SYNONYMS AND MACROS are two ways to customize the vrhotwires scripting experience to suit your own tastes.

They can

- help bring english commands into foreign languages.
-boil complex scripts into single commands (with paramaters!)
-provide compatability between different scripting languages by allowing the user to write a 'translator' file...

To use a synonyms file, change it's name to 'Synonyms' and put it in the same folder as the app.
After that anything you define in the file will write out in the new way, and when you enter it it will be understood.

EXAMPLE 1

So lets say that you would prefer the script

SpriteTrackSetVariable 7004 111 TrackIndex=7

to instead read:

Put the red key into the inventory

You might create defines in a synonyms file (or in the actual script's frameloaded event with a
// preceding) that read:

```
 define (SpriteTrackSetVariable) = (Put)
 define (111) = (the red key)
 define (TrackIndex=7) = (into the inventory)
```

Then your language could also have:

Put the blue key into the inventory
Put the big clue into the inventory
Put the purchaseItem#29 into the inventory
Put the magic spell into the inventory

without much trouble....

EXAMPLE 2

Let's say you were doing a project where it became annoying to say

```
SetPanAngle 99.7
```

and you instead just want to say:

```
Look at the yellow fish
```

You might create defines in a synonyms file (or in the actual script's frameloaded event with a
// preceding) that read:

```
 define (QTVRSetPanAngle) = (Look at)
 define (99.7) = ( the Yellow Fish)
 define (21.3) = ( the Red Fish)
 define (1.7) = (   the Green Fish)
 define (293.7) = (        the Blue Fish)
```

Now you can write scripts like:

```
On MouseDown Sprite 1
      Look at the Yellow fish
end
On MouseUp Sprite 2
     Look at the Red fish
end
On MouseEnter Sprite 3
     Look at the Green fish
end
On MouseDown Hotspot 22
     Look at the Blue fish
end
```

****
WARNING: these synonyms are very aware of spaces.
****

Using this we can eventually build a more natural-language-like parser:

When we add speech recognition to vrhotwires, we want to be able to speak phrases like:

```
If the user hasn't clicked in a while
                        Turn  up the music
                        Start Demo Mode
                          Go into screen saver fullscreen mode
 end
If the user clicks
                     if we're in demo mode
                                 End Demo Mode
                                   Mode is now "Interactive"
                     endif
end
```
****

Foreign languages:

Synonym substitution is also very useful for non-english speakers trying to learn quicktime wired scripting.

You can make a synonyms file that gives you hints in your own language as to how the calls work... yikes, this is sprenchlish:

```
 define (SetPanAngle) = (      aller un autre vista)
```

Then when you write out a number of different wired movies and study their scripts, they will write out 'in your own language'.

****
TIP: CUTAND PASTE A LOT IF YOU USE LONG SUBSTITUTIONS....typing in long strings is going to introduce errors...
****
There are really 3 ways that vrhotwires does substitutions:

1 is synonyms files

2 is //define comments in frameloaded triggers...

3 is macros where a block of code is substituted and parameters can be used...
#####
TIP: the first thing many people use synonyms for is to name variables. If you do this make sure your variable's number is long enough and complex enough to not occur all over the place...

If you have variable 1 then every 1 in every place and context will be substituted. This will cause all kinds of problems for you. So use something longer like variable 1234 instead. The odds of there being another random occurance of 1234 are minimal...
#####
MACROS:

Macros allow you to create commands with parameters that represent complex blocks of wired code.

for example, rather than writing out:

```
 SpriteTrackSetVariable 7777   1   tracktype=sprite
 SpriteTrackSetVariable 7778   99   tracktype=sprite
  SpriteTrackSetVariable 7779   3   tracktype=sprite
```

With a macro, the scripter can write out:

```
panto 99 3
```

Because the macro file reads:

```
 SpriteTrackSetVariable  7777   1   tracktype=sprite
 SpriteTrackSetVariable  7778   ^0   tracktype=sprite
   SpriteTrackSetVariable  7779   ^1   tracktype=sprite
```

and is called 'PanTo'

(so the name of the file in the macro folder is the new word in the language.)

If you study the macro files, you'll see the important marker is a ^ symbol and
passed parameters need to be numbered from 0 to 9

Also the file must be saved with DOS line endings...(use BBedit or codewarrior)

and you have to tell the macro engine what kind of data your new passed parameter is.

```
#      kFloatData = 1,
#      kFixedData,
#      kLongData,
#      kShortData,
#      kCharData,
#      kPStringData,
#      kCStringData,
#      kBooleanData,
#      kMatrixData,
#      kModifierTrackGraphicsModeRecordData,
#      kLanguageCodeData,
#      kPointData,
#      kTextStyleData
```

are the types of data.

So at the top, if you have 3 parameters, you need 3 numbers, each representing a datatype from the above list.

3 3 3 means long long long. (if this hangs you up, try guessing, or send it to bill and he'll
tell you what kind of data each call uses. (it's in movies.h, one of the universal headers.))

Other than that, I hope lots of people write good macros for everyone to use!