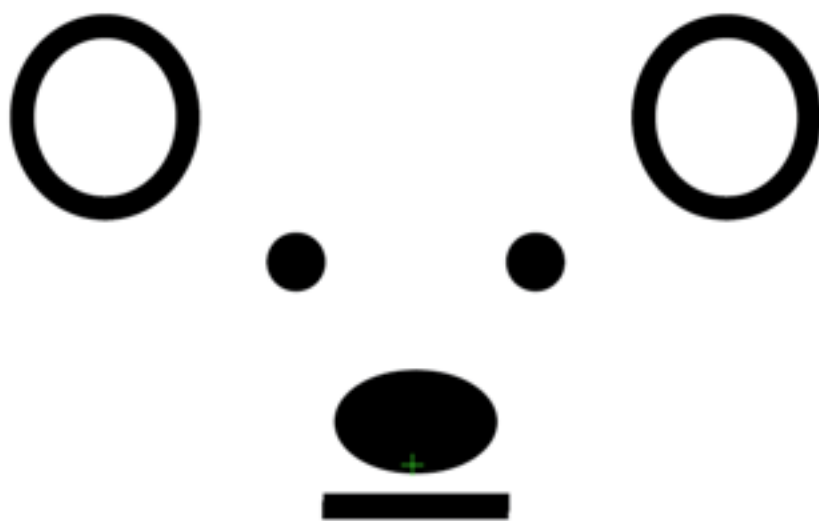


# ArchiveXojo Manual



*ArchiveXojo*

ArchiveXojo Manual	1
Introduction	1
What is Provided	2
Installing the Archive Xojo System	2
Learning to Use the Archive Xojo System	2
Help	3
Terminology	4
Code Listing or Handler	4
Method	4
Updating Xojo	4
Workflow	4
ArchiveXojo Database	6
ArchiveXojo – Location of Files on Disk	7
ArchiveXojo – Time Stamps	7
ArchiveXojo Menu Bar	7
Add	7
Import Code from XojoInfo Text File	7
Import Entire Xojo Project	8
View	8
Code Listing	8
Version - possibly to be introduced in the future	10
Import Log	10
Notes	10
Report	10
Characters Words Lines	10
Delete	11
Code Listing	11
Thin Out Stored Methods	11
Duplicate...	12
Export	12
Multiple Code listings	12

Newer Version in Other Project	13
Methods Unique to Project	13
Snapshot	14
GIT	14
Misc	14
User Preference	14
Advanced User	16
What would need to be modified	19
Appendix	20
Kaleidoscope - Black Pixel	20
Contact	21
Screenshots	22
Search Listing Window	22
Code Window	23
Code Size Report	24
Code Versions	25

# Introduction

*Archive Xojo System* is the composite of several things. It is a version control system. It is a project documentation tool. It is a central code repository making it easy to find and reuse code created in any of your *Xojo* projects.

*Archive Xojo System* stores code, as it is being developed, into an independent text file (**XojoInfo.txt**). At intervals, you can bring the content of this text file into the **ArchiveXojo** database.

Storing all your code changes as you work, the use of *Archive Xojo System* removes some of the anxiety inherent in altering code.

What if there is some bad *Xojo* crash during development session? Well, you have your code saved as text files. What if you embraced some new idea and enthusiastically set about refactoring existing code and then realized that it was a bad idea? Well, you have a bread crumb trail that basically allows you to back up from what turned out to be a mistake.

This is a great comfort and enhances your ability to be agile and move quickly with development. You can go back to any previous version of an individual bit of code.

*Archive Xojo System* is also a resource. All your old code from all your projects is available in one place for possible reuse in new projects. Refactoring old code and reusing old code is part and parcel of being a *Xojo* programmer. The **ArchiveXojo** database has all the tools of a database to find and grab bits of your previous work.

The *Archive Xojo System* relies on AppleScript for a part of its functionality so is provided only for users of the Mac. The **ArchiveXojo** database technically is dual platform and possible in the Windows environment, but requires licenses that I do not have and cannot afford for a freeware offering.

The **ArchiveXojo** database comes as an “empty” database. You fill it with your code from any or all of your *Xojo* projects.

# What is Provided

*Archive Xojo System* is available as a zip file from **Bearboat**. That zip files contains:

1. **ArchiveXojo** database
2. Xojo IDE script (**FormatCode**) to “pretty format” the code in the foremost *Xojo* window (Cowgar)
3. Xojo IDE script (**GrabAndAppend**) to place code from the foremost *Xojo* window into the Clipboard along with a small amount of metadata. It fires up the AppleScript (**CaptureXojoCode**) and passes this information on to that script.
4. AppleScript file (**CaptureXojoCode**) does some further processing, and finally appends the code and metadata to the **XojoInfo.txt** text file.



## Installing the Archive Xojo System

There is a separate document (*InstallationArchiveXojo*) that describes this process. The installation is a bit fiddly, particularly for someone who has never used IDE Scripts. The **ArchiveXojo** database itself can be placed in the *Applications* folder. There are two IDE Scripts that have to be placed in the *Scripts* folder of the the user’s current version of *Xojo*. The AppleScript file (**CaptureXojoCode**) has to be placed in some “reasonable” location and then one of the IDE scripts (**GrabAndAppend**) has to be edited to “tell” that script where your version of **CaptureXojoCode** has been stored. This process is described in the *InstallationArchiveXojo*.

## Learning to Use the Archive Xojo System

There are two learning resources: this Manual and Help windows contained in the application itself. The whole thing is not very complicated.

# Help

Many windows in the **ArchiveXojo** database application have an orange  in them. Click on the  to show a contextual help window.

## Terminology

---

### Code Listing or Handler

In the *Xojo* IDE, you write code in a code editing window. The contents of one of these code editing windows, including its parameters and return values if applicable, is considered a single code listing or a handler. These are what make up the individual records in the **ArchiveXojo** database. Sometimes this is informally referred to as simply “code” if the meaning is clear in context.

---

### Method

In different programming languages, the word method tends to connote slightly different things, but here I am referring to the code that *Xojo* refers to as methods, be those part of a class or a window or a module. Methods are a subset of code listings. In *Xojo*-speak they include functions.

---

## Updating Xojo

When new versions of *Xojo* are deployed by the user, the provided IDE scripts have to be copied into the *IDE Scripts* folder of that new version of *Xojo*.

## Workflow

When new code listings are created or modified, the user fires off an IDE Script (**GrabAndAppend**), generally by just typing a hot key that has been assigned to that script. That script grabs the code in the current window and saves its current state to a text file called **XojoInfo.txt**. This file is stored in a folder called *\_ArchiveXojo* in a location specified by an AppleScript (**CaptureXojoCode**). By default that location is the Desktop. So the path is: `~/Desktop/_ArchiveData/XojoInfo.txt`

In addition to the code itself, some metadata is also automatically stored in this text file. The date and time, the name of the *Xojo* project etc.

How often should the developer fire off the IDE Script? I do it whenever I plan to run *Xojo* to test my modified code or when I have modified and am preparing to leave some code window. I do it often before I am going to try some new, bold change in my code. Do it often! There is no penalty. The **ArchiveXojo** database easily handles dealing with many, many versions of code. The **ArchiveXojo** database will avoid importing exact duplicate code. It is always possible, but seldom really necessary, to selectively delete code versions from the database in the future should you desire. So basically, do not hesitate to fire off the IDE Script at will.

At some flexible interval, perhaps after the developer has completed some “task” or when the **XojoInfo.txt** is getting “big” or when there is a need to refer to the versions of code contained in that file, the developer imports the contents of this file into the database. The text file itself is human readable in a pinch, but its content is more approachable after it has been brought into **ArchiveXojo**.

In this way, the user builds a database of all versions of all the code that he is creating in all his *Xojo* projects. That **ArchiveXojo** database become a resource for developing further code. If some weird crash occurs in the course of development, most of your work will be retrievable from the **XojoInfo.txt** file that has been tracking your changes while you work.

**NOTE:**

The IDE Script, (**GrabAndAppend**), contains the (**FormatCode**) IDE script - available do to the efforts of Jeremy Cowgar. The **FormatCode** script makes the formatting not only “pretty” but more uniform which makes it easier to see the real differences between code versions. If the individual user does not like this formatting, he could remove the one line of code from the **GrabAndAppend** script that calls the **FormatCode** script.

You should be aware of one little thing. There is a “bug” in the **FormatCode** script related to the fact that it does not handle characters that are “outside” the ASCII range. Such characters will not occur in your code itself, but could occur in a comment or a string. So a character such as bullet, • , or a ¢ , or a § , etc. can disrupt the proper functioning of the **FormatCode** script and can cause drop out of a couple of characters from the saved version. I tend to avoid these characters, but if it becomes a major problem for you it is possible to remove the line of code from the **GrabAndAppend** script that calls the **FormatCode** script or to have two



versions of the **GrabAndAppend** script, with and without the call to the **FormatCode** script. If these two versions were assigned different shortcuts, these could be called selectively to deal with some particular handler code that might have a lot of characters out of the ASCII range.

## ArchiveXojo Database

The **XojoInfo.txt** text file will store all the versions of all the code created by the programmer. The text file is human readable. You could let this file just get bigger indefinitely and just use a text editor to search it for what you needed. But eventually, as it grows, it becomes unwieldy.

When the developer decides the time is right, she runs the **ArchiveXojo** database and imports all the contents of this text file into the database.

Open this database and go to the **Add** menu. Select *Import Code from XojoInfo Text File* and navigate to the **XojoInfo.txt** text file (it will be in a folder *\_ArchiveXojo*). Open the text file and all the individual versions that have been stored here will be imported into the database. The **XojoInfo.txt** file itself, having served its purpose, will be deleted. When changes in the code are made in the future, a new **XojoInfo.txt** text file will be automatically created and the cycle can begin anew.

Meanwhile, the **ArchiveXojo** database will allow you to search and copy and print all these code versions with the tools inherent in a database. When using **ArchiveXojo**, the code from a multitude of *Xojo* projects is all available in one location.

If a brand-new project is created, it is all very simple. All of the versions of all of the code in that database as they are being created can end up being entered into the **ArchiveXojo** database as described above.

However if a pre-existing *Xojo* project exists, built before the use of the *Archive Xojo System*, it is usually desirable to bring the existing code into **ArchiveXojo**. A tool to accomplish this is provided. Create a folder and save the pre-existing *Xojo* project in text format (Save As Xojo Project). Under the **Add** menu, select the menu item *Import Entire Xojo Project* and specify the folder. **ArchiveXojo** will scan the folder, find all the code within it, and bring it into the database.

## ArchiveXojo – Location of Files on Disk

Much of the functionality of **ArchiveXojo** involves creating various files, derived from the code archive, that are useful to the programmer and saving them to the hard disk. Small text files exported from **ArchiveXojo** are commonly simply created on the Desktop or the *RootFolder* for ready and immediate access. More complicated text files or folders of text files are created within the *RootFolder*.

The user can change the *RootFolder* in *User Preferences*.

## ArchiveXojo – Time Stamps

A common convention of this program is to prefix files or folders with a timestamp. This is generally of the form YYYYMMDD\_HHMM. For example, a file created on October 27, 2014 at 10:33 PM will be prefixed - 141027\_2233. This naming convention assures that files are arranged in the Finder in chronological order.

## ArchiveXojo Menu Bar

Here is a quick run-through of the Menu Bar of **ArchiveXojo** to provide a summary of the functions of this database.

### Add

---

#### *Import Code from XojoInfo Text File*

This menu item is used to import code listing versions from the **XojoInfo.txt** file.

When the file is imported, its contents end up in **ArchiveXojo**, and the file is erased from disk. It will be automatically recreated when the user modifies and creates new code in *Xojo* projects.

---

## Import Entire Xojo Project

A *Xojo* project that has been saved in text format in some folder can have **all** its code imported in one fell swoop into **ArchiveXojo**. The intent would be so do this once for an existing project and have subsequent data entered by importing from the **XojoInfo.txt** file.

It can take a minute or two to import a large *Xojo* project using this menu item.

Technically, since **ArchiveXojo** avoids importing duplicate records, you could repeatedly import an entire project and build up your versioned database in this way. It makes things initially simple, because most of the setup described in the document *InstallationArchiveXojo* is unnecessary.

For example, using this approach, every week you could import the current state of your code from a saved text format version. Only handlers that had changed since that last import would actually be brought in, since duplicates are ignored. While some might be content with this, it is not really how the *Archive Xojo System* was designed. The versioning would not be as fine grained. The date/time stamps would be relatively crude (simply representing the time that the code was imported rather than when it was actually modified). The “bread-crumbs” trail back into the past would be less complete.

## View

---

### Code Listing

The heart of the program is the ability to view all of the code listing versions that have been captured. Selecting this menu item provides a list of all of the code listing versions that exist in the database. In the listing form, across the top, are text entry boxes and pop-ups and checkboxes that act as filters on the list so the code listings visualized can be winnowed down to those desired. This filter area exists as a region of yellow and green occupying the top of the form. After changes are made in the filters, the user has to wait for a few seconds for the filtering to take effect.

It is possible to search within the text of a method. Two text entry boxes are available. If **One Line** is checked then the contents of **both** the text entry boxes have to appear somewhere in the **same** line for a match to occur.

When searching in the text of a code listing, there is a special technique to search for a complete word. Surround the word with spaces in the text entry box.

SpaceSinSpace ( **Sin** ) will find a code listing with the line { answer = Sin(number) } because the search will be modified to look for complete words and **Sin** appears as a complete word in the line. The search will not find code with the line { sum = 2 + Single }.

Double-clicking on any listed method opens a window containing the code of that method. On the left side is the code with the comments. On the right side, the code with all the comments stripped out is provided.

From this window, it is possible to do a variety of things with that code. It can be placed on the Clipboard. It can be exported as a text file to the Desktop (or *RootFolder*). A search box allows highlighting any particular string that occurs in that code.

Handler code from Xojo will be “wrapped” with something like Sub name\_method ... End Sub. These first and last lines are not appropriately pasted into another Xojo handler window. With Xojo, the name information for a method is entered elsewhere. If the goal is to paste code into a Xojo handler code window, hold down the Option-key when clicking on the Clipboard button. This will exclude the first and last lines of the handler code from what is placed in the Clipboard.

If there are multiple versions of a particular code listings, there will be a button, **Compare Versions (n)**, that will open a new window that provides access to all the versions of that single code listing that exist in **ArchiveXojo**, possibly from multiple projects. The window will show two versions of the code, the older on the left.

It can be hard to find the differences between two versions of code. It may require exporting the text of the two versions to text files and comparing them in an application that has DIFF functionality (can compare two text files and indicate all the differences between them).

On the left side of the window there is a colored circle with a percentage indicating just how similar the two versions are. If you click on that circle, the lines of code that are different, one version to the other, will show in orange text. This will often suffice to draw your eye to the differences of the versions without having to launch a full-strength DIFF application for a more complete evaluation.

---

### *Version - possibly to be introduced in the future*

This menu item offers slightly different functionality than *Method*. Only methods that have multiple versions are accessible through this menu item. The user selects a particular method and that method is opened in the form that shows all the versions of that selected method.

---

### *Import Log*

The import log tracks the date and time that records are imported into **ArchiveXojo**.

When you import methods, you are given an opportunity to store a comment about the methods that have just been imported. If you do not want bother with this extra step, then the dialog box can be turned off in the user preferences.

Some might find it useful to record a comment when some computing subtask has been completed and the code related to that subtask imported into **ArchiveXojo**. This gives the user date/time stamped notes on the progress of coding (similar to the comments that can be left in GIT with every commit).

---

### *Notes*

View and create notes. You can use the ability to store notes any way you want. There is no need to make any notes at all.

## **Report**

---

### *Characters Words Lines*

The user selects a particular project. The number of characters, words, and lines in that project is provided. It is possible create a change over time look at these metrics.

## Delete

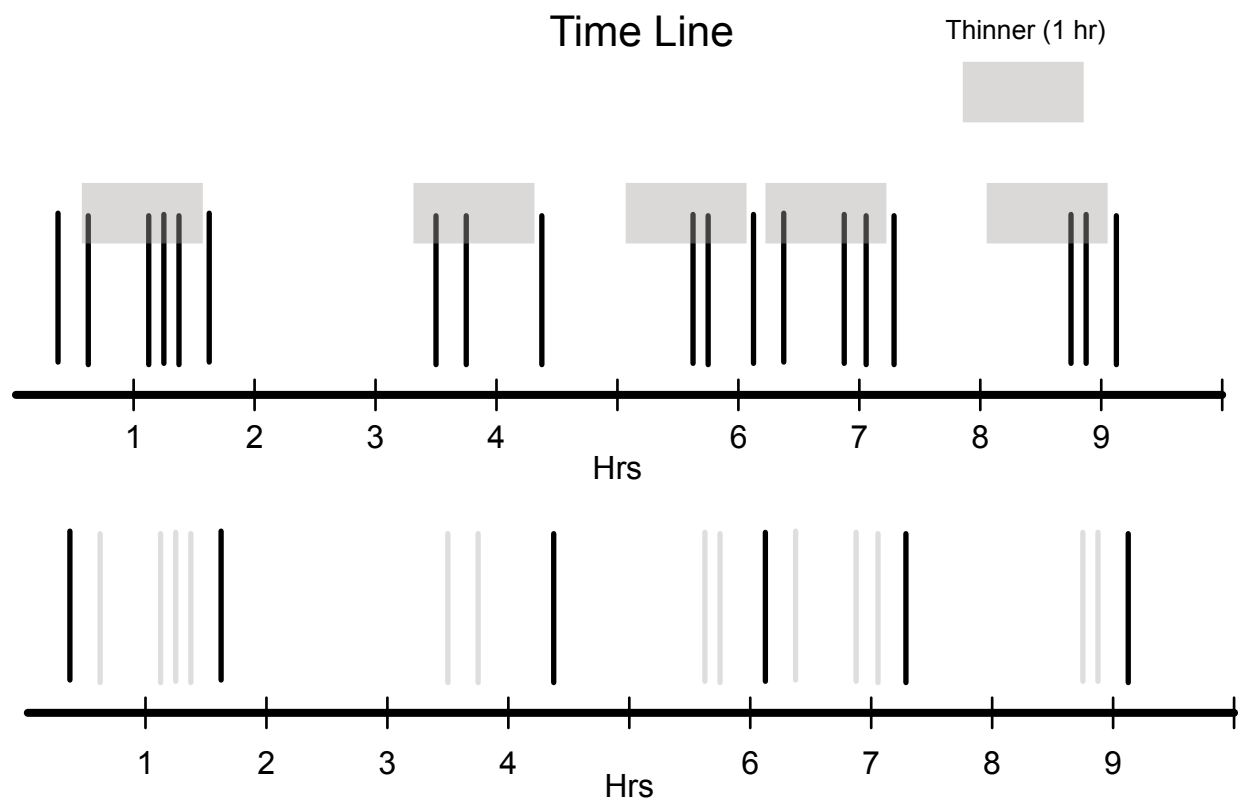
### Code Listing

It is possible that one might want to delete code listings from the database. Here the user can manually select code listing versions to remove from **ArchiveXojo**.

### Thin Out Stored Methods

Commonly, when working on the code of a method, a large number of versions will be generated by the text/edit cycle. In itself, this is not a big problem. Certainly, the **ArchiveXojo** database is capable of holding a very large number of versions. Nonetheless, some users might consider this to be uncomfortably cluttered, and this menu item allows an “intelligent” removal of versions that are separated by very short period of time.

The algorithm for this removal is that the most recent version is preserved and then everything within the time period chosen is removed. Then the next earlier version is preserved and then everything within the time period of that version is removed. Below is an attempt to show this algorithm graphically with a 1 hr period selected as an example.



Typically code development occurs in spurts and versions of a particular method might be distributed as seen in the example. After the thinning is applied, the versions shown in gray are deleted.

---

### *Duplicate...*

Sometimes duplicate code will be saved in the **ArchiveXojo**. This occurs when the identical code listings exists in multiple databases that are all being tracked by **ArchiveXojo**. Again, this is not really a problem. By default, **ArchiveXojo** stores code with identical content if it comes from different projects. But if this style does not suit you, this menu item offers a tool to remove such duplicates and get rid of this redundancy.

**ArchiveXojo** avoids storing identical code from the **same** project. Subtle differences such as capitalization suffice to make two versions not identical.

### **Export**

**ArchiveXojo** is a repository of lots of code that can be exported in various ways as text files. These can be used to help the programmer with current and future products.

---

### *Multiple Code listings*

Under the **View** menu, the *Code Listing* menu item allows the export, to text files, of the code of individual code listings. But there is also a time that it is useful to specify a large group of methods for which you want to have the code as text files. This menu item provides this functionality.

There are buttons in the window that determine what type of file(s) will be created out of the selected methods.

This button...	Does this...
<p><b>Export Simple Text Files</b></p>	<p>Exports a folder full of simple text files with all the selected methods as individual files.</p> <p>To export all the selected methods into a <i>single</i> text file use checkbox <b>One Document</b>. The single text file created will have a table of contents at the beginning. In this way, you could, if you wished, create a single text file that has the most recent version of all your code as one document. It is a form of backup, a source for copy/paste and a place where code can be reviewed. An example of this kind of file can be seen in the appendix.</p>

---

### *Newer Version in Other Project*

In the course of refactoring and hopefully improving code, it can come to be that a code listing used in multiple other databases is improved in the project currently under development. Perhaps you would like to consider updating these other projects but in the moment, you simply prefer to continue to perfecting the project currently being worked on.

At some later date, the developer may find himself working on another database. The thought kicks around in your head that there is an “improved” version of various bits of code that exist elsewhere which are more recent and presumably “better”. This menu item helps in discovering such methods.

---

### *Methods Unique to Project*

Some developers maintain a dummy project that contains many methods that have been found to be useful in other similar projects. Future projects might start as extensions of this dummy project. It may come to pass that the developer wants to know what methods in this new project are unique to it, i.e. not methods inherited from the dummy project. This menu item helps in identifying such code listings.



---

## Snapshot

Select a particular database and then specify a date. Click the button, **Create Snapshot**, and a text file will be created that contains all the code listings for that project as they existed on the date specified. If you create multiple text files with different dates, they can be compared with a DIFF tool such as *Kaleidoscope* to see the differences between the state of the project at different times.

---

## GIT

At intervals determined by the user, the code from a chosen project can be placed as a collection of text files into a folder, *GIT\_Xojo* in the *Documents* folder (you can specify an alternative location in *User Preferences*). A user that is familiar with GIT can designate this folder as a Repository and changes in that code overtime can be recorded in GIT.

When you do a GIT export, by default it exports the code as it exists at the present time. It is possible to adjust this by adjusting date/time controls to do an export of the code as it existed at some time in the past.

This would only rarely be useful. Perhaps if you worked on two distinct “things”, one a week ago and another in the last week, you might want to “split” this work into two groups in your GIT time-line and this function would enable that. All that work may have been imported into **Archive Method** in one import, but the export into your GIT repository would be divided into two by adjusting the date/time control that is provided.

This is a rudimentary use of the GIT tools. GIT is a big topic and one well beyond the scope of this discussion.

---

## Misc

---

### User Preference

#### Root Folder

The user can decide where to place the files that are created by **ArchiveXojo**. On the Desktop (which is the usual, or at least default choice) or in a subfolder of the *Documents* folder called *\_Athenaeum* or in *any* folder that the user decides to designate.

Even if you prefer exporting most files to a custom root folder, you might prefer exporting the text of **single** methods directly to the Desktop where they are readily accessible. There is a checkbox, [Export Single Method Text to Desktop](#), to enable that preference.

### GIT

For those familiar with GIT or interested in GIT, there is an option to create a folder on the local hard drive to which the most recent version of the text of methods can be exported whenever desired. **ArchiveXojo** simply creates that folder. Here the user determines where that folder will reside.

The user must use GIT tools to designate that folder as a GIT repository that GIT is “watching” so that versions of all the methods can be tracked by GIT. For those that pursue this, one has, in effect, a second version control system for the *Xojo* code you develop.

### Alerts

Many Alert boxes appear to inform the user of various things in the course of using **ArchiveXojo**. By default, many of these Alert boxes are self-dismissing. What is meant by this is that these small windows will go away on their own without the user actually having to do something (click on the [Done](#) button).

There is a countdown area showing just how much longer the Alert box will remain. If you click on that countdown, you get a little more time to read the contents of the alert.

When you first start using the program, you tend to need a little more time to absorb the content of these alerts. Later on, less time. The radio buttons here allow adjustment of just how long these Alert boxes stay up.

These Alerts tend to run in their own process so they can be ignored. The user can continue to work with the program and be confident that eventually they will go away by themselves. This is occasionally convenient when doing something like quickly creating text exports from old and never versions in the compare code listing versions window.

If the user does not want to bother with having self-dismissing Alerts, they can be turned off here.

## Notes

When a group of recent method versions is added, by default the user is provided with an option to create and leave a note about the import. This is analogous to the comments that are commonly created when committing versions with GIT.

If you do not find such comments to be useful, continually being asked whether you want to create a note is likely to be annoying. This can be turned off here.

## Updates

From time to time, it is possible that **Bearboat** will create updates for the Help files. **Bearboat** will provide text files containing the relevant information. Click on [Update Help...](#) to access and import those files.

# Advanced User

If you are to some degree familiar with AppleScript and you are annoyed by the particular location that the file XoyoInfo.txt files ends up, you can change this.

This location `~/Desktop/_ArchiveXoyoData/XoyoInfo.txt` is determined by the AppleScript. The AppleScript decides what folder XoyoInfo.txt is going to land in.

Lets look at the AppleScript. I have included the entire script below for completeness and then I look specifically at the areas that would have to change.

```
### Make sure that the folder _XoyoArchiveData exists in the appropriate
location ###
#           Courtesy of JMicheelTX http://macscripter.net/viewtopic.php?
id=41721 #

# Makes the initial folder on the Desktop
set newFolderPath to quoted form of (expandPath("~/Desktop/
_ArchiveXoyoData"))

set cmdStr to "if [[ ! -d " & newFolderPath & " ]]; then
mkdir -m 755 " & newFolderPath & "; fi"
do shell script cmdStr

# makes a subfolder in the folder created above if you want or need
```

```

# set newFolderPath to quoted form of (expandPath("~/Desktop/
  _XojoArchiveData/NewFolder"))

# set cmdStr to "if [[ ! -d " & newFolderPath & " ]]; then
# mkdir -m 755 " & newFolderPath & "; fi"
# do shell script cmdStr

on expandPath(pPathStr)
  local fullPath
  set fullPath to pPathStr

  if fullPath = "~" then
    set fullPath to (POSIX path of (path to home folder))
  else if fullPath starts with "~/ " then
    set fullPath to (POSIX path of (path to home folder)) & text 3
      thru -1 of fullPath
  end if

  return fullPath
end expandPath

### Append the contents of the code on the current foremost window of
  Xojo to the file XojoInfo.txt ###

set xojoClip to the clipboard
set newLine to return & return & "<newsave> * * * * * * * *
  * * </newsave>"
set briefDate to do shell script "/bin/date +%Y%m%d"
set briefTime to do shell script "/bin/date +%H%M%S"
set briefBoth to "<datetime>" & briefDate & "_" & briefTime & "</
  datetime>"

set delta to ((time to GMT) / 36)
set deltaTagged to "<timezone>" & delta & "</timezone>"

set xojoClip to newLine & return & briefBoth & return & deltaTagged &
  return & xojoClip

set this_file to (((path to desktop folder) as string) &
  "_ArchiveXojoData:XojoInfo.txt")

```

```

my write_to_file(xojoClip, this_file, true)

on write_to_file(this_data, target_file, append_data)
  try
    set the target_file to the target_file as string
    set the open_target_file to open for access file target_file with
      write permission
    if append_data is false then set eof of the open_target_file to 0
    write this_data to the open_target_file starting at eof
    close access the open_target_file
    return true
  on error
    try
      close access file target_file
    end try
    return false
  end try
end write_to_file

### Empty the clipboard
tell application "System Events"
  try
    set the clipboard to ""
  on error err_message
    display dialog err_message
  end try
end tell

### Tell user that AppleScript has done its job ###

#say "Finished!"
set variableWithSoundName to "Bottle"

display notification "Transferred to XojoInfo.txt" with title "Code Handler"
  subtitle "Captured" sound name variableWithSoundName

```

## What would need to be modified

At start of the script, it makes sure that the folder `_ArchiveXojoData` exists. By default this location is on the desktop. The advanced user could change this line to be another path. If the path contains subfolders, each subfolder needs to be checked for its “existence”, one after another. There is some commented out code that shows how to do make a subfolder.

```
### Make sure that the folder _XojoArchiveData exists in the appropriate
    location ###
#       Courtesy of JMichaelTX http://macscripiter.net/viewtopic.php?id=41721 #

# Makes the initial folder on the Desktop
set newFolderPath to quoted form of (expandPath("~/Desktop/
    _ArchiveXojoData"))

set cmdStr to "if [[ ! -d " & newFolderPath & " ]]; then
mkdir -m 755 " & newFolderPath & "; fi"
do shell script cmdStr

# makes a subfolder in the folder created above if you want or need
# set newFolderPath to quoted form of (expandPath("~/Desktop/
    _XojoArchiveData/NewFolder"))

# set cmdStr to "if [[ ! -d " & newFolderPath & " ]]; then
# mkdir -m 755 " & newFolderPath & "; fi"
# do shell script cmdStr

...

```

There is another area in the AppleScript that would be required to be changed as well. This first part above makes sure that the folder exists. The next part actually does the work of creating the text file `XojoInfo.txt`. This code exists in the middle of the script.

```

...
set delta to ((time to GMT) / 36)
set deltaTagged to "<timezone>" & delta & "</timezone>"

set xoyoClip to newline & return & briefBoth & return & deltaTagged &
    return & xoyoClip

set this_file to (((path to desktop folder) as string) &
    "_ArchiveXoyoData:XoyoInfo.txt")

my write_to_file(xoyoClip, this_file, true)

on write_to_file(this_data, target_file, append_data)
    try

...

```

## Appendix

### *Kaleidoscope* - Black Pixel

In the course of looking at multiple versions of code, the need will probably arise to detect just what are the differences are between two versions. This can be hard to do without computer assistance. **ArchiveXoyo** itself offers a relatively primitive form of this utility, but there are many third party utilities that can provide a more advanced comparison.

*Kaleidoscope* by Black Pixel is one such program available on the Macintosh (~\$50.00). It is warmly recommended for this purpose. Differences are displayed in an easy to understand window. It is easy and fast to bring two files into this program for the comparison. I frequently will have the two versions of a method sitting on the Desktop. I highlight them in the Finder and go to **Finder**>>*Services* and select **Open in Kaleidoscope** which will open the

appropriate comparison window in that program. You will probably find other contexts (outside of **ArchiveXojo**) where this program can be useful.

Recently, the popular text editor, *BBEdit*, has greatly improved its display of text differences when comparing files. The usefulness of this program for this purpose has become more competitive with the *Kaleidoscope*.

If you use GIT and *SourceTree* (on Mac), *Kaleidoscope* can be designated an external DIFF tool and work within *SourceTree*.

## Contact

Author: Robert Livingston

Company: **Bearboat**

Email: [rlivingston@me.com](mailto:rlivingston@me.com)

Feel free to write me a note. I would like to improve this archiving system and stamp out any bugs and improve the documentation.

If you have run into some problem — Write

If you have a suggestion — Write

If you have any comment, favorable or disparaging — Write

All communication is appreciated. One trouble with “free” software is that many users tread lightly. They do not feel entitled to complain or mention bugs. Frequently, they will just move on.

I would much prefer that users be more engaged and write. Bugs can be dealt with which improves the experience for everyone. Areas of confusion in the documentation can be clarified.

I cannot always address every suggestion, but I like to hear them.



# Screenshots

## Search Listing Window

Lists the name and modification dates of code versions.

The screenshot shows the ArchiveXojo search interface. At the top, there are filters for 'Any Project', 'Starts With', 'Short Name', 'Code Contains', and 'Date Range'. Below the filters, a table displays the search results. The table has the following columns: Last Modification, Project, Short Name, Code, Code Size, Comment Size, Code Type, and Path Name. The data rows show various code versions for projects like 'Practice' and 'BearboatSP'.

Last Modification	Project	Short Name	Code	Code Size	Comment Size	Code Type	Path Name
06/19/16 18:21:11	160606_1707Practice	PushButton1.Action	Unchanged	172	242	Event	winMain.Controls.PushButton1.Action
06/19/16 18:18:44	160606_1707Practice	PushButton1.Action	Changed	172	239	Event	winMain.Controls.PushButton1.Action
06/18/16 19:17:51	160606_1707Practice	PushButton1.Action	Changed	180	239	Event	winMain.Controls.PushButton1.Action
06/18/16 19:16:31	160606_1707Practice	PushButton1.Action	Changed	181	241	Event	winMain.Controls.PushButton1.Action
06/18/16 19:15:20	160606_1707Practice	PushButton1.Action	Original	182	243	Event	winMain.Controls.PushButton1.Action
06/14/16 20:09:45	140323_1348BearboatSP	displayAll	Changed	354	13	Method	OneIntTwoDou.displayAll
06/14/16 20:08:26	140323_1348BearboatSP	displayAll	Changed	354	13	Method	OneIntTwoDou.displayAll
06/14/16 20:03:40	140323_1348BearboatSP	Constructor	Original	984	187	Method	winShowArray3S.Constructor
06/14/16 20:03:40	140323_1348BearboatSP	pbOK.Action	Original	171	70	Events	winShowArray3S.Controls.pbOK.Action
06/14/16 20:03:40	140323_1348BearboatSP	Constructor	Original	1136	186	Method	winShowArray4I.Constructor
06/14/16 20:03:40	140323_1348BearboatSP	pbOK.Action	Original	171	70	Events	winShowArray4I.Controls.pbOK.Action

## Code Window

Displays a code listing with and without comments

ArchiveXojo

First Last Done

Previous Next Compare Versions (3) Search Go

Modified 06/14/16 20:09:45 Database 140323\_1348BearboatSP

**OneIntTwoDou.displayAll** Code Characters 354 Comment Characters 13 Record Created 06/14/16 8:10 PM Percent Code 96%

Send to Clipboard Text File On Desktop

```
Sub displayAll()  
// displayAll  
  
Dim nElement, upperBound As Integer  
Dim xMessage As String  
  
upperBound = Self.ubound()  
For nElement = 0 To upperBound  
    xMessage = xMessage + (Format(Self.get1(nElement), "-#") + " " +  
Format(Self.get2(nElement), "-#") + " " + Format(Self.get3(nElement),  
"-#")) + chr(ASCII_CARRIAGE_RETURN)  
Next  
  
Dim Frog As Integer  
Frog = 5  
  
MsgBox xMessage  
End Sub
```

Text File On Desktop Send to Clipboard Just Code

```
Sub displayAll()  
Dim nElement, upperBound As Integer  
Dim xMessage As String  
upperBound = Self.ubound()  
For nElement = 0 To upperBound  
    xMessage = xMessage + (Format(Self.get1(nElement), "-#") + " " +  
Format(Self.get2(nElement), "-#") + " " + Format(Self.get3(nElement),  
"-#")) + chr(ASCII_CARRIAGE_RETURN)  
Next  
Dim Frog As Integer  
Frog = 5  
MsgBox xMessage  
End Sub
```

## Code Size Report

Reports on the size of the code over time.

⓪ Analyse Database Size

? Calculates the number of characters, words and lines in the database. This is done for the code including the comments and also for the code without the comments. Done

### FoxIslCem

FoxIslCem Go

Code with Comments
Code Only

Characters	<b>617,827</b>	<b>391,443</b>
Words	<b>62,377</b>	<b>38,449</b>
Lines	<b>15,873</b>	<b>10,854</b>

Change Over Time
Export as Text Document

Date	Char	Word	Line	Char	Word	Line
12/9/15	617,827	62,377	15,873	391,443	38,449	10,854
8/30/15	604,963	60,989	15,593	381,972	37,477	10,630
8/29/15	547,942	55,157	14,279	346,560	33,939	9,709
8/27/15	542,319	54,633	14,143	342,760	33,608	9,603
8/26/15	397,705	39,960	10,141	235,205	23,280	6,369
12/30/14	274,936	28,601	6,779	155,340	16,248	3,964
7/16/14	273,941	28,493	6,741	154,771	16,182	3,937
7/14/14	272,602	28,361	6,705	154,088	16,110	3,916
7/8/14	259,915	26,894	6,419	149,835	15,658	3,811
6/9/14	240,213	24,910	5,866	135,393	14,204	3,375
6/8/14	208,570	21,453	5,097	115,050	12,044	2,831

## Code Versions

Differences highlighted in orange.

Total Versions of winMain.Controls.PushButton1.Action: 5

All versions from 160606\_1707Practice

Search  Go

85% Version # 2 **winMain.Controls.PushButton1.Action** Version # 5  Just Code

160606\_1707Practice 06/18/16 19:16:31 160606\_1707Practice 06/19/16 18:21:11

Prior Next Adjacent Oldest Export Clipboard Clipboard Export Newest Adjacent Prior Next

```

Sub Action()
  Dim A, B As Integer
  Dim compost As Integer
  B = 3
  A = 7
  A = B + B + B
  A = B + B + B * B

  Dim Ab As Integer
  Ab = Ab + 39 // a comment with lots of leading spaces and some
trailing

  If A > 7 Then
    Ab = Ab-3 // comment with intial single quote
  Else
    Ab = Ab + 4 // this is addition compost
  End If
  // some comment (no leading space after //)
  // pure comment with intial single quote
  // comment with bullet * in the com
  o
  t
  B = com
End Sub

```

```

Sub Action()
  Dim A, B As Integer
  Dim compost As Integer
  B = 3
  A = 7
  A = B + B + B
  A = B + B + B * B

  Dim Ab As Integer
  Ab = Ab + 39 // a comment with lots of leading spaces and some trailing

  If A > 7 Then
    Ab = Ab-3 // comment with intial single quote
  Else
    Ab = Ab + 5 // this is addition compost
  End If
  // some comment (no leading space after //)
  // pure comment with intial single quote
  // comment with bullet @ in the cdoe
End Sub

```