐ Click to toggle

☒ Fixed line height

Text height

First indent

☐ Find command ignores

Behavior

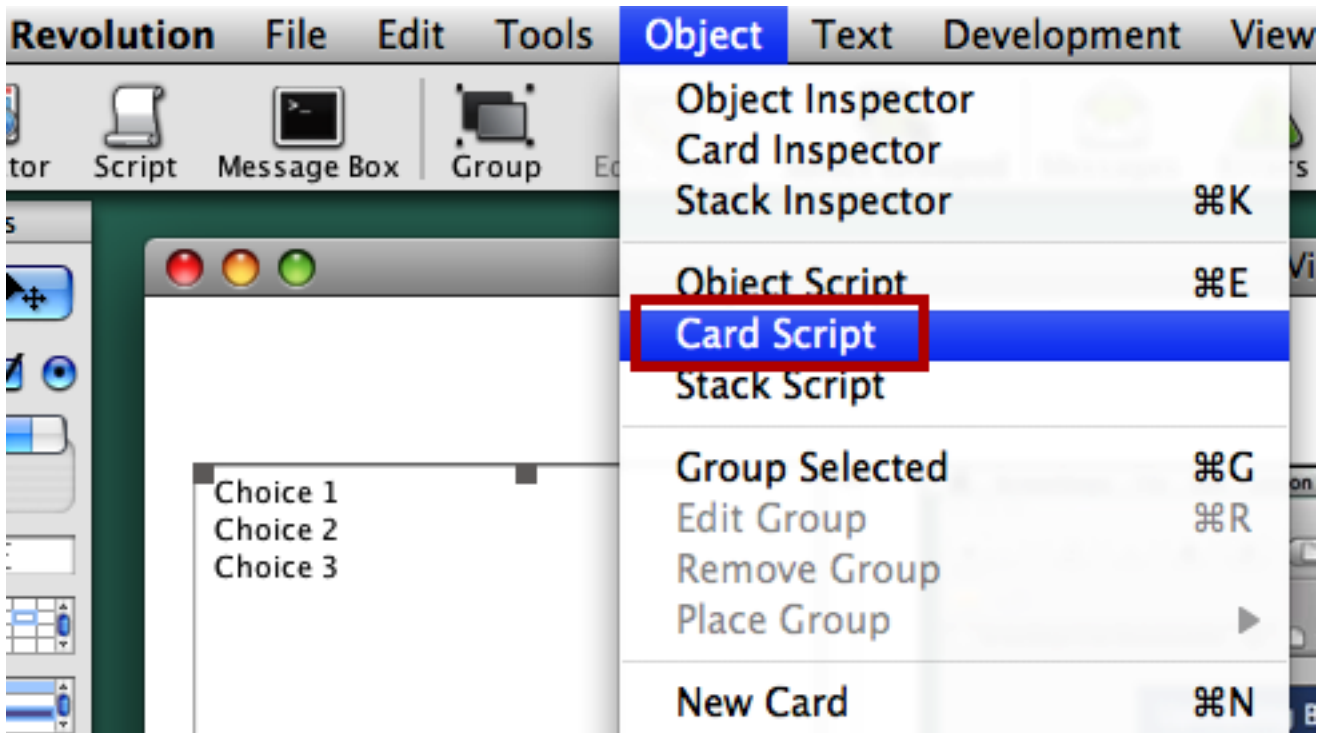
Name the List Field **Video Menu**.

Populating the Video Files Menu

Now that you have created a menu (the list field) to display your videos we will look at how to populate the menu so that it displays a list of available videos.

This lesson will introduce revTalk for the first time in the application. revTalk is the language you use in Revolution to control what the application does and how it responds to user actions.

Edit The Card Script

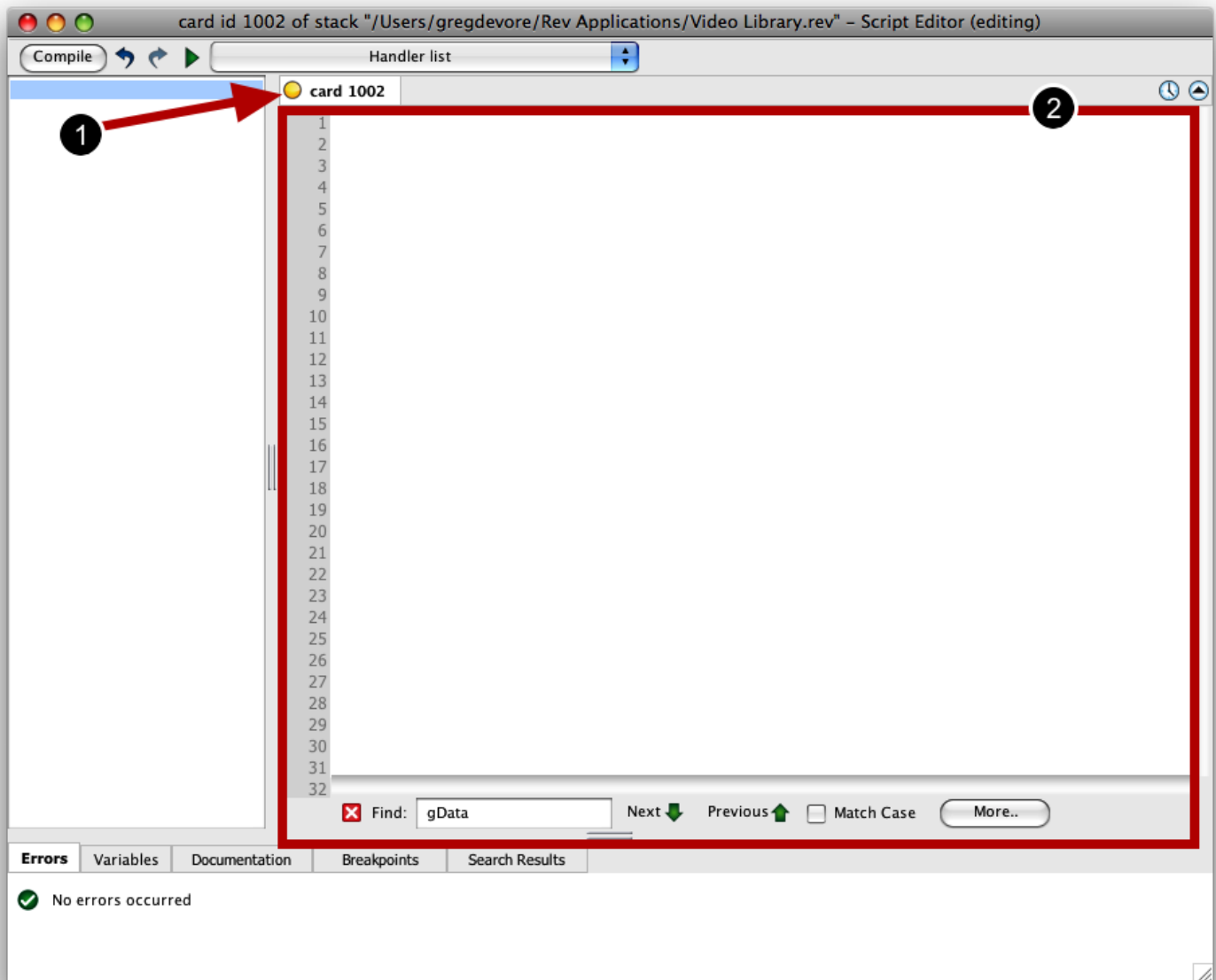


So far we have discussed the **Stack** window, the **Player** and the **List Field**. Now we will introduce the **Card** object. A Stack window is made up of one or more Cards. A Card can contain multiple objects such as a Player or Field.

We are going to add the revTalk necessary to populate the List Field to the **Card Script**. The Card Script stores revTalk that can affect any object on the card.

To edit the Card Script choose **Object > Card Script**.

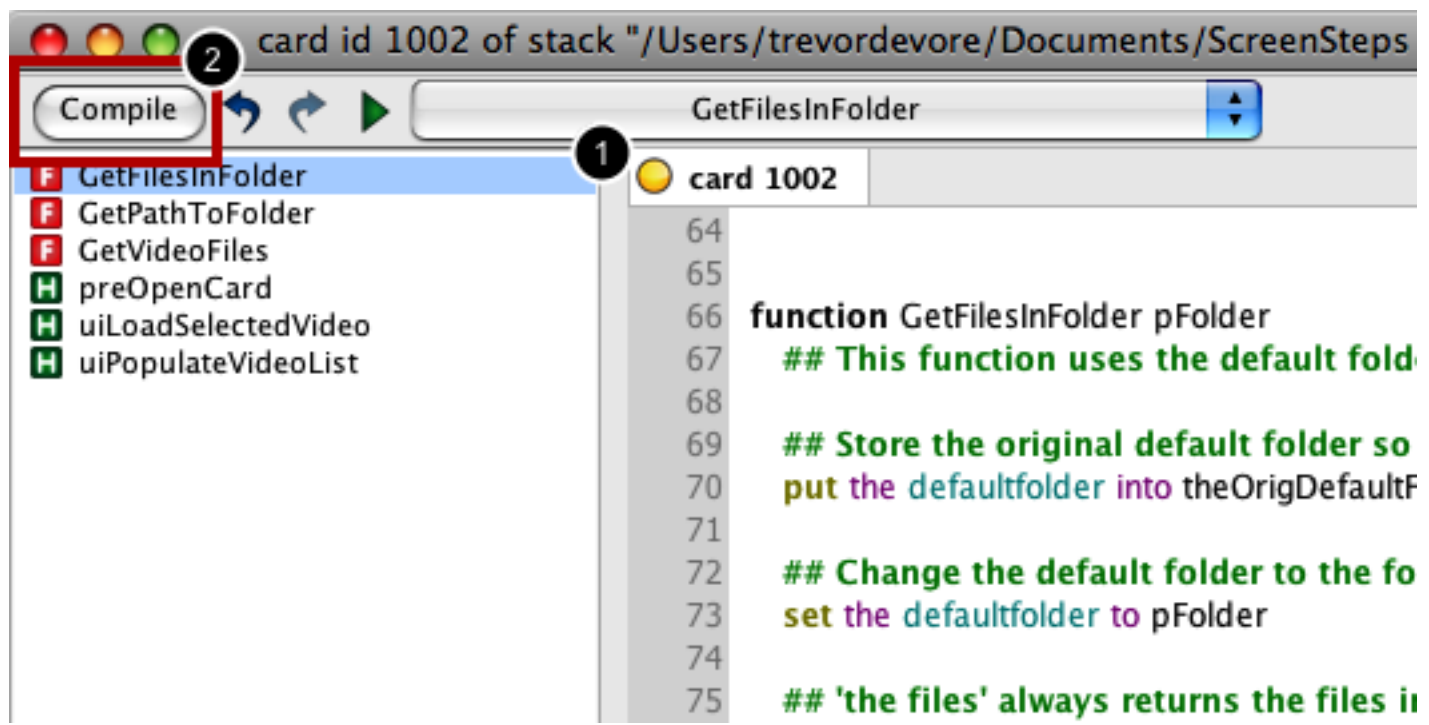
The Script Editor Opens



When you edit the script of an object you use the **Script Editor**. The Script Editor enables you to write revTalk that affects the behavior of your application. Notice that a tab is open in the Script Editor for the Stack window's card (1). We will begin typing revTalk into the script field (2).

The Card Script

Below you will find all of the revTalk that goes in the Card script. If you would just like to copy and paste all of the code into the Card script without worrying about what it does then you can do so. If you would like to walk through writing the revTalk that goes in the Card script then skip the next step go to the next lesson which describes the Card script in detail.



Copy and paste the following revTalk into the Card script. After pasting the revTalk into the Card script the dot in the Card's tab will turn yellow (1). Click the **Compile** button (2) so that Revolution applies the changes you've made to the Card's script.

Copy & Paste Into Card Script

on preOpenCard

Load the video list

 uiPopulateVideoList

If there is at least 1 video in the list then load the video

if the text of field "Video Menu" is not empty **then**

set the hilitedline of field "Video Menu" to 1

 uiLoadSelectedVideo

end if

end preOpenCard

command uiPopulateVideoList

Get list of video files in Video folder

put GetVideoFiles() into theFiles

Assign the list of files to the list field (our menu)

set the text of field "Video Menu" to theFiles

```
end uiPopulateVideoList
```

```
function GetVideoFiles
```

```
## Get the path to the "Video" folder on disk
```

```
put GetPathToFolder("Videos") into theFolder
```

```
## Get list of files in the folder
```

```
put GetFilesInFolder(theFolder) into theFiles
```

```
## Return list of files
```

```
return theFiles
```

```
end GetVideoFiles
```

```
function GetPathToFolder pFolderName
```

```
## Retrieving paths to folders that are relative to the stack can be tricky.
```

```
## Determine the location of this stack on disk
```

```
put the filename of this stack into theFolder
```

```
## Folder paths use "/" to separate each folder in the path
```

```
## By setting the itemDelimiter to slash we can refer to
```

```
## individual sections of the path by the 'item' token in revTalk.
```

```
set the itemDelimiter to slash
```

```
## When you build a standalone version of this stack on OS X the stack
```

```
## file will be located in side of an application bundle. These next few
```

```
## lines strip the application bundle portion of the path off.
```

```
if the platform is "MacOS" then
```

```
if theFolder contains ".app/Contents/MacOS" then
```

```
## Delete the last three items of the path that are specific
```

```
## to the application bundle
```

```
delete item -3 to -1 of theFolder
```

```
end if
```

```
end if
```

```
## Replace the last item in theFolder with the 'pFolderName' parameter
```

```
put pFolderName into the last item of theFolder
```

```

## Return the complete path
return theFolder
end GetPathToFolder

function GetFilesInFolder pFolder
    ## This function uses the default folder to get a list of files

    ## Store the original default folder so we can return to it later
    put the defaultFolder into theOrigDefaultFolder

    ## Change the default folder to the folder passed into the function (pFolder)
    set the defaultFolder to pFolder

    ## 'the files' always returns the files in the 'default folder'
    put the files into theFiles

    ## Restore the original 'default folder' setting
    set the defaultFolder to theOrigDefaultFolder

    ## Filter out invisible files (files that start with a "." in their name) from 'theFiles' variable
    filter theFiles without "."

    ## Return the list of files to the caller of the function
    return theFiles
end GetFilesInFolder

command uiLoadSelectedVideo
    ## Get the name of the video selected in the video menu
    put the selectedText of field "Video Menu" into theVideoName
    put "Videos/" & theVideoName into theVideoFile

    ## Set 'the filename' property the player object to the relative video path
    ## Revolution will locate the video as long as the "Videos" folder is
    ## alongside the stack file or the application executable.
    set the filename of player "My Video" to theVideoFile

    ## Reset the time of the Player object to 0
    set the currentTime of player "My Video" to 0
end uiLoadSelectedVideo

```

```

on preOpenCard
  ## Load the video list
  uiPopulateVideoList

  ## If there is at least 1 video in the list then load the video
  if the text of field "Video Menu" is not empty then
    set the hilitedline of field "Video Menu" to 1
    uiLoadSelectedVideo
  end if

end preOpenCard

```

```

command uiPopulateVideoList
  ## Get list of video files in Video folder
  put GetVideoFiles() into theFiles

  ## Assign the list of files to the list field (our menu)
  set the text of field "Video Menu" to theFiles

end uiPopulateVideoList

```

```

function GetVideoFiles
  ## Get the path to the "Video" folder on disk
  put GetPathToFolder("Videos") into theFolder

  ## Get list of files in the folder
  put GetFilesInFolder(theFolder) into theFiles

  ## Return list of files
  return theFiles
end GetVideoFiles

```

```

function GetPathToFolder pFolderName
  ## Retrieving paths to folders that are relative to the stack can be tricky.

  ## Determine the location of this stack on disk
  put the filename of this stack into theFolder

```

```

## Folder paths use "/" to separate each folder in the path
## By setting the itemDelimiter to slash we can refer to
## individual sections of the path by the 'item' token in revTalk.
set the itemDelimiter to slash

## When you build a standalone version of this stack on OS X the stack
## file will be located in side of an application bundle. These next few
## lines strip the application bundle portion of the path off.
if the platform is "MacOS" then
    if theFolder contains ".app/Contents/MacOS" then
        ## Delete the last three items of the path that are specific
        ## to the application bundle
        delete item -3 to -1 of theFolder
    end if
end if

## Replace the last item in theFolder with the 'pFolderName' parameter
put pFolderName into the last item of theFolder

## Return the complete path
return theFolder
end GetPathToFolder

function GetFilesInFolder pFolder
    ## This function uses the default folder to get a list of files

    ## Store the original default folder so we can return to it later
    put the defaultfolder into theOrigDefaultFolder

    ## Change the default folder to the folder passed into the function (pFolder)
    set the defaultfolder to pFolder

    ## 'the files' always returns the files in the 'default folder'
    put the files into theFiles

    ## Restore the original 'default folder' setting
    set the defaultfolder to theOrigDefaultFolder

    ## Filter out invisible files (files that start with a "." in their name) from 'theFiles' variable
    filter theFiles without "."

```

```

## Return the list of files to the caller of the function
return theFiles
end GetFilesInFolder

command uiLoadSelectedVideo
  ## Get the name of the video selected in the video menu
  put the selectedtext of field "Video Menu" into theVideoName
  put "Videos/" & theVideoName into theVideoFile

  ## Set 'the filename' property the player object to the relative video path
  ## Revolution will locate the video as long as the "Videos" folder is
  ## alongside the stack file or the application executable.
  set the filename of player "My Video" to theVideoFile

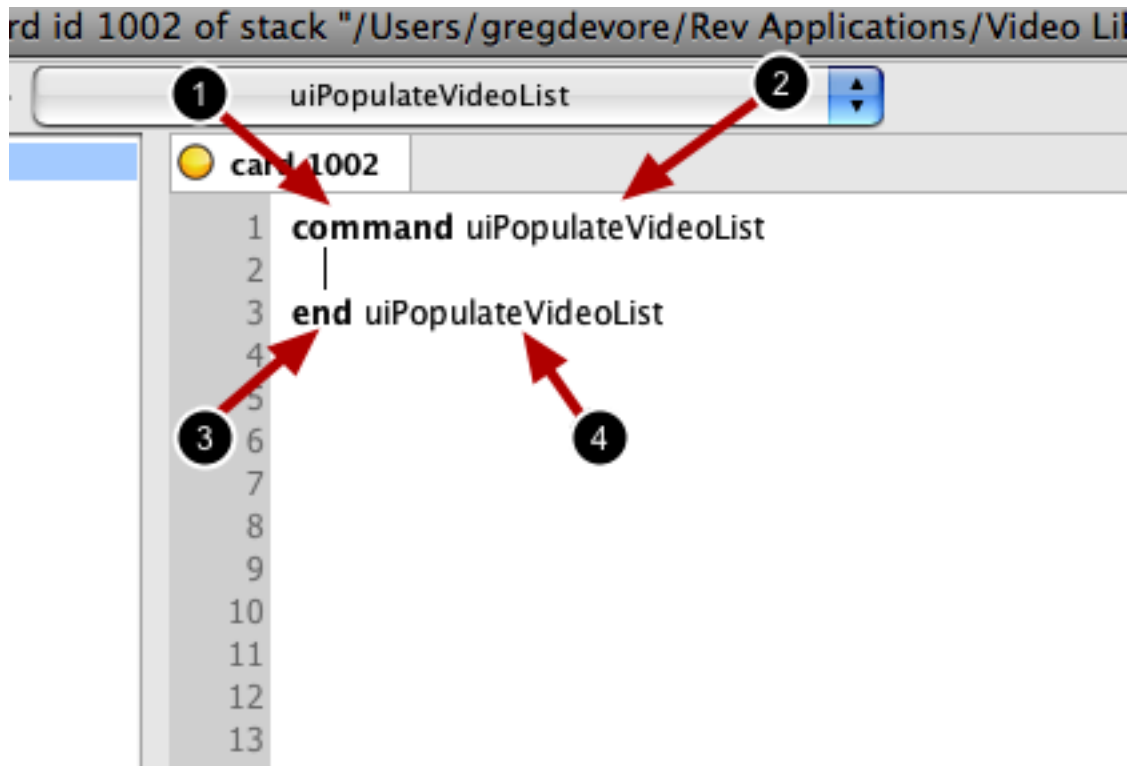
  ## Reset the time of the Player object to 0
  set the currenttime of player "My Video" to 0
end uiLoadSelectedVideo

```

The Card Script In Detail

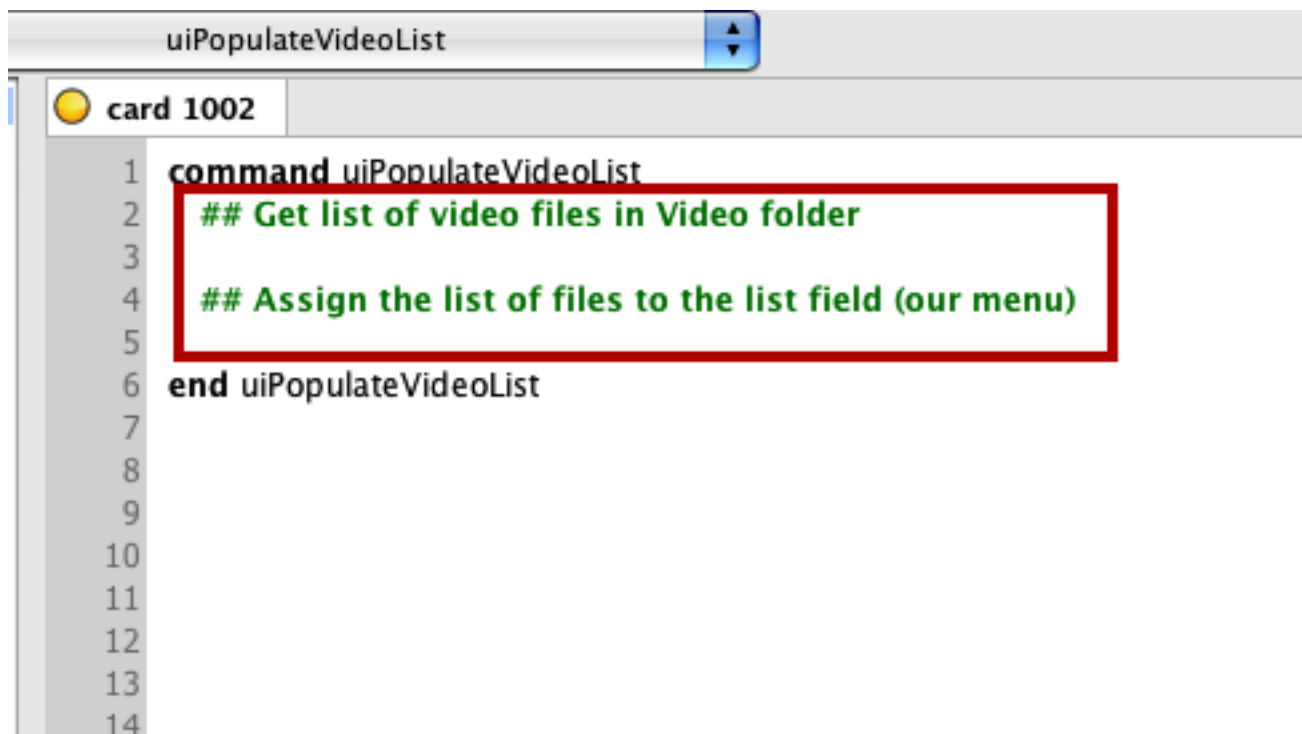
This lesson will describe the Card script in detail and may be of interest if you are trying to learn revTalk. If you would just like to get your Video Player working as quickly as possible the you can skip this lesson and come back to it later. In this case you should have copied and pasted the entire Card Script that was provided in the previous lesson.

Create The uiPopulateVideoList Command



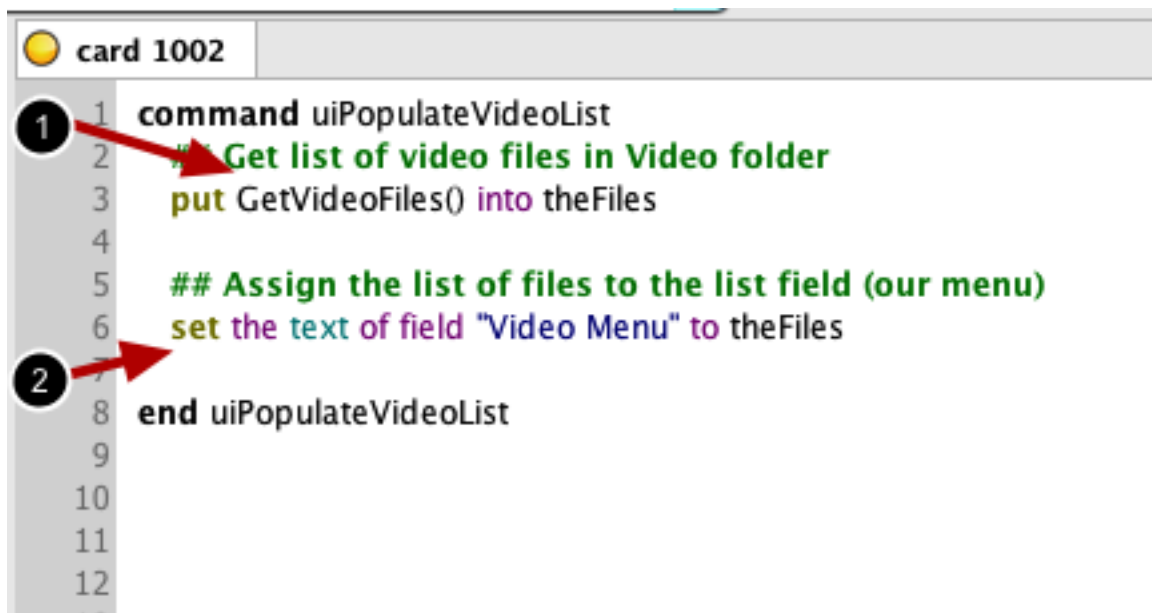
In revTalk you can write commands that perform certain operations. We are going to write a command that populates the List Field with available videos. The definition for a command starts with the word **command** (1), followed by the name of the command (2). To finish the definition of the command you write **end** (3) followed by the name of the command (4).

What Will The Command Do?



Before writing the actual revTalk that populates the List Field I've added some comments that describe what I want to do. This can be helpful as the comments tell the story of what needs to happen. I can then go back and write the revTalk that actually performs the required actions.

The Completed Command



Here is what the command looks like when it is finished. The command calls a **function** call **GetVideoFiles()** (1) that returns all of the available video files, each one separated by the return character. We will look at what a function is in the next step.

The command then assigns the list of video files to the **text** property of the **Video Menu** List Field (2). Assigning the **text** property of the List Field changes what you see in your Stack window.

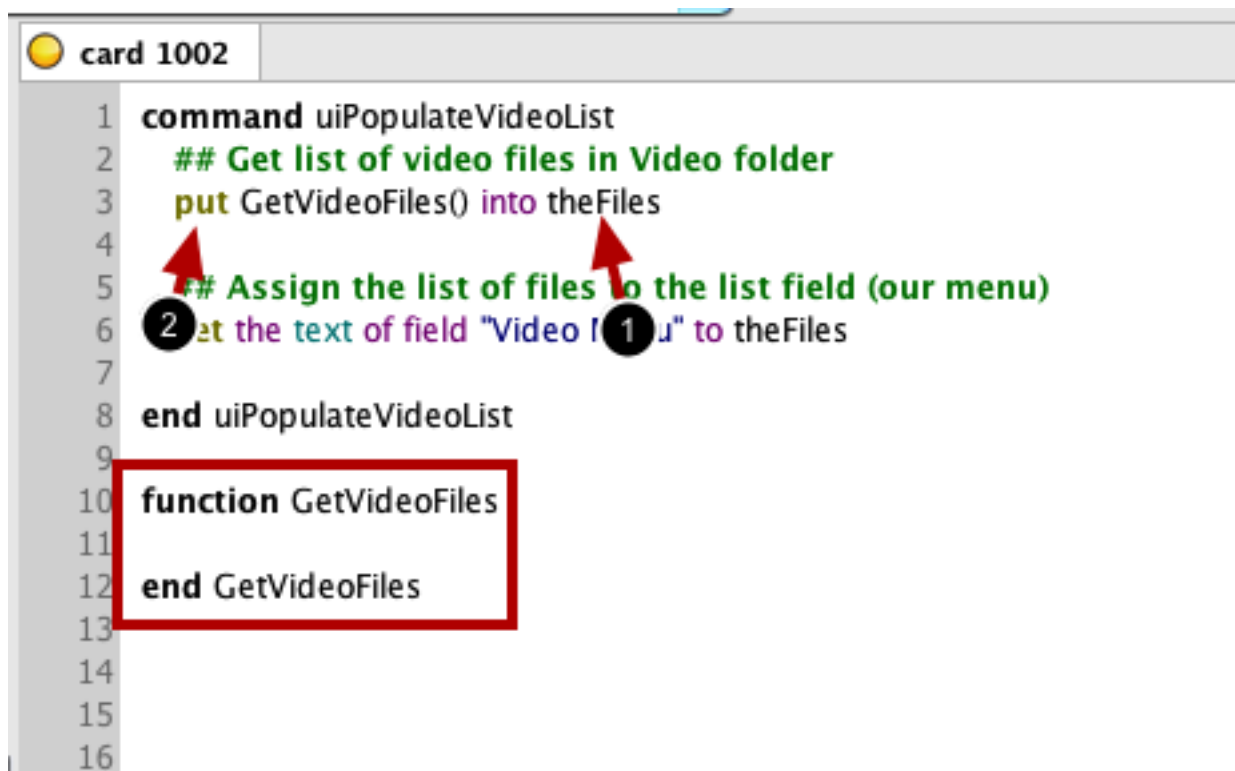
Copy & Paste Into Card Script

```
command uiPopulateVideoList
  ## Get list of video files in Video folder
  put GetVideoFiles() into theFiles

  ## Assign the list of files to the list field (our menu)
  set the text of field "Video Menu" to theFiles

end uiPopulateVideoList
```

The GetVideoFiles Function



In the `uiPopulateVideoList` command we called a function that returns a list of video files. As of right now there is no function named `GetVideoFiles` that has been defined. Let's do that now.

You define a function the same way that you do a command except that you use the word **function** rather than **command**. Notice how you can just add the definition of the `GetVideoFiles` function after the definition of the `uiPopulateVideoList` command. You can add any number of function and command definitions to a script.

```

function GetVideoFiles
  ## Get the path to the "Video" folder on disk
  put GetPathToFolder("Videos") into theFolder

  ## Get list of files in the folder
  put GetFilesInFolder(theFolder) into theFiles

  ## Return list of files
  return theFiles
end GetVideoFiles

```

The purpose of GetVideoFiles is to return a return delimited list of all video files in the **Videos** folder that you created earlier. In order to do this we will need to get the path to the Videos folder on disk and then get a list of all the files in the folder. GetVideoFiles makes use of two other functions that help do this: GetPathToFolder and GetFilesInFolder. We will define this two folders next.

Copy & Paste Into Card Script

```

function GetVideoFiles
  ## Get the path to the "Video" folder on disk
  put GetPathToFolder("Videos") into theFolder

  ## Get list of files in the folder
  put GetFilesInFolder(theFolder) into theFiles

  ## Return list of files
  return theFiles
end GetVideoFiles

```

The GetPathToFolder Function

When working with video the video files are stored separately from the application that we are creating. In this case the video files are stored in the Videos folder. When you share your application with others it is important that your application is able to find the video files on the computer it is running on.

GetPathToFolder helps do this. The purpose of the function is to take the name of a folder, in this case "Videos", and return the full file path to that folder.

You can read the comments in the function if you want to learn what it is doing but for now you can just copy and paste it into your Card script and now worry about the details.

Copy & Paste Into Card Script

```

function GetPathToFolder pFolderName
    ## Retrieving paths to folders that are relative to the stack can be tricky.

    ## Determine the location of this stack on disk
    put the filename of this stack into theFolder

    ## Folder paths use "/" to separate each folder in the path
    ## By setting the itemDelimiter to slash we can refer to
    ## individual sections of the path by the 'item' token in revTalk.
    set the itemDelimiter to slash

    ## When you build a standalone version of this stack on OS X the stack
    ## file will be located in side of an application bundle. These next few
    ## lines strip the application bundle portion of the path off.
    if the platform is "MacOS" then
        if theFolder contains ".app/Contents/MacOS" then
            ## Delete the last three items of the path that are specific
            ## to the application bundle
            delete item -3 to -1 of theFolder
        end if
    end if

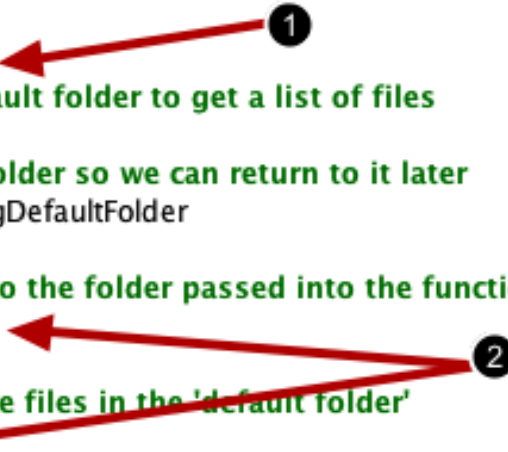
    ## Replace the last item in theFolder with the 'pFolderName' parameter
    put pFolderName into the last item of theFolder

    ## Return the complete path
    return theFolder
end GetPathToFolder

```

The GetFilesInFolder Function

```
26
27 function GetFilesInFolder pFolder
28     ## This function uses the default folder to get a list of files
29
30     ## Store the original default folder so we can return to it later
31     put the defaultfolder into theOrigDefaultFolder
32
33     ## Change the default folder to the folder passed into the function (pFolder)
34     set the defaultfolder to pFolder
35
36     ## 'the files' always returns the files in the 'default folder'
37     put the files into theFiles
38
39     ## Restore the original 'default folder' setting
40     set the defaultfolder to theOrigDefaultFolder
41
42     ## Filter out invisible files (files that start with a "." in their name) from 'theFiles' variable
43     filter theFiles without "."
44
45     ## Return the list of files to the caller of the function
46     return theFiles
47 end GetFilesInFolder
```

A diagram with two red arrows and two numbered circles. Arrow 1 points from circle 1 to the parameter 'pFolder' in the function signature on line 27. Arrow 2 points from circle 2 to the text 'the files' in the comment on line 36.

The **GetFilesInFolder** function is a handy way of retrieving a return delimited list of files located in a particular folder on a computer. The parameter you pass in, **pFolder** (1), is the path to the folder whose files you want to retrieve. The function then uses **the defaultFolder** and **the files** (2) to get the list of files.

Copy & Paste Into Card Script

```
function GetFilesInFolder pFolder
    ## This function uses the default folder to get a list of files

    ## Store the original default folder so we can return to it later
    put the defaultfolder into theOrigDefaultFolder

    ## Change the default folder to the folder passed into the function (pFolder)
    set the defaultfolder to pFolder

    ## 'the files' always returns the files in the 'default folder'
    put the files into theFiles

    ## Restore the original 'default folder' setting
```

set the defaultfolder to theOrigDefaultFolder

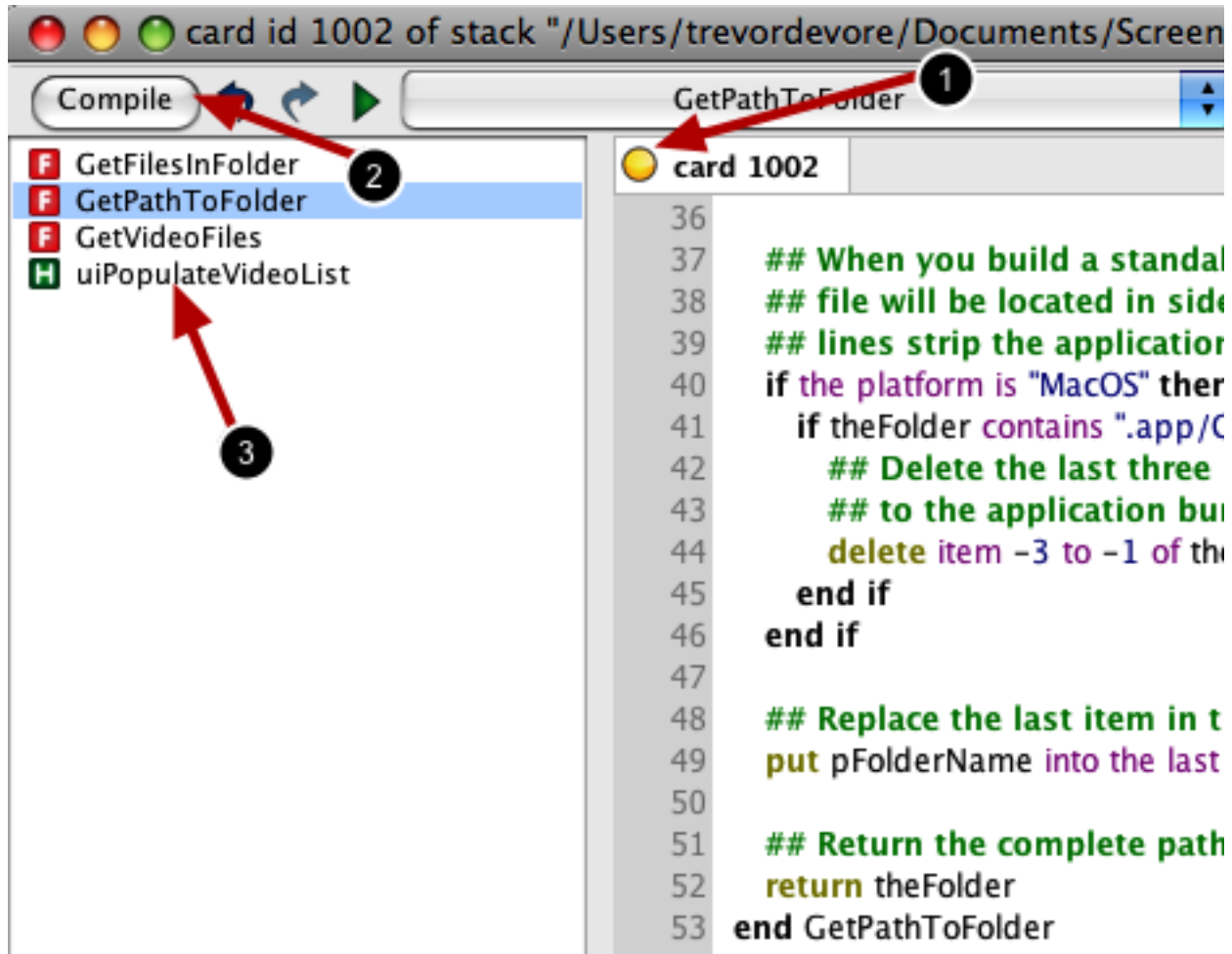
Filter out invisible files (files that start with a "." in their name) from 'theFiles' variable
filter theFiles without ".".*"

Return the list of files to the caller of the function

return theFiles

end GetFilesInFolder

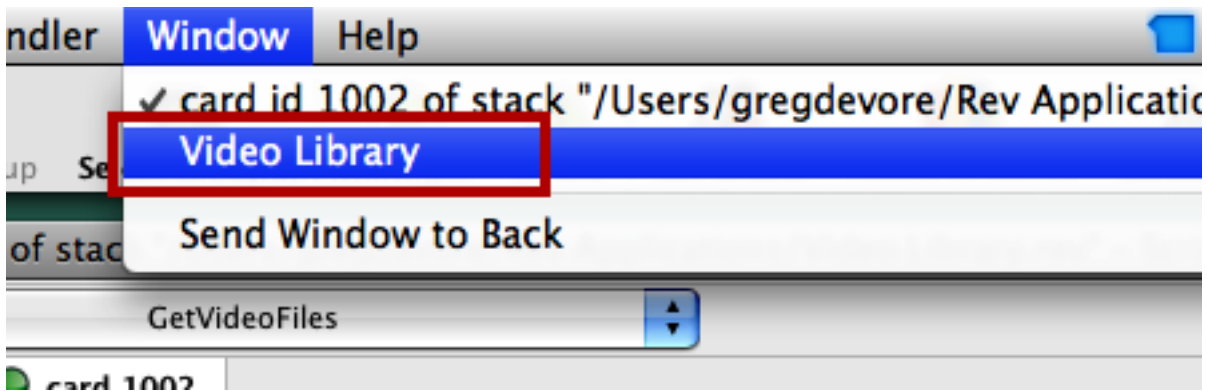
Compile the Card Script



Now that you have modified the revTalk in your Card script you need to compile the script. Whenever an object script has been modified the dot in the object tab will turn yellow (1). Click the **Compile** button (2) to compile the script. The dot will then turn green if the script compiled correctly or the Script Editor will alert you if there is something in your script that needs to be fixed, e.g. you typed something incorrectly.

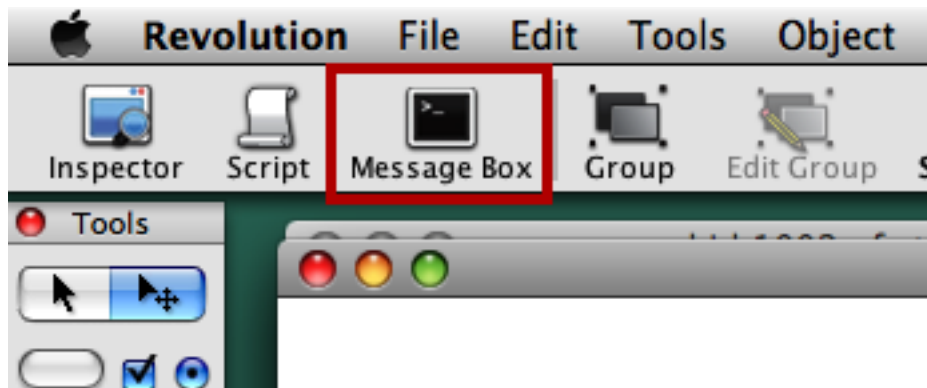
Also note that after you compile a script the list of commands and functions that you have defined in your script appear in the left column (3). This helps you quickly navigate to a particular handler (a handler is either a command or a function) in your script.

Test the uiPopulateVideoList Command

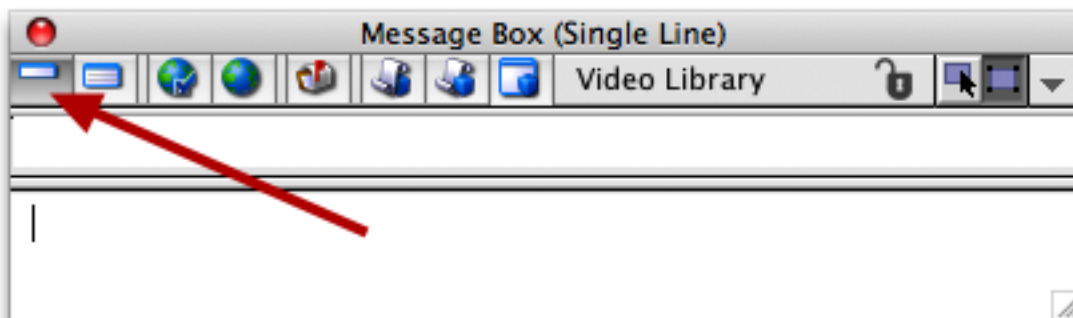


We have now added all of the commands and functions to the card script that are necessary for uiPopulateVideoList to work. Let's test it out. Use the **Window** menu to select the **Video Library** Stack window and bring it to the front.

Open The Message Box

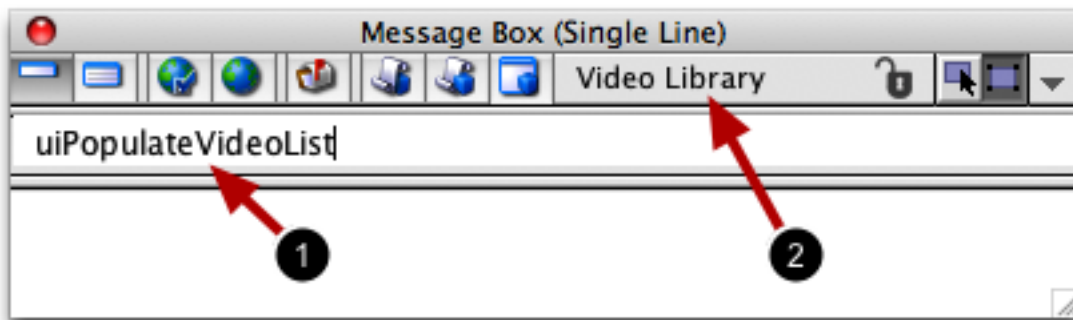


Now we are going to use the **Message Box** to test the uiPopulateVideoFiles command. The Message Box is a tool that allows you to execute revTalk statements. Click on the **Message Box** button in the toolbar.



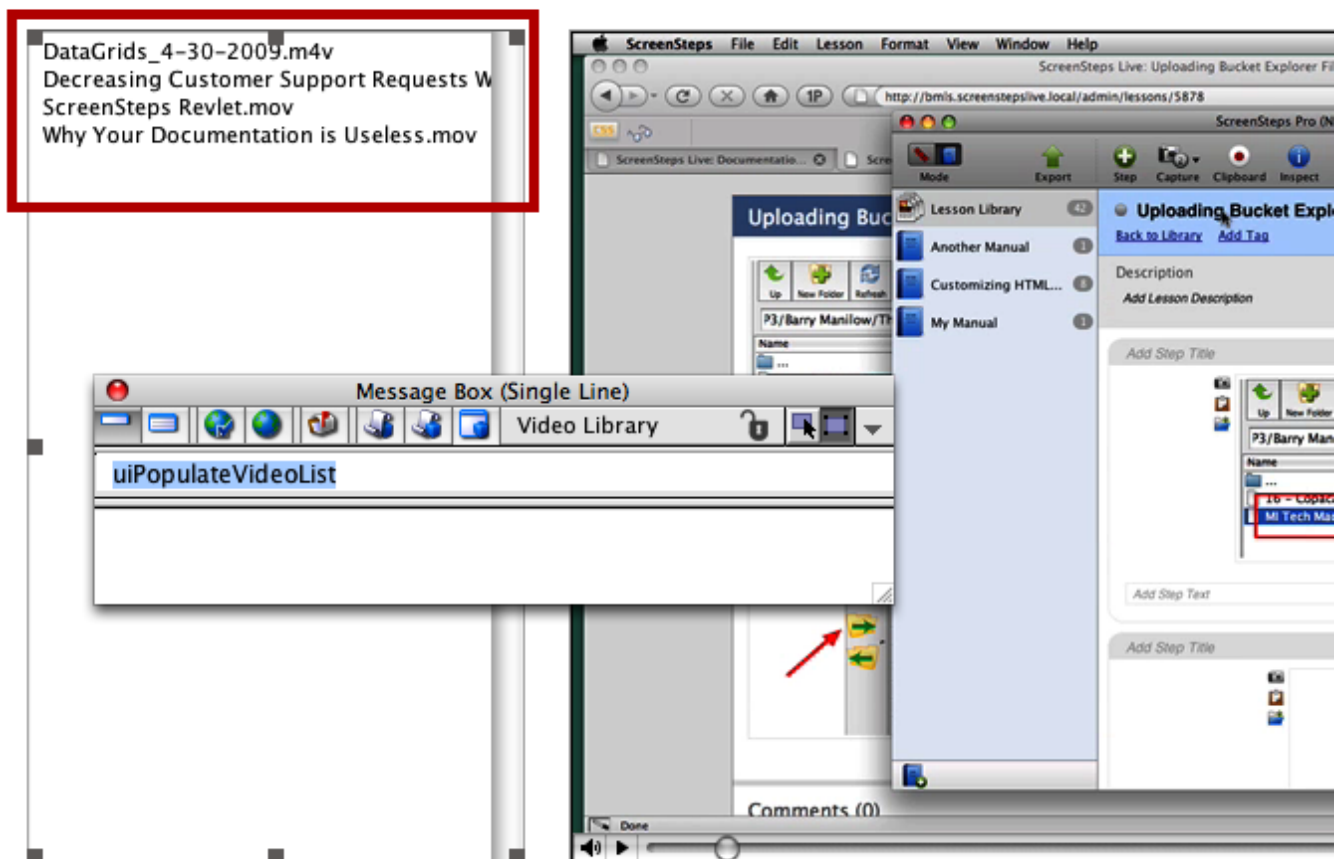
The message box will appear and should look something like this. Make sure that **single-line messages** mode is active.

Call uiPopulateVideoList



Type `uiPopulateVideoList` in top field and press the **Return** key (1). Because the **Video Library** stack (2) is the target of any commands executed in the **Message Box** the `uiPopulateVideoList` command you defined in your card script will be executed.

Done

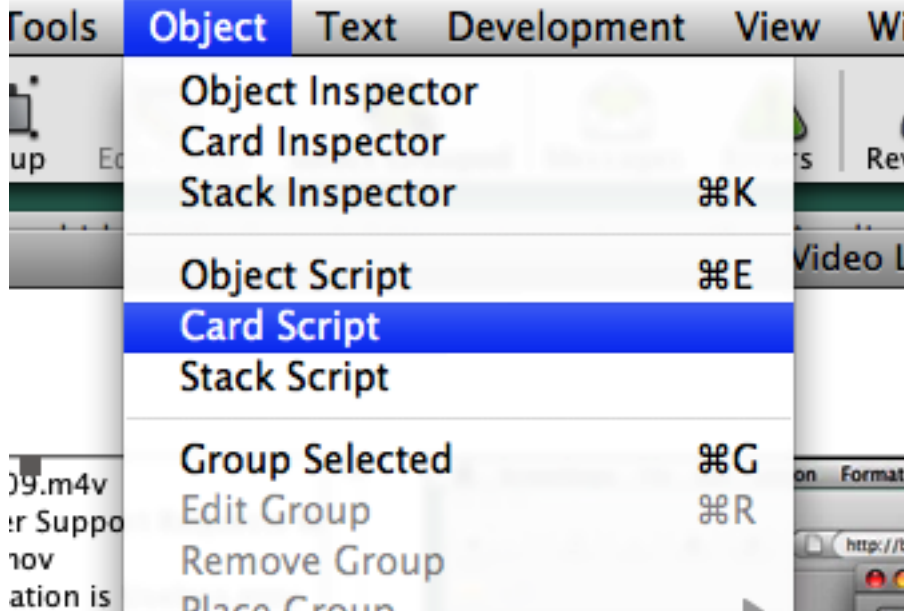


After pressing the **Return** key the List Field should populate with the videos in your **Videos** folder.

Loading Video Files When a User Selects a Video From the Menu

Now that the video menu is being populated we need to have the application load a video whenever the user selects a video in the menu. Let's look at how to do that.

Edit Card Script



With the **Video Player** Stack window at the forefront, choose **Object > Card Script** to edit the card script of the Stack window again.

Define The uiLoadSelectedVideo Handler

```
command uiLoadSelectedVideo
  ## Get the name of the video selected in the video menu 1
  put the selectedtext of field "Video Menu" into theVideoName
  put "Videos/" & theVideoName into theVideoFile

  ## Set 'the filename' property the player object to the relative video path
  ## Revolution will locate the video as long as the "Videos" folder is
  ## alongside the stack file or the application executable.
  set the filename of player "My Video" to theVideoFile

  ## Reset the time of the Player object to 0
  set the currenttime of player "My Video" to 0
end uiLoadSelectedVideo
```

To start we need to define a command that will load the video file that is selected in the menu. uiLoadSelectedVideo will do just that. It gets the video file that is selected and creates a path to the video (1) that looks like this:

Videos/the_selected_movie.mov

Note that the path to the video is a relative path rather than an absolute path. When you assign a relative path to the **filename** property of a Player object Revolution will try to locate the relative path using the folder that the stack file resides in as well as 'the default folder'. Since the **Videos** folder is alongside the stack file Revolution is able to find videos using the above relative URL.

Note for those using older versions of Revolution: Searching the folder that the stack file resides in in order to locate movies was added in Revolution 3.5. If you are using an older version of Revolution then you will need set the **defaultFolder** property to the folder the stack file is in.

Copy & Paste Into Card Script

command uiLoadSelectedVideo

Get the name of the video selected in the video menu

put the selectedtext of field "Video Menu" into theVideoName

put "Videos/" & theVideoName into theVideoFile

Set 'the filename' property of the player object to the relative video path

Revolution will locate the video as long as the "Videos" folder is

alongside the stack file or the application executable.

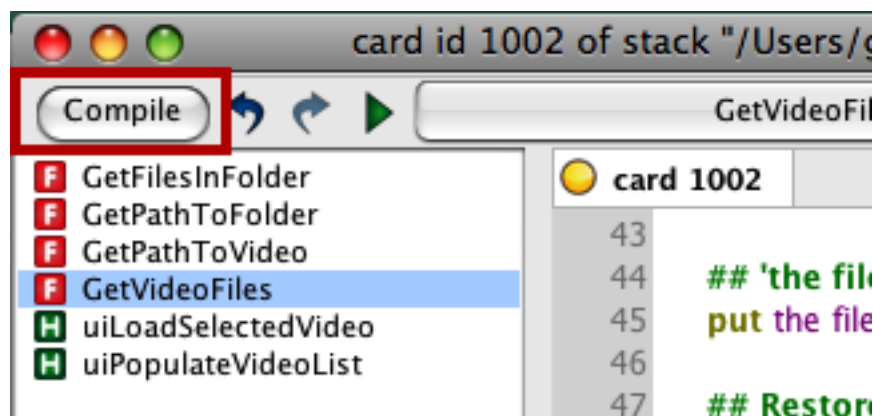
set the filename of player "My Video" to theVideoFile

Reset the time of the Player object to 0

set the currenttime of player "My Video" to 0

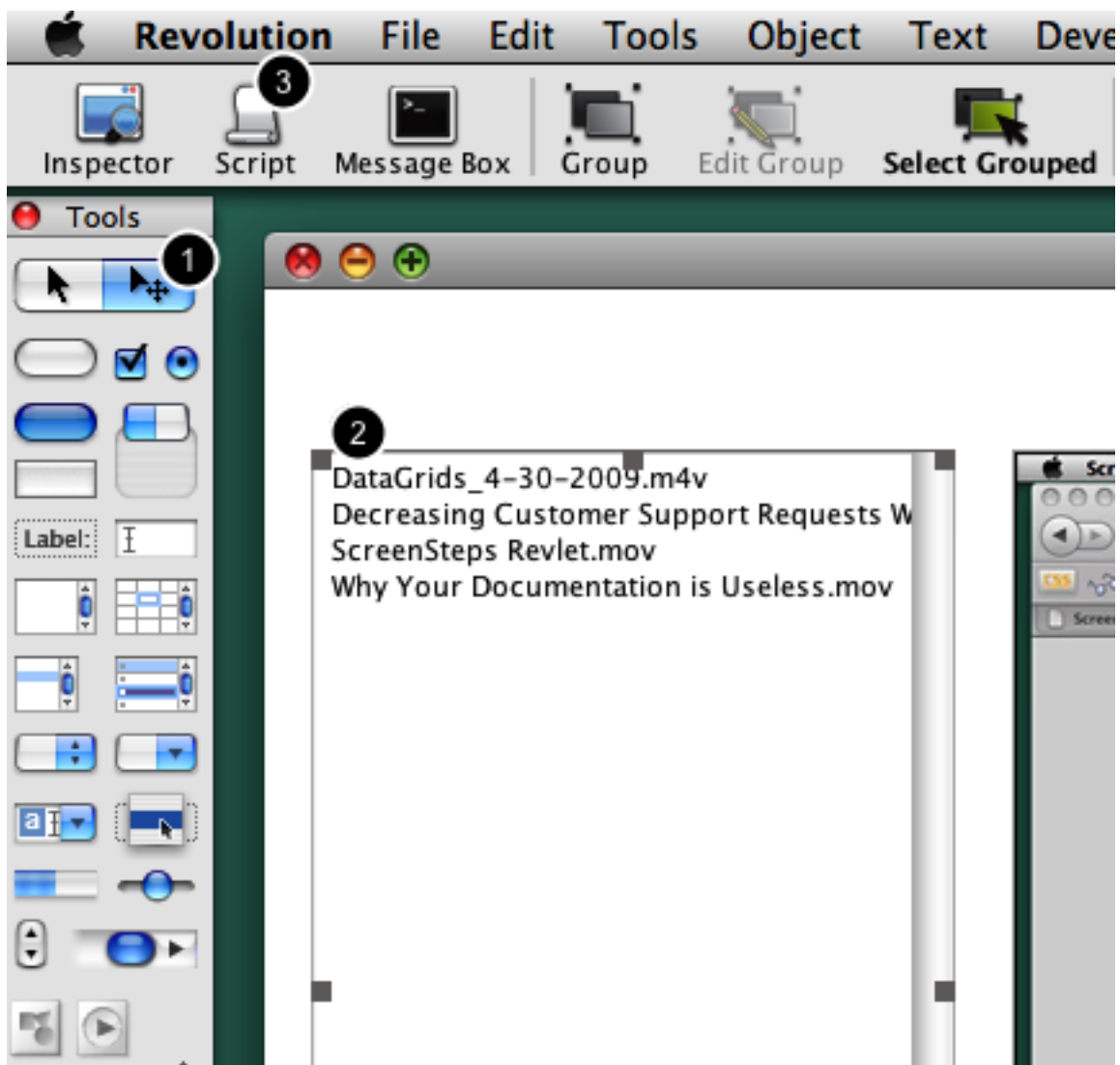
end uiLoadSelectedVideo

Compile The Card Script



Click the **Compile** button to save the changes to the Card script.

Edit the List Field Script



Now we just have to tell Revolution that we want the **uiLoadSelectedVideo** command to be called whenever the user changes the selected menu item in the video menu. With the **Edit** tool activated (1) select the **Video Menu** List Field (2). Click on the **Script** button in the toolbar (3) to edit the object's script.

Define selectionChanged



Whenever the user selects a line in a List Field Revolution sends a message to the field indicating that the selection has changed. This message is called **selectionChanged**. You can define what happens when this message is sent to the field by defining the **on selectionChanged** message in the field's script.

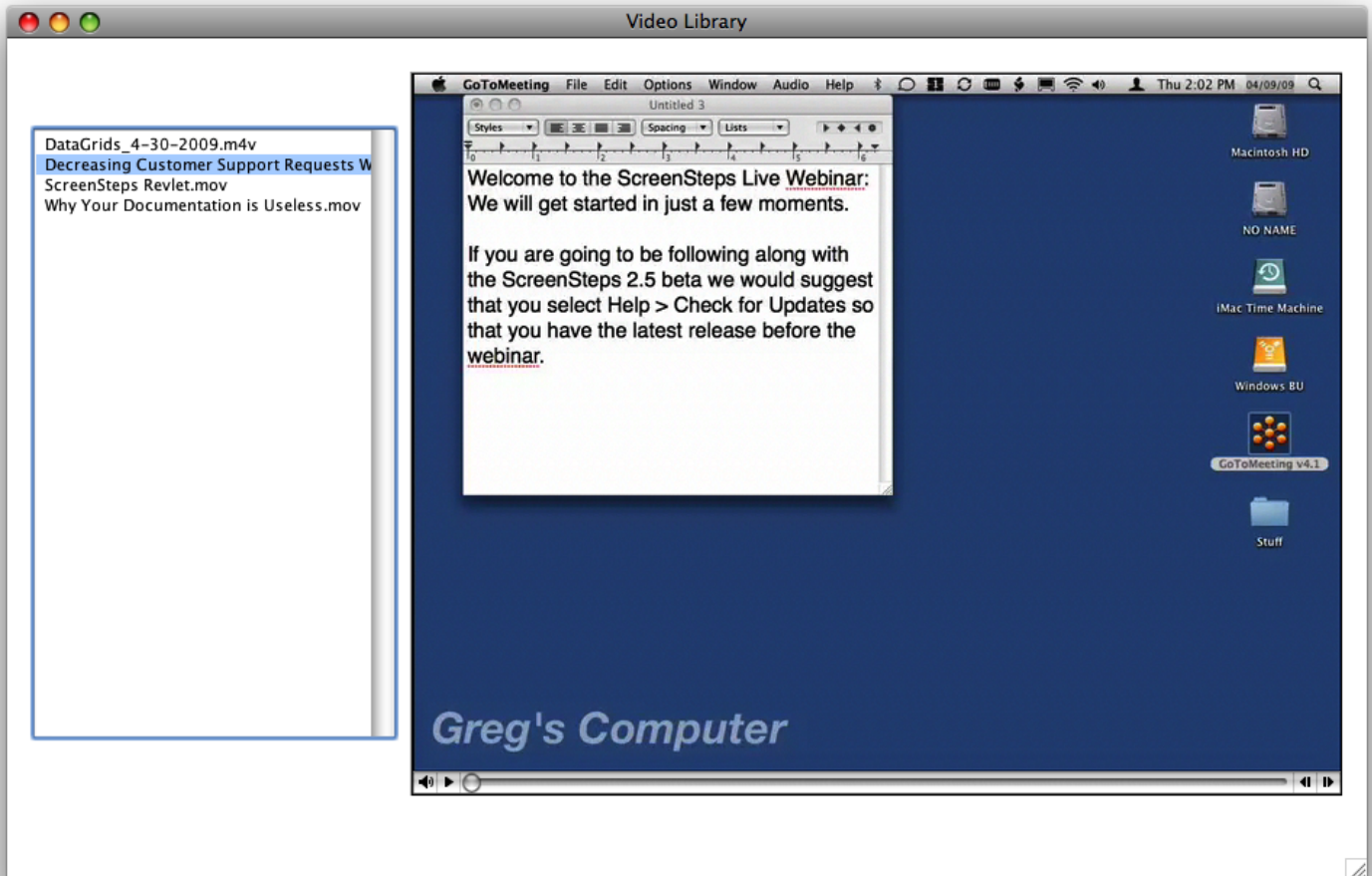
Remember that a Card script can have revTalk that affects all objects on a card. Since we have already defined the uiLoadSelectedVideo command in the Card Script we can call that command from the selectionChanged message of the List Field Script.

Copy & Paste Into List Field Script

```
on selectionChanged
  uiLoadSelectedVideo
end selectionChanged
```

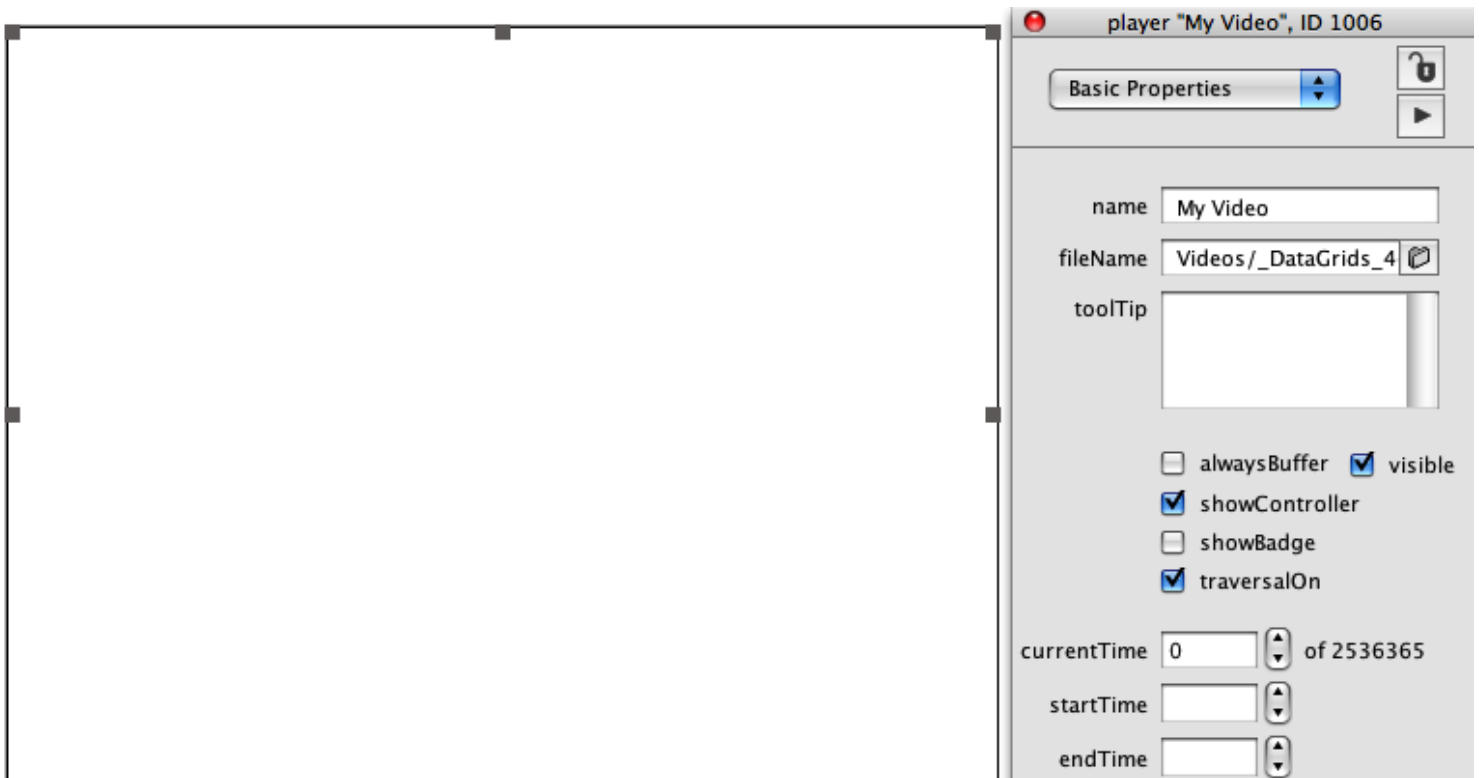


After adding the above script to the List Field and pressing the **Compile** button you can test your work. First activate the **Browse** tool.



With the **Browse** tool activated you can try selecting different selections in the video menu. As you make different selections the video in the Player object should change.

Troubleshooting: If Video Doesn't Show Up

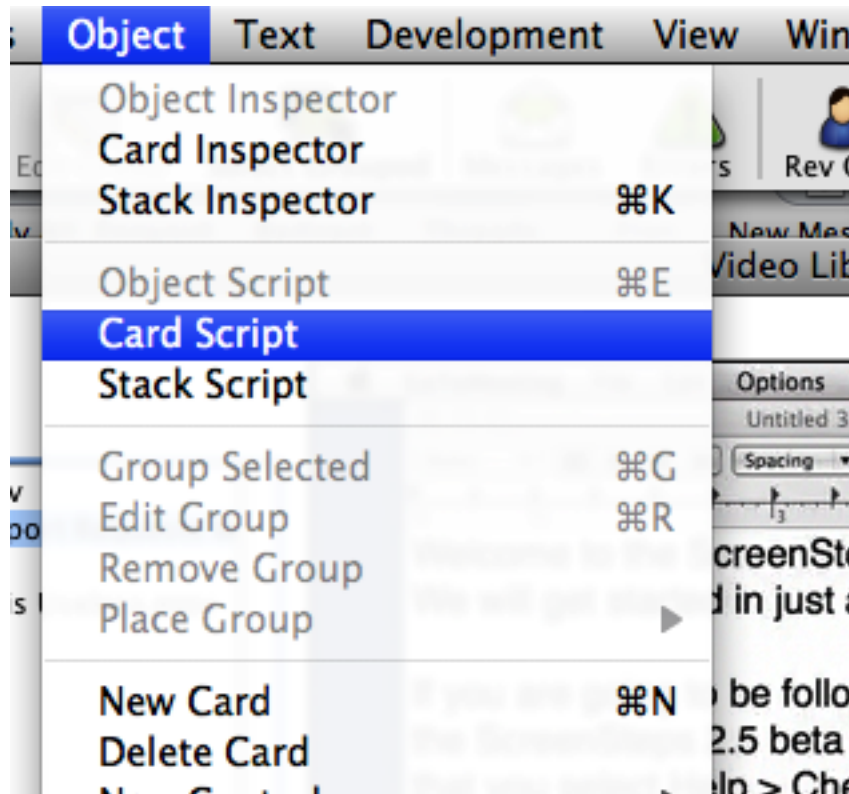


If the video doesn't show up in the Player object then check the Object Inspector for the Player object. Make sure the Source field (the Source field shows the 'filename' property of the Player object) contains a valid relative path.

Adding Code to Setup the Application Once it is Launched

At this point your Video Player application is working in the Revolution IDE and you are just about ready to build an application that you can share with others. The final step is to write a little revTalk that will populate the video menu and load the first video when your application is launched.

Edit Card Script

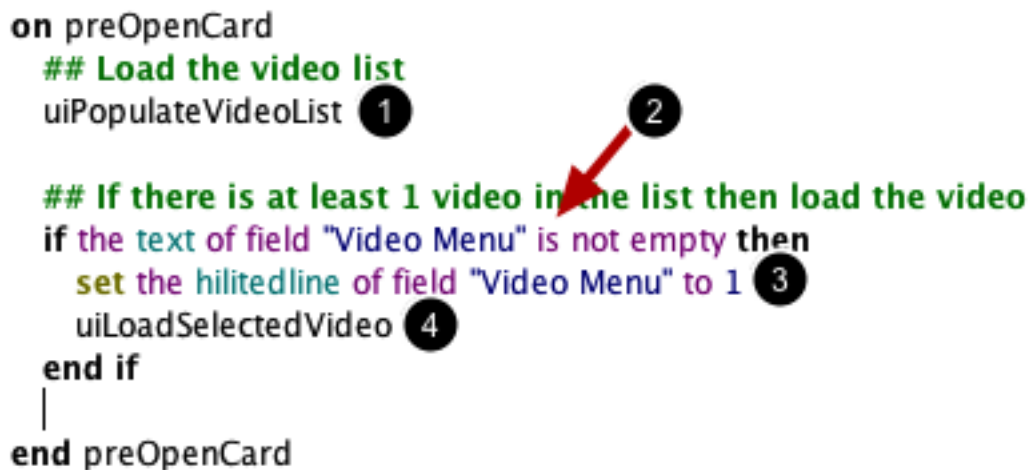


With the **Video Player** Stack window as the frontmost window choose **Object > Card Script**.

Define preOpenCard Message

```
on preOpenCard
  ## Load the video list
  uiPopulateVideoList 1

  ## If there is at least 1 video in the list then load the video
  if the text of field "Video Menu" is not empty then
    set the hilitedline of field "Video Menu" to 1
    uiLoadSelectedVideo
  end if
end preOpenCard
```



When a Stack window is about to be opened up and displayed on the computer Revolution sends the **preOpenCard** message to the Card that is about to be shown. This is a good place to initialize the menu and load a video.

To initialize the menu all we need to do is call **uiPopulateVideoList** (1) as that reads in the list of video files and populates the menu.

After the menu has been populated we can load the first video in the list. Before we load the video we check to make sure that at least one video exists in the menu. Checking that the text of field "Video Menu" is not empty (2) accomplishes this. If the **text** property of the field contains any text at all then there is at least one video file.

After confirming that there is at least one video we use **revTalk** to change the line that is selected by setting the **hilitedLine** property of the list field to 1 (3). This selects the first line of the list field.

With the first line selected we finally call **uiLoadSelectedVideo** which loads the currently selected video in the menu (4).

Copy & Paste Into Card Script

```
on preOpenCard
  ## Load the video list
  uiPopulateVideoList

  ## If there is at least 1 video in the list then load the video
  if the text of field "Video Menu" is not empty then
    set the hilitedline of field "Video Menu" to 1
```



```
    uiLoadSelectedVideo  
end if
```

```
end preOpenCard
```

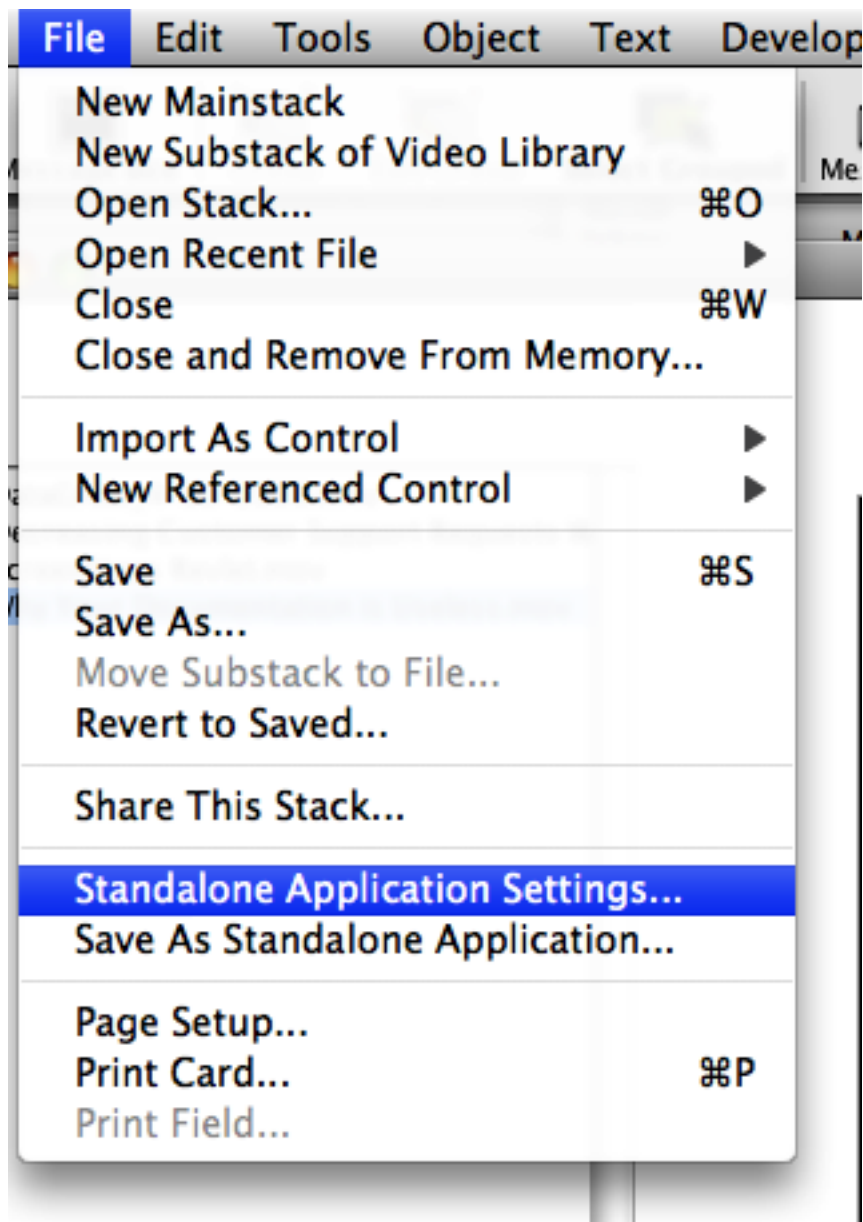
Compile The Script

Make sure and compile the Card script after adding the above revTalk.

Creating Your Executable Application for Mac and Windows

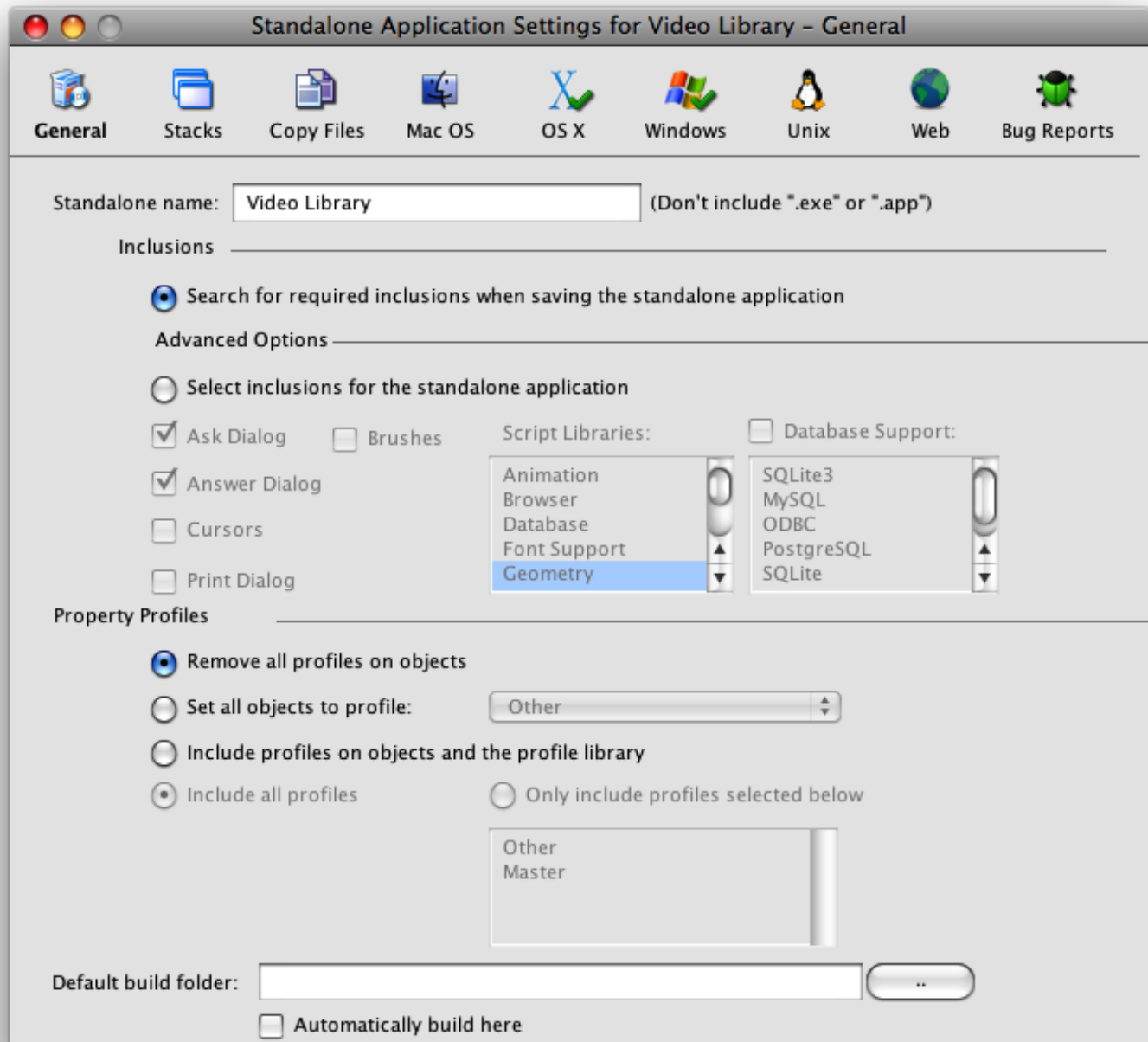
Now you are ready to build an executable application that you can share with others.

Open Standalone Application Settings Dialog



The first step in creating an executable application is to open the Standalone Application Settings dialog for the **Video Player** Stack. With the Video Player window as the frontmost window choose **File > Standalone Application Settings....** This menu item opens the dialog for the **Video Player** stack.

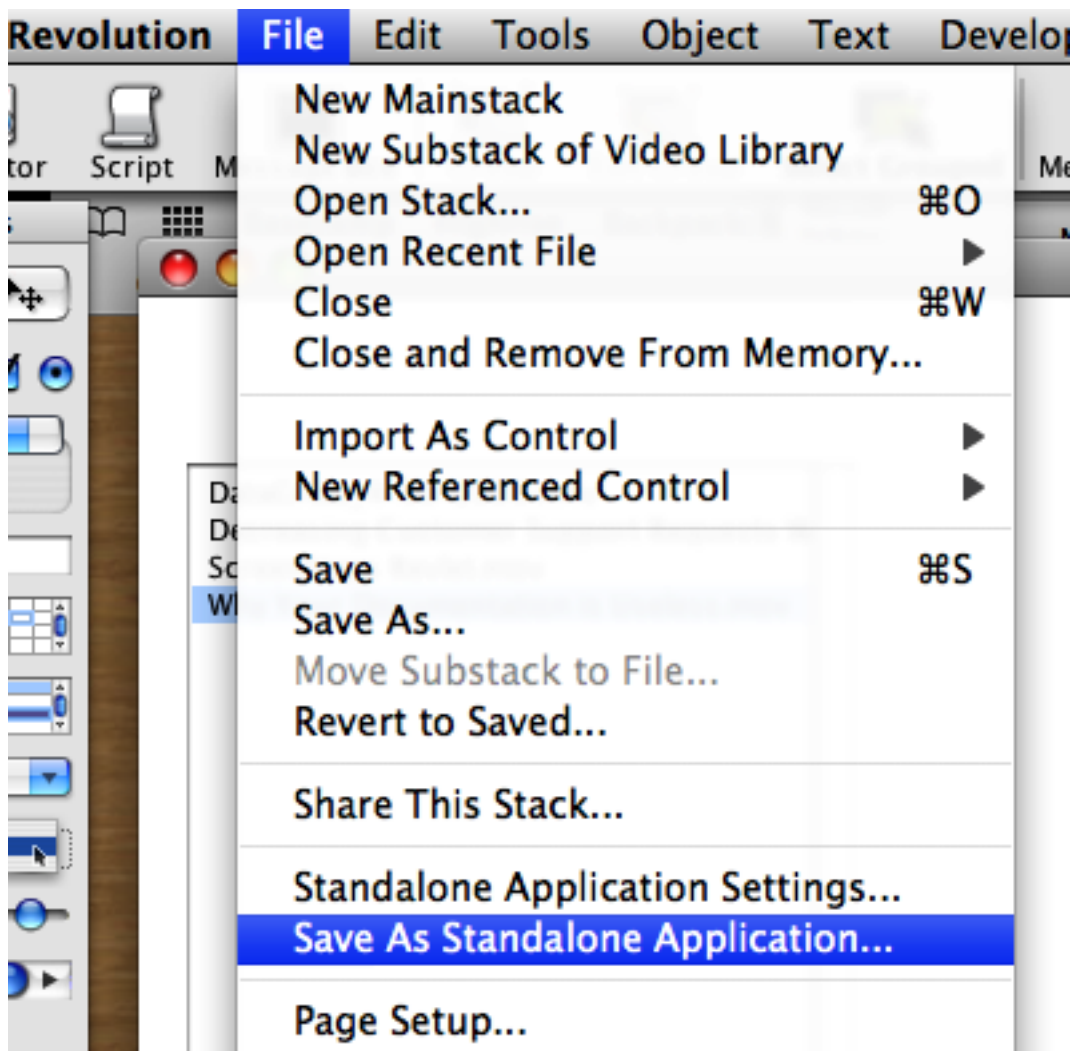
Standalone Application Settings Dialog



This dialog allows you to configure a number of options for the executable application that you will create. For our purposes the default settings will suffice as Revolution will build an executable for both OS X and Windows.

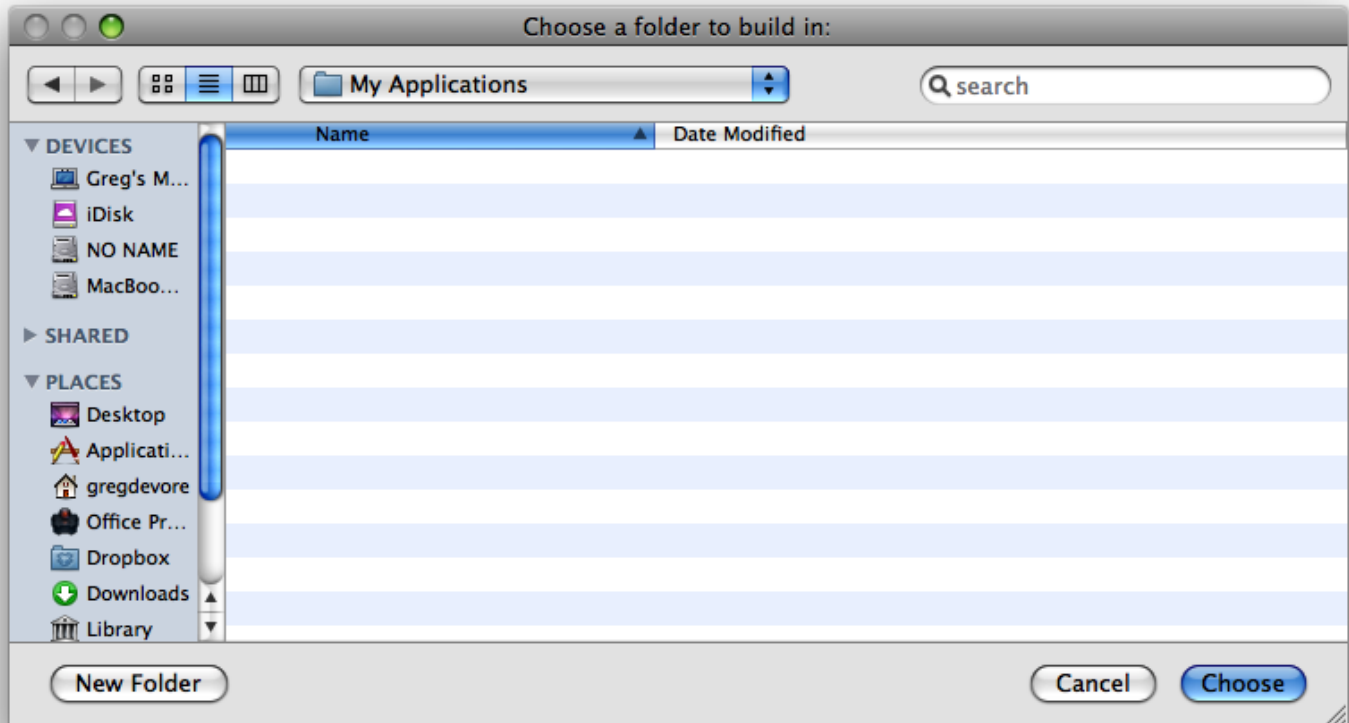
Go ahead and close the dialog using the close button in the dialog title bar.

Save As Standalone Application



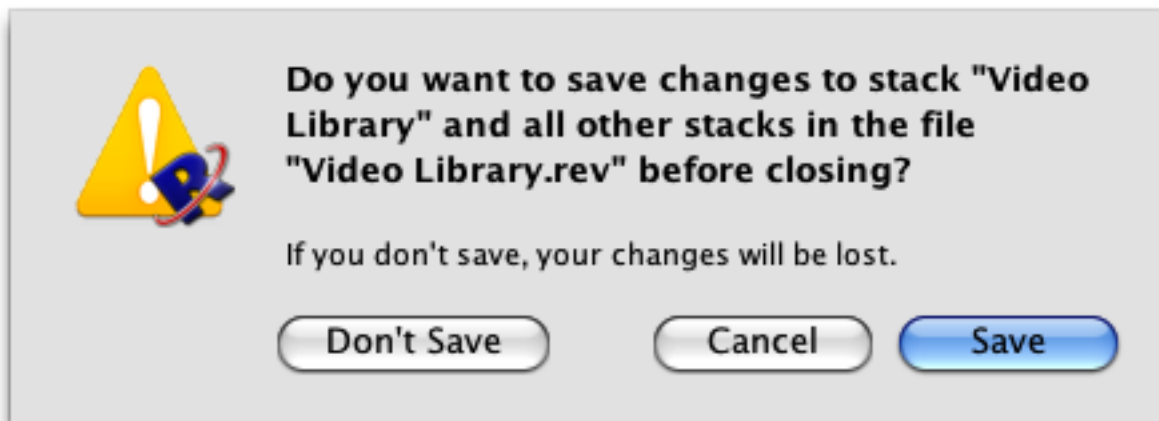
Now that you have configured the standalone application settings you can create an executable. With the **Video Player** window as the frontmost window choose **Save As Standalone Application...** to begin the process.

Select Output Folder



A folder selection dialog will appear asking you to select a folder to save your application in. Select the folder you would like to save to and click the **Choose** button.

Save Your Stack



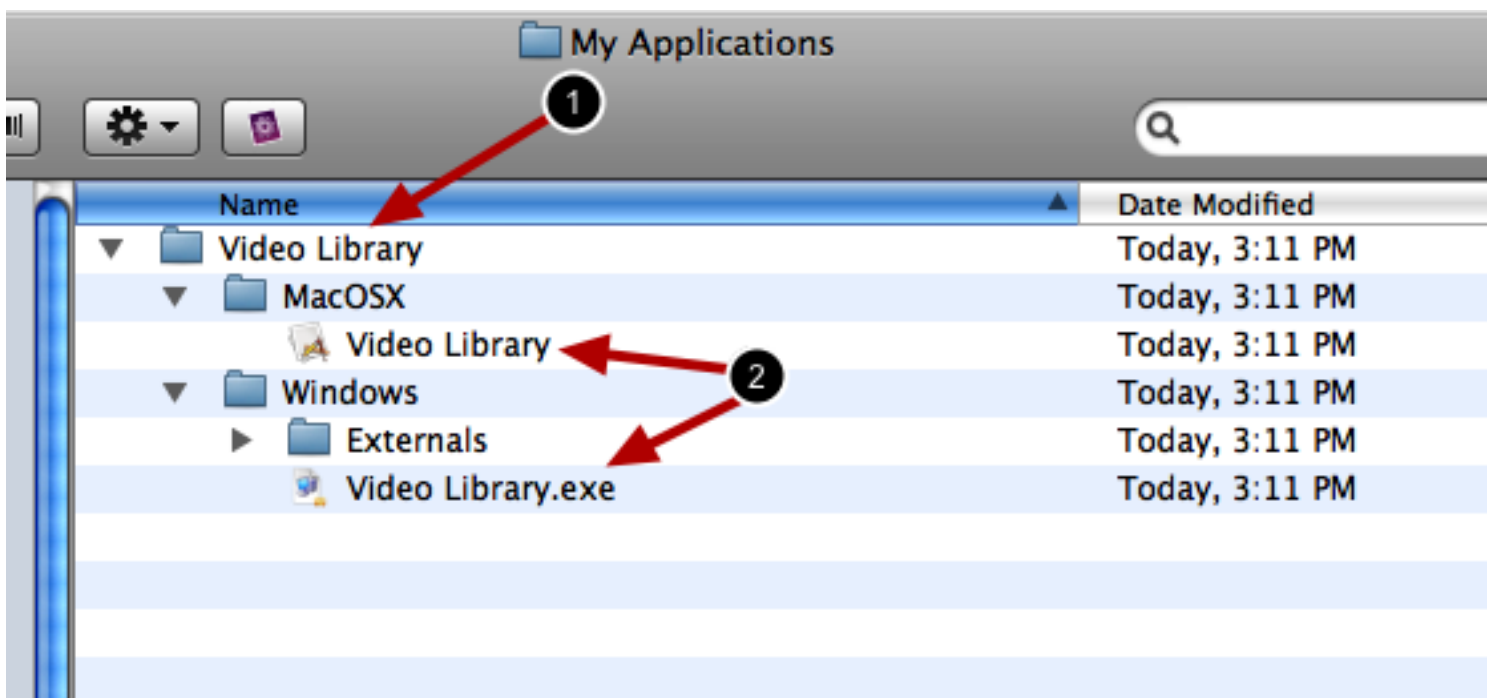
If you haven't saved your **Video Player** stack lately then Revolution will prompt you to save before the executable is created. Click the **Save** button.

Success



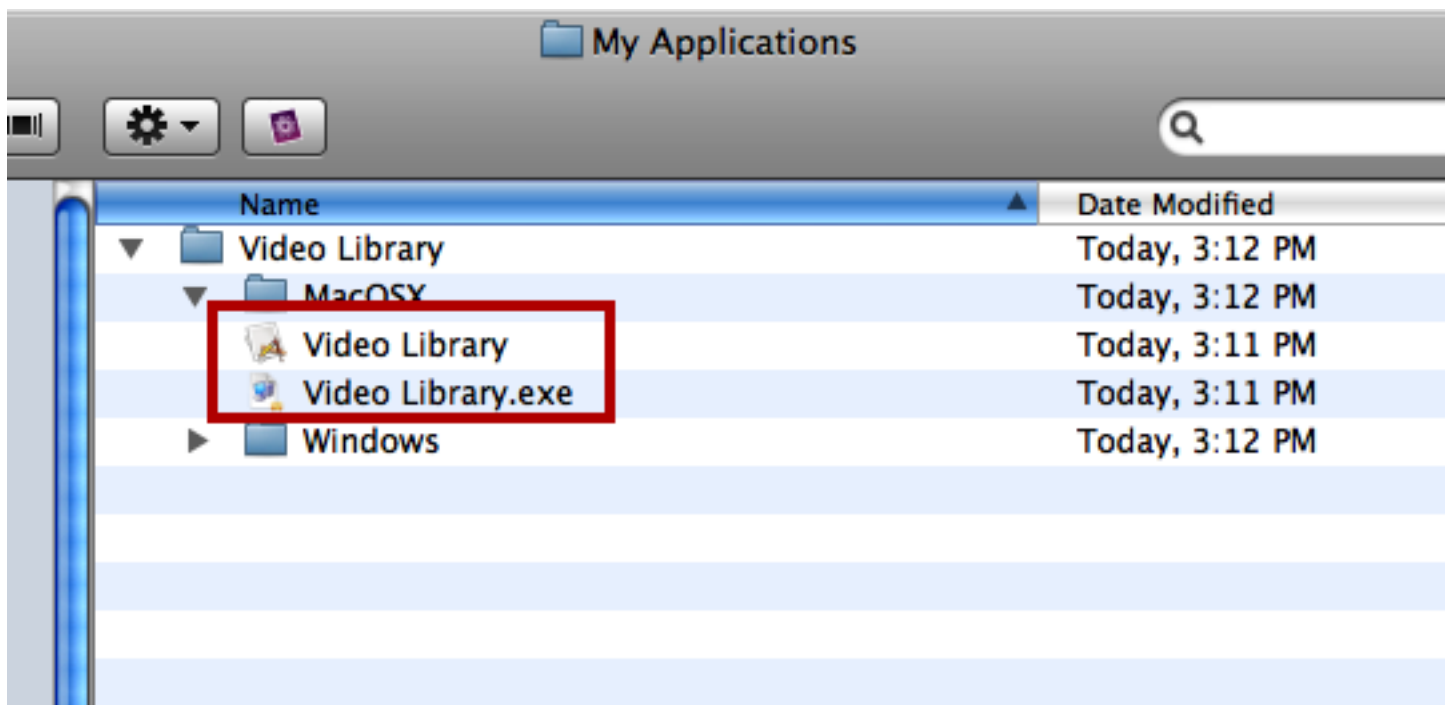
Revolution will display a progress window during the building process. When finished this dialog will appear. Click **OK**.

The Result



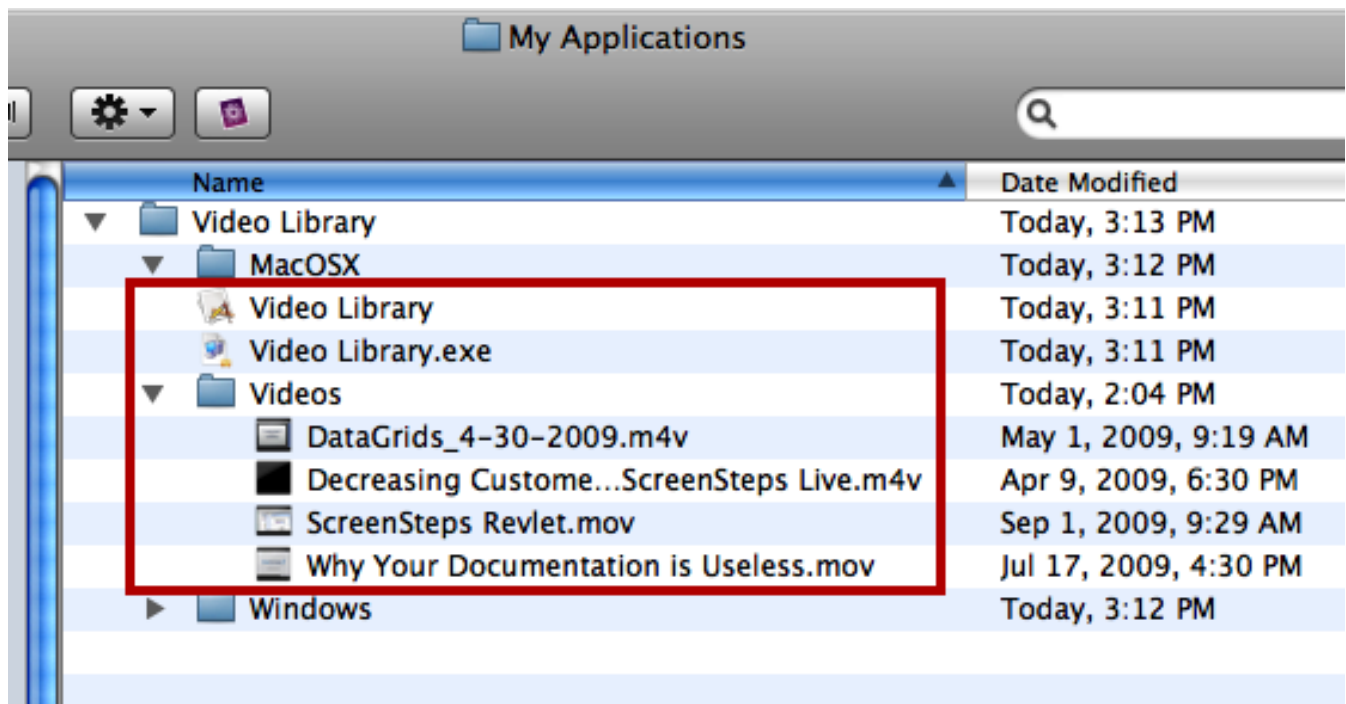
If you look in the folder you built the executable in you will find a folder named **Video Library** (1). In that folder you will find two other folders: **MacOSX** and **Windows**. In each of these folders you will find the executables for each platform (2).

Move Executables



Move the executables for each platform out of the **MacOSX** and **Windows** folders and into the **Video Library** folder.

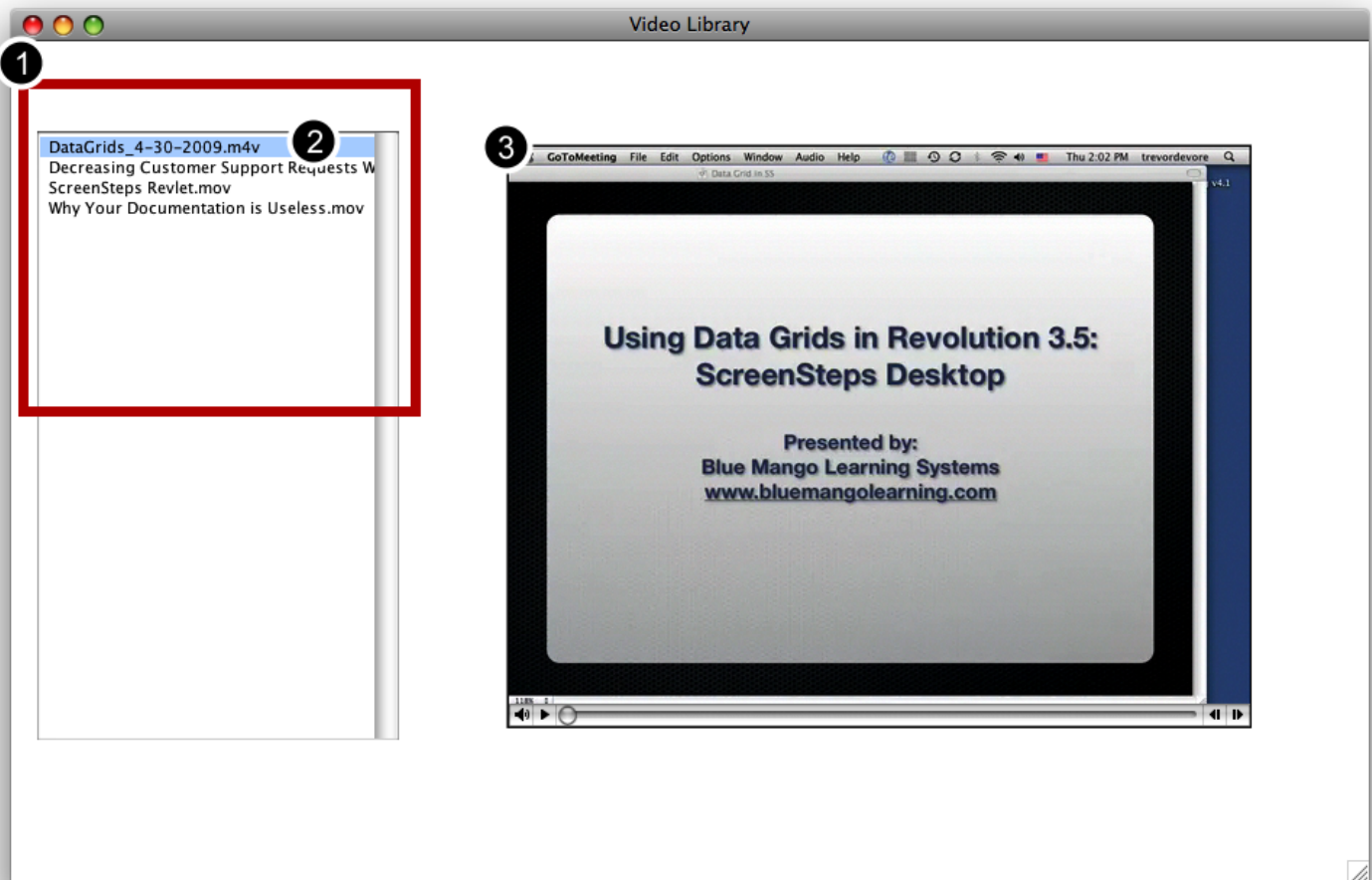
Copy Videos Folder



Now copy your **Videos** folder that you have been working with into the **Video Library** folder. It should now be alongside the Video Library executables for each platform.

You can safely delete the **MacOSX** and **Windows** folders at this point.

Test

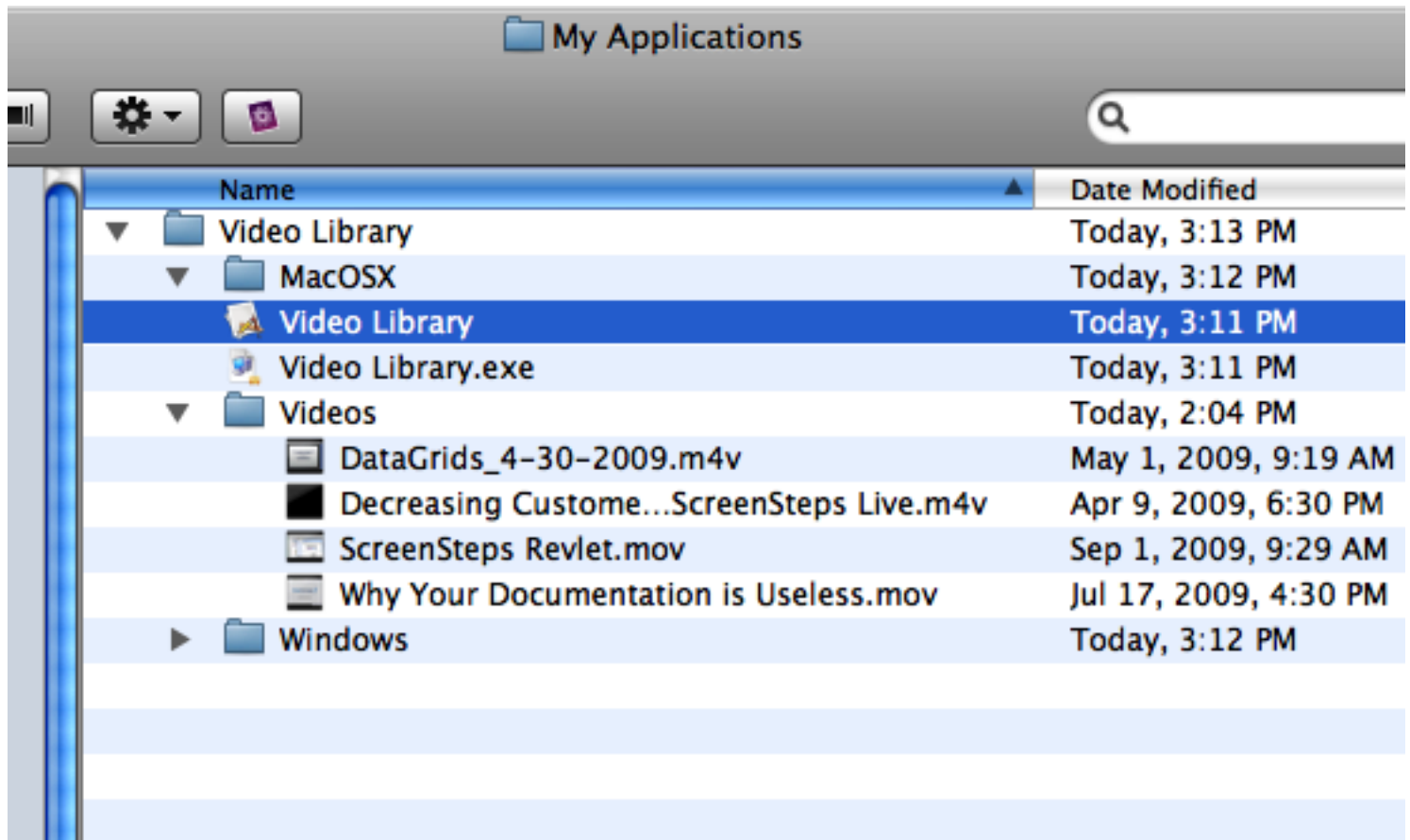


Double-click on the executable for the platform you are currently running on to launch it. The application should load and you should see the list of files in the Videos folder (1). The first video in the list should be selected (2) and the video should appear in the Player object (3).

Congratulations! You just created a Video Player application!

Sharing Your Application

What To Include



To share your application with others you just need to provide them with the executable for the platform they are running on and the **Videos** folder alongside it. Whoever you share the application with will only need to have QuickTime installed (guaranteed on OS X, optional install on Windows). Everything else is included in the executable file you created.