QuickBASIC to XBasic Conversion Notes

The following notes were provided by an *XBasic* programmer who converted an approximately 10,000 line QuickBASIC 4.5 program to *XBasic* in about two days, first running in character mode, then in another day with a GUI added.

First and foremost, keep the BASIC to *XBasic* conversion table nearby - that's an appendix in the *XBasic* programming language manual.

Most changes are easy to make, often with a monitored wildcard replace. I burned myself a few times with unmonitored wildcard replaces, so I suggest you look at each one before you replace. With *XBasic*, you can do this quickly with the F11 / F12 function keys.

Load your program as a text file, otherwise *XBasic* will have a real hard time trying to figure out what's going on. After all, *XBasic* understands *XBasic*, not QuickBASIC.

Save a new copy of the file you're converting every 5 minutes as a separate file. For example, you should end up with prog0000.x, prog0001.x, prog0002.x, ... prog0025.x by the time you finish. That way if you royally screw up some conversion you can go back to a recent decent working copy.

Type .h in the upper text area to get a list of shortcut "dot commands" including some real good find/replace examples. Entering dot commands in the upper text area is often much faster.

By the way, I've found that most of the find and replace stuff goes much easier if you set the find and replace to be case sensitive, which you only need to do once. Just select "Edit" then "Find" and depress the toggle button labeled "Case Sensitive". Also note that you can do reverse find and replace with F11 and F12 if you hold the shift key down.

0: Remember, *XBasic* is case sensitive. So **`FOR`** is a keyword and **`for`** is a variable. Also note that **`for`** and **`For`** and **`foR`** and **`FoR`** are all different variables because case matters.

If you don't find an equivalent for something that is pretty fundamental, check the standard function library. Check the "\xb\doc\library.doc" manual, or better yet, select "Help" + "StandardLibrary" from the menu bar in the main *XBasic* window. For graphics, take a couple hours and read the"\xb\doc\graphics.doc" manual.

1: Replace all occurrences of **`SUB`** with **`FUNCTION`**.

In *XBasic* every function is called a **`FUNCTION`**, whether it returns an argument or not.

You need to monitor this else you'll accidentally change all occurrences of **`GOSUB`** into **`GOFUNCTION`**.

2: Go through your program and find all subroutines, which in the meaning of *XBasic* is everywhere your program `GOSUBs` to. Find the beginning and end of each subroutine and:

- a: Put a `SUB` keyword before the subroutine name and remove the trailing colon from the name. For example, if the beginning of a subroutine is a line **dothis:**, change it to "**SUB dothis**".
- b: Replace the `RETURN` keyword at the end of the subroutine with `END SUB`.
- c: Find all `RETURN` keywords within the subroutine and change them to `EXIT SUB`.

3: Make sure no `RETURN` keywords remain in your program.

4: In every context, *XBasic* functions require parentheses after the function name, even if the function takes no arguments. Find every occurrence of functions that take no arguments and add `()` after the function name. For example, change `funcname` to `funcname()`.

5: As I recall, in QuickBASIC you return a value from a function by assigning a value to a variable with the same name as the function. In *XBasic* you simply say `RETURN (value)`. If you have a function named `a()` it can contain a variable named `a` that is just like every other variable in the function and has nothing whatever to do with returning values from the function.

6: *XBasic* array names are always followed by square brackets, as `array[n]` instead of `array(n)`. This makes it much easier to read *XBasic* programs because you can always tell when something is a variable, an array, or a function. So go find all your arrays and change the parentheses to square brackets.

7: Execution of *XBasic* programs always begins with the first declared function. The "**PROLOG**" of *XBasic* programs is for type declarations, for function declarations, and for constant declarations - nothing else. If you have any executable code before your first function, put it in an `Entry()` function and make sure it's the first function declared in the **PROLOG**.

8: I don't remember the scope rules for QuickBASIC very well any more, but with *XBasic* all variables are automatic local variables to the function they're in unless declared otherwise after the beginning of the function. If you want to share a variable with another function you need to put the variable in `SHARED` statements in both functions, or add a # prefix to the variable name wherever it appears.

Available scopes are:

AUTO	- automatic and local - maybe in a CPU register
AUTOX	- automatic and local - never in a CPU register
STATIC	- permanent and local - value retained between calls
SHARED	- permanent and shared - share in other functions too
EXTERNAL	- permanent and shared - with statically linked modules

9: Shared constants must be defined in the **PROLOG**, and have a `$$` prefix, as in `$$MyConstant`. So `CONST MyConstant = 32` in QuickBASIC becomes `$$MyConstant = 23` in *XBasic*.

10: Local constants are defined within a function must begin with a **$** prefix, as in **$ThisConstant**. So **CONST ThisConstant = 11** in QuickBASIC becomes **$ThisConstant = 11** in *XBasic*.

10: To convert strings to numbers in other BASICs you write:

```
value# = VAL (string$)   ' VAL returns double precision
```

In *XBasic* you can convert strings to any data type you want:

```
value  = SBYTE (string$)   ' signed byte (16-bits)
value  = UBYTE (string$)   ' unsigned byte (16-bits)
value  = SSHORT (string$)  ' signed short (16-bits)
value  = USHORT (string$)  ' unsigned short (16-bits)
value  = SLONG (string$)   ' signed long (32-bits)
value  = ULONG (string$)   ' unsigned long (32-bits)
value  = XLONG (string$)   ' native long (signed 32-bits)
value  = GIANT (string$)   ' signed giant (64-bits)
value  = SINGLE (string$)  ' 32-bit IEEE floating point
value  = DOUBLE (string$)  ' 64-bit IEEE floating point
value$ = STRING (string$)  ' character string
value$ = STRING$ (string$) ' character string
```

To exactly duplicate **VAL()**, replace it with **DOUBLE()**.

11: To generate a string of several of the same character, QuickBASIC has the **STRING$()** intrinsic. But **STRING$()** in *XBasic* converts any data type to a string representation. To generate a series of the same character with *XBasic*, change **STRING$()** to the two argument form of **CHR$()**. For example, change QuickBASIC **a$ = STRING$("a",5)** to **a$ = CHR$('a', 5)**.

12: To create formatted strings, see **FORMAT$()**.

13: *XBasic* replaces **MID$()** on the left side of the assignment operator (=) with **STUFF$().**

14: Remember, by default *XBasic* function arguments are passed by value, not by reference like QuickBASIC. To make an argument passed by reference, prefix the argument with a **@**, as in

```
a = func (@a, @b, @c$)
```

15: Remember, by default the type of *XBasic* variables/arrays is 32-bit integer **XLONG**, not **SINGLE**.