AWC

SeaBass
User's Manual

AWC
1279 FM 518 #2
Kemah, TX 77565
(281) 334-4341
http://www.al-williams.com/awce.htm
V1.0 15 July 2004

Table of Contents

Overview – What is SeaBass?

SeaBass is a unique compiler for a Basic-like language that uses standard C as an output language. That means that SeaBass allows you to write easy-to-understand Basic programs for any microcontroller that supports a C compiler!

In addition to the simplicity of Basic, SeaBass offers you full access to the power of C. You can mix C code with your programs as well as add in existing or new C libraries. If your C compiler supports embedded assembly code, you can even put assembly language code into your SeaBass program!

Because SeaBass integrates seamlessly with C, it is a perfect intermediate step between Basic and C programming.

If You Need Help

If you require assistance, please feel free to contact us. The best way to get support is via e-mail (stamp@al-williams.com). However, you may also call between 9AM - 4PM Central Time at (281) 334-4341. You can also fax to (314) 596-9925. Be sure to check out our Web page for updates at www.al-williams.com/awce.

What Else You'll Need

In addition to SeaBass, you must have a C compiler for the platform you wish to target. Most of the examples in this manual assume you are using the AWC APP-IV Atmel ATMega development board and the associated GNU C compiler. However, many compilers will process the code produced by SeaBass, so you may use it with GNU C on nearly any platform (including Linux or Windows) and other compilers such as SDCC, IAR, and others. The example directory has files required for using SeaBass with SDCC and the APP-II Microchip PIC development board.

Of course, you also need a hardware target and a means of programming the processor. This manual assumes you are using the APP-II or APP-IV for these functions.

Features

SeaBass includes all the features you'd expect of a modern Basic language including typed variables, looping constructs, and block structured control statements.

Although you don't need a detailed knowledge of C to use SeaBass, the language does use some C language constructs to improve efficiency and maintain compatibility with C code.

Differences from Basic

There are a few differences between SeaBass and traditional Basic that you should be aware of:

- All variables must be declared with **Dim** or **Array**
  Example: Dim x as int

- Comments at the end of a line start with ` (a back quote) instead of an ordinary quote
  Example: x=f() ` Set x to the current value

- Arrays use square brackets ([]) for indices:
  Example: sample[i]=getValue()

- Identifier names for variables, functions, and subroutines are case sensitive
  Example: X=x+10 ` two different variables

- Calling a subroutine or a function with no arguments still requires empty parenthesis
  Example: x=f()

- Functions return a value via the **RETURN** statement
  Example: RETURN 10

- You must avoid using words reserved by your C compiler as identifiers even if they are legal Basic identifiers
  Example: break=10 `break is reserved by C compiler!

- Parameters are passed to functions and subroutines by value, not by reference (that is, functions and subroutines receive copies of their arguments by default)

- There is no **GOSUB** statement (SeaBass uses named functions with formal parameters)

- SeaBass does not support Basic string handling (although you can use C-language strings with no problem)

- SeaBass uses C-language operators, most of which are the same as their Basic equivalents; a few are different (for example Basic's **MOD** operator is **%** in SeaBass)

There is one exception to the last rule. Although C uses == and **!=** as operators, SeaBas recognizes = and <> for these operations just as Basic does.

Controlling the Build Process

SeaBass' major function is to generate C source code that corresponds to your program. However, SeaBass can also launch a script that will complete the build process by invoking the C compiler and other tools necessary to generate an output object file.

By default, SeaBass tries to execute a program called ccompile.bat to complete the build process. It sends the base name of your program as the first argument, and any "linked files" (see **#LINK** in the keyword reference) as additional arguments.

What this file does is entirely up to you. For the APP-IV, for example, the file looks like this:

```
@echo off
set TARGET=%1
set EXTRA=%2 %3 %4 %5 %6 %7 %8 %9
make program
```

Of course, there is a makefile included which is used to perform the build process and start the programming software.

Command Line Options

If you start SeaBass with no command line arguments, a dialog will appear that allows you to select an input file, a build script, and set several options. You can leave this window open and simply press the Compile button when you wish to rebuild your program.

However, if you prefer, you can name the source file on the command line. This prevents the user interface from appearing. In this case, you can specify several options on the command line to control the build process:

- -c: Compile only; do not attempt to launch a build script
- -b script: Specify a build script to use instead of ccompile.bat
- -l: Prevent generation of **#line** directives; normally SeaBass includes these to allow the C compiler to generate more meaningful error messages

If you do not supply command line arguments, you'll see a dialog similar to the following:

This dialog lets you set all the parameters required to operate SeaBass. You may leave the “Optional Script” box empty if you do not require a customized build script (that is, this box corresponds to the –b option).

The dialog does provide one additional option: “Convert \ to /” which causes SeaBass to convert slashes in the file names before calling the build script. This is useful when using the GNU C compiler and other Linux-style tools.

Keyword Reference

Note that keywords are not case sensitive, although variable, function, and subroutine names are case sensitive.

@

The @ keyword passes the remainder of the line directly to the C source code output. This is an easy way to include C code in your SeaBass source.

```
@#include <io.h>
@x++;
```

@@

The @@ keyword causes subsequent lines from your SeaBass program to appear in the C language output. This copying continues until another @@ line appears. This is handy for including large amounts of C code into your SeaBass program.

@@

```
/* This is a piece of code copied from
somewhere else */
int dblit(int n)
{
   return n*2;
}
@@
```

‘ *(Apostrophe)*

The apostrophe or single quote is used at the start of a line to mark a comment (just like **Rem**). Note, however, that if the line has anything other than a comment on it, you must use the backquote (`) to mark the comment. It is acceptable to use the backquote in all cases.

```
' A comment that is OK
x=10 ' A bad comment (needs back quote)
```

` *(Back Quote)*

The back quote causes everything after it on the line to become a comment. You can use this anywhere on a line.

```
` A single line comment
x=10 `Set X to 10
```

#Link

The **#Link** keyword causes the compiler to send the named file (which must be in quotes) to the compile script. This is a handy way to add C or object files to your build process.

```
#Link "mylib.c"
#Link "yourlib.o"
```

Array

The **Array** keyword defines a single array with an optional type. If you don't specify a type, SeaBass assumes **int**. Note that SeaBass uses square brackets to denote array subscripts, which always start at 0. The size provided in the **Array** statement defines the length of the array, so a 10 element array has elements 0 to 9.

Array samples[20] ` int samples[0] to samples[19]

Array cs[10] as char ` cs[0] to cs[9]

Cinclude

You can use **Cinclude** to include a C-language header file. It is essentially the same as writing "@#include".

```
Cinclude "io.h"
```

Const

The **Const** keyword allows you to define constants in your source code.

```
Const limit=10
```

DefType

Variables have types in SeaBass. Each type must be a single word that corresponds to a C language type (for example, **int** or **char**). However, using **DefType** you can define more complex types and use a name of your choosing. For example, you might want to define an unsigned type **uint** that is equivalent to **unsigned int**.

```
DefType uint unsigned int
DefType pint int *
```

Dim

The **Dim** keyword allows you to define variables and assign their types. Unlike standard Basic, this step is required since memory is a scare resource on most microcontroller targets where you will

use SeaBass. The type names must be a single word that the C compiler will recognize (you can use **DefType** to define custom types that are more complex). The actual types available will depend on your C compiler, but typical values are: **char, int,** and **float**. If you omit the type name, SeaBass assumes the variable is an **int**. All **Dim** statements must appear outside of a **Function** or **Sub** or at the beginning of a **Function** or **Sub**.

```
Dim I    ` Assumed int
Dim x as int
Dim a as char, y as float
```

Do/Loop

SeaBas allows four types of controlled do loops. The first type always executes at least once and continues until the expression provided to the **Loop** keyword is nonzero:

```
Do
  x=x+1
Loop While x<>10
```

You can also loop until the expression is zero:

```
Do
   x=x+1
Loop Until x=10
```

It is also possible to do the test first in both cases:

```
Do While x<>10
  x=x-1
Loop

Do Until x=10
  x=x-1
Loop
```

End

The **End** statement can be used to end a subroutine or function by adding the **SUB** or **FUNCTION** keyword, as appropriate. You also end an **If** statement with **End If**.

When used by itself, **End** causes the processor to loop endlessly, effectively stopping your program.

```
Sub Demo(n as int)
If n=0 then
    n=1
End If
F(n)
End Sub
```

For/Next

The **For** loop allows you to execute code some number of times. There are two forms:

```
For I= 1 to 10
  F(I)
Next I
```

In this format, I assumes values of 1 to 10, with an increment of 1 (that is, the sequence is 1, 2, 3, 4… 10).

You can supply an optional **Step**:

```
For I = 1 to 10 Step 2
  F(I)
Next I
```

Now the sequence is 1, 3, 5, 7, 9.

It is permissible to make the loop go backwards with or without a **Step** clause. For example:

```
For I= 10 to 1
  F(I)
```

```
Next I
```
SeaBass will correctly set the default step value to -1 in this case.

Function

The **Function** keyword allows you to define a subroutine that returns a value via the **Return** keyword**.**

```
Function dblit(n as int) as int
  Return n*2
End Function
. . .
x=dblit(20)  ` x=40
```

Note that if the function doesn't take arguments, it still requires parenthesis both when defining it and when using it:

```
Function quibble() as int
   Return 100
End Function
. . .
x=quibble()
```

In addition, SeaBass programs start executing by running a function called **main**. Remember, SeaBass is case sensitive, so **Main** or **MAIN** won't work – it has to be **main**:

```
Function main() as int
. . .
End Function
```

Some C compilers will complain that **main** needs arguments, and you can supply them if the compiler is strict about it. For most embedded systems, it doesn't matter since these arguments pass information about a nonexistent command line.

Goto

Use **Goto** to jump to a named label in your program. Labels are simply an identifier ending with a colon. The underlying C compiler will place restrictions on where you can jump (for example, you can't jump from one **Sub** or **Function** to another.

```
toploop:
   x=x+1
   If x>0 Then
       y=y+1
       Goto toploop
   End If
```

Note that you can often use a **Do** loop or other SeaBass feature to avoid using **Goto**.

If/Then/Else/End if

There are two forms of the **If** statement. The first form jumps to a label if the indicated condition is non zero (see **Goto** for more on labels):

```
If x=0 Then Xwaszero
. . .
Xwaszero:
  Y=Y+1
```

The second form allows you to execute multiple lines of code based on the condition:

```
If x=0 Then
    Y=Y+1
    Z=Z-1
End If
```

You can also use an **Else** keyword with this form to execute statements if the condition is zero (that is, not true):

```
If x=0 Then
   Y=Y+1
   Z=Z-1
Else
   Y=Y-1
   Z=Z+1
End If
```

Include

The **Include** keyword allows you to bring another SeaBass file into your compilation. This is like copying the named file into your main file. For example, suppose your main file has the following:

```
X=1
Include “test.bas”
Z=3
```

And further assume that test.bas contains just one line:

```
Y=2
```

The final output would be just as if you had written:

```
X=1
Y=2
Z=3
```

Of course, you can include a file that includes another file, and so on (up to some practical limit set by your computer’s memory).

Let

The **Let** keyword is an optional way to introduce an assignment expression. For example, these two lines are equivalent:

```
Let X=0
X=0
```

Rem

The **Rem** keyword starts a remark. The entire line is then a remark. If you want to remark just part of a line use ` (backquote). The **Rem** keyword is equivalent to the apostrophe (').

Return

See the **Function** keyword for a description of **Return**. You can also use **Return** with no expression to end a **Sub** early.

Sub

The **Sub** keyword allows you to define a subroutine, which is similar to a **Function** but it does not return a value.

```
Sub BumpX(v as int)
  X=X+v
End Sub
```

Operator Reference

The exact operators supported by SeaBass actually depend on the underlying C compiler. However, here is a list of common operators:

Operator	Description	Sample
-	Negate	**-x**
~	Bitwise Inversion	**~x**
+	Add	**x+y**
-	Subtract	**x-y**
*	Multiply	**x*y**
/	Divide	**x/y**
%	Remainder From Integer Division	**x%y**
>>	Shift Right	**x>>y**
<<	Shift Left	**x<<y**
<	Less Than	**x<y**
>	Greater Than x>y	
<=	Less Than Equal To	**x<=y**
>=	Greater Than Equal To	**x>=y**
==	Equal	**x==y**
!=	Not Equal	**x!=y**
&	Bitwise And	**x&y**
|	Bitwise Or	**x|y**
^	Bitwise Exclusive Or	**x^y**
&&	Logical And	**x&&y**
||	Logical Or	**x||y**
!	Logical Not	!x

Note that in “logical” contexts (that is, the predicates of **If** statements or loops) SeaBass automatically converts = to == and <> to != so that you can continue to use Basic-like syntax for these operators.

Comparison operators like < or > return 0 for false or 1 for true. Bitwise operators operate on all the bits in their arguments (so 6 & 5 => 4). Logical operators treat any non-zero argument as a 1 and return a 0 or a 1 (so 6 && 5 => 1).

However, SeaBass does not perform this substitution in assignment contexts. If you wish to use “equal” or “not equal” in an assignment, you should use the C-language format.

APP-II Specifics

To use SeaBass with the APP-II, you must first install the SDCC compiler and tools (included on the CDROM). Once you have these installed, you’ll need to copy several files from the pic example directory. One of these files, ccompile.bat, will require modification to fit your system’s path.

Once you’ve modified these files, you can run Seabass in the usual way to generate a hex file. You download the hex file using any of the methods supported by the APP-II.

Note that the while SeaBass has specific support for the APP-II, you should be able to use SeaBass with any of the SDCC-supported processors including the PIC16F family or 8051-type processors. Naturally, you must have a way to program the device you want to use from a standard hex file.

APP-IV Specifics

To use SeaBass with the APP-IV, you must first install the Gnu C compiler and tools (included on the CDROM). Once you have these installed, you’ll need to copy two files from the avr example subdirectory: ccompile.bat is the compiler script for the compiler and makefile is a prototype make file for your projects.

You should edit both of these files and adjust paths or customize options that you want to change. For example, the makefile allows you to change the COM port you use, or the programming method (for example, you can use uisp or AvrDude). Once you change these files, you should not need to change them again. You can place ccompile.bat in a directory on your path, and simply copy the makefile to each directory you use. This will allow you to make further customizations, if necessary, to the makefile on a per project basis.

Once you have these files installed, you can compile your project by simply issuing the SeaBass command with arguments or by using the Windows dialog. SeaBass will automatically configure your makefile for your specific project and even launch the APP-IV programming software of your choice.

There are several .bh (SeaBass header) files that are useful with the APP-IV. The avrstdio.bh file, for example, provides routines for interfacing with the APP-IV serial port:

- uart_ready – Returns true if data is available from the UART
- uart_putchar – Writes a character to the serial port
- uart_getchar – Gets a character from the serial port
- uart_init – Set up the UART to the desired baud rate (64 is 9600 baud, see the file for constants for each baud rate)

In addition, the file enables you to use the C standard I/O functions such as **printf** and **scanf**, if you want to use them.

The avrio.bh file is just a wrapper around the APP-IV's I/O functions and provides easy-to-use I/O statements like **INPUT**, **OUTPUT**, **HIGH**, **LOW**, **TOGGLE**, etc. Keep in mind that you may use any of the APP-IV's C libraries, or any of the other libraries accessible to the C compiler from SeaBass. The example at the end of this manual shows a program that links in the C app4delay library and uses it directly.

Note that while SeaBass directly supports the APP-IV, you may use it with any target supported by GCC assuming you have a way to program the device.

Example

This program sends Morse code using an LED on port B, pin 0 of an APP-IV chip. It uses the SeaBass avrio.bh header to define I/O operations, and the APP-IV's C-language app4delay library to define time delay operations.

```
` Example SeaBass Program
` Sends a little Morse code on
` an LED on Port B pin 0

include "avrio.bh"        ` get I/O routines
cinclude "app4delay.h"    ` get delay from
#link "app4delay.c"       ` C code

` Define a string type
deftype string char *


` Speed of a dot in milliseconds
Const speed=200


` play a dot
sub dot()
  HIGH(B,0)           ` LED on
  delay_ms(speed)
  LOW(B,0)            ` LED off
  delay_ms(speed)
end sub

` play a dash
sub dash()
  HIGH(B,0)
  delay_ms(speed*3)
  LOW(B,0)
  delay_ms(speed)
end sub
```

```
` pause between characters
sub space()
  delay_ms(speed*3)
end sub


` Main program
function main() as int
  dim text as string
  dim i

  text="-.-. --.-"   ` message to send

` Set LED to output
  OUTPUT(B,0)

` Main code
top:
  for i=0 to strlen(text)-1   ` walk through
text
` do the right thing
     if text[i]='.' then
       dot()
     end if
     if text[i]='-' then
       dash()
     end if
     if text[i]=' ' then
       space()
     end if
  next
  delay_ms(10000)   ` wait 10 seconds
  goto top          ` and do it again
end function
```

Installing a C Compiler

The SeaBass CDROM contains two open source compiler distributions. The WinAVR distribution is a GNU C compiler for the AVR-family processors and supports the APP-IV board or nearly any Atmel AVR/ATmega processor. You can download the latest version of WinAVR from http://winavr.sourceforge.net.

The SDCC compiler supports the PIC and 8051/8052 processors (including our APP-II board). You can download the latest version of SDCC at http://sdcc.sourceforge.net.

Nearly any C compiler will work with SeaBass. To use a different C compiler, you'll need to create your own CCOMPILE.BAT to control the build process. Note that you can also ask SeaBass to compile only (using the –c option or the check box on the Windows dialog) and then manually compile the resulting C file.

SeaBass Keywords

@	C Pass through
@@	Multiline C pass through
‘	Comment at start of line
`	Comment at end or start of line
#Link	Add file to compile
Array	Define an array
Cinclude	Include C file
Const	Define constant
DefType	Define type
Dim	Declare variable
Do/Loop	Execute code repeatedly
End	Stop program or end subroutine/function
For/Next	Execute code repeatedly
Function	Define a function
Goto	Goto a label
If/Then/Else	Conditionally execute code
Include	Include a SeaBass file
Let	Optional assignment keyword
Rem	Remark
Return	End a function or subroutine
Sub	Define a subroutine

Notes

Notes

Notes

License

This is a legal agreement between the end user ("You") and Al Williams Consulting., its affiliates and subsidiaries (collectively "AWC"). This Agreement is part of a package (the "Package") that also includes, as applicable, executable files that you may download, a disc, or a CD-ROM (collectively referred to herein as the "Software") and certain written materials (the "Documentation"). Any patch, update, upgrade, modification or other enhancement provided by AWC with respect to the Software or the Documentation shall be included within the meanings of those terms, for the purposes of this Agreement, except to the extent expressly provided below.

BY DOWNLOADING OR INSTALLING THE SOFTWARE, YOU ACKNOWLEDGE THAT YOU HAVE READ ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT, UNDERSTAND THEM, AND AGREE TO BE BOUND BY THEM. YOU UNDERSTAND THAT, IF YOU PURCHASED THE PACKAGE FROM AN AUTHORIZED RESELLER OF AWC, THAT RESELLER IS NOT AWC'S AGENT AND IS NOT AUTHORIZED TO MAKE ANY REPRESENTATIONS, CONDITIONS OR WARRANTIES, STATUTORY OR OTHERWISE, ON AWC'S BEHALF NOR TO VARY ANY OF THE TERMS OR CONDITIONS OF THIS AGREEMENT.

If You do not agree to the terms of this Agreement, do not download or install the Software and promptly return the entire Package to the place You obtained it for a full refund. If you should have any difficulty in obtaining such refund, please contact AWC at 281-334-4341.

LIMITED LICENSE: You are entitled to download or install, and operate this Software solely for your own use, but may not sell or transfer reproductions of the Software or Documentation to other parties in any way. You may download or install, and operate one copy of the Software on a single terminal connected to a single computer. You may not network the Software or otherwise use it on more than one computer or computer terminal at the same time.

OWNERSHIP; COPYRIGHT: Title to the Software and the Documentation, and patents, copyrights and all other property rights applicable thereto, shall at all times remain solely and exclusively with AWC and its licensors, and You shall not take any action inconsistent with such title. The Software and the Documentation are protected by United States, Canadian and other applicable laws and by international treaty provisions. Any rights not expressly granted herein are reserved to AWC and its licensors.

OTHER RESTRICTIONS: You may not cause or permit the disclosure, copying, renting, licensing, sublicensing, leasing, dissemination or other distribution of the Software or the Documentation by any means or in any form, without the prior written consent of AWC. You may not modify, enhance, supplement, create derivative work from, adapt, translate, reverse engineer, decompile, disassemble or otherwise reduce the Software to human readable form.

LIMITED WARRANTY: AWC warrants for a period of ninety (90) days following original retail purchase of this copy of the Software that the Software is free from substantial errors or defects that will materially interfere with the operation of the Software as described in the Documentation. This limited warranty: (i) applies to the initial purchaser only and may be acted upon only by the initial purchaser; and (ii) does not apply to any patch, update, upgrade, modification, or other enhancement provided by AWC with respect to the Software or the Documentation or to any bonus game provided by AWC at no extra charge as part of the Package, which are provided on an AS IS BASIS ONLY. EXCEPT AS STATED ABOVE, AWC AND ITS LICENSORS MAKE NO OTHER WARRANTY OR CONDITION, EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, REGARDING THIS SOFTWARE. THE IMPLIED WARRANTY THAT THE SOFTWARE IS FIT FOR A PARTICULAR PURPOSE AND THE IMPLIED WARRANTY OF MERCHANTABILITY SHALL BOTH BE LIMITED TO THE NINETY (90) DAY DURATION OF THIS LIMITED EXPRESS WARRANTY. THESE AND ANY OTHER IMPLIED WARRANTIES OR CONDITIONS, STATUTORY OR OTHERWISE, ARE OTHERWISE EXPRESSLY AND SPECIFICALLY DISCLAIMED. Some jurisdictions do not allow limitations on how long an implied warranty or condition lasts, so the above limitation may not apply to You. This limited warranty gives You specific legal rights, and you may also have other rights which vary from jurisdiction to jurisdiction.

If you believe you have found any such error or defect in the Software during the warranty period, contact AWC as described in the Documentation. If you have a problem resulting from a manufacturing defect in the Software, AWC's and its licensors' entire liability and Your exclusive remedy for breach of this limited warranty shall be the replacement of the Software, within a reasonable period of time and without charge, with a corrected version of the Software. Some jurisdictions do not allow the exclusion or limitation of relief, incidental or consequential damages, so the above limitation or exclusion may not apply to You.

LIMITATION OF LIABILITY: AWC AND ITS LICENSORS SHALL NOT BE LIABLE FOR SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY OR OTHER INDIRECT DAMAGES, EVEN IF AWC OR ITS LICENSORS ARE ADVISED OF OR ARE AWARE OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL AWC'S AND ITS LICENSORS' AGGREGATE LIABILITY EXCEED THE PURCHASE PRICE OF THIS PACKAGE. Some jurisdictions do not allow the exclusion or limitation of special, incidental, consequential, indirect or exemplary damages, or the limitation of liability to specified amounts, so the above limitation or exclusion may not apply to You.

GENERAL: This Agreement constitutes the entire understanding between AWC and You with respect to subject matter hereof. Any change to this Agreement must be in writing, signed by AWC and You. Terms and conditions as set forth in any purchase order which differ from, conflict with, or are not included in this Agreement, shall not become part of this Agreement unless specifically accepted by AWC in writing. You shall be responsible for and shall pay, and shall reimburse AWC on request if AWC is required to pay, any sales, use, value added (VAT), consumption or other tax (excluding any tax that is based on AWC's net income), assessment, duty, tariff, or other fee or charge of any kind or nature that is levied or imposed by any governmental authority on the Package.

EXPORT AND IMPORT COMPLIANCE: In the event You export the Software or the Documentation from the country in which You first received it, You assume the responsibility for compliance with all applicable export and re-export regulations, as the case may be.

GOVERNING LAW: This Agreement shall be governed by, and any arbitration hereunder shall apply, the laws of the Texas, U.S.A.