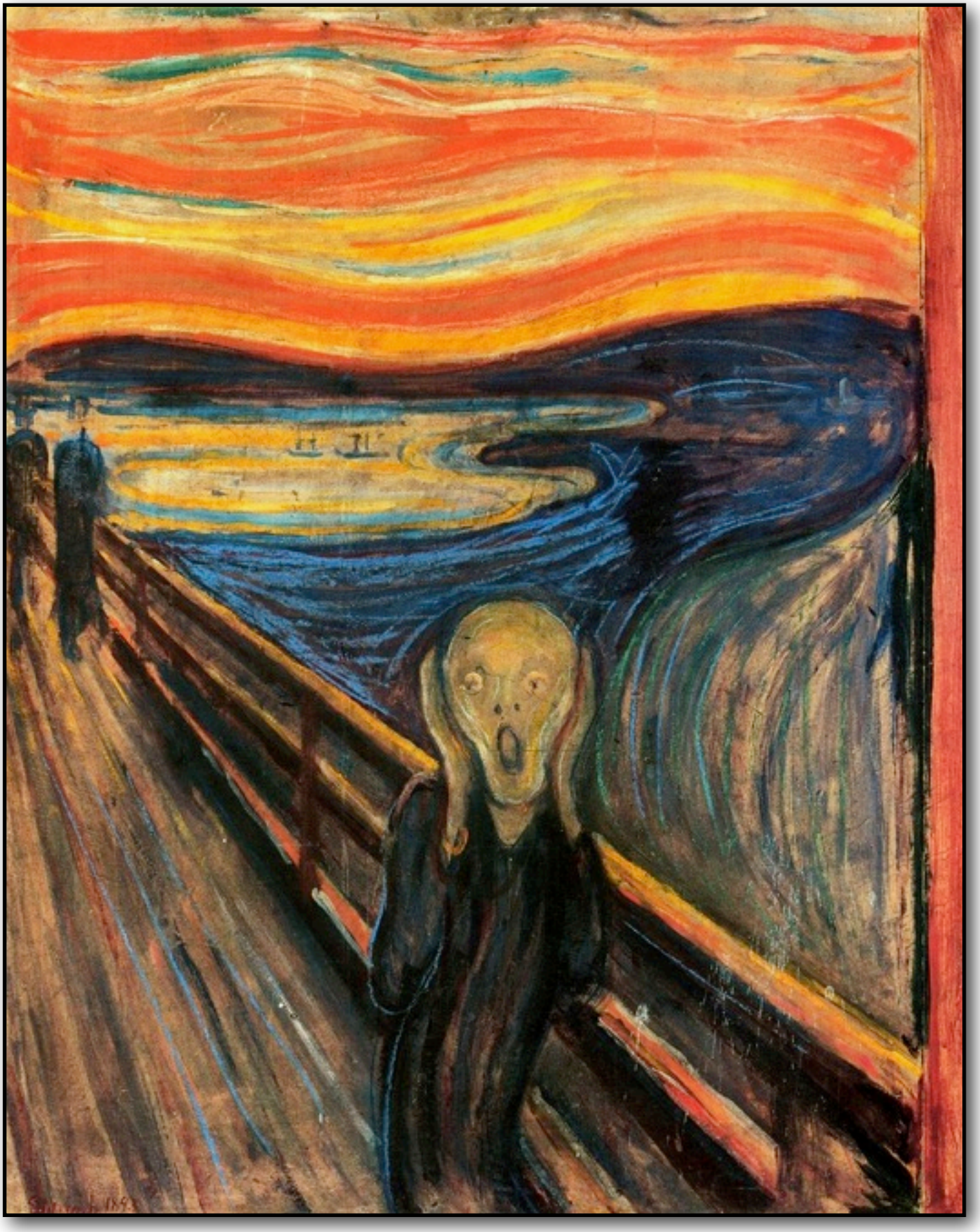
Apple® IIGS

BrkDown Reference Manual

Wester Ross - Scotland
A place where you wouldn't mind breaking down
Photo c1973 © Ewen Wannop



“The Scream” by Edvard Munch - 1893
Oil, tempera, pastel and crayon on cardboard. National Gallery, Oslo, Norway

Dedicated to the memories of Joe Kohn 1947-2010
and Ryan Suenega 1967-2011

BrkDown is Freeware and Copyright © 2015 Ewen Wannop

BrkDown and its supporting documentation may not be printed,
copied, or distributed for profit.
Distributing and/or archiving is restricted while in an electronic form.
Any "free" distribution must be given permission by Ewen Wannop
in advance -- please contact via email by sending mail to:

spectrumdaddy@speccie.co.uk

There is no guarantee that the right to redistribute this material
will be granted. The contents of this document may not be
reprinted in part or in whole.

Credits

My thanks go to all who originally helped me develop ChewBagger,
and the suggestion from Hugh Hood for a Disassembly function
which eventually led to the creation of BrkDown.
My thanks also go to Hugh Hood, Dagen Brock, and the rest of the
beta testers, for their invaluable help and suggestions during its creation.

Index

Preface

Welcome to BrkDown

Background

When it was suggested that I include a disassembly routine in ChewBagger, it was clear that this could only be of limited use within such a program. I felt there was the need for something more powerful to be created.

In the past, I had looked at various of the other available disassemblers, but had only found the ORCA Desktop.DISASM to be useful, as it gave me a quick way of seeing code in a format that I could easily work with. However I found that for me, it has its limitations, and I felt some aspects of it as a desktop program could be improved on.

There are many other disassemblers to be had, some of them being quite powerful,. Most are text based programs using the 80 column display, and require the construction of templates to do a thorough job. I wanted to break away from that, and write an easy to use desktop disassembler, that could be used by those with limited programming experience.

So ChewBagger has a new sibling, and BrkDown has been born...

BrkDown is Freeware and Copyright © 2015 Ewen Wannop

The BrkDown Application

Requirements

BrkDown is a stand alone application, and requires no fonts or tools other than those already supplied with System 6.0.1. For its displays, BrkDown uses custom fonts built into the application itself. There are also six files that are required when generating source code, and these can be found in the Data folder within the archive. They should remain within that folder, which should itself be inside the folder that you run BrkDown from.

If you plan to disassemble large 16 bit segmented applications, such as Spectrum, you will need as much free memory as possible. If you are using an emulator such as Sweet16, then give it at least 8Mb to work with, or preferably 14Mb if you can. Files are first loaded into memory for disassembly, in order that cross referencing information between segments can be preserved, and this takes much more memory than BrkDown would require for smaller programs, such as P8 applications, when the usual 4Mb of memory of the IIgs will suffice.

You will also need some spare hard disk space to hold all the working files.

What is BrkDown

BrkDown is a point and click desktop disassembler, for both eight and sixteen bit files. It will also disassemble the resource forks of forked files, and after suitable editing, can generate either ORCA/M or Merlin source files for later assembly.

Legal

BrkDown is Freeware, and may be distributed freely, provided the archive stays intact, no alterations are made to it, and full attribution is always made to the author Ewen Wannop. BrkDown may not be sold in any form, but may be included in other distributed archives, provided you first seek my permission.

BrkDown Strategy

It is easy to simply disassemble a program into source code, but to turn that into code that can later be reassembled, takes a certain amount of programming skill. BrkDown handles the first step for you automatically, and then helps you with the second step, by using direct desktop interaction, with various tools and features, rather than using the template based control of other disassemblers.

The problem with creating disassemblies is that any disassembler you use, cannot know when a section of data it sees is actual code, or is a variable, or just a block of data. You need to interact with the disassembly to indicate what these sections might be, before you can generate code that will be meaningful when reassembled.

It is suggested that your workflow with BrkDown should be this:

Create a Project

Disassemble the target file

Define strings, code, and other data sections

Define any Global Equates

Manually adjust any further code that is needed

Define any code Directives

Build cross referencing Labels

Build Source files

How BrkDown Works

BrkDown works with Projects, which you create from your target application or binary file. Within the working BrkDown folder, you will find the Projects folder, a Src.Files folder, a Data folder, and an Archives folder.

The Projects folder holds a number of named folders, with a single folder for each Project. Within each of those named folders, you will find a number of files, depending on the kind of application or file you are working with:

Code.xx	Multiple source data files
Global.Equates	Defined global equates
Link.Template	Cross segment label references
Resource.equ	Resource fork equates
Resource.src	Resource fork source
Segment.xx	Working source files
Status	Project status data

After Building Source, within the Src.Files folder, you will find a number of files:

Resource.equ	If the original has a resource fork
Resource.src	If the original has a resource fork
Source.xx.x.asm	Files from ORCA/M disassemblies
Source.xx.x.s	Files from Merlin disassemblies

The Archives folder holds any Projects that you may have archived:

Spectrum.arc	Named GSHK archive

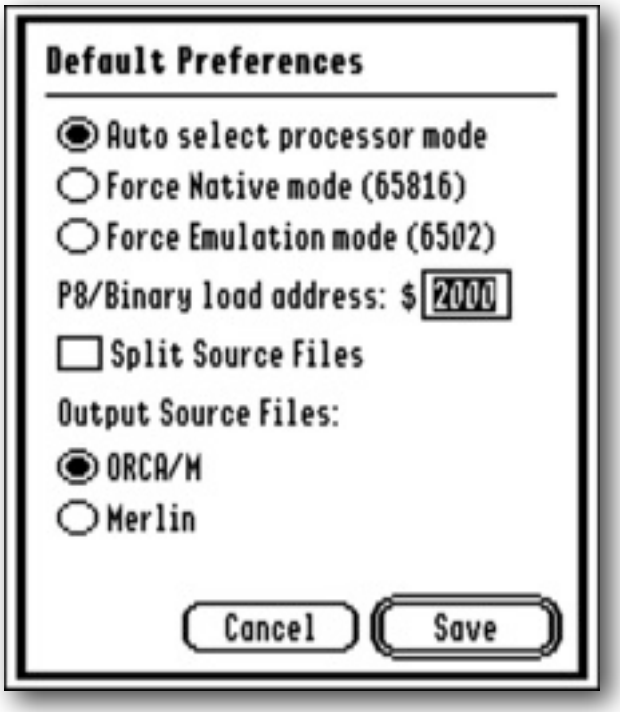
After creating a Project or Projects, you Select and open the project, and you will see the disassembled raw source in the main window. If it is the first time you have opened that Project, it will be automatically disassembled. If you have already disassembled the Project, you will be given the option to open the existing files, or to disassemble again. See “Projects” for more details.

So the final Source files can be built correctly, you will need to edit or work with the disassembled Project files, in order to make sense of them. BrkDown gives you a number of tools to help with the process. See “Editing Files” for more details.

Finally, you will need to generate the output Source files. This will create either ORCA/M or Merlin compatible source files, which can then be reassembled. See “Building Source” for more details.

General Operation

Preferences



Before you start to work with BrkDown, you should set Preferences to suit your working methods.

Auto Select Processor Mode - The initial processor mode used for a disassembly will be determined by the filetype. For eight bit or binary files, this will be to force Emulation 6502 mode, and for Forked or 16 bit files, this will be Native 65816 mode.

Force Native Mode - The initial processor mode will be set to 16 bit wide 65816 native mode.

Force Emulation Mode - The initial processor mode will be set to 8 bit wide 6502 emulation mode.

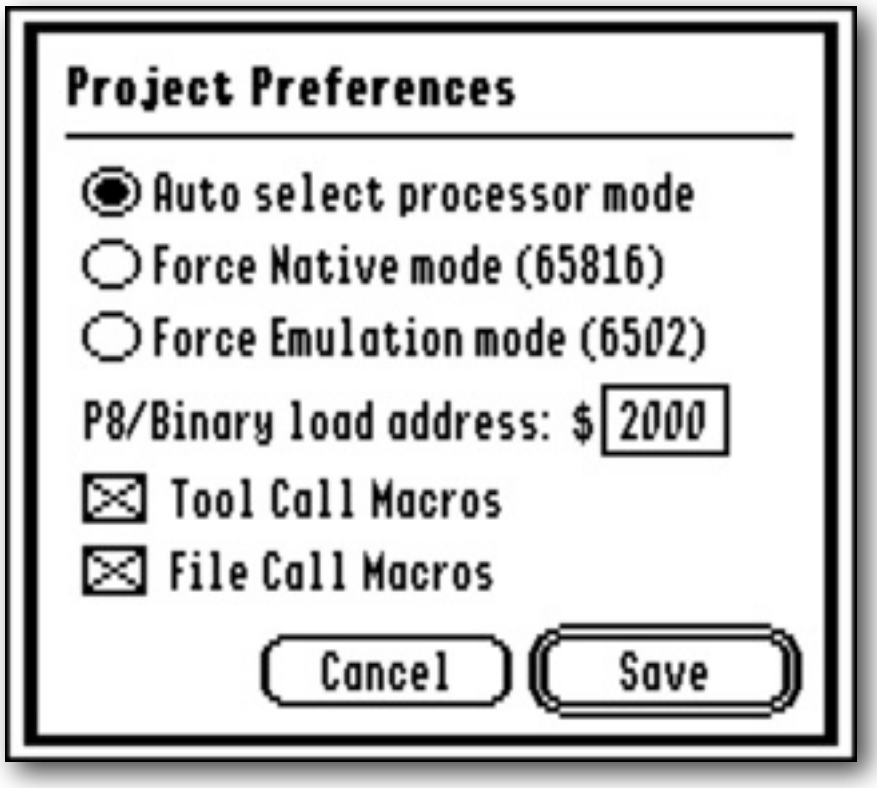
P8/Binary load address - Relocatable 16 bit files will always be loaded automatically to free memory, and disassembled from those memory positions to preserve the validity of any cross-referenced variables or entry points.

As 8 bit files are usually loaded into memory at $2000, this will be the default load address used for the disassembly. However, when the Project is created, if the Auxtype has a non-zero value, that value will be used to prime the Load Address instead. If you wish to change the default value, then enter a new value here. It can be changed later if you wish on a Project by Project basis.

Split Source Files - When building Source files, you can optionally choose to split the output into files of approximately 32K long. If you are editing on the IIgs, this might be advantageous. If you are editing on a Macintosh or Windows computer, you might prefer to keep the output as a single large file.

Output Source Files - Although BrkDown editors work with ORCA/M style files, it can output either ORCA/M or Merlin 16 compatible source files. You should select your preferred method here before you start, so any Resource forks will be correctly generated on disassembly.

Note: Some of these settings can be changed later, and applied to individual Projects:



Projects

Selecting Projects

This is the starting point for any new Projects, and working with existing Projects.

New - For each Project, you will need to create a new entry in the list. Using the standard file chooser, select the target application or binary file, and then either accept the default name, or change it to a new one of your choice. If a Project already exists with the same name, you will be prompted to change the name to a different one.

Delete - Deletes the selected Project folder and all its files. It also deletes any Source files from its related folder as well. It does not delete any Archived files for the Project, and you will be asked for confirmation before it proceeds.

Info - Displays information for the target file linked to the Project. Optionally you can change the Project name here, or set specific Preferences for that Project.

In addition to the settings described in the main Preferences section above, there are two settings in this dialog that apply only to individual Projects:

Tool Call Macros The default is for toolbox calls to be expanded into their respective macros. Uncheck this box to stop those from being generated.

File Call Macros The default is for GS/OS and P8 file calls to be expanded into their respective macros. Uncheck this box to stop those from being generated.

Archives - If you have Balloon installed on your system, this option will allow you to Archive or Restore the Project folder and its related Source files folders. The default is to save to the Archives folder, but optionally you can save to any folder of your choice.

Select - Selects a Project to work with. The first time a Project is selected, it will be automatically disassembled, and any resource forks will also be disassembled. The data fork is disassembled to an ORCA compatible format that is used for displaying and any subsequent editing within BrkDown. The resource fork is disassembled to either an ORCA/M or Merlin compatible format, depending on the option you set in Preferences.

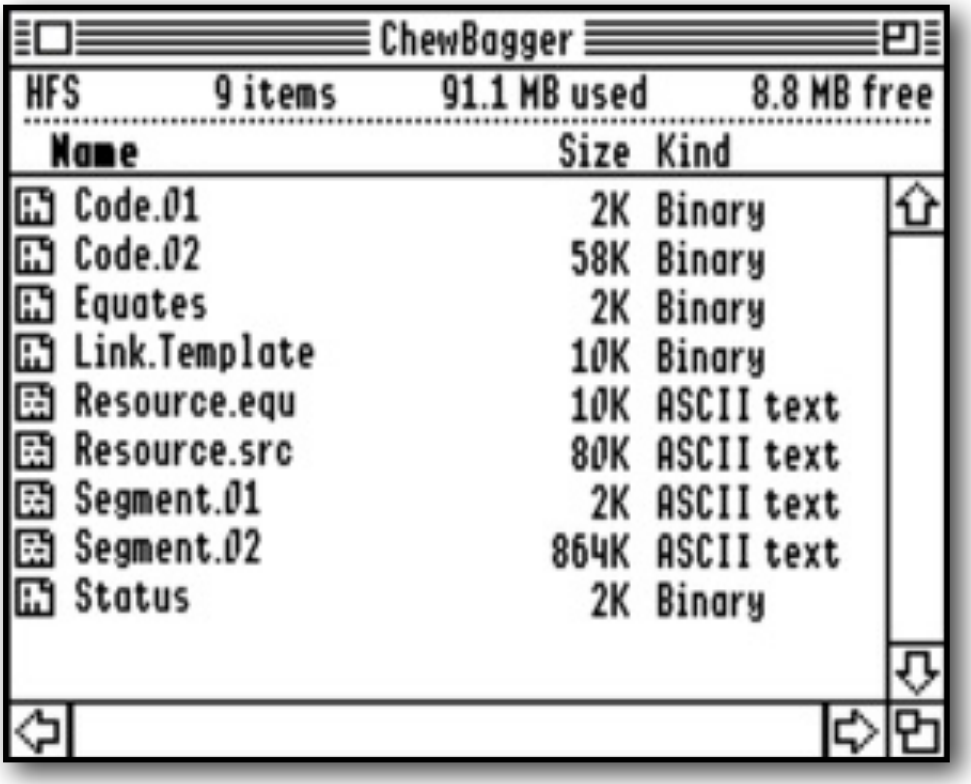
If the Project has already been disassembled, you will have the option to either disassemble once more, or use the current files for further editing. Please be patient, as disassembling can take some time, depending on the size of the target file you have selected, and the speed of your computer.

Note: If you disassemble once more, then the entire Project folder will first be emptied. This will delete any existing labels, and global equates you may have already defined.

Note: If you are using ORCA/M, you will need to put the supplied 'types.rez' file into the 'RInclude' folder within the 'Libraries' folder on the drive where your ORCA/M files are kept. This should replace any existing file with that name. This updated file holds some enhanced resource information which BrkDown supports.

Note: The original target file for a Project must remain in its original folder if you wish to disassemble for a second time.

The Projects Folder - The Project folder holds a number of files as outlined on Page. 6. It is important that you do not delete or edit any of these files yourself, as subsequent operations in BrkDown may fail. If you need to make a copy of these files, use the 'Archives' function to keep a GHSK compatible copy of the two related folders.



You will see one or more of these files within a Project folder:

Code.xx - Multiple files, numbered according to their segment number within the original file. For a P8 or Binary target file, there will be only one Code file.

Code files hold the data from each segment after it was loaded into memory for the disassembly process. For relocatable GS/OS files, the Code files will hold the segment data after it was loaded in memory, rather than the raw data from the original file.

Global.Equates - If you have defined any global Equates, they will be stored in this file for later use by the 'Build Labels' or 'Apply Labels' functions.

Link.Template - When you 'Build Labels', the value for each label that is generated is kept in this file for later use by the 'Build Labels' or 'Apply Labels' functions. This allows labels to be correctly referenced across multiple segments in different memory banks.

Resource.equ - If the original target file had a resource fork, an equates file, `Resource.equ,` will be created, holding the equates required for the resource fork to be re-assembled correctly.

Resource.src - If the original target file had a resource fork, the `Resource.src` file will be built, holding the disassembled resource fork. It must be kept along with, and used in conjunction with, the `Resource.equ` file for any subsequent re-assembly.

Note: The code format of the two resource files will depend on whether ORCA/M or Merlin had been selected as the operating environment when the disassembly was made.

Segment.xx - This holds the disassembled data for each segment within a file. For a P8 or Binary target file, there will only be one Segment file created, and it will be numbered as Segment.01. For GS/OS files, there may be one or more sequentially numbered segment files created.

Status - This file holds various items of information used by BrkDown during the disassembly and editing processes.

Note: The `Segment.xx` and `Resource.src` files, are the data files you will see displayed on the Project screen while you work.

To display the other segments from a segmented file, either choose the segment directly from the Segments menu, or use the 'Open-Apple-N' key to move between them.

You will not be able to display an `~ExpressLoad` segment. This is deliberate, as it serves no further purpose for the final building of the source files, and so cannot be edited.

Note: If you change the format from ORCA/M to Merlin, or vice versa in the Preferences dialog, when you next select the Project, if you want the Resource forks to be correctly formatted, you will need to disassemble the target file once again.

Inserted Calls: Six data files, PDOSA.Calls, GSOSA.Calls, and ToolA.Calls, for ORCA/M, and PDOSM.Calls, GSOSM.Calls, and ToolM.Calls, for Merlin, are supplied with the BrkDown archive. They are required, and should remain within the '`Data`' folder in the same folder as BrkDown.

These files can be edited and expanded if you wish. Just follow the same format of a `value/space/label,` and you can add your own custom calls, or any that you feel are missing from the lists. Enter values as a hexadecimal word, without a preceding $ sign., and finish the file with a CR (Return). You can add comment lines if you wish, by preceding the comment with an asterisk at the start of the line.

The values contained within these files are used in this way:

PDOSA.Calls & PDOSM.Calls When disassembling code in 6502 Emulation mode, any ProDOS calls in the file matching the two byte operand values from the file, will be replaced by the ProDOS call from the file. In the same way, any Monitor ROM values in commands with two byte operands, such as JSR, LDA, LDX, CMP, INC etc., will be checked against the values in the file, and the equate from the file will be inserted in place of the operand.

If you add any new Equates to the files yourself, these will also be checked, and replaced if they are found. If these Equates lie within the range of the program code, you may also need to Insert a matching Label at the appropriate point in the code.

GSOSA.Calls & GSOSM.Calls When disassembling code in 65816 Native mode, any GS/OS calls in the file matching the two byte operand values from the file, will be replaced by the GS/OS call from the file.

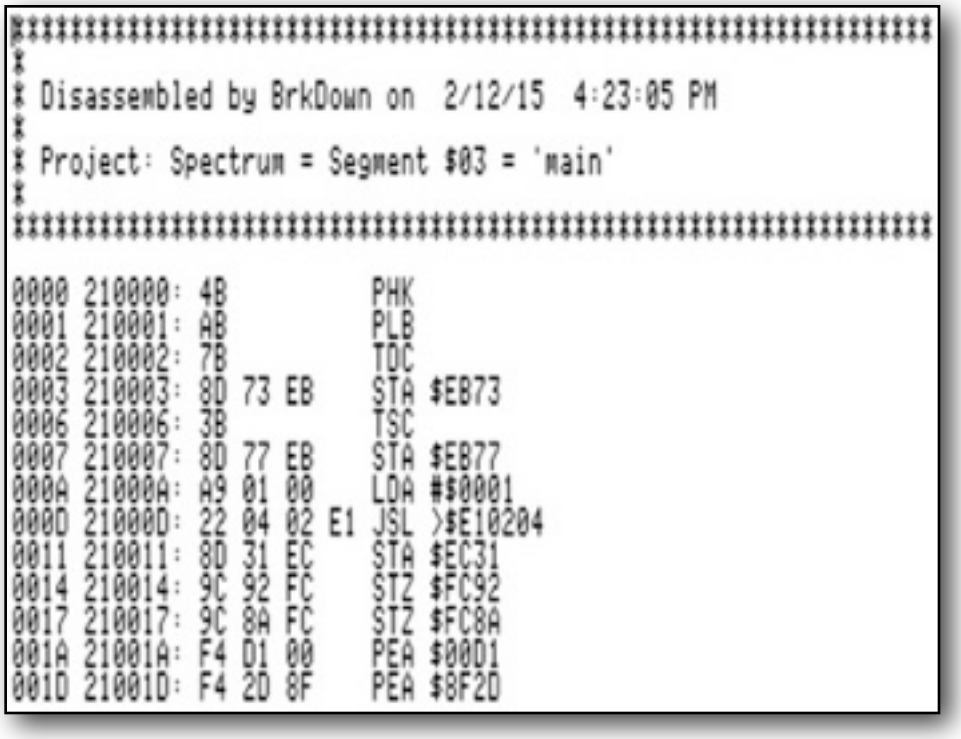
TOOLA.Calls & TOOLM.Calls When disassembling code in 65816 Native mode, any Tool calls in the file matching the two byte operand values from the file, will be replaced by the Tool call from the file.

Note: As ProDOS, GS/OS, ROM, and Tool, calls can be named differently in ORCA/M and Merlin source code, two sets of files are provided. As supplied, these files contain the same calls, but in the appropriate format for each operating mode. If you wish to edit these files, make sure you select the correct `'A'` files for ORCA/M and the `'M'` files for Merlin.

Editing Projects

You cannot directly edit the Project text from the Project window. This is to ensure that any changes you make will be correctly structured for BrkDown to build source files later on. A number of functions are provided however, which will let you modify the code as required from its raw state. These are explained in detail below.

Note: The Editor works with a file structure compatible with ORCA/M, even when you have selected to work in Merlin. The working Segment.xx files, will be converted into Merlin format when you Build the final Source files.



Except for files representing a resource fork, which are displayed for information only, each file will display in a 'standard' format. This will be familiar to those who have worked with the ORCA/M 65816 assembly language.

There are up to six columns of data that are displayed:

Column	Function
1	Offset into the code from the start of the file
2	Relative address and bank number for the line of code
3	Up to four bytes of code for this statement
4	The Operation code for this statement
5	The Operand field for this statement
6	Optional comment field

For most of the functions, you will select the range of code you wish to change or define. Simply select the byte range you wish to work on, and apply the required function. You will see an Alert if a selection is invalid for any reason for that function.

Most lines of code will display as having from between 1-4 bytes of code, but for some lines, such as lines defined as 'STR' or 'ASCII', there may be a hidden number of extra bytes of code. In such cases, you will not be able to select directly from within the hidden bytes without first turning the line into visible data in the form of 'Byte', 'Words', 'Long Words', or 'Hex Bytes'. This will then display any hidden bytes so they can be selected.

If you select the line from the Operation Code or beyond, all the extra hidden bytes in a line, to the start of the next line, will be included in the selection.

It is a good idea to regularly 'Save' changes as you go along, though if you do, you will not be able to use the 'Revert to Saved' feature later on. Refer to "Revert to Saved" for more details.

Many of the Editing functions reference the Code.xx source files to retrieve original data.

The Editing Functions

As has been described earlier, the raw output from the disassembler will in most cases need to be worked on to turn it into usable source code. The most likely data that will need to be changed will be any variable space, which will need to be set to a 'DS' data block from what will appear as junk code, and other areas of code that may need to be changed from a native 65816 to emulated 6502 code, or vice versa.

Note: Any orphaned bytes from a selection, will be displayed as leading or trailing data statements, in lines either side of the resulting lines from the selected function. An odd number of trailing bytes for a 'Define as Data' selection, will be displayed as shorter 'Words' or 'Bytes' data statements.

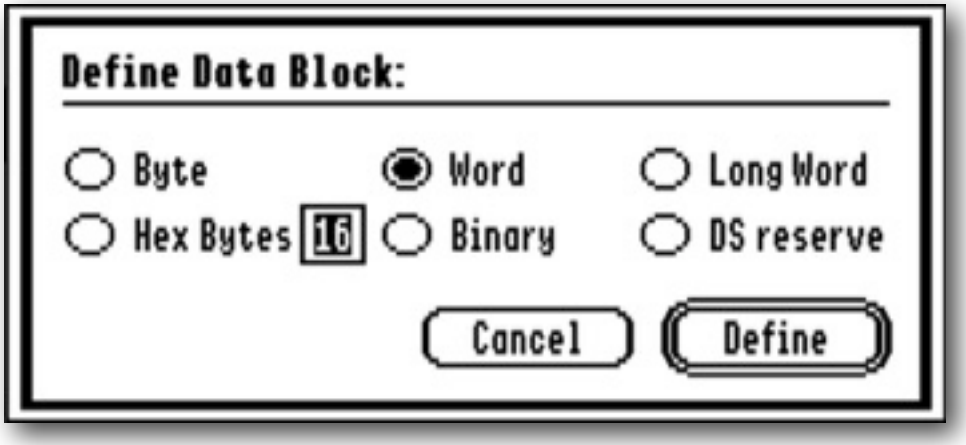
Define As Data - This allows you to change interpreted code to six forms of data statements:

Byte: Changes the selected bytes into data statements in the form:

```
DC I1'$00'
```

Word: Changes the selected bytes into data statements in the form:

```
DC I2'$0000'
```



Long Word: Changes the selected bytes into data statements in the form:

```
DC I4'$00000000'
```

Hex Bytes: Changes the selected bytes into data statements in the form:

```
DC H'000102030405'
```

The maximum number of bytes for each line of data can be set for a range of between 1 to 16. If the selected data exceeds that range, the data will be split over multiple lines.

Binary: Changes the selected bytes into data statements in the form:

```
DC B'11100100'
```

Up to four bytes can be displayed for each line.

DS Reserve: Changes the selected bytes into data statements in the form:

```
DS XX
```

Where `'XX'` is the decimal count of the bytes you selected.

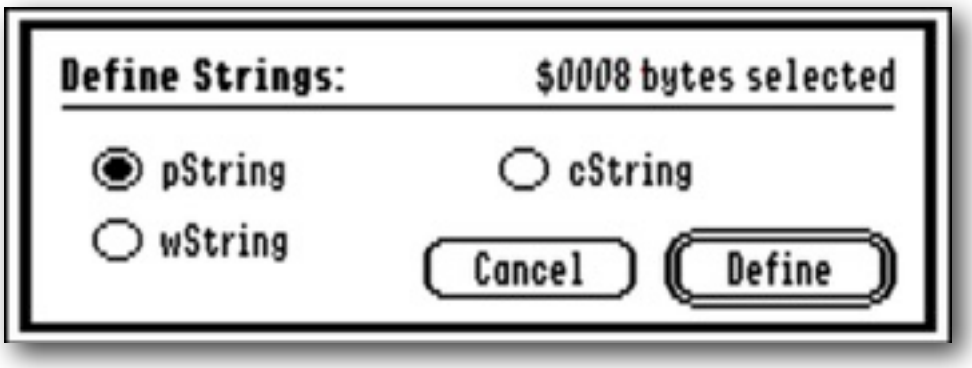
Define As ASCII - Changes the selected bytes into data statements in the form:

```
DC C'A string of text'
```

Printable ASCII will be displayed within a `'DC C'` statement, and any unprintable ASCII will show as interspersed `'DC I'` data statements. If printable text exceeds 32 characters, the text will be split over multiple lines.

Define Strings - Defines the selected data as either a pString, a cString, or a wString. If the selection number of bytes is invalid for the chosen format, you will see an Alert explaining what was required:

All three string formats display the selected data in the same basic format as used with 'Define as ASCII'. At the top right of the selection dialog, you will see how many bytes you have selected. The amount you will require will vary depending on the String format you choose. If you are working with Merlin Source Output selected, the resulting strings will have either single or double quotes, depending on the ASCII content of the selection:



pString: Will display the selected data in the form:

```
DC I1'$06'   ; length pString
DC C'String'
```

Note: The preceding single length byte. For a valid pString, you must select that number of bytes, as well as the preceding length byte itself.

cString: Will display the selected data in the form:

```
DC C'String'
DC I1'$00'   ; end of cString
```

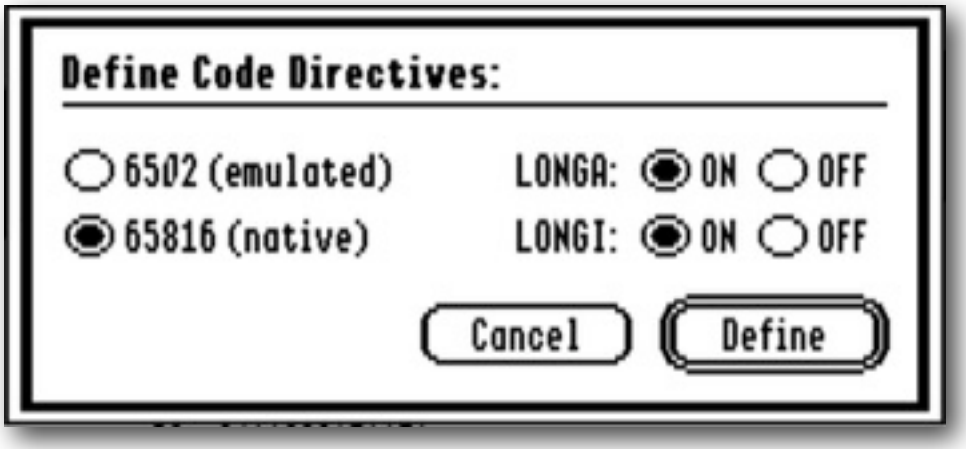
Note: The trailing single zero byte marker. For a valid cString, you must select a number of bytes as well as the trailing zero byte.

wString: Changes the selected bytes into data

```
DC I2'$06'   ; length wString
DC C'String'
```

Note: The preceding length word. For a valid wString, you must select that number of bytes, as well as the preceding length word itself.

Define As Code - This function allows you to disassemble a selected section of code again, allowing you to override the default format that was first used. You can use this to clean up sections of code back to their original raw state, or to change processor directives for the disassembled code selection:



Note: In some cases you may also need to insert a preceding or trailing directive as well as defining the code. Please refer to 'Insert Directive' for more details.

6502 (emulated): Disassembles the selection into 8-bit code, as used by P8 programs. If necessary, insert a preceding 'MX Off/Off' directive, and if the following code is 16-bit, a following 'MX On/On' directive.

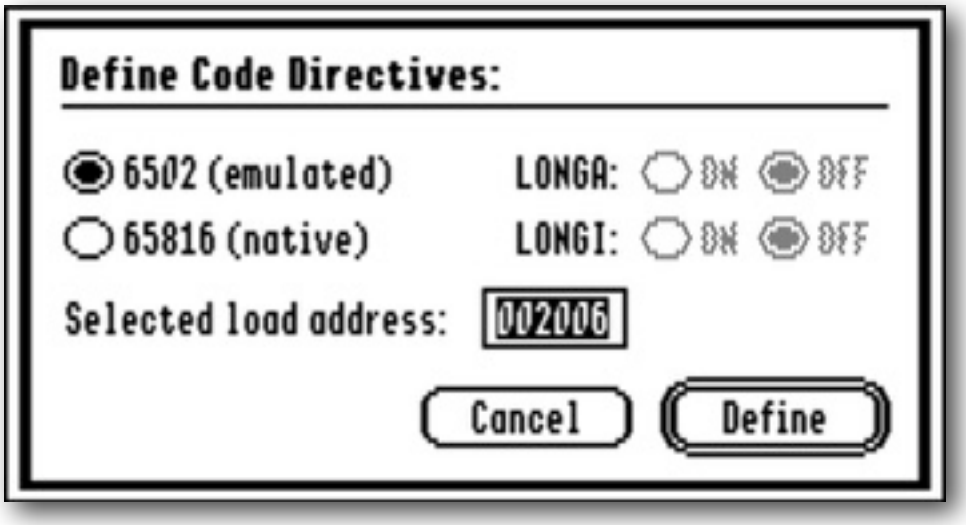
65816 (native): Disassembles the selection into 16-bit code, as used by GS/OS programs. If necessary, insert a preceding 'MX On/On' directive, and if the following code is 8-bit, a following 'MX Off/Off' directive.

These four options can only be selected when 65816 (native) mode is also selected, and sets the accumulator and X/Y register widths for the disassembly process:

LONGA ON: Sets the accumulator to 16-bits wide.
LONGA OFF: Sets the accumulator to 8-bits wide.
LONGI ON: Sets the X/Y registers to 16-bits wide.
LONGI OFF: Sets the X/Y registers to 8-bits wide.

If necessary, insert in the code a preceding 'MX On/On', 'MX On/Off', 'MX Off/On', or 'MX Off/Off' directive, and then if necessary, the opposite directive following at the end.

When working with a ProDOS 8 or Binary file, you will see a modified dialog:



In addition to the functions listed above, you can optionally change the load address of the selected code. A section of code that has had its load address changed, will be marked by comments at the start and end of the section. You may need to later delete or edit these in the word processor, if you need to add an origin 'ORG' command in the code:

```
0007 * Load address changed:
0007 008000:              org $8000
0007 008000: 53           DC  I1'$53'
0008 * End of load address change
```

Note: Changing the load address may interfere with Label generation for that section.

Add Comment - Adds a comment to the end of a line, or lets you edit or remove any existing comment:


```
0007 210007: 82 0D 00     BRL L210012 ;This is a comment
000A 21000A: 71 77        DC  I2'L217771'
000C 21000C:              STR 'ChewBagger'
```

Optionally, you can specify a Line Number to add the comment to. Some lines cannot have comments added, and you will then see an error message if you specify one.

The 'Remove' button will only display if there was an existing comment in the line. When clicked, it will remove that existing comment from the line.

Insert Comment - Adds a new comment line before the current line, or lets you edit or remove an existing comment line.

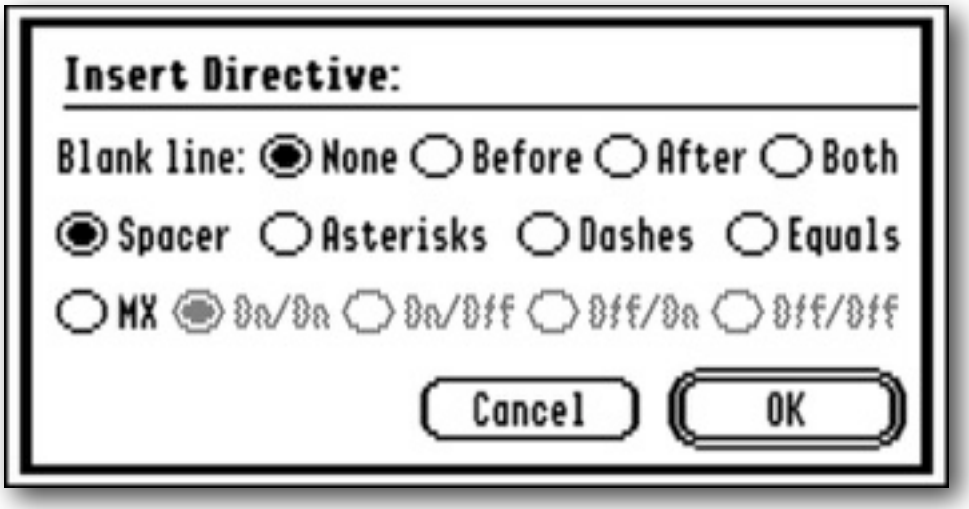

```
0002 *Insert as a new comment line
0002 210002: 82 0D 00     BRL $0012
0005 210005: 71 77        DC  I2'$7771'
0007 210007:              STR 'ChewBagger'
```

Optionally, you can specify a Line Number where to add the comment line.

The 'Remove' button will only display if the cursor was on an existing comment line.

Note: You can also use the 'Delete' function to remove a comment line.

Insert Directive - This function lets you insert blank, or divider lines, or insert processor directives, which tell the assembler how to handle the following lines of code:



Note: All inserted lines will display a Code Offset and Relative Address. These will be removed when the Source files are built.

Blank Line: This inserts a blank line in relation to the other functions you have selected. If you select 'Spacer', then you will insert from between one to three blank lines, depending on the settings for 'Blank Line'. For the other functions, it will insert a blank line either side of the inserted line.

Spacer: Inserts a blank separator line before the current line.

Asterisks, Dashes, Equals: Inserts a separator line before the current line, consisting of 2 asterisks, followed by 62 asterisks, dashes, or equal signs.

MX: Inserts a processor directive. There are four options, representing the widths of the Accumulator and X/Y registers. For instance, to set a wide 16-bit Accumulator, with short 8-bit X/Y registers, you would select 'MX' and 'On/Off'.

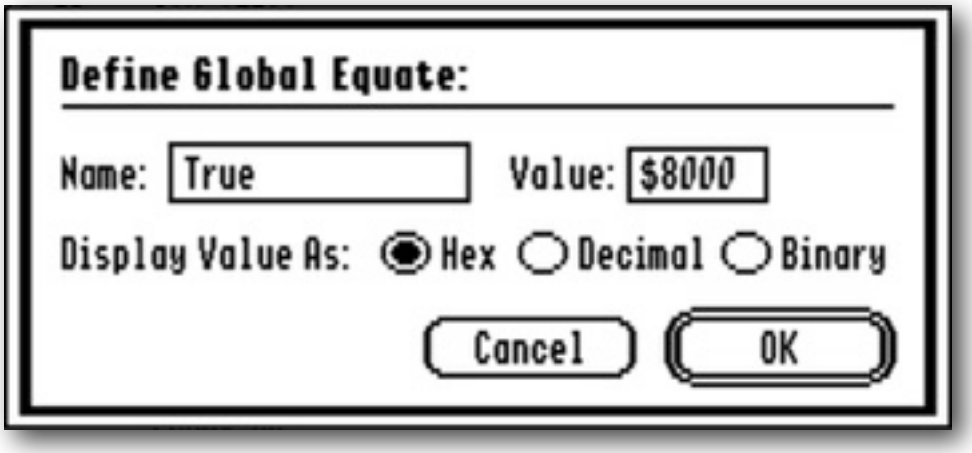
These are directives that instruct the assembler at a later point when the Source files have been built. If you have changed the processor widths for a section of code, make sure you insert the correct directives either side of that section of code.

During editing, they will show as ORCA/M format directives, such as the example given. If you are working in Merlin format, these will be changed to their correct Merlin form when the Source files are built:

```
0012 210012:              LONGA ON
0012 210012:              LONGI ON
0012 210012: 3B           TSC
0013 210013: 8D D8 9A     STA L219AD8
```

Define Equate - Inserts a Global Equate at the head of the first code segment. When you later use 'Build Labels' or 'Apply Labels', this Equate will then be applied to any matching immediate '#$' fixed values in the code within the segments.

These are inserted as Global Equates, so they will be valid across multiple segments.



Name: Give a name for the Global Equate. No checks are made as to whether this name already exists as an Equate, or a Label of the same name already exists. There should only be underscores, numbers, or alpha characters in the name.

Value: The value for the Equate. It can be given as either a Hex value, preceded by a '$' sign, or as a decimal value.

Display Value As: The Equate will be displayed at the head of the source code as either a Hex, Decimal, or Binary value. This does not affect the way the value is later used for substitutions within the code:

```
** Global Equates:

     True gequ $8000
     False gequ 0
     Marker gequ %0000010011010010

0000 210000: 4B           PHK
```

Label Functions

The output files from the disassembler hold limited information with regard to referenced address and variables. If Source files are built from the files at this point, they will have insufficient information to be able to be usefully re-assembled. You must first convert the referenced addresses and variables into Labels. These cross-referenced labels will be inserted into the code at the correct points across multiple segments.

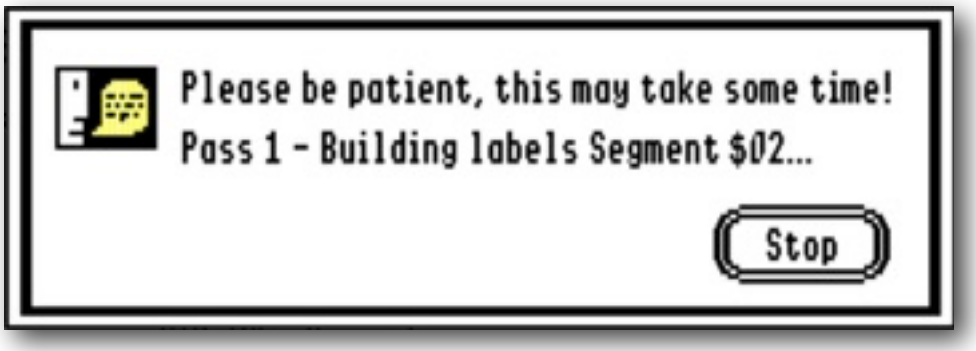
Building Labels is a three step process. The multiple code files are first scanned, and the references collected, changing them into a Label value, then the list of references is sorted, and finally the Labels are written back to the code as a new line entry.

If you have 'Defined As Code' any data, and so lost the Labels on screen, you will need to 'Build Labels' once again to restore them, though in some cases, just 'Apply Labels' will suffice. See 'Apply Labels' for a full explanation.

The referenced variables are held in the Link.Template file within the Project folder. This file will be constantly added to as you work with the code, and is used for the 'Build Labels' and 'Apply Labels' functions.

If you disassemble the Project once more, all collected Labels will be lost, and you will need to start over and 'Build Labels' once again.

Build Labels - The three stage process, as outlined above, will scan through all the code files, building a list of all the referenced addresses. These are then used for the Labels.



Labels will be generated for Branch, Jump, and most Addressing modes, and display in this format while editing:

```
0002 210002: 82 0D 00    BRL L210012
0012 _L210012 entry
0013 210013: 8D D8 9A    STA L219AD8
```

In the example above, the referenced address from the disassembler is first converted into a Label in the form `L210012`. This appears in every operand field where this Label is referenced, and then as the Label on its own line, with an Underscore marker.

In the final ORCA/M generated source file, the example above will look like this:

```
          BRL L210012
L210012 entry
          STA L219AD8
```

The referenced Labels show as a six byte value, generated from the disassembler output. Even with an 8-bit file, Labels will still show as a six byte value. The Labels are only references, but by using their original six byte addresses, cross-references across multiple banked segments of a GS/OS file will be maintained correctly.

Note: The 'Build Labels' process can take some time, depending on the size of the original source file, and the processor speed of your IIgs. Multi-segment files obviously will take much longer. It can take many minutes to complete for a large file, so please be patient while the process completes.

Apply Labels - This will re-apply any Labels that may have got lost in the coding process. It only sorts and applies label entries, without checking for any new ones.

If you have defined any 'Equates', it will be necessary to 'Build Labels' from scratch, in order that any equates are correctly applied in the code.

If you have 'Defined Labels' for sections of code, you will also need to 'Apply Labels', so the definitions are inserted in the correct places.

Define Label - Labels will mostly be automatically generated, but there are some instances of code where they cannot easily be determined, or rather it could be dangerous to interpret them automatically as a referenced address.

This function lets you manually define these immediate referenced addresses. It is only available when working with native 65816 files.

Define Label can be used in two instances, either where a PEA instruction is pushing an immediate address onto the stack, or where a processor register is being referenced with an immediate address.

In the case of the PEA instruction, you must select two lines, making a split valid address from within any of the segments of that file. The lines need not be adjacent.

This is an example explains how the function works:

```
025F 21025F: F4 21 00    PEA $0021
0262 210262: F4 7C 9B    PEA $9B7C
```

Here the first PEA operand pushes the Hi word of the effective address onto the stack, followed by the second PEA operand pushing the Lo word of the effective address onto the stack.

The 'Define Label' function picks up the two values, checks they make a valid 16-bit address within one of the segments, then turns them into a Label in this ORCA/M format:

```
025F 21025F: F4 21 00    PEA |L219B7C|-$10
0262 210262: F4 7C 9B    PEA |L219B7C
```

The generated Label is then added to the Link.File, and will be turned into a full Label line when you later use 'Build' or 'Apply' Labels. If using the Merlin format, the lines will be converted into Merlin compatible format when you later build the source files.

The second instance is where the processor registers references two immediate values, which make a valid address within one of the segments. The lines need not be adjacent.

This is an example explains how the function works:

```
0A24 210A24: A9 21 00    LDA #$0021
0A27 210A27: 48          PHA
0A28 210A28: A9 A0 4B    LDA #$4BA0
0A2B 210A2B: 48          PHA
```

Here the first operand holds the Hi word of the effective address, followed by the second operand holding the Lo word of the effective address. The Label will then be generated to look like this:

```
0A24 210A24: A9 21 00    LDA #^L214BA0
0A27 210A27: 48          PHA
0A28 210A28: A9 A0 4B    LDA #L214BA0
0A2B 210A2B: 48          PHA
```

Note: In both cases the last line must not be selected, so it has been shown in grey for this example.

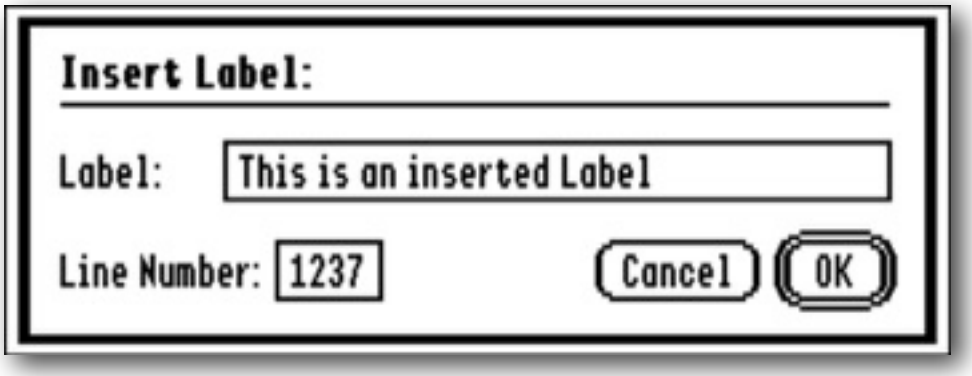
You can also select a single line, when the current bank will be used to build the Label:

```
0AA7 210AA7: A0 10 C0    LDY #$C010
```

```
0AA7 210AA7: A0 10 C0    LDY #L21C010
```

Note: These Labels will only be applied later when you 'Build' or 'Apply' labels.

Insert Label - Labels will in the main be generated either automatically, or using the Define Label function. Optionally you can generate a Label line, using any name or description you like.



Optionally, you can specify a Line Number where to add the Label line.

Note: A Label will always display in the code with a preceding underscore, but when the Source is built, that first underscore will be removed. A Label must start with either an alpha character, or an underscore, but can contain any alphanumeric character after that. Any spaces will be translated into underscores.

Delete Line - Deletes a selected line. You will first see the selected line highlighted, and must confirm that you wish to delete it.

Tip: To delete a range of code, first turn the code into a DS data line.

Revert to Saved - You can revert to the last Saved version of the displayed segment. This will undo any changes you have made.

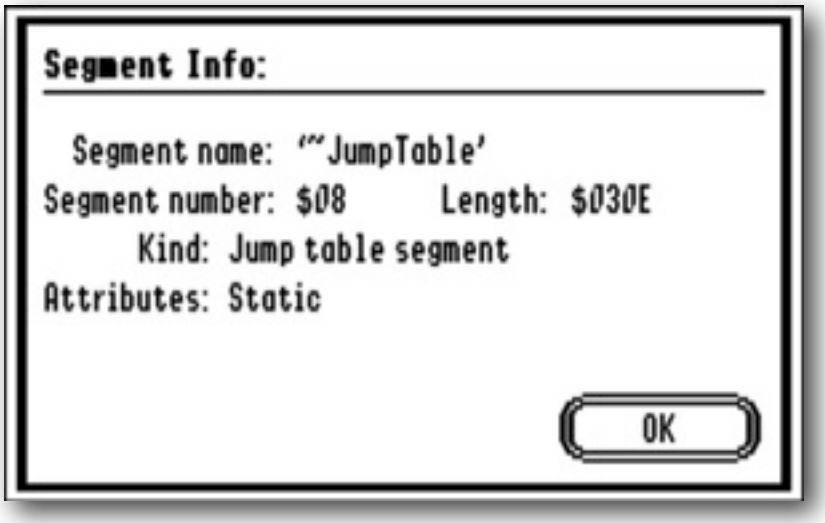
Other Editing Functions

Next Segment - If you are working with a multi-segmented GS/OS file, this will move to the next segment in the file, wrapping round to the first segment. If an ~ExpressLoad segment is present, it will not be displayed, as there is nothing meaningful that you can do with it.

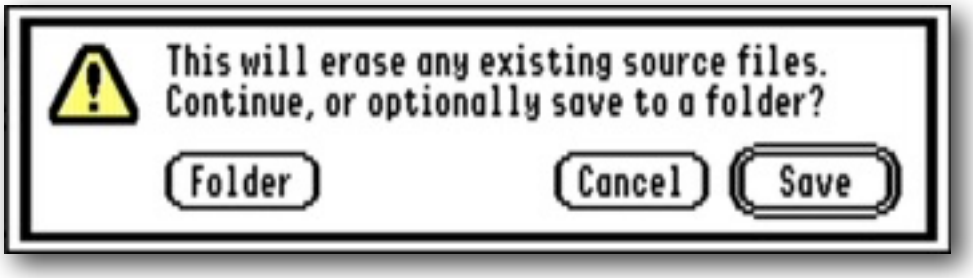
If a segment has been named, you will see its name in the menu, and at the top of the screen when you edit. In some applications, segments may not be named, and will show in the menu as (Blank Segment Name), and as an empty string at the top of the screen.

The Resources can be displayed, but cannot be edited directly. If you need to edit the Resources, please refer to the General Word Processor for more details.

Segment Info - For GS/OS files, displays the segment information:



Building Source - As the main reason for disassembling an application will be to edit and re-assemble it, you will need as a final step, to Build Source files from the disassembled and edited code. The output will depend on whether you are working with ORCA/M or Merlin format. Many of the functions, and formatting of the output file, will depend on that selection.



The default is to save the Source files to the Project folder in the Src.Files folder, but optionally, you can choose to disassemble to a folder of your choice.

Note: Merlin files are output with the Hi-bits set for most of the data in a source file.

The Resource files will already have been correctly structured when the application was disassembled, but in the case of Merlin files, these will have the Hi-bits set on the output.

This is a short example from the start of a code file:

```
****************************************************************
*
* Disassembled by BrkDown on  2/20/15 11:06:21 AM
*
* Project: ChewBagger = Segment $02 = ''
*
****************************************************************

** Global Equates:

     True gequ $8000
     False gequ $0000

0000 _L210000 entry
0000 210000: 4B          PHK
0001 210001: AB          PLB
0002 210002: 82 0D 00    BRL L210012
0005 210005: 71 77       DC  I2'L217771'
0007 210007:             STR 'ChewBagger'
0012 _L210012 entry
0012 210012: 3B          TSC
0013 210013: 8D D8 9A    STA L219AD8
0016 210016: A9 01 00    LDA #$0001
0019 210019: 22 04 02 E1 JSL >$E10204
```

This shows how it will look when built as an ORCA/M source file:

```
*************************************************************
*
* Disassembled by BrkDown on  2/20/15 11:06:21 AM
* Source built by BrkDown on  2/20/15 11:09:13 AM
*
* Project: ChewBagger = Segment $02 = ''
*
*************************************************************

        65816 on

Begin   start

** Global Equates:

True gequ $8000
False gequ $0000

L210000 entry
        PHK
        PLB
        BRL L210012
        DC  I2'L217771'
        STR 'ChewBagger'
L210012 entry
        TSC
        STA L219AD8
        LDA #$0001
        JSL >$E10204
```

This shows how the same code will look when built as a Merlin source file:

```
*************************************************************
*
* Disassembled by BrkDown on  2/20/15 11:06:21 AM
* Source built by BrkDown on  2/20/15 11:12:34 AM
*
* Project: ChewBagger = Segment $02 = ''
*
*************************************************************

 XC  ; enable 65C02
 XC  ; full 65816 asm
 MX %00 ; 16 bit entry

Begin start

** Global Equates:

True equ $8000
False equ $0000

L210000 ent
 PHK
 PLB
 BRL L210012
 DW L217771
```

```
 STR 'ChewBagger'
L210012 ent
 TSC
 STA L219AD8
 LDA #$0001
 JSL >$E10204
```

This shows an example of a Control Template and Menu Item from an ORCA/M Resource file:

```
// --- rControlTemplate Definitions

resource rControlTemplate (CTLTMP_00000001, $0000) {
                    $00000001,                  // ID
                    {  0,481, 27,483},          // rect
                    rectangleControl {{
                              $0002,            // flag
                              $1000,            // moreFlags
                              $00000000         // refCon
                    }};
};

// --- rMenuItem Definitions

resource rMenuItem (MENUITEM_000000FA, $C018) {
                    $00FF,                  // itemID
                    "Z","z",                // itemChar, itemAltChar
                    NIL,                    // itemCheck
                    $80C0,                  // itemFlag
                    PSTR_000000FA           // itemTitleRef
};
```

This shows the same Control Template and Menu Item from a Merlin Resource file:

```
** rControlTemplate Definitions

CTLTMP_00000001 equ *
 dw 6 ; pCount
 adrl $00000001 ; ID (1)
 dw 0,481,27,483 ; rect
 adrl $87FF0003 ; rectangle control
 dw $0002 ; flag
 dw $1000 ; moreFlags
 adrl 0 ; refCon

** rMenuItem Definitions

MENUITEM_000000FA equ *
 dw 0 ; menuitem template version
 dw 255 ; menuitem ID
 asc 'Z' ; alternate characters
 asc 'z'
 hex 0000 ; check character
 dw $80C0 ; menubar flag
 adrl PSTR_000000FA ; itemTitle
```

General Functions

Open Projects - Opens the Projects Dialog. See 'Selecting Projects' for more details.

Close Project - Closes the currently displayed Project, and returns you to the Projects dialog.

Load File - Loads a text file into a general word processor. This editor is provided so you can do any final tweaking or editing of a code file.

Be very careful that any changes you make here are compatible with your assembler. If necessary, you can always return and make further changes from the Project edit window.

Save - If any changes have been made, saves the current open file back to disk.

Save As - Saves the current open file to a new file on disk. This function is not available from the Projects window.

Derez File - Normally you will work with the disassembled resource forked files that are generated when you select a Project. If you just need to disassemble a resource fork for other reasons, this will let you save the files to a folder of your choice. You should first select the format you wish to work with.

Note: There have been new resources added since Genesys 1.2.4 was written. BrkDown not only supports those new resources, but also clears up a few bugs that Genesys introduced into the ORCA/M disassemblies.

Preferences - Opens the Preferences dialog. See 'Preferences' for more details.

Printer Setup - Opens the standard dialog for the printer you have chosen in the Direct Connect Control Panel.

Print - Opens the printer dialog for your chosen printer, and prints either the selected text, if none has been selected, the entire text.

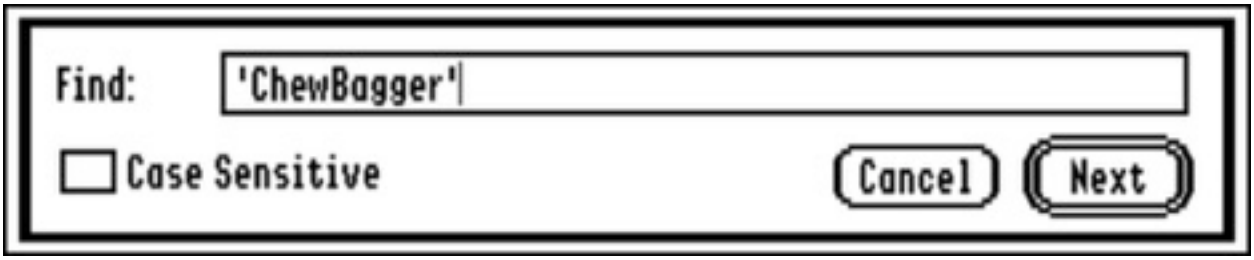
Quit - Quits BrkDown. If you were viewing the Projects window at the time, when BrkDown is reopened, it will load the same file, and go to the same selection.

Next - For segmented files, displays the next segment, wrapping round to the first.

Show Clipboard - Displays the current contents of the Clipboard.

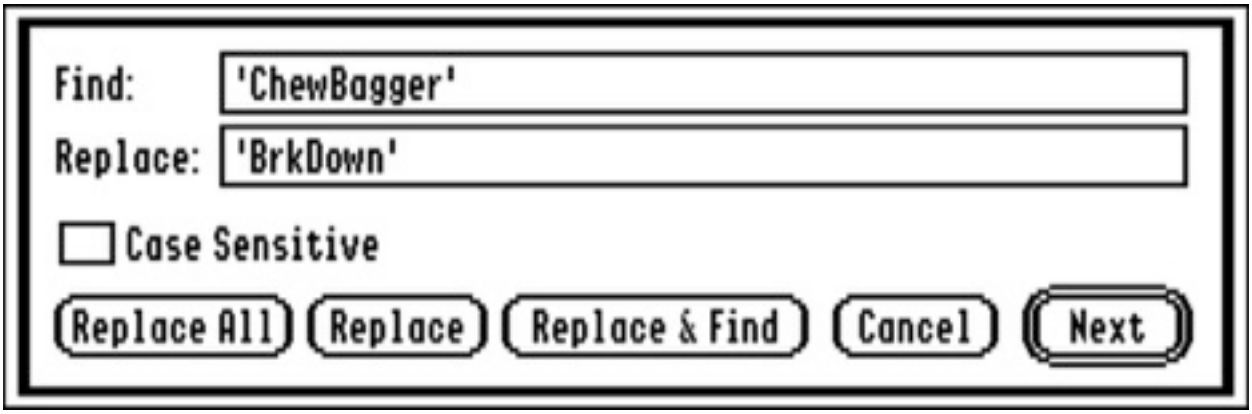
Go To - When working with the Projects window, if you select a number in any of these formats 'L012345', '$00012345', '$012345', or '$0000', it will Jump or Go To that line number and select it. If the current bank number is not given, it will assume that the address is within the current bank. If the bank number is given, it must match the current bank for the Jump to work.

Find & Find/Replace - Opens a Find or Find/Replace dialog. When the Project window is open, you can only Find text:



Enter the search string in the top box, and click the 'Next' button to start searching. Searching starts from the top of the file. Once found, you can look for more instances, using the Open-Apple-G key.

For the general word processor, more options are available to you:



Enter the search string in the top box, and an optional replace string in the bottom box. Choose from the various buttons what it is you would like to do.

Appendix

Useful Links

There are other 16 bit disassemblers available which you may like to look at:

'**ORCA.DISASM**' can be found on the CD '**Opus][: The Software**', available from the Syndicomm online store:

http://store.16sector.com/

The '**Flaming Bird Disassembler**' is available from Brutal Deluxe:

http://www.brutaldeluxe.fr/products/french/tfbd.html

'**Sourceror**' is part of the Merlin suite of software written by Glen Bredon. This is not commercially available, but may be found on some download sites.

Assemblers you might be interested in:

'**ORCA/M**' is on the CD '**Opus][: The Software**', available from the Syndicomm online store:

http://store.16sector.com/

'**Merlin32**' is available from Brutal Deluxe:

http://brutaldeluxe.fr/products/crossdevtools/merlin/

'**Merlin 16**' is part of the Merlin suite of software written by Glen Bredon. This is not commercially available, but may be found on some download sites.

Contacts - Problems

Hopefully you will have none, but if you do, and they cannot be answered by reading these notes, please contact me on:

spectrumdaddy@speccie.co.uk

Contacts - Other information

If you do not already know about Spectrum™, please drop by my home pages and read more. Apart from all the other wonderful things it does, Spectrum™ offers many useful tools for processing files, such as post processing text files that you have received that may have obstinate formatting.

Spectrum™ is now Freeware, and with all my other applications, is available from my web site:

http://www.speccie.co.uk

Someone once said to me, 'Spectrum™ does everything!'

BrkDown © 2015 Ewen Wannop