XOJO

User Guide

Book 4
Development

2015 Release 4

Xojo, Inc.

Preface

Xojo Guide

Book 4: Development

© 2015 Xojo, Inc.

Version 2015 Release 1

Section 1

About the Xojo User Guide

This *Xojo User Guide* is intended to describe Xojo for both developers new to Xojo and those with significant experience with it.

The *User Guide* is divided into several “books” that each focus on a specific area of Xojo: Fundamentals, User Interface, Framework and Development.

The *User Guide* is organized such that it introduces topics in the order they are generally used.

The Fundamentals book starts with the Xojo Integrated Development Environment (IDE) and then moves on to the Xojo Programming Language, Modules and Classes. It closes with the chapter on Application Structure.

The User Interface book covers the Controls and Classes used to create Desktop and Web applications.

The Framework book builds on what you learned in the User Interface and Fundamentals books. It covers the major framework areas in Xojo, including: Files, Text, Graphics and Multimedia, Databases, Printing and Reports, Communication and Networking, Concurrency and Debugging. It finishes with two chapters on Building Your Applications and then a chapter on Advanced Framework features.

The Development book covers these areas: Deploying Your Applications, Cross Platform Development, Web Development, Migrating from Other Tools, Code Management and Sample Applications.

Copyright

All contents copyright 2014 by Xojo, Inc. All rights reserved. No part of this document or the related files may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher.

Trademarks

Xojo is a registered trademark of Xojo, Inc. All rights reserved.

This book identifies product names and services known to be trademarks, registered trademarks, or service marks of their respective holders. They are used throughout this book in an

editorial fashion only. In addition, terms suspected of being trademarks, registered trademarks, or service marks have been appropriately capitalized, although Xojo, Inc. cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark, registered trademark, or service mark. Xojo, Inc. is not associated with any product or vendor mentioned in this book.

Section 2

Conventions

The Guide uses screen snapshots taken from the Windows, OS X and Linux versions of Xojo. The interface design and feature set are identical on all platforms, so the differences between platforms are cosmetic and have to do with the differences between the Windows, OS X, and Linux graphical user interfaces.

- **Bold type** is used to emphasize the first time a new term is used and to highlight important concepts. In addition, titles of books, such as *Xojo User Guide*, are italicized.

- When you are instructed to choose an item from one of the menus, you will see something like “choose File ↠ New Project”. This is equivalent to “choose New Project from the File menu.”

- Keyboard shortcuts consist of a sequence of keys that should be pressed in the order they are listed. On Windows and Linux, the Ctrl key is the modifier; on OS X, the ⌘ (Command) key is the modifier. For example, when you see the shortcut “Ctrl+O” or “⌘-O”, it means to hold down the Control key on a Windows or Linux computer and then press the “O” key or hold down the ⌘ key on OS X and then press the “O” key. You release the modifier key only after you press the shortcut key.

- Something that you are supposed to type is quoted, such as “GoButton”.

- Some steps ask you to enter lines of code into the Code Editor. They appear in a shaded box:

```
ShowURL(SelectedURL.Text)
```

When you enter code, please observe these guidelines:

- Type each printed line on a separate line in the Code Editor. Don’t try to fit two or more printed lines into the same line or split a long line into two or more lines.

- Don’t add extra spaces where no spaces are indicated in the printed code.

- Of course, you can copy and paste the code as well.

Whenever you run your application, Xojo first checks your code for spelling and syntax errors. If this checking turns up an error, an error pane appears at the bottom of the main window for you to review.

Table of Contents

7. Sample Applications

7.1. Sliders

7.2. Eddie’s Electronics

Chapter 1

Deploying Your Applications

In this chapter you will learn how to deploy your desktop and web applications.

CONTENTS

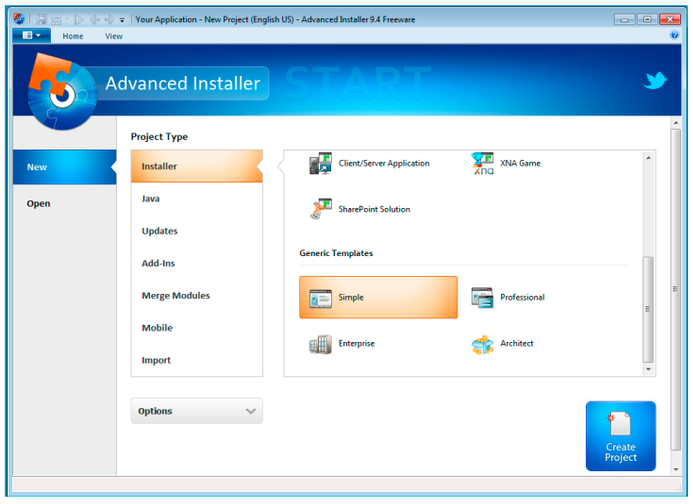
Windows Deployment

On Microsoft Windows, applications are deployed using installers.

Any installer tool will work on your apps, including: InnoSetup, Advanced Installer, NSIS and InstallShield.

***Note**: These installer tools all have to be run on Microsoft Windows in order to create a Windows installer. However, InnoSetup does work with WINE to allow it to be run on OS X or Linux.*

Figure 1.1 Advanced Installer



Installer Files

When you create your installer, you need to tell it to include all the files necessary to run the application. At a minimum, this includes the EXE file and the contents of its associated Libs folder.

For example, an application called Sliders would create a file called “Sliders.exe” and a folder called “Sliders Libs”. The Libs folder contains DLLs for libraries, plugins and other associated files needed by your application.

If your application has other support files or folders, such as a Resources folder, then make sure that your installer includes them as well.

Setup.exe or MSI

Most installer tools allow you to create your installer as a Setup.exe file or as an MSI (Microsoft Installer) file.

Either work fine, but MSI files have the advantage of being the current recommended method from Microsoft and can be used by IT departments for better control of installations. Choose what works best for your customers.

Location and Shortcuts

Windows applications are installed to the Program Files folder. On 64-bit systems, your applications are installed to the Program Files (x86) folder (because they are 32-bit).

Windows users expect to have easy access to your application, so this means you should create easily accessible shortcuts.

Your installer tool should provide you with the option of creating a shortcut for the user on the Desktop and in the Start Menu.

Zip

For very simple distribution, you can Zip the application and its supporting files (such as the Libs folder). You can create a Zip by right-clicking on the parent folder in Windows Explorer and selecting Send To->Compressed (zipped) Folder.

Once unzipped, the application can be run from any location.

Although this can be useful for testing purposes, it is not recommended for proper Windows application deployment.

Microsoft Redistributable Files

Your applications require the Microsoft Visual C++ Runtime. The required files are included in the Libs folder (msvcp100.dll and msvcr100.dll) of your applications for your convenience.

However, for increased Windows compatibility, you should instead consider including the Microsoft Visual C++ Runtime Redistributable Installer as part of your installer.

To do this you do not include the msvcp100.dll and msvcr100.dll files that are in the Libs folder and instead embed the Microsoft Visual C++ Runtime Redistributable as part of the installer. Most installer tools have a way to do this automatically.

The advantage of doing it this way is that the Visual C++ Runtime Redistributable will install its files into the Windows system folder, which allows them to be updated by Microsoft Windows Update for security and other reasons.

If you leave your Visual C++ Runtime files in the Libs folder then they cannot be updated by Windows Update.

As of this writing, you can download the Visual C++ Runtime Redistributable from here:

http://www.microsoft.com/en-us/download/details.aspx?id=5555

Section 2

OS X Deployment

Generally, OS X applications consist of a single App file, called the **Application Bundle**. You can embed other files into the Application Bundle because it is technically a folder that OS X treats as a file. In Finder, you can right-click on any App and select “Show Package Contents” to see the actual contents of the Application Bundle as a folder.

Of course, your application can still have separate files not in the Bundle, in which case you want to make sure that your application is in a folder of its own.

Since applications are technically a folder, you have to include them in some sort of container in order to distribute them, such as disk images (DMG), installers and even simple Zip files.

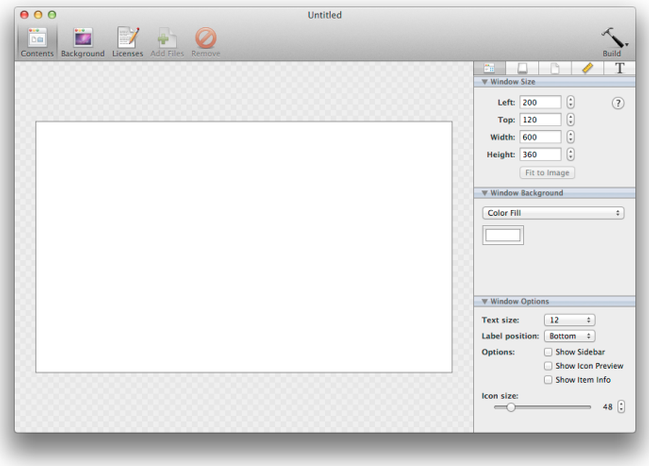
Disk Images (DMG)

A disk image is a file that the user downloads. After downloading the file (and double-clicking it), it appears in the sidebar of the Finder as a drive. This is called “mounting” the disk image. When the user clicks on the drive in the Finder sidebar, they see your application and typically drag it to the Applications folder to install it.

To create a disk image, you can use the Disk Utility application included with OS X or try one of the many specialized disk image creation tools such as DMG Canvas.

A disk image is probably the most common way to install OS X applications, but keep in mind that some users (especially those new to OS X) may find the concept of mounting a drive

Figure 1.2 DMG Canvas



and dragging a file to the Applications folder very confusing.

Also keep in mind that if the user mistakenly tries to run the Application directly from the disk image, it may not behave as expected because the disk image is read-only.

Installers

OS X can also use an actual installer to install your application, but it is not common for applications distributed outside the Mac App Store. To create an installer you can use the PackageMaker tool (included with the OS X Development tools) or you can use the free Packages installer.

An installer gives you more control over permissions and other settings, but they can be much more difficult to create than a simple disk image.

Also, remember that an installer on OS X (a pkg file) is actually a bundle so you have to distribute it in a disk image or a Zip.

Zip

A Zip file is an archive of your application. A zip is easy to download and most users understand what they are. They can usually be unzipped by simply double-clicking on them, which reveals the application itself. The application can then be copied to the Applications folder.

You can create a Zip by right-clicking on your application in the Finder and selecting Compress.

Code Signing for GateKeeper

OS X 10.8 introduced a new feature called GateKeeper. GateKeeper is designed to provide a level of security for users installing applications. Essentially, if the application (or its installer) is not digitally signed using an Apple-provided certificate then OS X will not allow the application to be installed by default.

This can be overridden by the user right-clicking on the application (or installer) and selecting Open, but not many users will know about this.

This means you are probably going to want to digitally sign your OS X applications. To get a certificate, you have to join the Mac Developer Program ($99). Your certificate is good for five years.

You use Certificate Utility in the Mac Dev Center to create a certificate (you can use the same certificate for all your applications).

Once you have followed the instructions to create the certificate and have installed it on your computer, you can use it to code sign your application using this command:

```
codesign -f -s "Developer ID Application:
YourName"  "YourXojoApp.app"
```

Code signing must be done as the absolute last step. If you modify anything inside your application bundle (such as Info.plist), you will invalidate the signature and have to code sign again.

***Note**: If you are using an installer then you have to sign the installer separately using a special installer certificate.*

Mac App Store

The Mac App Store is a great way to distribute your OS X applications. People find it easy and convenient to purchase applications from the Mac App Store. Unfortunately, getting applications into the Mac App Store is not easy or convenient for the developer.

Sandboxing

Sandboxing is used to restrict what your application is able to access. This serves as a security feature, because if an app were to become compromised for some reason then it would be unable to do as much damage as if it had full control of the computer.

Sandboxing works best with Cocoa applications, although it does work for Carbon applications (with some Apple restrictions).

Certificates and Code Signing

To be able to submit an application to the Mac App Store for Apple to review, you need to first create your Mac App Store certificates using the online Certificate Tool that is part of the Mac Dev Center. You also need to create a bundle ID for your application.

When your application is complete, you then need to code sign your application (and all its dynamic libraries). You also need to create the installer and code sign that as well.

Finally you can create a submission using iTunes Connect, fill in all the required information and then upload your application using the Application Loader tool that is part of the OS X Development Tools.

Once you have submitted something to the Mac App Store, it can take several weeks before your application is approved by Apple and ready for sale.

Validating The Apple ID

The above steps work great for any apps, including free apps, but if you are selling your app, you will want to prevent people from copying the purchased app to another computer and running it there (without having to log into their Apple ID). To do this you also need to verify the Apple ID.

Verifying an AppleID requires calling a Cocoa API. The code is far too involved to include here, but there is a sample project in the “Platform Specific” folder for OS X that you can reference.

Section 3

Linux Deployment

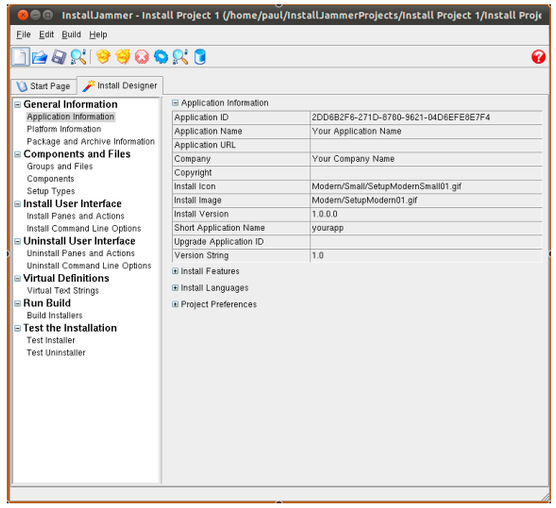
Linux applications also have a variety of deployment options. The simplest is to use GZip and let the user put the application wherever they want. You can also use a tool such as InstallJammer that creates a generic installer that works with a variety of Linux platforms. Lastly, you can create separate installers for each Linux distribution. Common installer formats are deb (used by Debian and Ubuntu) and RPM (RedHat Package Maker) used by RedHat.

Generic Installer

InstallJammer is an open-source product that has a simple user interface for creating an installer that works on a variety of Linux distributions.

Unfortunately, this tool is no longer being actively developed, but it still works well and is a good choice if you do not use Linux often enough to master creating dedicated installers.

Figure 1.3 Install Jammer



Debian Installer

Debian installers are used by Debian-based Linux distributions, such as Ubuntu. They can be installed by the Synaptic Package Maker or from the terminal.

You create Debian installers using the **`dpkg-deb`** terminal application. Unfortunately, its usage is far more involved than can be discussed in this book.

This tutorial describes how you can create a Debian package:

http://tldp.org/HOWTO/html_single/Debian-Binary-Package-Building-HOWTO/

Redhat Installer

The Redhat installer format (RPM) is used by Redhat-based Linux distributions.

You can create RPM installers using the rpmbuild terminal application. Unfortunately, its usage is far more involved than can be discussed in this book.

The Fedora Project does have a good walkthrough, which is available here:

http://fedoraproject.org/wiki/How_to_create_an_RPM_package

Web Deployment

You can create two types of web applications, Stand-alone and CGI. In addition to differences in how they operate, they are also deployed differently.

Stand-alone Application

When you create a stand-along web application, you get a single application that consists of both your application and a standalone web server. This web server listens on the port specified in the Build settings.

To start a stand-alone web application, you simply run the application from the command line or terminal. This command runs a standalone web app on OS X or Linux:

```
./mywebapp
```

You can run multiple stand-alone web applications on the same server, but each web application has to be listening on a unique port. To make this easier, you can specify the port as a command-line parameter:

```
./mywebapp --port=8080
```

If you have the port set at 8080 for example, then you can access the web server using the URL followed by the port like this:

http://www.myserver.com:8080

Command Line Parameters

You can use these command-line parameters to change settings when you launch the web application.

Command	Description
--Port	The port the web app listens to for a connection. This can be used to change the port that was used to build the web app.
--MaxSockets	The maximum number of allowed connected sockets. This is not the number of maximum users as the number of connections does not map one-to-one with the number of connected users. Do not set the value too high as it will increase memory and CPU usage. The default is 200. If you do not want any unsecured connections, set this value to 0.
--SecurePort	The port the web app listens to for a secure connection. This can be used to change the port that was used to build the web app.
--MaxSocketsSockets	The maximum number of allowed secure connected sockets. This is not the number of maximum users as the number of connections does not map one-to-one with the number of connected users. Do not set the value too high as it will increase memory and CPU usage. The default is 200.
--NetworkInterfaceIndex	The index value (of NetworkInterfaces) to use as the NIC for connections.
--SecureNetworkInterfaceIndex	The index value (of NetworkInterfaces) to use as the NIC for secure connections.

Using Port 80

In order to navigate to a stand-alone web application without specifying the port, you need to use port 80, the standard port used by web browsers. However, OS X and Linux do not allow you to use port 80 (or any port lower than 1024) unless you have root access.

This means that on OS X and Linux you need to use the *sudo* command to start your web application if you are using port 80:

```
sudo ./mywebapp
```

***Note**: The sudo command prompts you for your user name and password.*

Background Processing

When you run your web application directly from the command line, it will continue running as long as you don't quit the command/terminal window or log out of the account.

This technique does not work well on true servers or remote servers. In these cases you want to run the web application as a background process. On Windows this means, you would run the web app as a Windows Service. On Linux it means you would run it as a Daemon. On OS X, you can run it as a daemon or use the preferred launchd command to start it as a daemon.

Deploying with SSL

In order to deploy a standalone web application so that it uses SSL, you first need an SSL certificate. You can use a self-signed certificate for local testing, but always purchase a real certificate for publicly available web sites.

The certificate file used by Xojo is a text file containing the components of your SSL certificate. Generally, a self-signed certificate will have two components (the certificate and key file) while a purchased certificate will likely have three or more (certificate, key, CABundle or intermediates). Create the text file with the same name as your application (minus the app extension if any) and add the ".crt" extension. Then paste the contents of your certificate files in this order:

1. Certificate
2. CABundle
3. Private Key

Start each file on a new line.

This Xojo certificate file must be placed next to the built web app when it runs. This can be done using a Copy Files Build Step with Build Automation.

Now you can start your web app and tell it to listen for connections on a secure port. This uses the *secureport* and *maxsecuresockets* command-line parameters described in the preceding section. The secure port must be different than the (non-secure) port selected in the Shared Properties within Xojo. It also must be different than a port specified with the port command-line parameter.

For example, if you want to launch your web app on secure port 8081, use this command:

```
MyWebApp --secureport=8081
```

Once you have launched the app, you can connect to it in the browser like this (for a web app running locally):

```
https://127.0.0.1:8081
```

If you also want to prevent unsecured access to the web app, you can set the number of unsecured sockets to 0:

```
MyWebApp --secureport=8081 --maxsockets=0
```

WINDOWS SERVICE

To run a web application as a Windows Service, you use the sc command from the Command Line to install the app as a service:

```
sc create MyWebAppSvc type= own start=
auto binpath= c:\Path\To\Exe\mywebapp.exe
```

Now you can go to the Services Manger in the Control Panel and see the “MyWebAppSvc” service listed. Use Services Manager to Start, Stop or Pause the service.

DAEMON

To make your stand-alone web application run as a daemon background process, you call the *Daemonize* method, usually in the **App.Open** event:

```
#If Not TargetDebug Then
  If Not Daemonize Then
    Print(“Unable to Daemonize app.”)
    Quit
  End If
#Endif
```

When you now run the stand-alone web application from the terminal, it will immediately return you to the terminal prompt because the app is now running in the background.

This also works on OS X, but Apple prefers that you use the *launchd* command to start daemons. Using *launchd* means you have to create a Property List file with the specific settings.

For more information, refer to Apple’s documentation on this:

http://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingLaunchdJobs.html

CGI Application

When you build your web application as a CGI application, you get an executable file that communicates to your existing web server using CGI (common gateway interface).

The way this works is that a small Perl CGI script is created when you build your web application. Its only purpose is to communicate between your web application and the web server. The Perl script starts your web application if it is not running and routes all communication between the web server to your web application and vice versa.

When you build your web application for CGI, you have the option of choosing a port or having it be chosen automatically. This is an internal port that is used for communication between the Perl script and your web application. Because this is internal communication, the firewall is not affected. The only requirement is that nothing else on the server is also using the port.

Nearly all web servers support CGI and Perl, but some are much easier to configure than others.

Apache

The most common type of web server you will see is Apache. This is typically preinstalled on all Linux servers.

Most Apache servers have a specific location where CGI applications should be uploaded. This is often called the “cgi-bin” folder.

On a properly configured web server, deploying a web application is as simple as using your FTP client of choice to copy the entire web application folder’s contents to the cgi-bin folder (be sure to enable Binary mode for the transfer). The files include:

- The web application itself
- The Libs folder
- config.cfg
- the Perl CGI file
- The .htaccess file (which may be hidden in the default view of your FTP software)

Then you can start the web application by accessing the cgi script:

http://www.myserver.com/mywebapp.cgi

Troubleshooting

Unfortunately, not all servers are going to be properly configured to run Xojo web applications. You may find that you have to change the configuration yourself. These are some common areas to check. Unfortunately, step-by-step instructions are not

possible because every installation of Apache is different, having configuration files in different places with different web server users and different permissions. Use these tips as guidelines, but you'll still need an understanding of Apache and how it has been installed on the OS you are using.

- Platform: Verify that you uploaded the correct platform for your web application. Although you may be developing and testing on OS X, if you are using a Linux server you need to remember to make sure that you create a Linux build to upload.

- 32-bit Libraries: Your application is a 32-bit application. To run it on a 64-bit version of Linux, you need to ensure the 32-bit compatibility libraries are installed. The command to do this varies depending on the Linux distribution. With Debian, you can use apt-get:

```
# apt-get update
# apt-get install ia32-libs-multiarch
```

- Application Identifier: Verify that the Application Identifier is unique. If there is another web app running on the server with the same Application Identifier, then your new app will not start.

- Permissions: Verify that your CGI application and all the libraries in the Libs folder have been copied to the server and have execute permissions (755 is best). You may also need to check the parent folder as well the config.cfg and Perl CGI script. Permissions of 777 are not secure and some web servers (that use seEXEC) will not run anything with that permission.

- AddHandler: If your web application displays the page source rather than running, you may need to add the command "AddHandler cgi-script .cgi" to your Apache configuration file or to the .htaccess file generated with your web application.

- Internal Server Error: If the Perl CGI script is having trouble starting you may see this error. Verify that Perl is installed correctly and permissions are correct.

- Unable to Launch Application: The Perl CGI script is unable to start your application. Usually you will see additional information such as "permission denied". Verify that your web application was uploaded using FTP Binary mode (and not ASCII, which can corrupt the application files).

- Unable to Connect to Port: The port may be blocked for some reason. Perhaps the firewall is blocking a local port or the port is in use by another application or service. If you have selected a port, make sure it is > 1024 and < 65536.

- Web Server Logs: Your web server logs may have additional useful information. Unfortunately, the location of these logs varies depending on the Linux distribution and version.

IIS (Microsoft Internet Information Services)

Included with Windows is a web server called IIS. This web server is ideally suited for running simple HTML web sites, .NET web applications and other specific Microsoft web products.

By default, IIS does not have a Perl installation and does not work well with CGI.

Although Xojo web applications do work with IIS, properly configuring it is an advanced task. Unless you absolutely require IIS, it is recommended you instead configure Apache to run on Windows and use it instead.

These are the official instructions for installing Apache on Windows:

http://httpd.apache.org/docs/2.2/platform/windows.html

Deploying with SSL

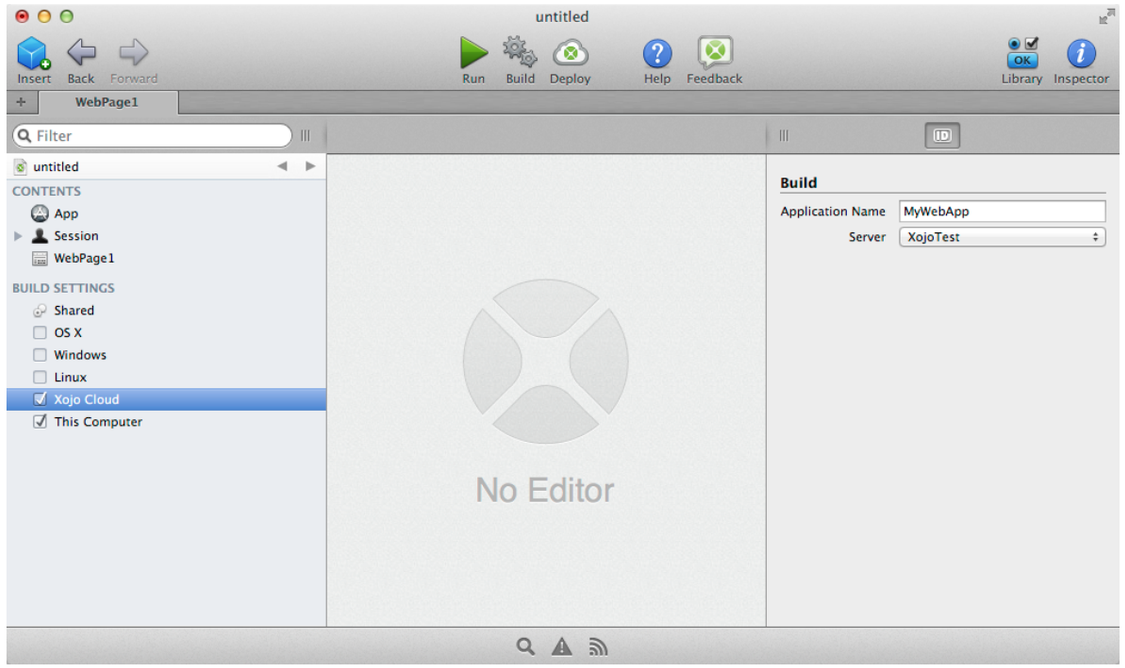
Because a CGI app uses a separate web server, you have to ensure your web server (typically Apache) is configured to work with SSL and your SSL certificate.

You do not have to do anything specific with your Xojo CGI web app in order to deploy it as SSL.

Xojo Cloud

If all this setup and configuration is too much to bother with, you should consider hosting your web application using Xojo Cloud. This service allows you to deploy a web application directly from Xojo in one step.

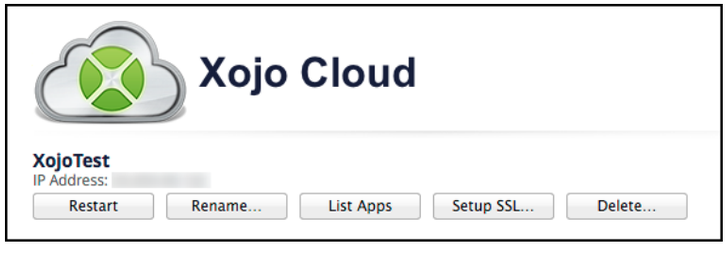
Figure 1.4 Xojo Cloud Build Setting



With Xojo Cloud, you just check the “Xojo Cloud” target in Build Settings for your web project and then select it to specify the name of the application and the Xojo Cloud server on which to deploy. Then you simply click the Deploy button in the toolbar and your app is built and uploaded to your Xojo Cloud server.

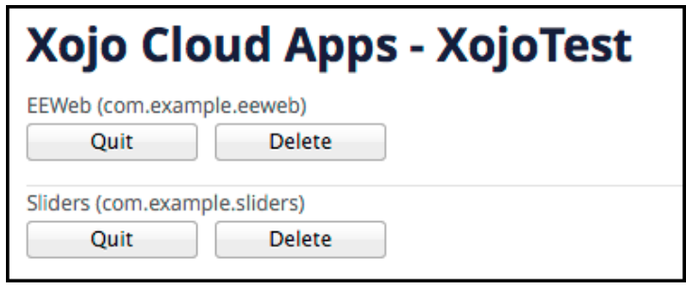
Control Panel

You can purchase a Xojo Cloud server from the Xojo Store. Once it is provisioned, you can access the Control Panel from your Accounts page on the Xojo web site.



The Control Panel displays your servers (including the IP address) and can performs these actions:

- Restart: Restarts the Xojo Cloud server.
- Rename: Give the server a new name.
- List Apps: Displays all the Xojo apps that have been deployed to the Xojo cloud server. From here you can Quit or Delete apps.



- Setup SSL: Walks you through the process of getting a CSR (Certificate Signing Request) to use to purchase an SSL Certificate that you can then use on your server.
- Delete: Deletes the Xojo Cloud server.
- PostgreSQL: Enables or disables PostgreSQL database server, providing you with a username and password.
- MySQL: Enables or disables MySQL database server, providing you with a username and password.
- Tunnel: Enables or disables an SSH tunnel (providing you with a username and password) to allow you to connect to PostgreSQL/MySQL from your own computer.

Deployment

The built app's name has a suffix if the Stage Code (in the Shared Build Settings) is set to Development (-Dev), Alpha (-Alpha) or Beta (-Beta).

This allows you to deploy apps for testing without affecting an app that is running in production.

Set the Stage Code to “Final” to remove any suffix.

PostgreSQL and MySQL Usage

Once you enable PostgreSQL or MySQL database on your Xojo Cloud server, connecting to the database in a deployed app works exactly as if the DB was on your local computer. There are no special steps to do.

If you want to access a DB on Xojo Cloud from your computer, you will have to enable the Tunnel and then connect to the tunnel from the computer. On OS X and Linux, you can use this command to access the tunnel:

```
ssh -L 5432:localhost:5432 dbadmin@ipaddress -N
```

If you are using Windows, you will need to use an external app such as PuTTY. There are a number of tutorials available on the internet, just put “putty ssh tunnel” into your favorite search engine.

Firewall

When using classes that have to communicate outside the server (database or socket classes, for example), you need to use the XojoCloud.FirewallPort class to first open the port.

```
Dim fwp As New XojoCloud.FirewallPort(587, _
  XojoCloud.FirewallPort.Direction.Outgoing)
fwp.Open() // This call is synchronous
If fwp.isOpen() Then
  // Do what you need to do
  fwp.Close() // Close port when done
End If
```

Section 5

iOS Deployment

All methods of iOS deployment require an iOS Developer membership with Apple. You can sign up at http://developer.apple.com.

Device Deployment for Testing

Running in the iOS Simulator is fast and convenient, but it does not work exactly like an iOS device. To build for a device, you need to create a certificate, App ID, devices and a Provisioning Profile in the iOS Dev Center. You can find information on how to do this in the online documentation:

https://xojo.helpdocsonline.com

App Store Deployment

Once your app has been tested, you can submit it to the review process for the App Store. To do this you need additional certificates and provisioning profiles. Plus, you will need to set up the app information in iTunes connect. You can find information on how to do this in the online documentation:

Online Documentation

Enterprise Deployment

For apps that you want to distribute to your organization, but do not want to submit to the App Store, you can the Enterprise deployment option and the Apple Configurator tool to install the apps on the devices in your organization. You can find information on how to do this in the online documentation:

Online Documentation

Chapter 2

Cross-Platform Development

This chapter describes how you can create cross-platform applications. Topics include, user interface layout, conditional compilation, localization, and OS-specific features.

CONTENTS

Section 1

User Interface Layout

When designing your applications to be cross-platform, you should keep in mind the differences in user interface across the different platforms.

An application that is well designed for Windows may look drastically out of place on OS X or Linux. And vice versa.

Menus

ApplicationMenuItem

Desktop applications usually have an About menu that displays an About window with the name of the application and its copyright information.

On Windows and Linux, this About menu appears in a Help menu, which you can add by clicking the “Add Menu” button in the Menu Editor.

On OS X, the About menu instead should appear in the Application menu.

To make your About menu automatically move on OS X, you add your About menu to the Help menu so that it appears as expected for Windows and Linux. You then change its Super property in the Inspector from “MenuItem” to “ApplicationMenuItem”. Any menu that uses this class will be automatically moved to the Application menu when it is run on OS X.

PrefsMenuItem

Similarly, the Preferences menu is also located in the Application menu on OS X. On Windows and Linux, the Preferences menu is often located in the Edit menu and is instead called “Options”.

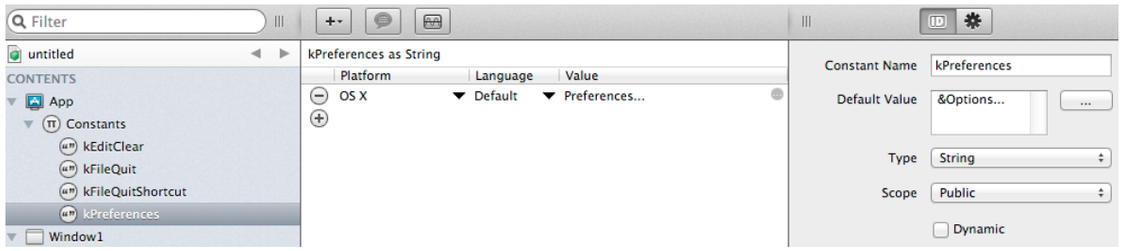
There is always a fixed Preferences menu in the Application menu on OS X. To attach your Preferences menu to it, you set its Super property to “PrefsMenuItem”. Only one menu in your project should be set to PreferencesMenuItem.

To change the name of the Preferences menu for OS X and Windows/Linux, you would use a constant. By default there are several constants on the App class that control the text for Edit->Clear/Delete and File->Quit/Exit. You can add another to handle preferences.

Add a new constant and call it kPreferences, setting its default value to “&Options...”.

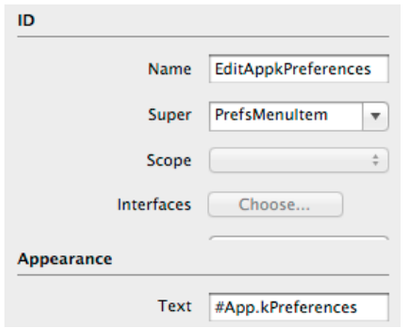
In the Constant Editor, click the “+” to add a new entry and select “OS X” as the Platform. For the Value, enter “Preferences...”

Figure 2.1 Adding a Preferences Constant



Now that you have created the constant, you can use it as the text of the menu. Add a new MenuItem to the Edit menu and set its Text property to “#App.kPreferences”. This tells it to use the value of the constant. Also set its Super to “PrefsMenuItem”.

Figure 2.2 Preferences Menu Properties



You can use the preview buttons in the Menu Editor toolbar to see the text change between OS X, Windows and Linux.

In addition, when you run the application on OS X, the Preferences menu appears in the Application menu instead of the Edit menu.

Dialog Buttons

Perhaps you have never noticed it, but when you use a dialog box on Windows and Linux, the buttons are in a different order than they are on OS X.

On OS X, the default OK/Cancel buttons display as Cancel followed by OK. On Windows and Linux they appear as OK followed by Cancel.

For your applications to look proper on each platform, the buttons should appear in the appropriate positions. The easiest way to do this is to us a ContainerControl to swap the buttons for you at run-time.

Figure 2.3 Print Dialog Showing Cancel Followed by Print (OK) Button

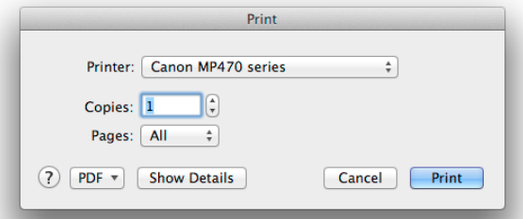
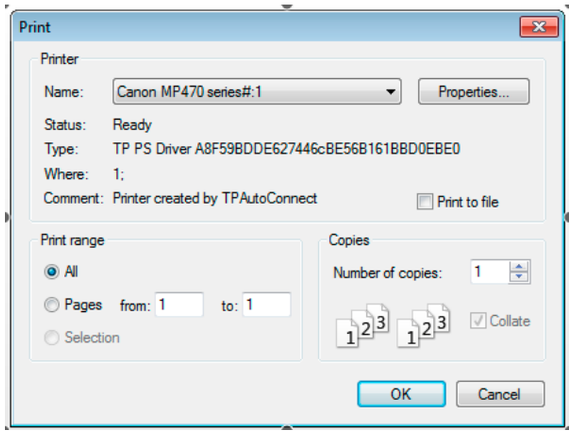


Figure 2.4 Print Dialog on Windows Showing OK Followed by Cancel Button



The example project OKCancelContainer in the Desktop->Custom Controls folder demonstrates how to do this.

Fonts

Normally you will use the System font as the Font for your controls. This tells your application to use the default system font as the font for the control. As you might expect, the default system font varies by platform.

This means that some controls may end up being too small on some platforms to fit the text you provided. It is important that you run your project on each platform and review the layout to make sure that everything is readable and fits as you expect.

You may find that you need to increase the size of some controls so that they display properly on all platforms. You can do this in the Layout Editor by increasing the size of a control. Or you can do it at runtime by increasing the size of the control in its Open event depending on the platform being used (see the Conditional Compilation section). This code in the Open event handler of a PushButton increases its size when running on Linux:

```
#If TargetLinux Then
  Me.Height = Me.Height + 20
#Endif
```

Graphics and Flickering on Windows

OS X and Linux use a technique called double-buffering for all window drawing. This means that updates to a window are done offscreen and then shown on the actual screen in one update. This results in stable, flicker-free windows and graphics updates.

Windows, however, does not use this technique. It sends updates to the Window immediately. This can often result in unsightly flicker when running an application on Windows, but there are strategies to mitigate this.

Do not Overlap Controls

The easiest thing you can do to prevent flickering is to not overlap any controls. Overlapped controls result in more requests to redraw the controls which results in flickering.

Use a Canvas

For best results, display any graphics using the Paint event of a Canvas control. Stay away from using the Window Paint event, the Canvas.Backdrop property or the ImageWell control. Although those techniques work fine in certain situations, they often lead to flickering in more complex window layouts.

On the Canvas, the first thing you want to do is enable the DoubleBuffer property and disable the EraseBackground property. The DoubleBuffer property allows the Canvas to do its updates offscreen to minimize flicker. The EraseBackground property prevents the Canvas from being erased (and showing as

a white rectangle) before it is redrawn, which is a common source of flicker.

Now you can do all your drawing in the Paint event using the supplied graphics object, g.

***Note**: Do not do any drawing directly to the Canvas.Graphics property. This will likely increase flickering and will definitely slow down graphics updates.*

You can have separate methods that update the graphics, but they need to be called from the Paint event with the graphics object supplied to the methods as a parameter.

When you want to update the graphics in the Canvas, you call the Invalidate method:

```
Canvas1.Invalidate(False)
```

You can also call the Refresh method:

```
Canvas1.Refresh(False)
```

The difference is that Invalidate tells the Canvas to update itself when it gets a redraw request from the operating system. The Refresh method tells the Canvas to update itself immediately. Normally you want to use Invalidate.

Both of the above have an EraseBackground parameter, which defaults to True. This means that the Canvas is erased before the contents are redrawn. In most cases, you do not want the erasing to occur, so you should specify False as the EraseBackground parameter.

By using these techniques, you can create stable, flicker-free graphics in your Windows applications.

Section 2

Conditional Compilation

In the process of creating an application that runs on multiple platforms, you may find that you have some code that is only needed on a single platform.

An example of this might be code that saves preferences. On Windows you might want to use the Registry, but on OS X you might want to use a plist.

You can handle these special cases using conditional compilation.

Conditional Compilation

With conditional compilation, you are telling the compiler to include or exclude specific parts of your code depending on what is being compiled. This is done using the #If command in conjunction with #Else, #ElseIf and #Endif.

You can use these commands along with any constant to selectively include or exclude code when building.

These are some of the built-in constants you can use:

- **TargetMacOS**: True when doing a build for OS X, False for anything else.
- **TargetCarbon**: True when doing an OS X Carbon build, False for anything else.
- **TargetCocoa**: True when doing an OS X Cocoa build, False for anything else.
- **TargetHasGUI**: True for desktop and web applications, False for console applications.

- **TargetLinux**: True on apps built for Linux, False for anything else.
- **TargetWeb**: True for web applications, False for anything else.
- **TargetWin32**: True on apps built for Windows, False for anything else.
- **TargetCloud**: True on apps deployed to the Xojo Cloud, False for anything else.
- **TargetRemoteDebugger**: True on apps running through the Remote Debugger, False for anything else.
- **DebugBuild**: True when your app is running in the Debugger, False when it is running as a standalone build.
- **VersionString**: Returns the version being used in the format: 2014r1

The constants can be used like this:

```
#If DebugBuild Then
  // Activate logging
  App.Logging = True
#Else
  App.Logging = False
#Endif
```

This structure could be used to handle saving preferences differently:

```
#If TargetWin32 Then
  // Use Registry
  SavePreferencesToRegistry
#ElseIf TargetMacOS Then
  SavePreferencesToPList
#ElseIf TargetLinux Then
  SavePreferencesToXML
#Endif
```

You can even use your own constants, such as this to check if a beta has expired:

```
#If App.BetaBuild Then
  // Check if expired
  Dim now As New Date
  If now.Year > 2012 Then
    Quit
  End If
#Endif
```

Selecting a Target for a Class/Method in Inspector

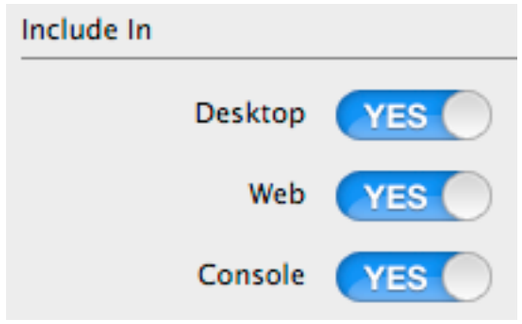
You can specify if a particular project item (excluding Windows) should only be included for certain project types.

For example, you may have a class that you only want to include when you are creating a web application.

You do this using the “Include In” properties in the Advanced tab of the Inspector. To get to the Advanced tab, click the gear button:  .

In the inspector, you now see the Include In section, which allows you to specify the targets to include. Your choices are Desktop, Web and Console. Select No to exclude the class (or selected method) from the appropriate target. By default, each target is set to Yes.

Figure 2.5 Include In Targets



You can also set “Include In” for methods, properties, constants, event definitions, enums, structures, delegates and anything else you can add to a project item.

OS X Features

Cocoa

Cocoa is the current user interface framework supported by Apple on OS X.

If you are moving an older project from Carbon (an older OS X UI framework) to Xojo, your existing applications should just work for the most part, but there are some differences to consider.

General UI Differences

- Resizable windows can be resized from any window edge.
- Spellchecking and grammar checking are now available in TextField and TextArea controls by setting AutomaticallyCheckSpelling property to ON in the Inspector.
- PushButtons have a ButtonStyle property than can be used to change the look of the button.
- The Window.Composite property is ignored since Cocoa draws all windows as composite.
- Windows in your apps can be dragged around the screen while code is running.
- Timers, threads and sockets continue to get events when menus or OS modal dialog boxes are displayed.
- StyledText draws much faster.

Graphics

All drawing using a Canvas must be done from the Paint event or a method called from it. Cocoa is very restrictive about this. If you try to access the Graphics object of a Canvas directly, you will drastically slow down the display performance of your app.

You should also use Canvas.Invalidate to refresh the Canvas display instead of Canvas.Refresh. The Refresh method tells the Canvas to redraw itself immediately, which can slow things down by redrawing too often. Invalidate tells the OS to redraw the Canvas when it is ready, which is more efficient.

All drawing now takes place in a ‘generic’ colorspace that is independent of the current output device. This means that colors will render more similarly between screens and printers.

Threads

Cocoa does not allow any code that is in a thread to modify anything related to the user interface (which runs on what is

called the main thread). If your user interface needs to be updated based on information in a thread, you should instead have the UI request the information from the thread.

The best way to do this is to use a Timer that periodically updates the UI based on information that it gets from the thread.

MenuItems

Cocoa does not allow the same MenuItem to be used in two different places. To work around this, simply use the MenuItem.Clone method to create a copy of the MenuItem instead.

Font Display

Cocoa is more restrictive about the font styles that it displays. In particular, if a font does not have built-in bold or italic variations, then the font cannot be displayed as bold or italic. The Bold or Italic properties will have no effect in this case.

To check if the font you wish to use has the variation you need, use the Font Book application included with OS X.

Keyboard Handling

Unhandled KeyDown event handlers cause a Beep instead of behaving silently. This is intentional and gives the user feedback that their action did not do anything. To eliminate the Beep, simply Return True from the KeyDown event handler.

Additionally, the KeyDown event handler now literally means “a key was pressed” and does not attempt to interpret the key event for dead key sequences or input methods. For example, pressing Option+E and then E again results in two key presses sent to KeyDown instead of a single key press with the value “é”.

Unsupported Items

- Drawer windows
- ResourceFork class
- Cursors stored in the resource fork
- TextArea.Save method

AddressBook

The AddressBook classes are used to access Address Book data in OS X. The available classes are:

AddressBook, AddressBookAddress, AddressBookContact, AddressBookData, AddressBookGroup.

The AddressBook class gives you access to the AddressBook. You can also access the AddressBook using System.AddressBook:

```
Dim book As New AddressBook

// or

Dim book As AddressBook
book = System.AddressBook
```

To get the contacts in the Address Book, use the Contacts property:

```
Dim contacts() As AddressBookContact
contacts = book.Contacts
```

Each contact has properties for the various fields, such as FirstName, LastName, etc. To add a contact, create it and then add it using the Add method:

```
Dim contact As New AddressBookContact
contact.FirstName = "Bob"
contact.LastName = "Roberts"
book.Add(contact)
```

AddressBookAddress is used to get address information for a contact. AddressBookData is used to get information such as email addresses, phone or fax numbers.

AddressBookGroup gives you information about all the groups in the address book. Use the AddressBook.Groups property to get an array of groups.

KeyChain

The KeyChain is an OS X feature used for storing account passwords for applications. By taking advantage of the keychain, your users do not have to type their password if their keychain is unlocked on their system.

Use the KeyChain class to access OS X Keychains for your applications. The classes are: KeyChain, KeyChainItem and KeyChainException.

You should always ask for permission from the user before storing anything in a keychain.

You use the System module to get a reference to the default KeyChain. You use the KeyChainItem class to create, update or find items in the keychain.

If you have more than one keychain, then you can use the KeyChain constructor to access specific key chains by number.

This code stores a password in the default Keychain:

```
Dim newItem As KeyChainItem
If System.KeyChainCount > 0 Then
  newItem = New KeyChainItem
  // Indicate the name of the application
  newItem.ServiceName = "MyApplication"

  // Assign a password to the item
  System.KeyChain.AddPassword(newItem,
"SecretPassword")
Else
  Beep
  MsgBox("You don't have a key chain.")
End If

Exception e As KeyChainException
  MsgBox("Keychain error: " + e.Message)
```

And this code retrieves the password:

```
Dim itemToFind as KeyChainItem
Dim password As String
itemToFind = New KeyChainItem

// Name to find
ItemToFind.ServiceName = "MyApplication"

// Get the password
password =
System.KeyChain.FindPassword(itemToFind)
MsgBox("Password: " + password)

Exception e As KeyChainException
```

Adding to a plist

Apps built for OS X consist of an “application bundle”. This bundle contains the app itself, resources and other components such as frameworks. It also contains an “Info.plist” file, which is an XML file containing specific settings that tell OS X about your application.

For certain apps, you may need to modify the plist to enable features, such as for UTI and Retina (refer to the following sections for more information).

To make it easier for you to include your own settings in the application plist file, you can create your own Info.plist file with the specific settings you need and drag it into your project.

When your app is built, the settings in your plist file are copied to the application plist file.

Notes

- Do not add more than one plist file to the project.
- Any items in your plist file that are duplicates of what is created during the build process are overwritten by the build process.
- Only top-level keys and their entire value are copied. For example, if a key specifies a dict for a value then the entire dict is copied. Top-level keys are keys that are immediate children of PLIST > DICT in the plist XML structure.

- The file must have both a plist header and a the extension ".plist".

Uniform Type Identifiers (UTI)

Before OS X 10.4, OS X used File Types and Creator Codes to identify documents created by applications. Since then Apple has been phasing out Creator Codes and File Types. It now recommends that you use a Uniform Type Identifier (UTI) to identify documents.

A UTI is a special text string that identifies both data and application documents. For example, the UTI for a PNG picture or file is *public.PNG* and the UTI for a PDF file is *com.adobe.pdf*.

If you are working with common file types, you should use the public UTI for it.

Here are some links for reference:

- http://en.wikipedia.org/wiki/Uniform_Type_Identifier
- http://developer.apple.com/library/mac/#documentation/Miscellaneous/Reference/UTIRef/Articles/System-DeclaredUniformTypeIdentifiers.html

***Note**: UTIs work with both Cocoa and Carbon applications.*

Custom UTI

For your own documents, you can create your own UTI. A custom UTI typically has a "reverse-domain name" format plus an additional entry for the document type name. So you might have a UTI such as "com.company.app.doc". The first part of this UTI

must match the Application Identifier specified in the App properties (com.company.app in this case).

You create a UTI for your documents using the File Type editor. The last column in the editor lets you specify the UTI.

When you build your application, the UTI information is included in the Info.plist file in the application bundle. But in order for this to work properly, you have to add an additional "UTExportedTypeDeclaration" section to the plist manually.

Setting up a Custom UTI

These are the steps to create a custom document type that can be opened by your applications. You need to do these steps after each build of your application because the Info.plist that you need to modify is recreated with each build:

1. In the OS X Build Settings:
    a. Specify the Bundle Identifier for your application using reverse domain name format: com.company.app
    b. Enter "????" as the Creator Code. This is simply a placeholder; it is not used.
2. Add a File Types Set to your project and then add a File Type to it:
    a. Enter a name for Display Name and Object Name (such as MyCustomDocType).
    b. Enter "????" for MacType and MacCreator. These are placeholders and are not used.
    c. Specify your extension (without a period).
    d. Enter the UTI using your application identifier as the base. For example, this could be your doc UTI: com.company.app.doc
    e. Choose the icon to use to identify files of this type.
3. Go back in the App properties.
    a. Specify your File Type in AcceptFileTypes
4. Build your application.
5. In Finder, go to your application and select "Show Package Contents" from the contextual menu. Navigate to the Contents folder and open Info.plist file in a text editor. Add the following code to the end (before </dict>), replacing the UTI (com.company.app.doc) and extension (myextension) as

appropriate:

```
<key>UTExportedTypeDeclarations</key>
  <array>
    <dict>
      <key>UTTypeConformsTo</key>
      <array>
        <string>public.data</string>
        <string>public.item</string>
      </array>
      <key>UTTypeIdentifier</key>
      <string>com.company.app.doc</string>
      <key>UTTypeTagSpecification</key>
      <dict>
        <key>public.filename-extension</key>
        <array>
          <string>myextension</string>
        </array>
      </dict>
    </dict>
  </array>
```

6. Save Info.plist.

7. Navigate back to the your application in the Finder and Run your application to register the types

When you use GetSaveFolderItem (or SaveAsDialog), you supply the FileType to use as the Filter. A custom document created with the selected file will be associated with your application so that the specified icon appears and so it can be opened by your application.

```
Dim f As Folderitem
f =
GetSaveFolderItem(FileTypes1.CustomType,
"Untitled" +
FileTypes1.CustomType.Extensions)

If f <> Nil Then
  Dim output As BinaryStream
  output = BinaryStream.Create(f, True)
  output.Write("CustomType: Sample Text")
  output.Close
End If
```

When you go view the saved document in the Finder, you will see that it has the specified icon. Double-clicking it will launch your app (if it is not already launched) and call the OpenFile event.

Opening files works by supplying the name of your FileType as the filter of an open dialog:

```
Dim f As FolderItem
f = GetOpenFolderItem(FileTypes1.MyType)
```

Migrating to UTIs from MacCreator and MacType

If you have been using the MacCreator and MacType properties of a FolderItem and specifying them in File Type Sets, then you need to start using UTIs soon. Apple has deprecated both MacCreator and MacType.

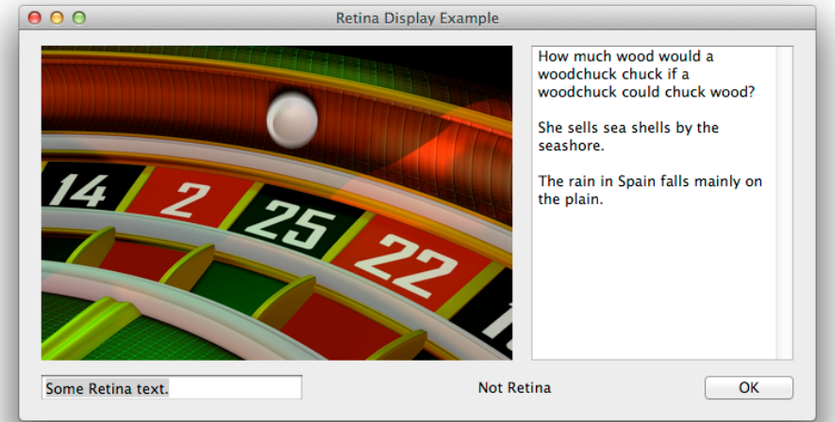
To switch to using UTIs, you simply need to start using UTIs in your File Type Sets and updating Info.plist as described earlier. You can leave the old MacCreator and MacType setting in place or you can remove them so that you do not get deprecation warnings when analyzing your projects.

Figure 2.6 A Custom Document in Finder

Retina Displays

Retina displays are displays that have extremely high pixel resolution. When text and UI elements are drawn at the same size as non-retina displays, these smaller pixels result in incredibly clear and sharp text and controls. A retina display also allows for higher resolution pictures to fit in a smaller space on the screen. Currently, Apple offers retina-type displays with iOS and Mac products.

Figure 2.7 Non-Retina Application



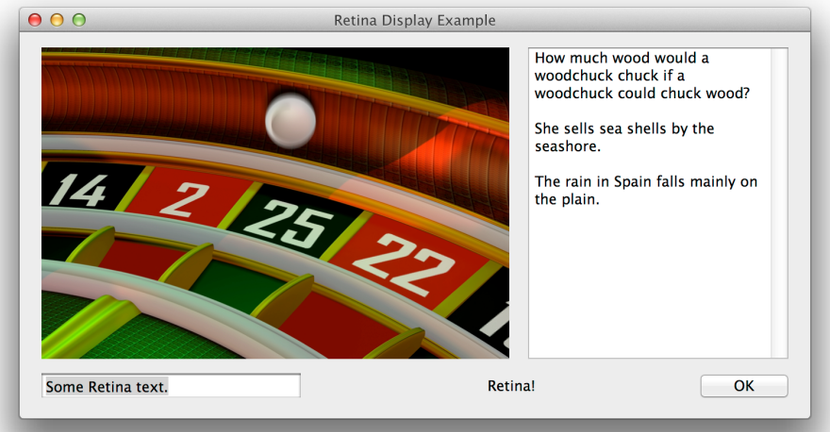
With a bit of tweaking, your Cocoa applications can be made retina-aware so that UI elements and graphics draw at high quality when run on a retina display.

***Note**: Only Cocoa applications can be made retina-aware. Apple does not support retina-aware Carbon applications.*

This is a two-step process. First, you need to update the Info.plist in your application bundle to enable retina support.

Second, if your application has graphics, you need to update them in order for them to appear at retina quality. Generally this means that you need to provide much larger graphics and then scale them down to the size you need. Usually you supply a 2x resolution image and then scale it to fit the available area. OS X will automatically handle this.

Figure 2.8 Retina Application



For some graphics, you may find that having separate pre-scaled graphics for retina and non-retina displays work best. For example, you might want to swap out a 1x image for non-retina displays, but use a 2x image with a retina display. To do this, you need to ask OS X what the scale factor is so that you can determine which image to use.

Enabling Retina Support

To enable retina support, you need set the NSHighResolutionCapable key to True in the Info.plist in the application bundle.

You can do this by creating your own Info.plist file as follows and dragging it into your project:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
     <key>NSHighResolutionCapable</key>
     <true/>
</dict>
</plist>
```

This enables retina support for the UI elements, such as buttons and text.

Alternatively, you can use Build Automation to add this key to the plist file after your application is built.

***Note**: If you do not have a retina Mac, you can still see how a retina application looks. One option is to use an iPad with retina display as a extra display for your Mac (using an app such as AirDisplay). Or you can download Quartz Debug from Apple (in their Graphics Tools) and use it to enable HiDPI display modes for your display. Once you do this (select Window->UI Resolution and "Enable HiDPI display modes), extra display modes (with greatly reduced resolution) appear in your display preferences.*

For example, on a 1920x1200 display, the best available HiDPI mode is 960x600.

Retina Graphics

If you just add large (2x) graphics to your projects and scale them down to fit when drawing them, then you automatically get the high resolution graphics used on a retina display. A non-retina display uses the scaled version.

But using a scaled version can sometimes lead to non-optimal results, so Apple recommends having two separate images included in your application, one for retina displays and one for non-retina displays.

In order to detect if your application is running on a retina display, you can check the BackingScaleFactor function in the Cocoa API. Add this code on a Window:

```
Function ScalingFactor() As Single
#If TargetCocoa
  Declare Function BackingScaleFactor Lib
"AppKit" Selector "backingScaleFactor"
(target As WindowPtr) As Double
  Return BackingScaleFactor(Self)
#Else
  Return 1
#Endif
End Function
```

The ScalingFactor is 2 for a Retina MacBook Pro (or other HiDPI modes) and 1 for anything else, but the values could change depending on type of retina display. You can use this value to determine the image size to use or for other scaling you may need.

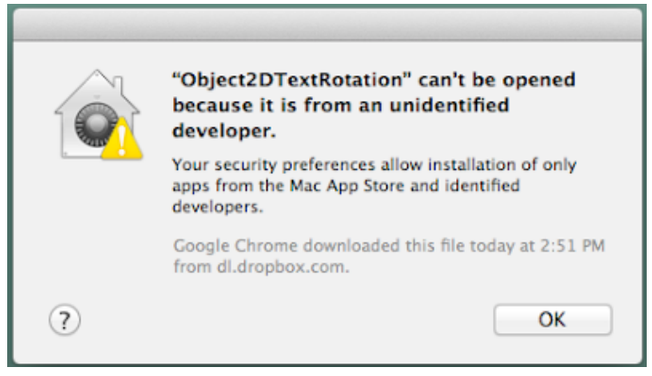
Note that this only applies to bitmap graphics that are drawn on the Graphics instance in the Paint event handler. Vector graphics scale automatically.

If you are manually double-buffering your graphics, you may find that they are not automatically drawn in retina quality. Rather than double-buffering, you should instead draw directly to the Graphics instance in the Paint event handler. The DoubleBuffer property of Canvas works with retina displays.

Code Signing

With the release of OS X 10.8 Mountain Lion in 2012, the new GateKeeper functionality is now in effect. This means that new apps that are downloaded or copied to a Mac with OS X 10.8 or newer, but that are not digitally signed using your Apple Developer Certificate, display this error when run:

Figure 2.9 OS X Error for Unsigned Applications

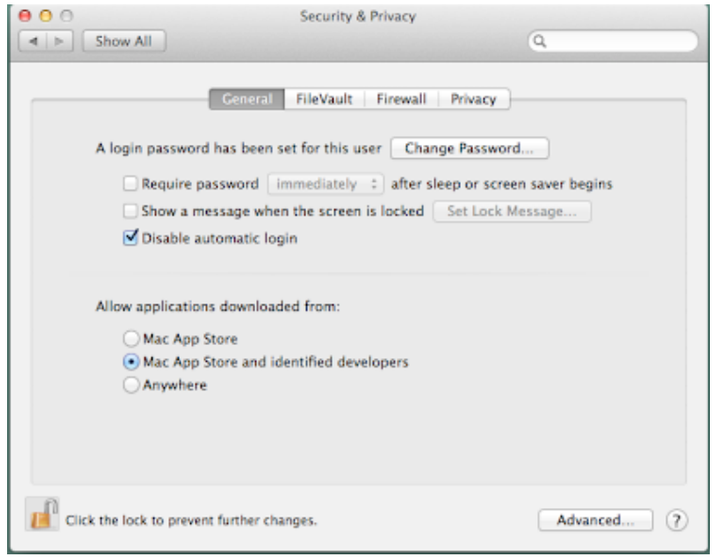


This error can be overridden in System Preferences, by changing the "Allow applications downloaded from" setting to "Anywhere" (Figure 2.10).

In addition, you can right-click on the app and click Open in the menu to tell OS X, "I'd really like to run this app, thank you very much."

Note that this only matters for new apps that you transfer to a Mac running OS X 10.8 or later. You'll be able to run the apps you create on your developer machine without this warning. You'll only run into this warning when you copy the app to another Mac, either by making it available for download or by copying it via a USB stick, the network or anything else.

Figure 2.10 Security & Privacy System Preferences



So even though you don't technically need to sign your OS X applications in order to avoid this warning, you are probably going to want to. The truth is that most people will just leave the setting at the default and will not know that when they get the warning message that they can right-click on the app to open it. You could try explaining all this to them, but either way it is going to be a hassle for your users. Odds are they just won't bother with your app.

To sign your apps you need a developer certificate from Apple. And the only way to get a Developer Certificate is to sign up for

the Mac Developer Program, which costs $100 a year. However, the certificate you get is good for 5 years, so it looks like you do not need to pay the $100 fee each year unless you also want to distribute apps in the Mac App Store.

You can find out more about the Mac Developer Program at the Mac Dev Center:

https://developer.apple.com/devcenter/mac

Once you have joined, you can create your own certificates using the Developer Certificate Utility at the Mac Dev Center. The steps are a bit involved, but essentially you will request a Developer ID certificate using the Developer Certificate Utility.

The Utility then walks you through the process of starting KeyChain Access and downloading and uploading files until you have the certificate installed. It's a little tedious, but relatively straightforward.

That's the hard part. With the certificate installed, you can now use it to code sign any of your applications. You do this using the Terminal command codesign (pronounced "code sign").

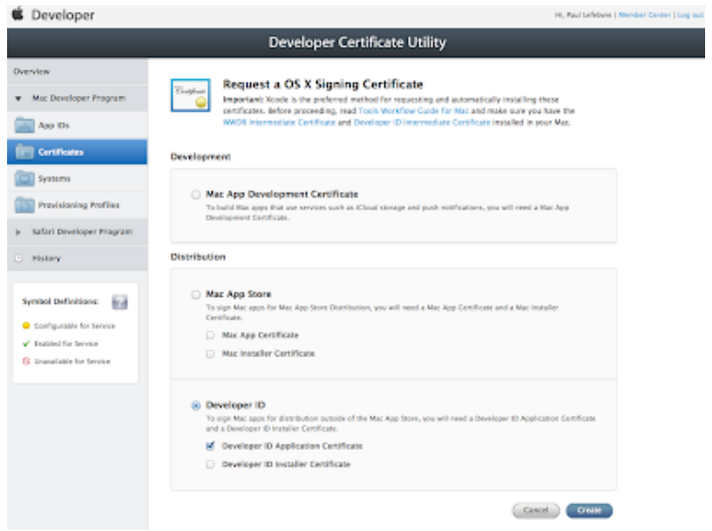
But before you begin, make sure you have the Intermediate Developer ID certificate installed. Go to this page:

http://www.apple.com/certificateauthority/

and download the Developer ID certificate. Double-click it to install it into Keychain Access.

Now you are ready to code sign your application. Navigate to its folder using Terminal. There you can enter this command to code sign your application and all its libraries. Obviously you want to replace "YourXojo.app" with the name of your application and "Developer ID Application: YourName" with the name of your signing certificate specified in Keychain Access.

Figure 2.11 Developer Certificate Utility at Mac Dev Center



```
codesign -f -s "Developer ID Application:
YourName"  "YourXojoApp.app"
```

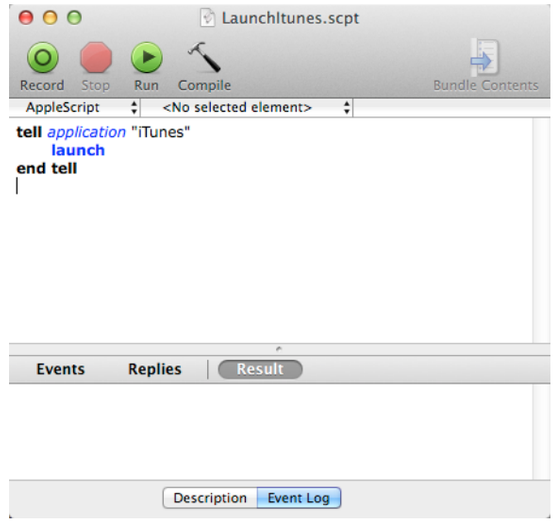
That's it. Now you can compress your app and transfer it to another computer.

AppleScripts

AppleScript is the OS X system-level scripting language than can be used to control applications. You can create AppleScripts using the AppleScript Editor.

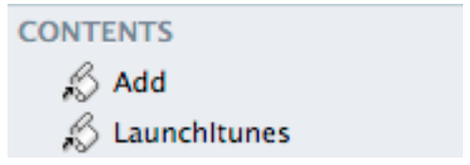
In order for Xojo to run an AppleScript, the script must be saved as a “compiled” script. You can do this using the Script Editor and selecting Compile on the toolbar and saving the script.

Figure 2.12 AppleScript Editor



Now you can use the AppleScript with Xojo by dragging the compiled script onto the Navigator. The script appears in the Navigator with a script icon next to it. This item is added to Xojo as an “external” item and includes it with the application when you build.

Figure 2.13 AppleScripts in Navigator



To run the AppleScript, just call it by the name it has in the Navigator.

Here is a simple script that you can compile and drag into your

project to launch iTunes:

```
tell application "iTunes"
  launch
end tell
```

Passing Parameters

To pass parameters, add an “on run” handler to contain your script and specify the parameters using curly brackets:

```
on run {value1, value2}
  // your script code goes here
end run
```

Xojo Integers passed to AppleScripts are sent as Integer values and are treated as Integers by AppleScript. All other Xojo types (including other numeric types such as Int8 and Double) are sent as Strings and are treated as Strings by AppleScript.

Returning Values

You can also return values from an AppleScript back to your Xojo code by using the return command in the script. This example adds two values and returns the result:

```
on run {value1, value2}
  return value1 + value2
end run
```

AppleScript does not use types like Xojo does. All values are returned as Strings to Xojo.

Calling AppleScripts from Xojo

AppleScripts are called just like built-in global methods and functions. You use the name it has in the Navigator. If it has parameters, you supply them after the name as you would any other method that has parameters.

Scripts that return values can be assigned to a Xojo variable. This command calls the above script that adds two values:

```
Dim sum As String
sum = Add(5, 10)
```

***Note**: If there is an error in your AppleScript, it is logged to the Messages pane at run-time.*

AppleEvents

AppleEvents are a way for OS X applications to communicate with one another. An AppleEvent is a self-contained block of data which consists of a sequence of key-type-value data (called an AppleEvent Descriptor, or AEDesc). Each descriptor can contain other descriptors as an ordered array or as a mixture of keyed data.

The AppleEvent as a whole is both itself and the AppleEvent Descriptor.

Refer to the Language Reference for specific details on how to use AppleEvents.

Section 4

Windows Features

ActiveX, COM and OLE

OLE (Object Linking and Embedding) and COM (Component Object Model) are ways to communicate with Windows objects in your applications. ActiveX describes controls that utilize COM or OLE.

You use the OLEObject, OLEContainer, OLEParameter and OLEException classes to access these Windows features.

OLEObject

OLEObject can be used to send messages to other Windows applications that support OLE, such as Internet Explorer.

Use the Value method to get and set values of the OLE object. Use the Invoke method to call methods (with or without arguments) on the OLE object.

This code creates a connection to Internet Explorer and then tells it to display Wikipedia:

```
Dim obj As OLEObject
Dim v As Variant
Dim params(1) As Variant
obj = New
OLEObject("InternetExplorer.Application",
True)
obj.Value("Visible") = True
params(1) = "http://www.wikipedia.org"
v = obj.Invoke("Navigate", params)
Exception e As OLEException
  MsgBox(e.message)
```

OLEContainer

OLEContainer is used to embed ActiveX controls into your applications.

To use an OLEContainer, you drag it from the Library onto a Window. In the ProgramID property of the Inspector, you specify the program ID for the control.

You can access the properties and methods of the ActiveX by using the Content property, which returns an OLEObject where you can use the Value and Invoke methods.

This code in a Button, displays a PDF in an Adobe Reader ActiveX OLEContainer that has been added to a window:

```
PDFContainer.Content.Value("Src") =
"C:\Document.pdf"
```

***Note**: Depending on the version of Adobe Reader, you may need to click on the container before the PDF is displayed.*

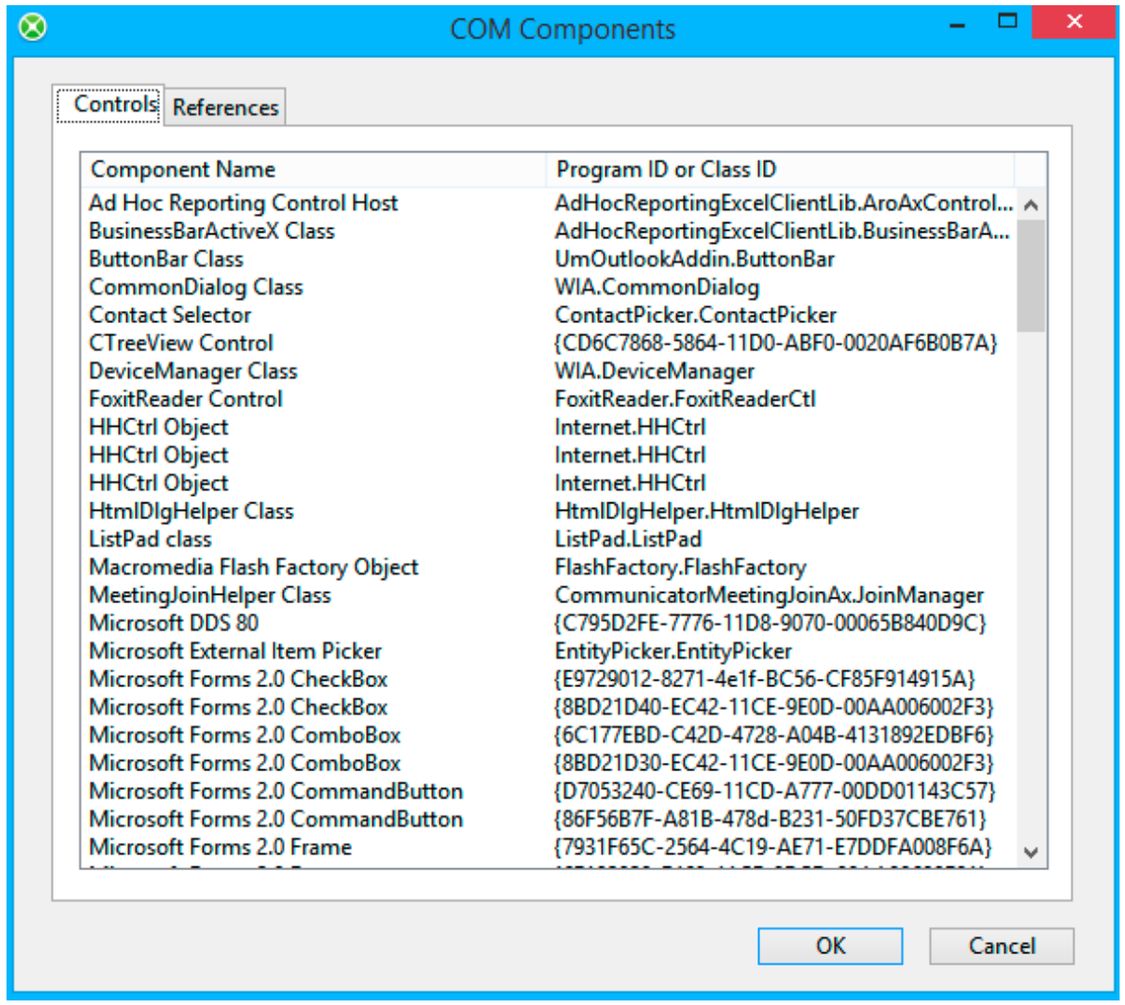
To print the PDF in the OLEContainer, you can call the "printWithDialog" method of the Adobe Reader ActiveX control:

```
PDFContainer.Content.Invoke("printWithDialog")
```

Using Insert ActiveX Component

You can also directly add ActiveX controls and automatable/OLE objects by selecting Insert -> ActiveX Component from the menu. This displays a window with two tabs: Controls and References. The Controls tab lists the ActiveX controls that you can add to a

Figure 2.14 Insert COM Components Window



window. The References tab lists the automatable COM objects are are not controls, like the iTunes Library, Microsoft Word, etc.

When you select an item and click OK, a module (containing classes for the component) is added to your project for you to use. Refer to the docs for the component to understand how to use its classes, methods and properties.

Office Automation

Microsoft makes it possible to automate Word, Excel and PowerPoint, which you can do using the three classes: WordApplication, ExcelApplication and PowerPointApplication.

These classes expose the Microsoft Object Model for each of the Office Applications. The Object Model for Word, Excel and PowerPoint are each immense, but they are fully documented by Microsoft.

- Word Object Model Reference: http://msdn.microsoft.com/en-us/library/ff837519
- Excel Object Model Reference: http://msdn.microsoft.com/en-us/library/ff194068
- PowerPoint Object Model Reference: http://msdn.microsoft.com/en-us/library/ff743835

By way of example, this code takes text entered into a TextArea and displays it in a Microsoft Word document:

```
Dim wordDoc As New WordApplication
wordDoc.Visible = True
Dim doc As WordDocument
doc = wordDoc.Documents.Add
doc.Range.Text = TextArea1.Text
```

***Note**: WordDocument is a class in the Word Object Model.*

Registry

The Registry is a system-wide Windows feature for storing application preferences and settings.

The RegistryItem class is used to find and add information to the Registry.

This code gets the location for the “Program Files” directory from the registry:

```
Dim r As New
RegistryItem("HKEY_LOCAL_MACHINE\Software\Micro
soft\Windows\CurrentVersion", False)

If r <> Nil Then
  MsgBox(r.Value("ProgramFilesDir"))
End If
```

Section 5

Localization

You localize your applications using dynamic constants, which is a special form of a constant that is added to a project item such as a module, window, web page or class. Only a constant of type String can be marked as dynamic, allowing it to be more easily used for localization.

Figure 2.15 A Dynamic Constant

To identify a constant as dynamic, create a string constant and then check the dynamic check box. A recommended approach is to create a separate module for your localizable constants, but you can put these constants anywhere.

You can enter different localized values for the dynamic constant based on platform and language. Do this using the Constant Editor.

Use the “+” and “-” buttons to add or remove a specific localization. You can choose the Platform and the Language for which to specify a value.

When the user runs an app, the language specified on their system is used to look up the localized value of the constant to use.

Constants created in this manner can then be used as the Text or Caption of controls so that the control displays the appropriate localized value. To use a dynamic constant as the Text or Caption, you prefix it with the “#” character when entering it in the appropriate property. If you have a module called LocalizedStrings and have a protected constant called kWelcomeMessage, then you add it to a Label as the Text property like this:

```
#LocalizedStrings.kWelcomeMessage
```

Of course you can also refer to the dynamic constant in your code just as you would any other constant.

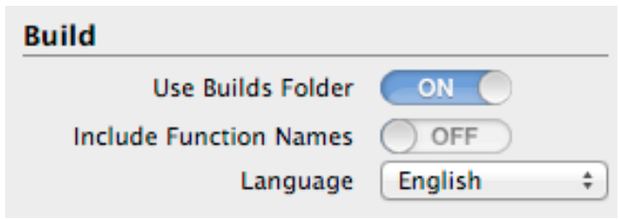
But with a dynamic constant, you can also specify which localization you want to use. You do this by supplying the language code (usually two-character) and optionally the region code as a parameter to the constant. For example, if you had a constant called *kHello* that had localizations for both French (“Bonjour”) and English (“Hello”), you could force the French version to be displayed by doing this:

```
Dim s As String
s = kHello("fr") // s = "Bonjour"
s = kHello("en") // s = "Hello"
s = kHello("en_UK") // s = "Welcome"
```

Building a Localized App

On the Shared Build Settings, there is a Language property in the Build section of the Inspector. This property determines the language that is used by any constants that have “Default” as the language.

Figure 2.16 Application Language Setting



It is important that you specify a specific language in this build setting. If you also leave this setting at “Default”, you will run into confusion if the project file is shared with people that do not have the same system language as you.

For example, if you leave both the constant language and build language as “Default” then “Default” becomes “English” for users that build with an English system and becomes “French” for users that build with a French system.

To prevent this confusion, always at least choose the correct language in the Build setting. Alternatively, don’t use “Default” for your constants and instead always specify the exact language for each constant.

Lingua

If you have a lot of text to localize, it can get tedious to enter all the values using the Constant Editor. The Lingua application is used to simplify the localization process. With Lingua you can localize your application strings outside of the project by using the dynamic constants you have already created.

When all of the strings in your project have been defined as dynamic constants, choose File ↠ Export Localizable Values. A dialog box appears, asking you to select the language you want to localize to. Select the language you are localizing to and click Export. (Also make sure that the Language in the Build Settings is set to either Default or any other language that you want to use as the originating language in the export.)

Figure 2.17 Lingua Main Window

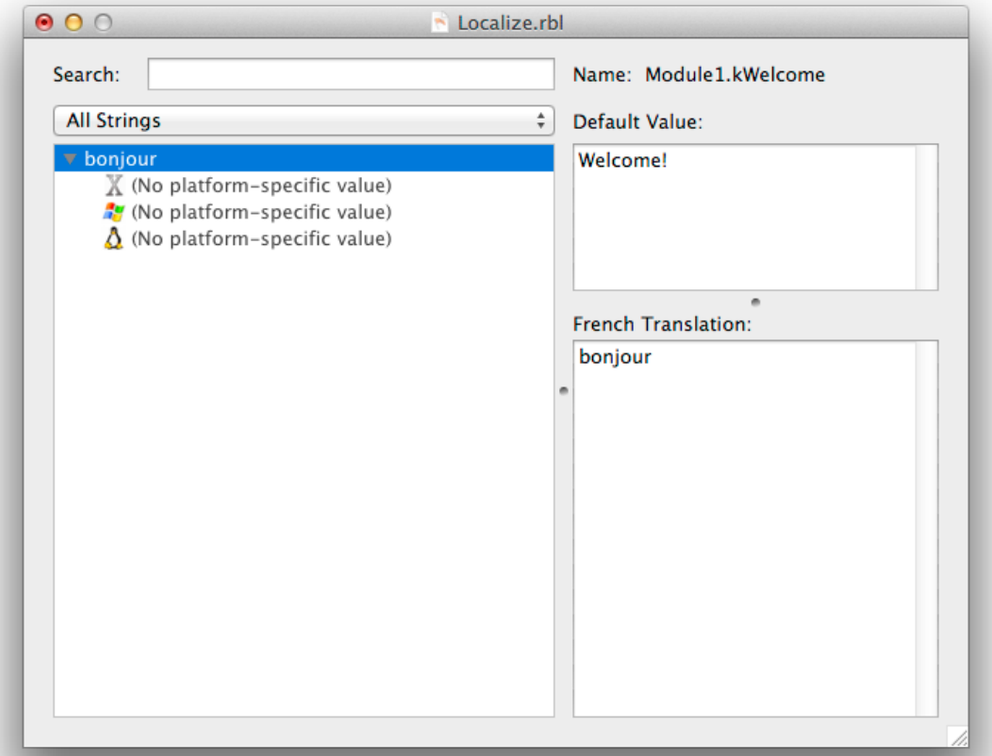


Figure 2.18 Lingua Welcome Screen

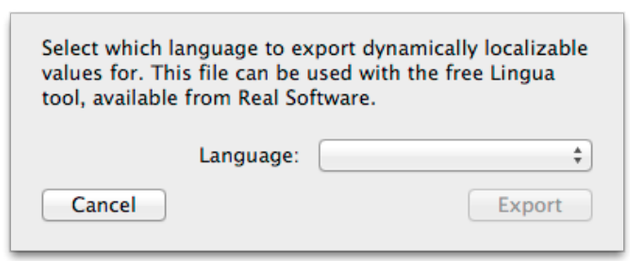
When you click Export, a file is created that can be opened by Lingua.

Launch Lingua and then open the file you exported. The main Lingua window opens, showing a list of all the dynamic strings in your application. The values are grayed out when there is no localized text yet. If there are any different values specific to Windows, Linux, or OS X, there will be an icon to the far right of the string in the list. Like the strings, the icons are grayed out if the value hasn't been localized yet.

To localize a string, select it in the list. The original value is displayed in its entirety in the upper right pane, and in you can type the translated text in the lower right panel.

To add a value specific to a platform, expand the string in the list, and select the individual platform to edit it.

Figure 2.19 Export Localized Values Dialog



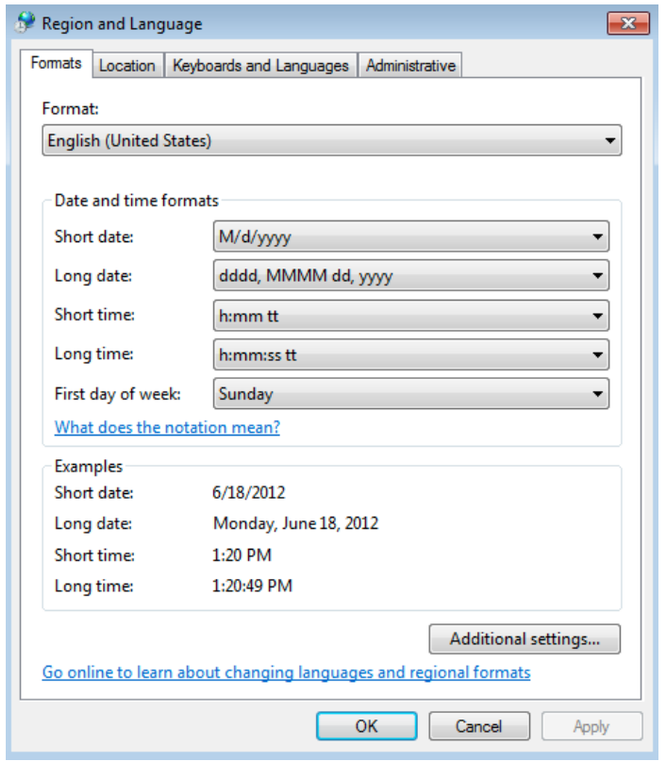
To test the strings, choose File ↠ Export to Application. Lingua presents an open-file dialog box. Select the target application and click Open. When the import is complete, switch back to Xojo and debug the application normally.

When finished localizing, you can save the file from within Lingua and then import the strings file back into your project by dragging it into a project or choosing File ↠ Import.

Changing the Language on Windows

To use a different language on Windows, you need to change the Language setting on the Format tab. Changing the Language on the Keyboard and Languages tab will not affect your applications.

Figure 2.20 Changing the Language for an Application



Localizing Web Applications

Web applications are also localized using dynamic constants, but the language that is displayed is determined by the language setting in the browser being used to access the web application.

***Note**: More specifically, it is using the constant value that is most appropriate for the HTTP header language setting of the current session.*

There are also several properties of the Session that are helpful for localization:

- LanguageCode
- LanguageRightToLeft

In addition, there are several dynamic constants that are used for system messages that you can localize:

- ErrorDialogCancel
- ErrorDialogMessage
- ErrorDialogQuestion
- ErrorDialogSubmit
- ErrorThankYou
- ErrorThankYouMessage
- NoJavascriptMessage
- NoJavascriptInstructions

Chapter 3

Web Development

Web development can be a bit different than desktop development. This chapter covers ways to optimize your web application and suggestions for porting desktop applications to web applications.

CONTENTS

Section 1

Optimizing Web Applications

Unlike a desktop application that is on the same computer as the user, a Web app could be thousands of miles away from the user. Also, different users are connecting with varying degrees of speed in between. Some may truly be using a state-of-the-art broadband service, while others are making do with less than an optimal mobile connection. Therefore it is important that you take into consideration how to build your app to optimize for performance on the internet. Here are a few techniques you can use to optimize your web application for performance.

Graphics

Use PNG

Although pictures in JPEG format can be smaller than those in PNG format, JPEG tends to be fuzzy for text and smooth color, does not support transparency and has possible licensing issues.

You can create a WebPicture in PNG format by passing Picture.PNGFormat to the second parameter of the WebPicture constructor. Here's an example:

```
WebImageView.Picture = New
WebPicture(source, Picture.FormatPNG)
```

Store Pictures in Properties

Pictures stored in properties of a web application are cached by the web browser so they are sent only once from the app to the browser. As a result, storing pictures as App properties reduces the amount of data that is transmitted between the app and the

browser. Storing your pictures in Modules will also allow the browser to cache them.

Use Rectangles

Rather than creating pictures, use Rectangles when possible. They can be dramatically altered using styles. For example, by setting the corner radii to 50, you can turn a rectangle into a circle. Rectangles are very small in terms of the amount of data that needs to be sent from your app to the browser.

Use the Style Editor to Create Styles

Styles are small (in terms of the amount of data sent from the app to the browser) so they are a very efficient way to make visual changes.

Latency

Remove Unused Event Handlers

Event handlers cause communication between the browser and the web app on the server even if they have no code in them. For this reason, remove any event handlers that do not have code in them.

Send Large Objects in the Background

If you have large objects you know you will likely need, you can use a Timer control or use Push to assign them to properties of the page while the user is busy doing something else. For example, Google Maps sends in the background map segments that are around the area you are viewing in case you scroll the map. You can use this same technique in your applications.

Be Careful with Query Results

If you’re accessing a database and loading the results into a ListBox, be careful with large numbers of records. The more records you load, the longer it will take for that information to be transmitted to the browser. And there’s only so much information a user can realistically view anyway. Avoid filling ListBoxes with huge numbers of rows.

Be Careful with Key and MouseEvents

Every event causes data to be transferred between the browser and your app. That means you want to avoid using frequent events such as KeyUp, KeyDown and MouseMove if possible. If

your app is running on the local network, these events will probably be fine but if the user is accessing it over the Internet and you have a lot of simultaneous users, it may cause too much of a lag to work responsively. Test and experiment!

Framework Usage

Don't Use Implicit Page Names

Store a reference to a page in a variable or property rather than using the name of the page as an implicit reference. Implicit references have to be looked up by the framework which takes longer than simply accessing a reference that is stored somewhere. For example, rather than doing this:

```
WebPage1.Show
```

Do this:

```
Dim page As WebPage1
page.Show
```

And then continue to access your page via the variable or property ("page" in this case).

Use InsertText/AppendText when Updating Text Areas

Each time you update the Text property of a Text Area, all the text is sent from your app to the browser. If you just need to append text or insert some text, use the AppendText and InsertText methods instead. These send only the text being inserted or appended to the browser.

Deployment and Development

Use CGI Rather than Standalone

You can build your app as a CGI or standalone HTTP Server. Tests suggest that a standalone HTTP Server app should handle a few hundred users without a problem. However, if you need more performance, use CGI instead.

Memory Leaks

Memory leaks occur when objects are created but never destroyed. As more and more objects are created and not destroyed, the amount of memory used increases. Eventually, the app will crash because the machine runs out of available memory. In a desktop app this may not be a big deal because the user will eventually quit the app and that will clear memory. However, in a web app it is more serious because the web app may be running for days, months or even longer. If your app is running as a CGI, once the number of users (sessions) accessing the app gets to zero, the app will quit and this will release any memory that app is

using. However, your app may never reach the point where there are no users so you need to be careful about not leaking memory.

Local Laws

Web applications are sometimes affected by the local laws in your area. For example, the European Union recently passed a directive requiring web sites to ask visitors for their consent before they can install most cookies.

Section 2

Porting Desktop Applications

To create a web version of an existing desktop application, there are several things to consider.

Obviously, you cannot use your project exactly as it is, but with an appropriate design, you may find that you can re-use (or even share) a significant amount of code.

Project Type

First, desktop applications and web applications have different project types. You cannot change the type of a project, so you will need to create a new web application project in order to create a web application.

User Interface

The user interface for a web application is completely different than the user interface for a desktop application. Not all of the desktop controls have equivalent web controls (Tab Panel, for example) and not all the features of the desktop controls are available in web controls (such as List Box). There are also web controls that do not have an equivalent desktop control (for example, Map Viewer).

In addition, web applications do not have a concept of a Menu Bar, which is something that almost every desktop application uses.

You are going to need to completely re-implement your existing desktop user interface using web application controls. And while you are doing this, you should also consider redesigning things to work better in a web application.

Web Pages Replace Windows

Generally speaking, each window in a desktop application can be designed as a web page in a web application. You use the

Show method to display different pages based on user actions, similar to how you might show additional windows.

Keep in mind that a web application can only show a single page at one time. If your desktop application relied on having multiple visible windows, then you will need to rethink that design.

Dialogs

Dialogs in desktop applications can use the MessageDialog class or be modal windows. In web applications, those options are not available.

To create a dialog in a web application, you instead add a Web Dialog to your project and add your layout to it. Then you add this web dialog to the page or pages on which it should appear and call it using Show where appropriate.

When the dialog is closed, its Dismissed event handler is called where you can determine what action to take.

***Note**: Web dialogs are not modal. After you call Show to display the dialog, the rest of the code in your method runs.*

Shown Event Handler

In desktop applications, you often use the Open event handler to do initial set up of your controls or windows. In web applications, you should instead use the Shown event handler.

Styles

In desktop applications, you can modify the style of a control by changing properties on it for color, font, etc. With web applications, you instead create a Style that has the settings you want and you apply the Style to the control.

This has the benefit of allowing you to use the same style for controls throughout your web application. If you then need to change something in the style (say a font size), you can do that in one place (the Style itself) and the change will take effect anywhere that the style is used in the web application.

Multiple Users

One fundamental difference between desktop and web applications is that desktop applications are designed to be used by a single user at a time while web applications are designed to be used by many users at a time.

Sessions

Having to deal with multiple users means you may need to manage global data differently. In desktop applications, properties and methods on App are global to the entire application, which can often be useful.

In a web application, App is also global to the entire application, which means it is globally available to all users of the web application.

So you do not want to store global information that is specific to the current user in App, such as the UserName that was used to log in. Instead, web applications have a concept called a Session. Each user that connects to your web application gets its own session in the form of a Session object. Use the Session object to manage global information just for the user. For example, saving the UserName to a property on the Session means that it is only visible to the one user.

Databases

Similarly, with databases, desktop applications often keep a global reference to the database in App. With a web application, you should instead use Session for the database reference. Each user that connects to the web application should have its own connection to the database so that transactions work properly and so that you can isolate database access to prevent other users' data from being visible.

For desktop applications, if you want to allow multiple users to access a database (using multiple installations of the desktop app), then you usually want to use a database server.

With web applications, you may find that you do not always need a database server. Because your web application is itself running as a server, SQLite is often more than sufficient for handle light to medium web application loads.

Cookies

Most applications need to save preferences or settings of some kind. With web applications, you can easily do this using Cookies, a web technology that provides a way for a web browser to save settings that can be requested by web applications.

```
Session.Cookie.Set("UserName") = UserNameField.Text
```

Log In

Mentioned briefly above was the concept of a UserName used to log in. Most web applications require a user to log in so that the web application knows what data it should be displaying. This is often the first page of your web application. This log in information can be saved in Session and stored in Cookies for easy access and retrieval.

This code (in the Shown event handler for the page) could fetch the saved Cookie and use it to pre-fill the UserName field:

```
UserNameField.Text  = Session.Cookies.Value("UserName")
```

Code Sharing

If you tend to keep most of your code in your user interface objects, then you will not be able to share your code between desktop and web projects.

But if you instead separate your code out into classes that are called by your user interface objects, then you can start to share code between web and desktop projects.

This is sometimes referred to as Model-View-Controller design. The View is the user interface, either web or desktop. This cannot be shared. The Model is your data. This is shareable. The Controller is the interface between the Model and the View.

Using this design in conjunction with conditional compilation allows you to create shared code that works for both desktop and web applications.

For details on how to set up projects that share code, refer to Chapter 5: Code Management, Section 1: Sharing Code Between Projects.

Navigator Example

This simple example uses a class to open a new web page or a new window, depending on the type of app.

Create a new web or desktop project and add a new class, called Navigator. In it, add a method called ShowScreen2 with this code:

```
#If TargetWeb Then
  WebPage2.Show
#ElseIf TargetDesktop Then
  Window2.Show
#Endif
```

This code displays either WebPage2 or Window2 depending on the type of application.

In Window1, add a Button and put this code in its Action event handler:

```
Dim n As New Navigator
n.ShowScreen2
```

Now add a new Window (it should default to Window2 as the name). You should give this window a title that says “Window 2” so you know when it has opened.

Run the project and click the button on the default window. Window2 opens.

Now create a new web project. Add a Button to WebPage1 and add the same code to its Action event handler:

```
Dim n As New Navigator
n.ShowScreen2
```

Copy the Navigator class from the desktop project to the web project.

Lastly, add a second web page, called WebPage2. Give it a title so you know when it displays.

Run the project and click the button on the default web page. Web Page 2 appears.

You have now created a (very simple) class that can be used in either a desktop or web application. This technique can apply to just about anything. Code that refers to web-specific objects or features should be included in “#If TargetWeb” and code that is for desktop apps should use “#If TargetDesktop”.

Although you are using the same Navigator class, you are not actually sharing the exact same class between the two projects. Changes made to the Navigator in one project do no affect it in the other project. If you want to share the exact same class so that a change in one project is reflected in another, then you need to use an External Project Item as described in Chapter 5.

Section 3

Mobile Support

Web applications can be a great way to provide mobile applications for you users and customers.

General Mobile Support

Use the Session.Platform property to determine the platform that your web application is running on. With this information, you can choose to display a page that is specifically designed for the smaller screen of an iPhone, for example. This code in the Session.Open event handler displays the appropriate page for the device being used:

```
Select Case Session.Platform
Case WebSession.PlatformType.iPhone
  iPhonePage.Show
Case WebSession.PlatformType.AndroidPhone
  AndroidPhonePage.Show
Case Else
  MainPage.Show // All other devices
End Select
```

Pages are resized to the screen size or the page MinimumHeight and MinimumWidth. If either minimum value is larger than the actual screen size, the user is able to scroll the web application. If the minimum is smaller than the actual screen size, then the entire web page is visible.

The Session.OrientationChanged event allows you to know when the user has changed the device orientation. You can use this to display a different page if appropriate.

Zooming is locked to 100%, which prevents the page size from changing allowing it to look best on mobile devices.

iOS and Android

With iOS and Android, the user can “bookmark” a web page to the home screen. When the user does this with a web application, the icon you have specified for the loading screen is used as the icon on the home page.

Chapter 4

Migrating from Other Tools

Are you coming to Xojo from another development tool? Here are some tips.

CONTENTS

Visual Basic

Visual Basic (6 or earlier) and Visual Basic .NET use a language very similar to the Xojo language. You will notice that many of the commands are nearly the same, but there are differences as well.

Similarities to Visual Basic

Visual Basic 6 (VB6) is no longer supported by Microsoft, which recommends you instead migrate to Visual Basic .NET (VB.NET). But Visual Basic .NET is large and complex, not to mention not cross-platform. Xojo is usually a better choice for Visual Basic 6 applications because it has the simplicity of VB6, but is a fully object-oriented language like VB.NET.

To start with, the language syntax of VB is very similar to Xojo.

Figure 4.1 VB and Xojo Data Types

VB Data Type	Xojo Data Type
Boolean	Boolean
Byte	Byte
Currency	Currency
Date	Date class
Double	Double
Integer	Short
Long	Integer
Object	Object
Single	Single
Variant	Variant

You'll see familiar syntax for If..Then..Else, For..Next, While..Wend, Dim and many other commands. Someone who has used either VB6 or VB.NET will have no trouble understanding the Xojo language.

Data Types

Although Xojo data types are not always named exactly the same as VB6 data types, all the equivalent types are there. For example, Integer is equivalent to a VB6 Long.

Controls

The default UI controls included with VB are, for the most part, also included with Xojo.

Figure 4.2 VB and Xojo Controls

VB Control	Xojo Control
PictureBox	Canvas
Label	Label
TextBox	TextField
Frame	GroupBox
CommandButton	PushButton, BevelButton
CheckBox	CheckBox
ComboBox	ComboBox
Listbox	ListBox, PopupMenu
HScrollBar, VScrollBar	ScrollBar
Timer	Timer
Shape	Oval, Rectangle
WebBrowser	HTMLViewer
TreeView	ListBox
Toolbar	Toolbar
MediaPlayer	MoviePlayer

But Xojo also has several controls that are not included by default with VB. Of course, VB had plenty of additional, but Windows-specific, controls that could be added to its default setup.

Differences from Visual Basic

A big difference is that Xojo cannot create DLLs, ActiveX controls or any kind of shared libraries. Since these are all Windows-specific technologies, they are not useful for cross-platform applications.

Xojo can access DLLs and many ActiveX controls, but using them means your application will only run on Windows and cannot be cross-platform.

Of course, Xojo can easily create web applications, something VB6 cannot do.

File I/O

File input and output in VB6 uses direct, path-based access to files. This is not something that works for cross-platform applications, so Xojo consolidates all file processing into a few classes: FolderItem, TextInputStream, TextOutputStream and BinaryStream.

Data Types

Xojo has a strongly typed programming language. VB6 (and older versions) would allow you to use a variable that had not been previously declared. It would infer a type based on a special character in its name (name$ would be a String, for instance).

Before trying to migrate VB6 code, you should use the OPTION EXPLICIT command to make sure that all your variables are declared.

Visual Basic Migration Assistant

Visual Basic Migration Assistant (VBMA) is a free tool that can help you to begin migrating VB6 and VB.NET code to Xojo.

VBMA creates a Xojo project from the contents of your VB project. Specifically, it moves over forms, modules, classes and their code.

What it Does

VBMA takes the specified VB project and creates a Xojo XML project file containing the forms, modules, classes and source code from the VB project. The purpose of this tool is to get your project into Xojo so that you can work on it from a single place. The tool does not create a working Xojo application from VB code.

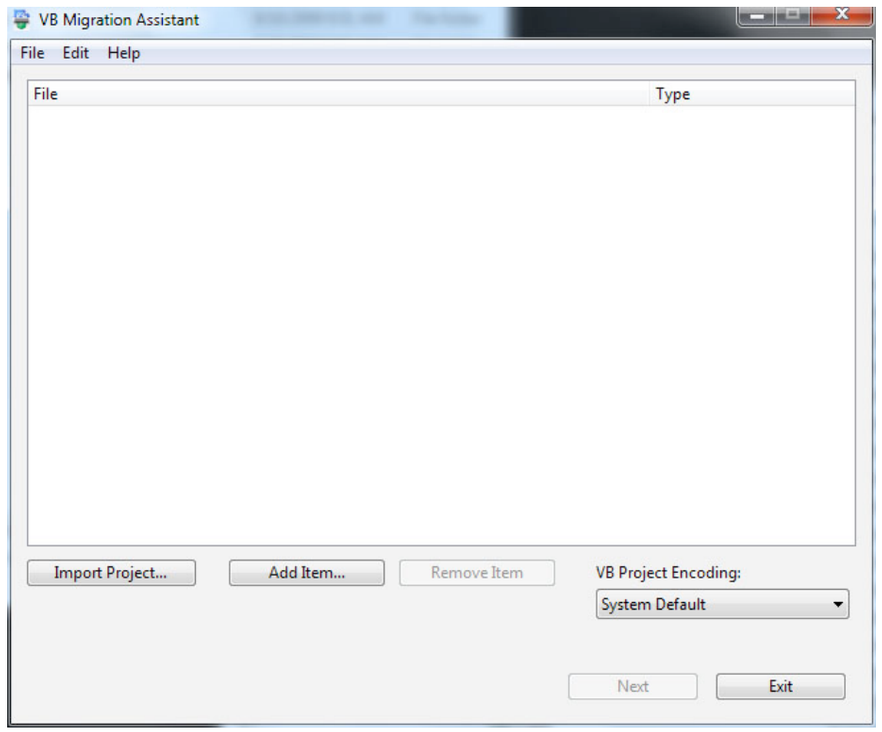
Since VB forms do not exactly match Xojo windows, VBMA maps VB controls to their equivalent Xojo controls as it migrates the project.

Source code is not converted or modified in any way. The code is migrated to the Xojo project, but is completely commented out and is primarily for reference.

For Best Results

VB 5 and 6 and VB.NET are supported. If you are using an older

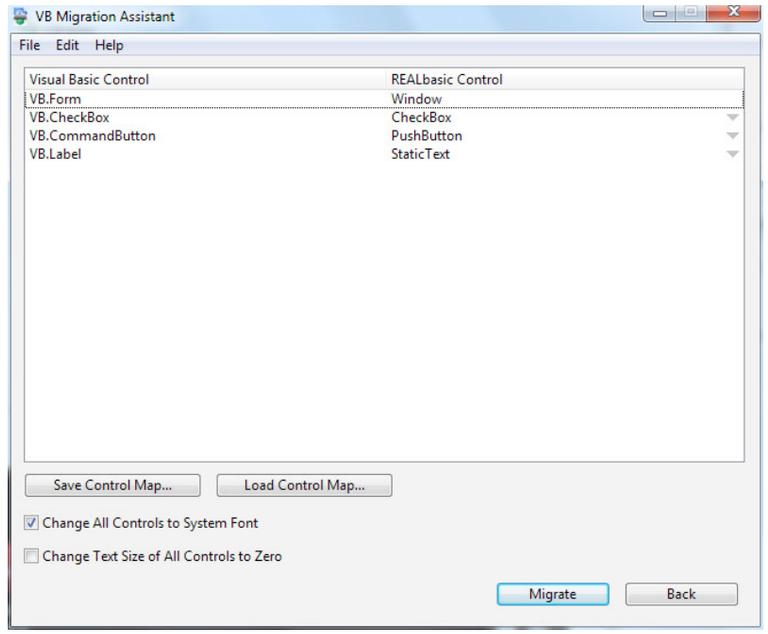
Figure 4.3 Visual Basic Migration Assistant Project Selection



version of VB, upgrade your code to a later version before attempting to use VBMA.

Review and try to reduce your usage of 3rd party VB controls. Not only are they not cross-platform, but not all of them work with Xojo.

Figure 4.4 VBMA Control Mapping



Converting a Project

When you launch VBMA, you are presented with a wizard that walks you through the process. The first screen is for Project Selection.

Select the “Import Project” button and choose your VB project file. Alternatively, you can drag individual files to the List or you can use the “Add Item” button to select individual files.

Specify the encoding as necessary. This is the encoding/language that was used to create the VB project. This is especially important if your VB project uses any non-English filenames.

Click the Next button to go to the Control Mapping screen.

VBMA analyzes the VB project and displays the type of controls that it is using. In the Mapping screen, you can select the Xojo control to use for each VB control.

You can choose to save the Control Map to a file so that you can use it again for other projects.

Click the Migrate button to migrate the VB project to a Xojo XML project file. You will be prompted for a location to save the file.

When VBMA is finished it will attempt to have Xojo open the project file.

With your VB project in Xojo, you can now begin work on creating a Xojo version.

Section 2

Microsoft Access

Microsoft Access is database software that runs on Windows and is part of specific versions of Office. It is often use to create specialized in-house database applications. But Access cannot create real, stand-alone applications. If you have an access application and are running into its limitations, Xojo is a great choice to take your application to the next level.

Similarities

Access has a form designer, database designer and a programming language (VBA: Visual Basic for Applications).

Xojo has all these components as well, but expands on each of them. It has a form designer with many more controls than Access provides and allows you to layout your user interface in any way you want. It uses SQLite as its built-in database and has a database designer for designing your tables. And of course, Xojo has a much more robust programming language.

Migrating

Migrating an Access application is typically a three-step process where you migrate the database itself, the forms that are used to manipulate the data and the source code.

Database

When migrating a Microsoft Access application, you first need to consider the database. If you are using the Access "Jet" database engine, you will most likely want to migrate it to another database engine. Although you can connect to a Jet database using ODBC or ADO on Windows, Jet is not a cross-platform database format. OS X can only connect to a Jet database using ODBC in read-only mode.

Your best bet in this case is to use SQLite, which is much faster than the Access Jet database and is fully cross platform. You can easily migrate your database tables and data from an Access database to SQLite. This can be done using ODBC, ADO or a variety of 3rd party products.

If your Access database is connecting to another database as its data source, then you can use the ODBC plugin and an ODBC driver to connect. Or you can use the built-in plugins for connecting to PostgreSQL, MySQL, Oracle and Microsoft SQL Server.

Forms

Your Access forms are likely used to edit data in tables. You can recreate these forms as Windows or web pages in your Xojo application.

For desktop applications, you may find the DataControl control to be a simple way to map your fields to database tables and columns without having to write a lot of code.

Source Code

Access is programmed using the Visual Basic for Applications language which is quite similar to the Xojo programming language.

You will have to rewrite your code, but at the same time will find the Xojo programming language to be familiar in its syntax and commands.

Section 3

FileMaker

FileMaker is a database tool that runs on both Windows and OS X. It is often called the OS X version of Access.

Much like with Access, FileMaker has its own database engine, form designer and scripting language.

Migrating

Migrating a FileMaker application is typically a three-step process where you migrate the database itself, the forms that are used to manipulate the data and the scripting code.

Database

When migrating a FileMaker application, you first need to consider the database. Although you can connect to a FileMaker database using ODBC, you will need to get the appropriate drivers.

Instead you can migrate your data to SQLite (a fast, cross-platform database) by first converting the FileMaker data to XML.

Forms

Your FileMaker forms are likely used to edit data in tables. You can recreate these forms as Windows (or web pages) in your Xojo application. In most cases you will use Label, TextField and TextArea to recreate form fields, but there are lots of other available controls as well.

For desktop applications, you may find the DataControl control to be a simple way to map your fields to database tables and columns without having to write a lot of code.

Either way, you would likely have your application connect to the database at startup and then populate the form with the first record. Next and Previous buttons can be added to fetch and display the appropriate information from the database.

Source Code

FileMaker is programmed using a scripting language that is somewhat similar to the Xojo programming language.

You will have to rewrite your code, but at the same time will find the Xojo programming language to be familiar.

Figure 4.5 Some FileMaker Commands and their Xojo Equivalents

FileMaker Command	Xojo Command
Exit Script	Return
Set Error Capture	Try..Catch..End Try
Set Variable	Dim
If..End if	If..End If
Loop..End Loop	Loop..Until
Go to Field	TextField.SetFocus
Field assignment	TextField.Text = "value"

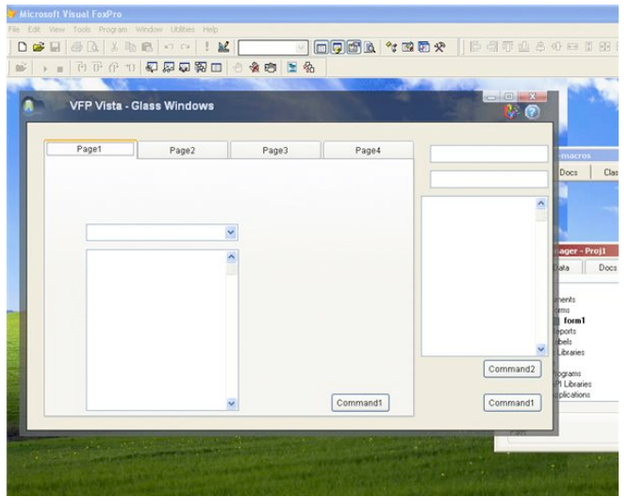
Special thanks to Hal Gumbert of Camp Software for assistance with the information in this section.

Visual FoxPro

Visual FoxPro (VFP) is a Windows programming tool made by Microsoft. It has been given its end-of-life and is no longer supported by Microsoft.

VFP has its own tightly integrated database engine, a form designer and a programming language.

Figure 4.6 Visual Fox Pro



Migrating

Migrating an Visual FoxPro application is typically a three-step process where you migrate the database itself, the forms that are used to manipulate the data and the scripting code.

Database

When migrating a VFP application, you first need to consider the database. You can connect to a VFP database using ODBC on Windows, but it makes more sense to migrate your data to SQLite, which is a fast, cross-platform database.

Forms

Your VFP forms are likely used to edit data in tables. You can recreate these forms as Windows (or web pages) using drag and drop just as you can with VFP.

For desktop applications, you may find the DataControl control to be a simple way to map your fields to database tables and columns without having to write a lot of code.

Source Code

VFP is programmed using a proprietary language that is quite similar to the Xojo programming language.

You will have to rewrite your code, but at the same time will find the Xojo programming language to be familiar.

Cully Technologies makes a product that can help you migrate your Visual Fox Pro projects to Xojo:

http://cully.biz/

Language Syntax

The syntax of the two languages is different, but the concepts are quite similar. For example, to create an instance of a new class in VFP, you might use this code:

```
LOCAL oMyClass
oMyClass = CREATEOBJECT("MyClass")
```

In Xojo you would write:

```
Dim oMyClass As New MyClass
```

The VFP MessageBox command is similar. Instead of writing this:

```
MessageBox("Hello, World!")
```

You would write this:

```
MsgBox("Hello, World!")
```

Other VFP commands and their Xojo equivalents are listed in the accompanying figure.

Figure 4.7 Some VFP Commands and their Xojo Equivalents

VFP Command	Xojo Command
ON ERROR	Exception
TRY..CATCH..END TRY	Try..Catch..End Try
DO WHILE..ENDDO	While..Wend
FOR EACH..ENDFOR	For Each..Next
FOR..ENDFOR	For..Next
IF..ENDIF	If..End If
LOOP	Continue
DECLARE	Dim
DO CASE..ENDCASE	Select Case..End Select

Special thanks to Kevin Cully of Cully Technologies for assistance with the information in this section.

Chapter 5

Code Management

This chapter covers two important concepts for managing your code: code sharing and source control.

CONTENTS

Section 1

Sharing Code Between Projects

Normally your projects are independent of each other. But there are certainly times when you may have common project items that you want to share amongst multiple projects.

Copy and Paste with a Master Project

The simplest way to do this is with copy and paste. You can select the project items in the Navigator, copy them and then paste them into another project. This creates two separate copies of the project items for each project.

The best way to use this technique is with a “master project” that contains the shared code you use. Keep the master project updated with all your shared code and any changes you make.

You can copy shared project items from the master project as needed to your other projects. This allows you to re-use project items without having to worry about changes affecting other projects.

External Items

You can also convert project items to “External Project Items”. An External Project Item can appear in multiple projects, so changing the shared project item in one project changes it for all projects. When you create an External Project Item, the item becomes a file on disk (in either binary or XML format).

To convert a normal project item to an External Project Item, open the contextual menu for the item in the Navigator (usually by right-clicking on it) and select “Make External...”

This prompts you to choose a location for the file and to select the type of the file. Your choices are Binary or XML. If your External Project Item is going to be stored in a source control system, you should choose XML instead of Binary.

External Project Items appear in the Navigator with a small arrow icon superimposed on the normal icon.

You can also add existing External Project Items that are on disk to a project. You do this by dragging the file from the disk to the Navigator while holding down ⌘+Option (on OS X) or Shift+Ctrl (on Windows and Linux).

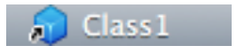


Figure 5.1 External Item in the Navigator

Note: If you make a change to an external

project item in one project, the change is NOT automatically reflected in any other projects that are open and also use the same external project item. You have to close and reopen the other projects in order to see the change.

Section 2

Using Source Control

A good developer always has backups of their source code. But backups are not enough. You also want to use a Source Control System. This is also sometimes called a Version Control System.

A source control system is able to track changes to individual files, keeping a history of the changes and allowing you to go back and look at prior versions of the file.

In order to use a source control system with your projects, you first want to make sure you are using the Text Project (Xojo Project) format. The Text project format saves each project item as a separate text file on disk. Having separate files allows the source control system to track changes to individual project items. Text files allows you to use “Difference” tools to compare differences between different versions of a file.

Subversion

Subversion (SVN) is probably the most commonly used version control system. SVN uses a central “repository” or database that contains your source code and all its versions.

The process for working with Subversion is as follows:

- Create a repository on the Subversion server
- Check Out the repository to a local computer. This is where you edit your files.
- When you want to make your changes permanent, you Commit them to the Subversion server. This also allows others on your team to see the changes.
- If you have multiple developers working on a team, you use the Update command to get changes added to the server by others on your team.

You can control Subversion using the command-line tool, called svn. But there are plenty of easy-to-use graphical tools that work with Subversion, such as SmartSVN, RapidSVN, Versions and Cornerstone.

Subversion Hosting

You can run a Subversion server on your development machine, but having Subversion on an external server is a great way to have an easy off-site backup of your source code.

You can choose to install Subversion on your own servers and some web hosting companies provide Subversion hosting as an optional feature.

There are also companies that offer Subversion hosting, such as Code Spaces, Source Repo, Assembla and Beanstalk.

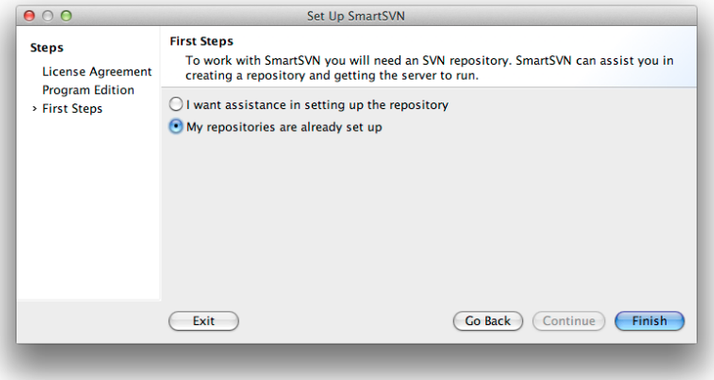
Setting up a Project with Subversion using Assembla and SmartSVN

Assembla offers free Subversion hosting. And SmartSVN has a free tool for working with Subversion. These steps walk you through creating a Subversion repository at Assembla, adding a project to it and then making and committing changes.

1. First, create an account with Assembla and then create a repository. In this case, the repository Xojo is used at this location: https://subversion.assembla.com/svn/xojo/

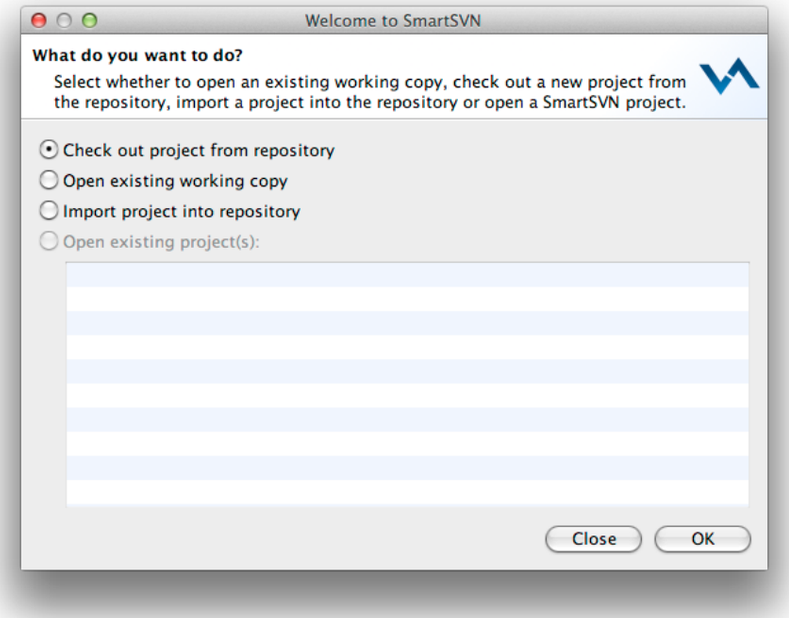
2. Download and install SmartSVN for your operating system: http://smartsvn.com

3. Run SmartSVN and let it finish its setup. It will eventually get you to the first steps where you tell it you are ready to choose a repository. Since you already created a repository with Assembla, select “My repositories are already set up” and click Finish:

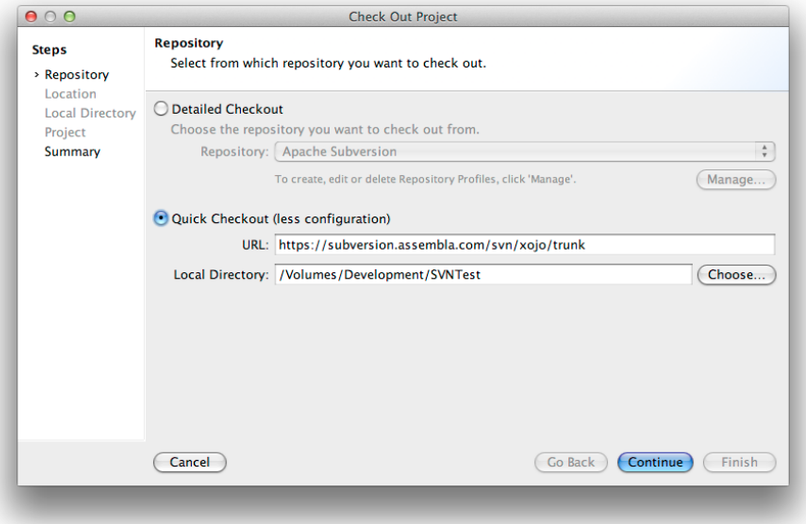


4. Click Cancel at this prompt regarding importing settings:

5. Now you will check out the empty repository from Assembla. Select “Check out project from repository” and click OK:

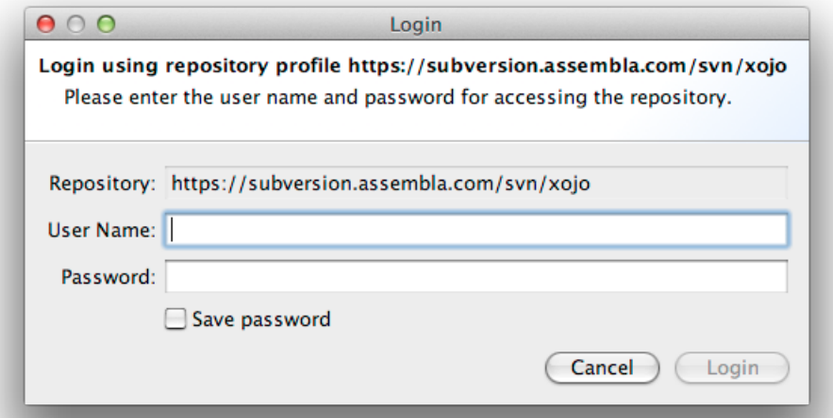


6. Now it is time to specify the repository information. Fill in the repository URL from Assembla (add trunk to the end of the URL) and provide a location for the local files then click Continue:



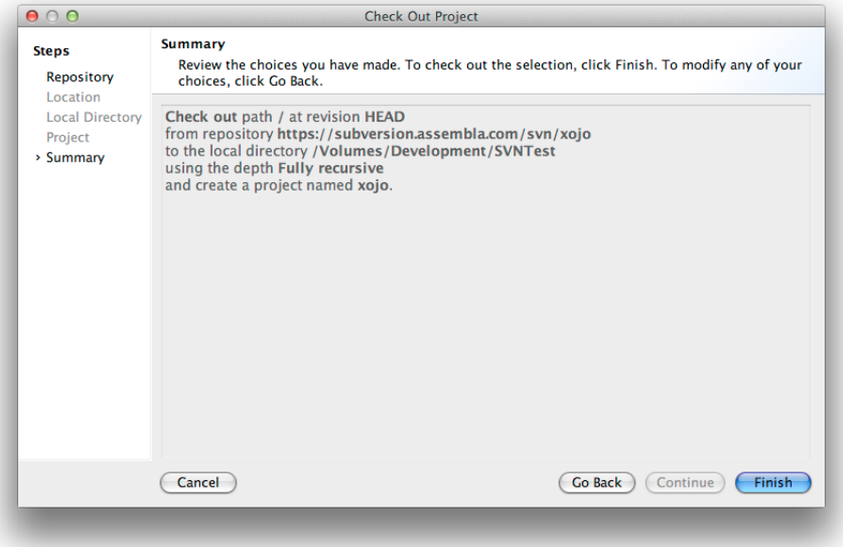
7. Click Accept on the Server Fingerprint dialog.

8. At the Login window, enter the User Name and Password that you used to create your Assembla account then click Login:

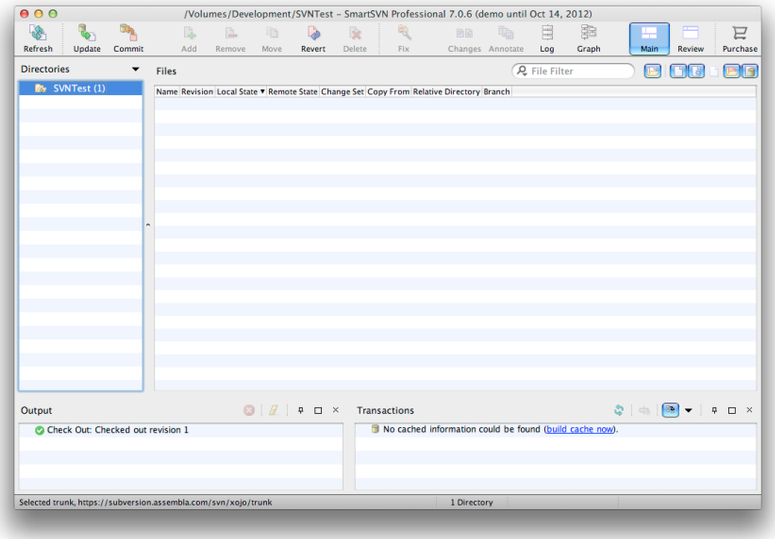


9. Decide whether you want to create a Master Password on the Master Password dialog, then click OK.

10. The repository will be checked out to the local folder and the Summary window appears. Click Finish to check everything out:



11. When the checkout has completed you will see the main window:

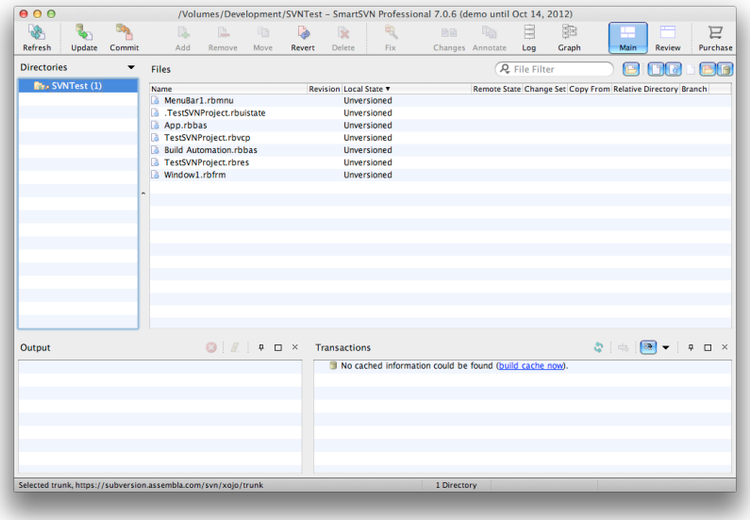


12. In Finder or Windows Explorer, navigate to the local folder so you can see the files that were checked out.

13. Create a new project in Xojo and select Save. Choose Xojo Project as the format and enter a name such as TestSVNProject. Choose your local Subversion folder as the save location.

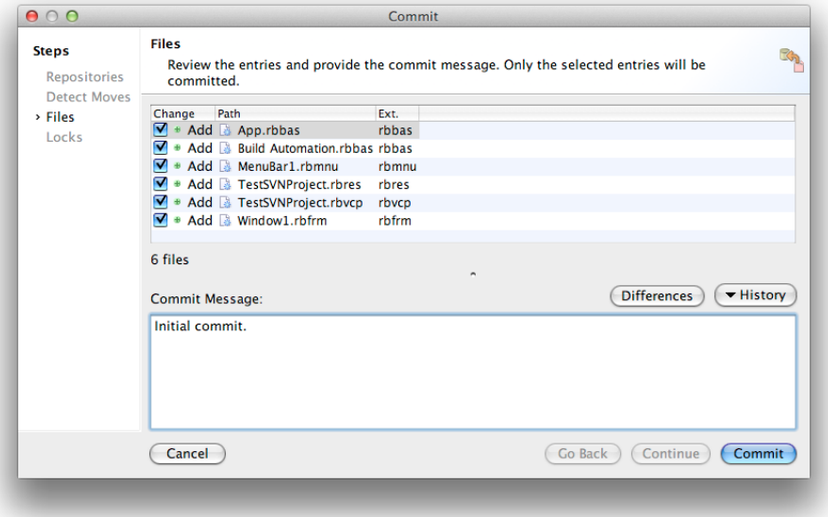
14. Go back to SmartSVN. You'll see the project items are displayed with "Unversioned" displayed as the "Local State". This means the files have not been added to the repository

yet.



15. To add them to the repository, you Commit them. But before you do this, you want to Ignore one of the files. The file called ".TestSVNProject.xojo_uistate" tracks the position of the main Xojo window, open tabs and other settings (there is no leading "." on this file on Windows). It should not be added to Subversion.
    Right-click on it and select "Ignore" from the contextual menu. Click OK on the Ignore dialog and you'll see the file no longer appears in SmartSVN.

16. Now you can commit the other files. Select them all and click the Commit button on the toolbar. Click Continue on the warning that appears.
    You are now prompted to enter a Commit Message. This is text that you should enter to help describe the commit. Having useful information in your Commit Messages is an important part of successfully using any source control system. In this case you can enter "Initial commit." as the message and click the Commit button:



17. Now your project has been added to your Subversion repository. As you make changes to the various items in the project, you will see their "Local State" appear as "Modified" in SmartSVN to indicate that it has changed. You should Commit any local modifications you make to the repository so that others can access the changes.

18. To see how a change works, go back to Xojo and change the Title property of Window1 to "Test" and click Save. When you go back to SmartSVN, you will see "Modified" as the Local State for Window1.

You have now set up your first project with Subversion.

Additional Information

Subversion has other features for managing your project, including:

- Ignore: For excluding files in your Subversion folder from being managed by Subversion.
- Revert: For retrieving older versions of a file from the repository.
- Merge: Used to combine files that have been separately modified by two or more people.

To learn everything there is to know about Subversion, read the official Subversion book, *Version Control with Subversion*, which is available for free online: http://svnbook.red-bean.com/

Git

Git is a source control system that is gaining in popularity. Unlike Subversion, Git is a distributed source control system which does not rely as heavily on a central repository for your source code.

Git Hosting

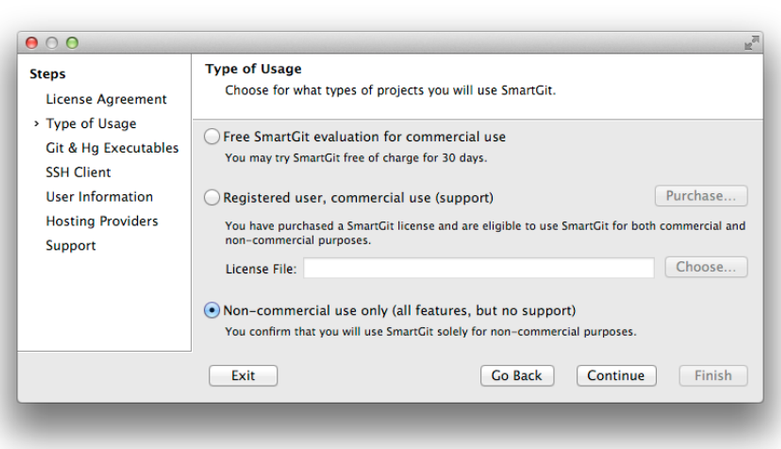
You can use Git completely on your local workstation, but there are also hosting companies that offer centralized Git services such as BitBucket, Code Spaces, Source Repo, Assembla and Beanstalk.

You can control Git using the command-line tool, called git. But there are plenty of easy-to-use graphical tools that work with Git, such as SourceTree, SmartGit, GitHub and Gitbox.

Setting up a Project with Git and SmartGit

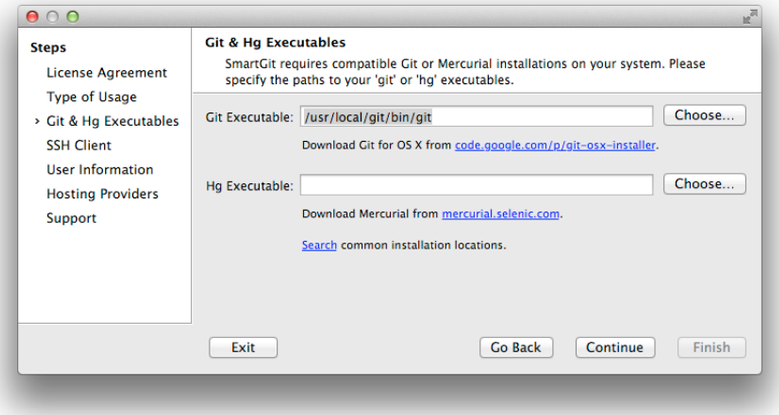
These steps describe how to use Git with your Xojo projects.

1. Download and install SmartGit: http://www.syntevo.com
2. Run SmartGit and accept the License Agreement.
3. Select “Non-commercial” use for Type of Usage and click Continue:



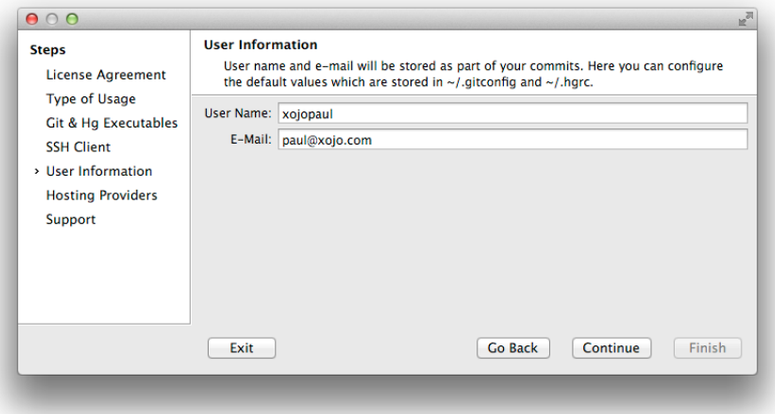
4. In the next screen, you need to specify the path to the Git command-line executable. You will likely need to download and install this separately (a link is included on the setup screen). On OS X, it is located in /usr/local/git/bin/git. Enter

the location and click Continue:



5. Leave the SSH Client setting at the default and click Continue.

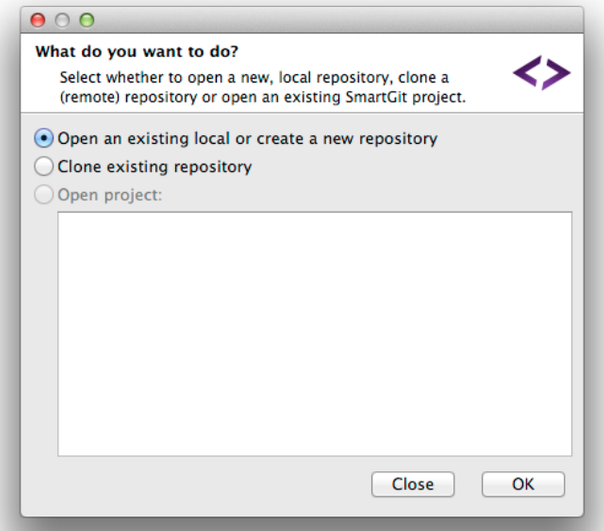
6. Now you can specify a User Name and E-Mail which are stored along with your Commits:



7. On the next screen, you can specify a hosting provider. For this example, a hosting provider is not used, so select “I don’t use a hosting provider.” and click Continue.

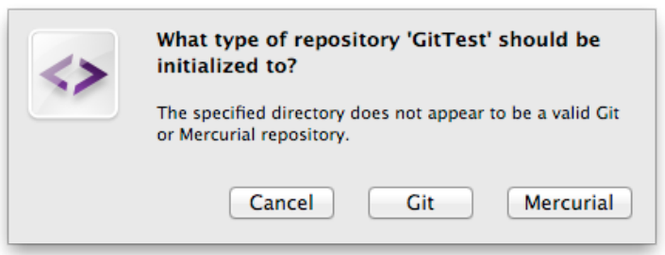
8. Click Finish on the last screen.

9. In the Project window that appears, select “Open an existing local or create a new repository” and click OK:

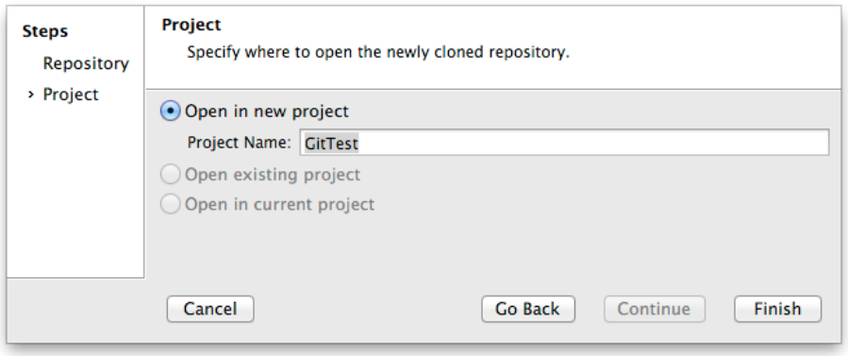


10. Select a folder for the Git repository and click Continue.

11. You will be prompted to choose the type of repository to create. Click Git:



12. Click Finish on the Project screen to open the project in SmartGit:



13. Now you can start Xojo and create a new project. Save the project to the folder you selected, ensuring that you select Xojo Project as the format.

14. Go back to SmartGit. You'll see the project items as files in the viewer. They will all have a "Working Tree State" of Untracked. Select the file with the xojo_uistate extension, right-click to open the contextual menu and select Ignore. Click Ignore in the dialog that appears. This adds a file called ".gitignore" to the folder.

15. Now you need to commit the files. Select all the files (including the new .gitignore file) and click Commit on the toolbar.

16. In the Commit window, enter a Commit Message, such as "Initial commit." and click the Commit & Push button.

17. The files will disappear from the SmartGit window. To make them appear, select "View->Show Unchanged Files" from the main menu. When you do this they all appear with a Working Tree State of "Unchanged".

18. Go back to the project and change the title of the default window and save the project.

19. When you go back to SmartGit you will see the default window now has a Working Tree State of Modified.

You have now set up your first project using Git.

Mercurial

Mercurial is a distributed version control system that is similar to Git. Use the Git instructions above to use Mercurial with SmartGit.

Chapter 6

Unit Testing

This chapter covers unit testing and the Regressinator unit testing framework.

CONTENTS

XojoUnit

What is Unit Testing?

Unit Testing is a concept for testing discrete components of your application. Unit tests typically test behind-the-scenes objects, methods and such. They do not test the user interface of your application. You typically run the unit tests frequently to verify that your application is behaving as expected.

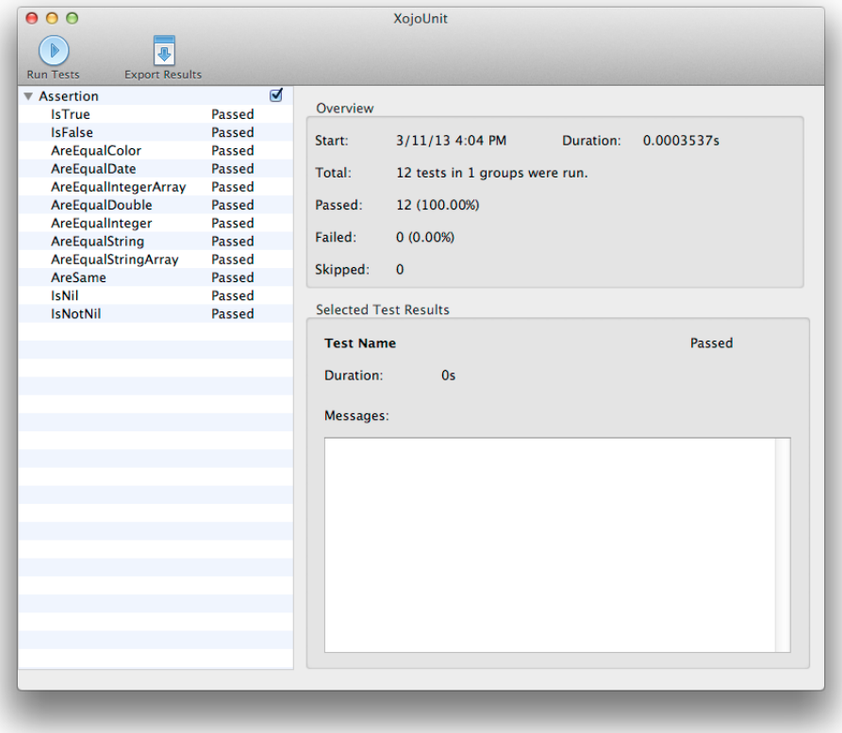
As an example, a unit test might be used to validate a calculation. Or it might be used to verify data retrieval from a database.

About XojoUnit

XojoUnit is a framework that makes it easy for you to create your own unit tests. The XojoUnit unit testing framework is available on GitHub:

https://github.com/xojo/XojoUnit

Figure 6.1 XojoUnit Window



XojoUnit works with desktop, web, console and iOS apps. Use the appropriate version with the type of app you are creating.

XojoUnit Desktop has a window that displays the tests in your application, provides a way for you to run selected tests and displays test results. XojoUnit Web displays the results in a web page. XojoUnit Console displays results in a terminal or command prompt and also outputs a text file with the results.

To use XojoUnit, copy the XojoUnit folder from one of the example projects to your project.

Adding your own tests is a simple process:

•Create a test class (e.g. MyTests) as a subclass of TestGroup.

•Create a subclass of TestController

(e.g. XojoUnitController).

- Add your test subclass (MyTests) to InitializeTestGroups event handler on XojoUnitController:

```
group = New MyTests(Self, "My Tests")
```

- Create methods in your test subclass class (MyTests) to do the tests. The methods must end in “Test” in order for them to appear in the XojoUnit results.

- Provide a way to display the results and run the tests. For desktop apps, you want to show the TestWindow, for web apps you want to show the TestPage and for console apps you want to run the tests manually.

A XojoUnit Example

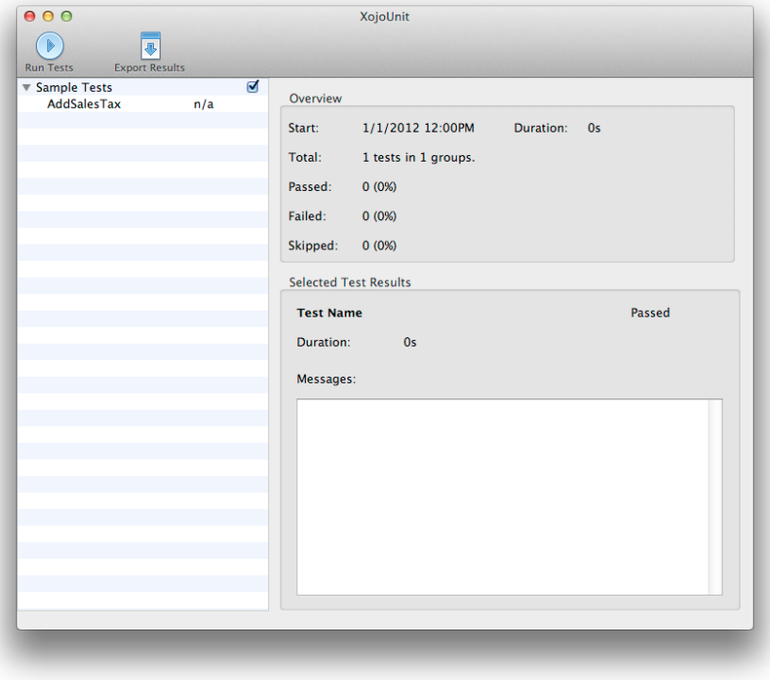
Consider an application that has a method in a Calculate class that is responsible for calculating sales tax and adding it to the supplied amount. The code might look like this:

```
Sub AddSalesTax(amount As Currency,
pct As Double) As Currency
  Return amount + (amount * 0.10)
End Sub
```

Granted this is a very simple method, but it does have a bug in it. The bug is obvious perhaps, but unit testing will help uncover it.

Here is an example of a XojoUnit test that tests a variety of calculations to make sure the correct result is returned. First, you need to create a test class to hold the unit test, so create a new class called “SampleTests” and set its Super to “TestGroup”.

Figure 6.2 XojoUnit with AddSalesTax Test



In XojoUnitController (a subclass of TestController), add this code to the InitializeTestGroups event handler:

```
group = New SampleTests(Self, "Sample Tests")
```

In the SampleTests class, add the following method to test the Calculate.AddSalesTax function:

```
Sub AddSalesTaxTest
  Dim calc As New Calculate
  Dim result As Currency
  result = calc.AddSalesTax(10.00, 0.10)

  Assert.AreEqual(11.00, result)

  result = calc.AddSalesTax(20.00, 0.05)
  Assert.AreEqual(21.00, result)

  result = calc.AddSalesTax(10.00, 0.07)
  Assert.AreEqual(10.70, result)
End Sub
```

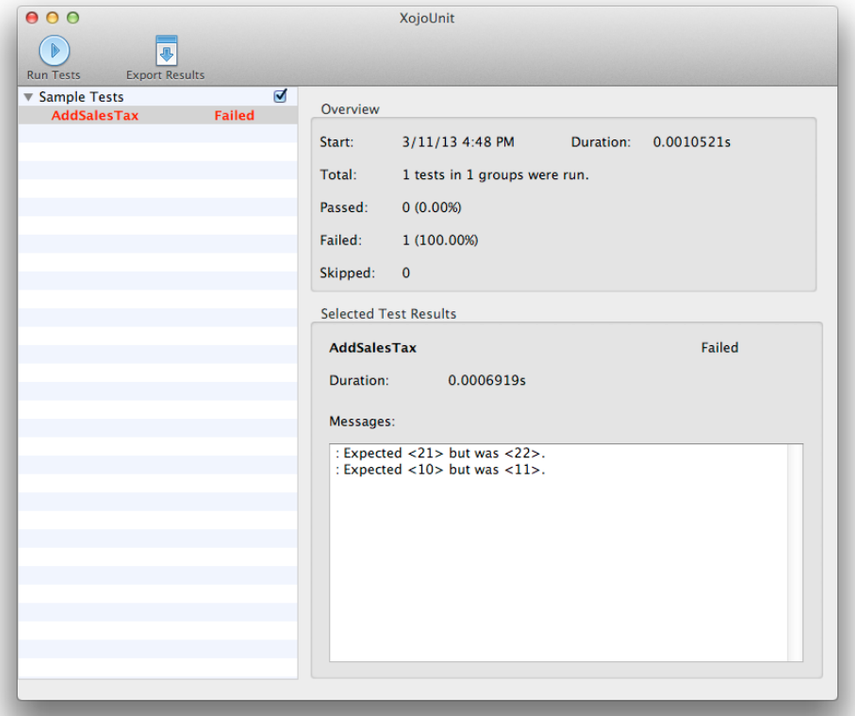
This unit test method creates an instance of the Calculate class so that the AddSalesTax method can be called. It is called three times and its result is compared with what is expected. The comparison is done using the Assert.AreEqual method. If the two values are the same, then the test passes. If the two values differ then the test fails.

Run the project to see the XojoUnit window with the AddSalesTax test displayed in the list.

Now click the Run Tests button on the toolbar. This will run the test and display the results. In this case you'll see that the test failed.

Clicking on the test shows the details for the test, including how long it took to run and the messages. Here you can see the

Figure 6.3 Test Result Failure



messages indicate that the values that were calculated are different from what was expected. This is telling us that the AddSalesTax method has an error in its calculation. Looking back at the code, you should notice that it is using a hard-coded value of 0.10 as the percent rather than using the pct parameter.

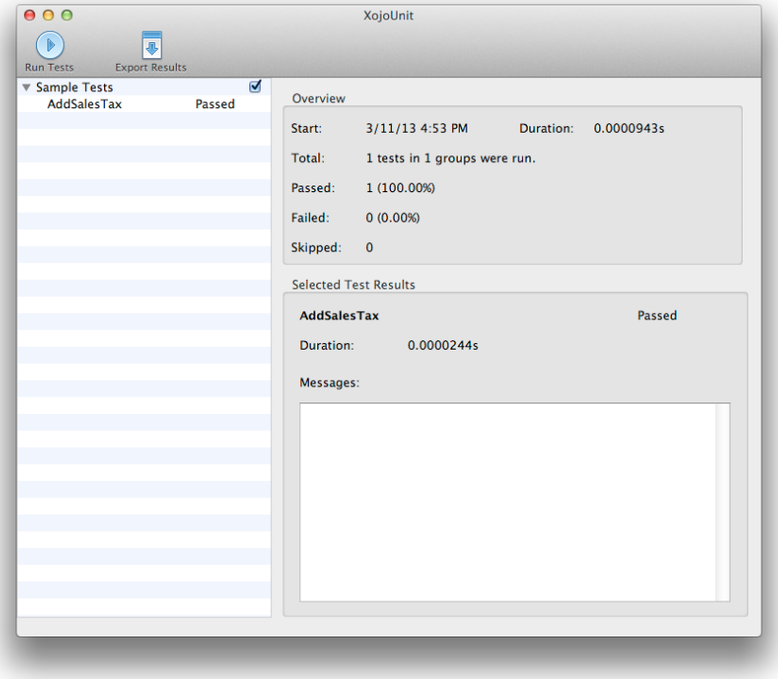
Quit the app and go back to the code for Calculate.AddSalesTax. Update it so that it uses “pct” in place of “0.10”:

```
Sub AddSalesTax(amount As Currency,
pct As Double) As Currency
  Return amount + (amount * pct)
End Sub
```

Run the project again and click the Run Tests button.

Now you will see that the test has completed successfully.

Figure 6.4 Successful Test Results



The XojoUnit Window

Each test class can contain any number of tests. You can also have any number of test classes.

You can uncheck a test class in the XojoUnit window to prevent its tests from running.

The Export Results button will export the test results as a text file.

XojoUnit Assertions

In the above example, you used a method of the Assert class called AreEqual. These are the other methods available to help you write your tests:

- AreDifferent
  Checks if the two objects point to difference references or instances.

- AreEqual
  This is the method you will use most often. It is overloaded for these data types: Color, Currency, Date, Double (and array), Int64, Integer (and array), String (and array).

- AreSame
  Checks if the two objects point to the same reference or instance.

- Fail
  Used to manually fail a test and display a message.

- IsFalse
  Tests if the supplied boolean expression is False.

- IsNil
  Tests if the supplied object is Nil.

- IsNotNil
  Tests if the supplied object is not Nil.

- IsTrue
  Tests if the supplied boolean expression is True.

- Message
  Displays a message in the message area for the test.

- Pass
  Manually passes a test and displays a message.

Chapter 7

Sample Applications

These applications are complete, working applications that demonstrate a variety of the available features.

CONTENTS

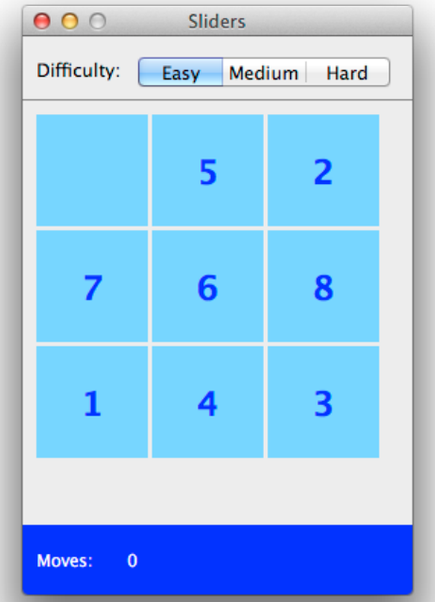
Section 1

Sliders

Sliders is an application that lets you play the “slider puzzle” game. In this game there are a series of number tiles on a grid in a jumbled order. Your job is to put all the tiles in numerical order in as few moves as possible.

This sample application is available as both a desktop and web application. It is located in Examples/Sample Applications.

Figure 7.1 Sliders Desktop App



Desktop

The desktop version of Sliders (Sliders/Sliders.xojo_binary_project) works on Windows, OS X and Linux and demonstrates several features, including:

- Canvas
- Dynamic controls
- Object-oriented design

You play the game by clicking on a tile next to the empty square. The tile you clicked then “slides” to the empty square. The object is to put the tiles in numerical order in as few moves as possible.

Each click counts as a move. There are three levels of play, which correspond to the size of the grid. Easy is a 3x3 grid, medium is a 4x4 grid and hard is a 5x5 grid.

Code Overview

Sliders primarily consists of two main objects: SlidersWindow and SliderTileCanvas.

SlidersWindow is the main window that displays the tiles and controls the gameplay.

SliderTileCanvas displays a single tile with a number.

SLIDERTILECANVAS

SliderTileCanvas is a subclass of Canvas (which is why it has Canvas as a suffix). It is responsible for drawing the tile on the

screen, including its background color and the number of the tile. When a tile is clicked, the Action event definition is called.

Most of the code is in the Paint event, where it simply sets the background color and fills the size of the tile to the color and then draws the text in the Caption property (this contains the tile number):

```
g.ForeColor = &c66CCFF
g.FillRect(0, 0, g.Width, g.Height)

g.ForeColor = &c0000FF
g.TextSize = 24
g.Bold = True
g.DrawString(Caption,
(g.Width-g.StringWidth(Caption))/2,
```

SliderTileCanvas is used on SlidersWindow to draw the tiles.

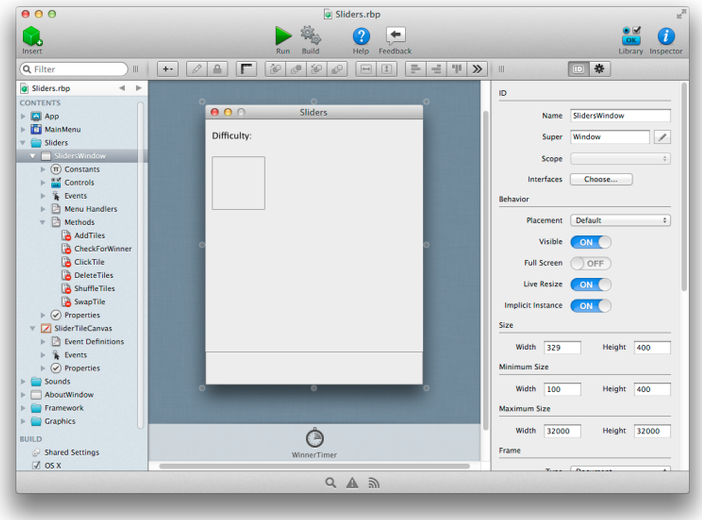
SLIDERSWINDOW

When you look at the window, you see that there is only a single tile on it. This tile is part of a Control Set. At run-time the rest of the tiles that are needed are added dynamically by the **AddTiles** method based on the selected difficulty.

The **DeleteTiles** method removes all the tiles (except the first one) so that the right number can be recreated when starting a new game or changing the level.

When the user clicks on a tile, the **ClickTile** method is called. This method verifies that the clicked tile can be moved and if so switches it with the blank tile. This method also increases the move counter.

Figure 7.2 Sliders Project with SlidersWindow Selected



ShuffleTiles shuffles the tiles for a new game. So that each game is solvable, the tiles always start in numerical order. ShuffleTiles essentially randomly clicks on tiles a large number of times to put them in a random order. This is much like what you would do to mix up a real Slider game.

Lastly, the **CheckForWinner** method is called by the WinnerTimer to check if all the tiles are in the correct order. It is done in a timer so that there is a slight pause before the “You won” message appears.

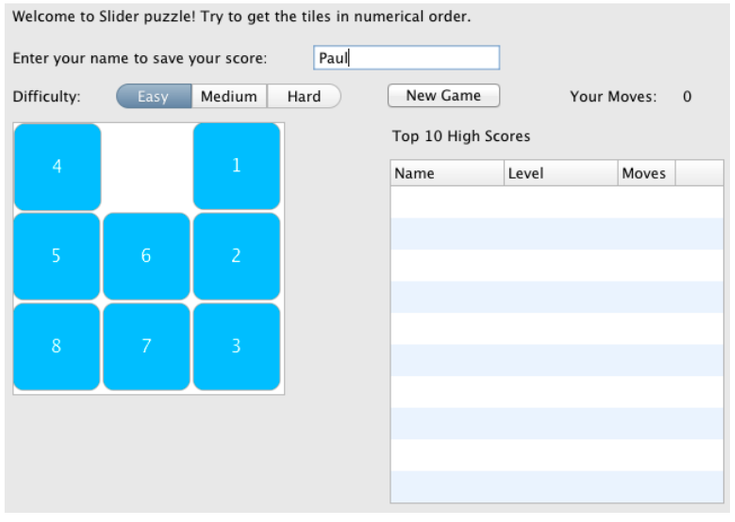
Web

The web version of Sliders (SlidersWeb.xojo_binary_project) works in all supported browses and demonstrates several features, including:

- Animator
- WebCanvas
- Dynamic Controls

In addition to counting moves and having the same levels of difficulty as the desktop app, the web version can save the high scores if you enter your name. Since the high scores are global to the web application, players that connect to the web application can see the high scores of other players.

Figure 7.3 Sliders as a Web Application



Code Overview

Sliders primarily consists of one main object: SlidersPage, the main web page that displays the tiles and controls the gameplay.

In addition, SliderTileStyle is a Style that controls the look of the tiles.

SLIDERPAGE

When you look at the web page, you see that there are no tiles on it. SliderTileContainer is a ContainerControl that contains a single tile. At run-time the tiles that are needed are added dynamically to the web page by the **AddTiles** method based on the selected difficulty.

The **DeleteTiles** method removes all the tiles so that the right number can be recreated when starting a new game or changing the level.

When the user clicks on a tile, the **ClickTile** method is called. This method verifies that the clicked tile can be moved and if so switches it with the blank tile. This method also increases the move counter.

ShuffleTiles shuffles the tiles for a new game. So that each game is solvable, the tiles always start in numerical order. ShuffleTiles

essentially randomly clicks on tiles a large number of times to put them in a random order. This is much like what you would do to mix up a real Slider game.

Lastly, the **CheckForWinner** method is called by the WinnerTimer to check if all the tiles are in the correct order. It is done in a timer so that there is a slight pause before the “You won” message appears.

HIGHSCORE

The HighScore class is used to track high scores for each player. The LoadScores method on the web page loads high scores for display on the page.

Section 2

Eddie’s Electronics

The Eddie’s Electronics sample application is a working application for a fictitious company that sells electronics which is available in the Sample Applications folder. The application demonstrates many features, including:

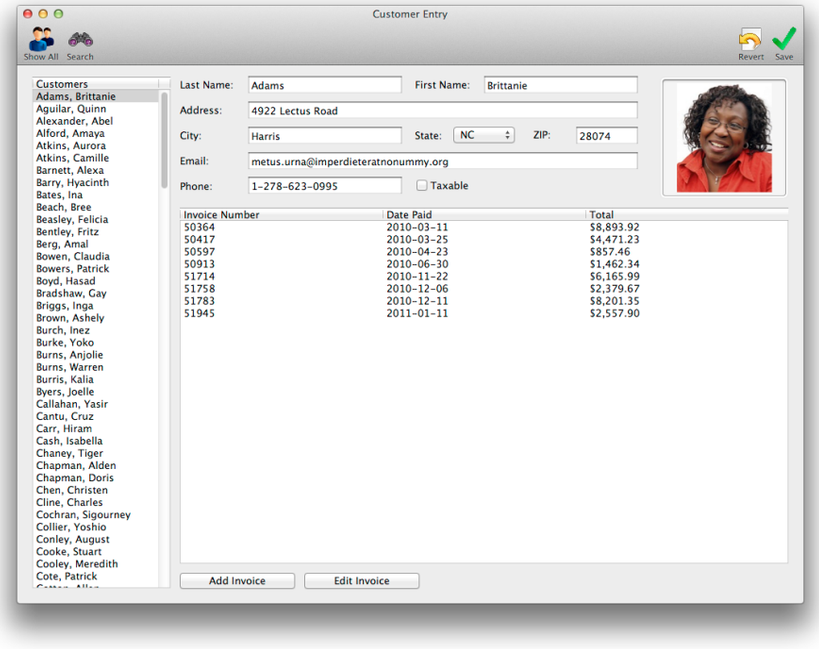
- Use of a SQLite database
- Similarities between desktop and web applications
- Layout design and interaction between different windows and web pages

This application is available as desktop, web and iOS apps, which work similarly.

Desktop

The desktop application includes a pre-populated database with sample data for many customers and orders. The database is recreated each time the application runs, so any changes you make to it are discarded when you quit the application.

Figure 7.4 Eddie’s Electronics Desktop Application



Click on a customer to view their information, a picture and to see the orders that they have placed.

You can click on individual invoices to edit the order or you can add new invoices for new orders.

Code Overview

The desktop application has four windows for displaying information. The main window, CustomerDetailsWindows, displays the customers and their orders. SearchWindow is a dialog (a drop-down sheet on OS X) that lets you search for customers. AboutWindow displays the logo and version. InvoiceDetailsWindow is used to add or edit invoices.

The OrdersDatabase class handles all access to the database. It has methods for finding customers, invoices and handling errors.

APP

The App class initializes the database in the **Open** event by calling the shared method SetupNewDatabase.

ORDERSDATABASE

The OrdersDatabase class is a subclass of SQLiteDatabase. It is used for all database access. The methods that find or get data return a RecordSet that you can then process to display or update data.

The shared method, **SetupNewDatabase**, creates a new in-memory database for use by the app each time it starts.

CUSTOMERDETAILSWINDOW

The CustomerDetailsWindow is the first window that appears when the application starts. It loads all the customers into the CustomerList ListBox on the left (using the **LoadCustomers** method).

When you click on a customer, its data is loaded into the fields using the **LoadCustomerFields** method. Similarly, LoadInvoices loads all the invoices for the customer into the InvoiceList ListBox.

The **SearchForCustomer** method is used to find one or more customers by name and is called when the Search feature is used.

The **Save** method saves any changes made to the customer fields back to the database.

INVOICEDETAILSWINDOW

The InvoiceDetails window displays invoice details for an existing invoice and allow the data to be changed. It is also used to create new invoices.

SELECTABLEPOPUPMENU

This is a subclass of PopupMenu and adds a method, **SelectValue**, that will select a specific value in the popup. It is used to select the appropriate state for a customer.

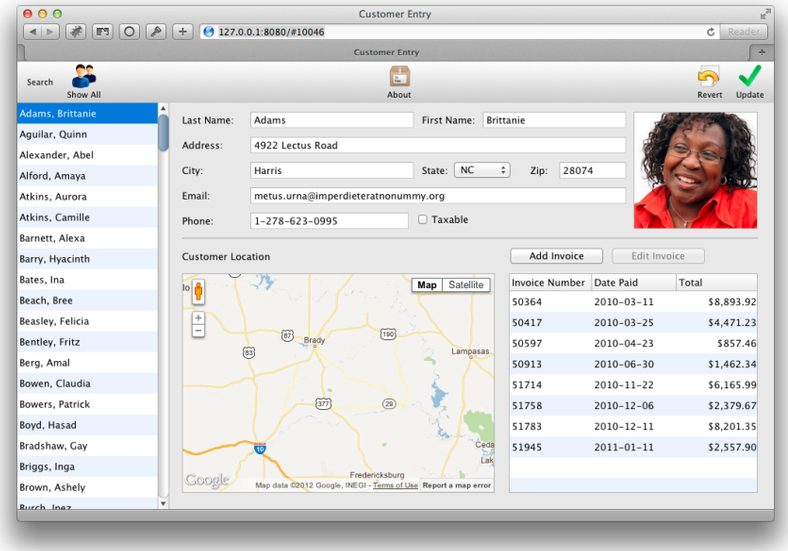
Web

The web version is very similar to the desktop version. You'll notice that the major Windows in the desktop application are Web Pages in the web application.

The web application has error logging (to both a database and a file).

The web application also uses the MapViewer control to display the customer's location on a map.

Figure 7.5 Eddie's Electronics Web Application



CODE OVERVIEW

The web application is structured similarly to the desktop application, but there are several changes needed in order to accommodate the needs of a web application.

First, since a web application is multi-user by default, a new database is created for each session that connects to the web application.

The web application logs errors to a table and text file since it is not possible to display a message unless a session is connected.

The Search feature is embedded in the toolbar instead of showing a separate dialog.

The AboutDialog and InvoiceDetails are WebDialogs instead of modal Windows.

The web version also has separate pages that are designed specifically for small screen mobile devices such as an iPhone.

In short, much of the user interface layout is different for the web application, but you will see that a lot of the code is almost exactly the same or is extremely similar to the desktop version.

In particular, the web edition also uses the OrdersDatabase class for all database communication.

SESSION

The Open event of the Session object is called each time a user connects to the web application. In this event, a new in-memory database is created.

CUSTOMERDETAILSPAGE

This is equivalent to CustomerDetailsWindow in the desktop application. It displays when a user first connects to the web application. It loads all the customers into the CustomerList ListBox on the left (using the **LoadCustomers** method).

When you click on a customer, its data is loaded into the fields using the **LoadCustomerFields** method. Similarly, LoadInvoices loads all the invoices for the customer into the InvoiceList ListBox.

The **SearchForCustomer** method is used to find one or more customers by name and is called when the Search feature is used.

The **Save** method saves any changes made to the customer fields back to the database.

CUSTOMERINVOICEDIALOG

Equivalent to InvoiceWindow, the InvoiceDetailsDialog window displays invoice details for an existing invoice and allow the data to be changed. It is also used to create new invoices.

MOBILE SUPPORT FOLDER

The Mobile Support folder has several alternate pages that are displayed when a user on a small mobile device connects to the web application.

These pages have a different layout to show less information, but generally work the same.

iOS

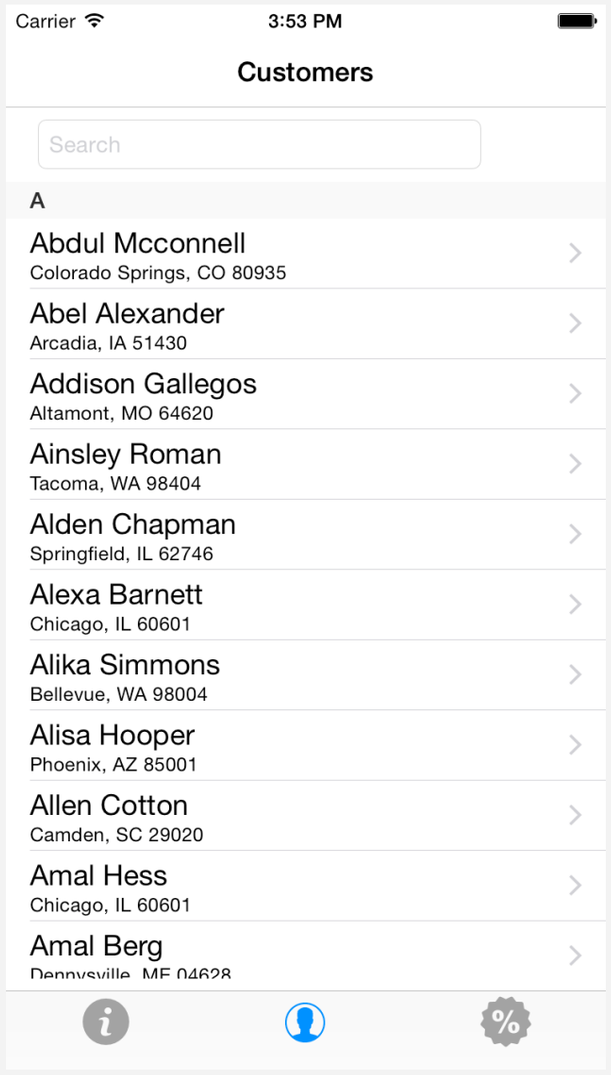
The iOS version of Eddie's Electronics works similarly to the desktop and web versions. The app consists of several views that

control each major area of functionality, all arranged within a TabBar.

CUSTOMERVIEW

This displays a list of all the customers in the database. Clicking on a customer displays the CustomerDetailView.

Figure 7.6 iOS Version of Eddie's Electronics



CUSTOMERDETAILVIEW

The view displays the detail information for the selected customer. From here, the user can return to the list of customers, click the Edit button to edit the customer, show the invoices for the customer or open the Maps app with the location of the customer. When the user clicks the Edit button, the CustomerEditView is displayed.

CUSTOMEREDITVIEW

The CustomerEditView displays the customer information so that it may be edited. Clicking Done saves the data back to the database.

INVOICEVIEW

The InvoiceView displays all the invoices associated with the customer.

SALESVIEW

The SalesView is accessible by clicking on the sales button in the TabBar. This displays all sales data by quarter. Swipe left and right to move between quarters.

CUSTOMER

The Customer class contains information about a specific that gets loaded from the database. When the app starts, the database table is loaded into an array of Customer instances. This array serves as the DataSource for the table that displays the list of customers.

INVOICE

The Invoice class contains information about the invoices for a specific customer. This information is loaded from the database.