readme

Allan Savage

COLLABORATORS			
	TITLE : readme		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Allan Savage	June 4, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

readme

1.1 Prog_Bar v1.1 Link Library

```
                                Prog_Bar.Lib


             A progress bar link library for Amiga Programmers

                       Written by Allan Savage © 1996
```

Contents

1.2 Introduction

Introduction

Prog_Bar.lib is a progress bar link library designed to make it easy to create, manage and delete progress bars within most programming languages. It has been tested using Dice C, Devpac 3 Assember, SAS/C and Turbo Modula-2.

The library is easy to use and provides a large range of options for your progress bars. It has been designed to operate in a similar way to the GadTools functions for managing gadgets, so the ideas involved should not be new to anyone who might be using them (except possibly assembly programmers.

See Usage for more details).

The library is capable of producing many different types of progress bars using a simple set of parameters. These allow the programmer to completely customise a progress bar by changing its border, colours, size, location, dimensions, direction and text.

To give you an idea of what is possible using this system I have included a few demonstration programs complete with source code so that you can learn from the techniques used to control a progress bar.

1.3 Installation and Usage

Installation

How to install and use the library really depends on which environment you want to use it in. I will provide a general outline of the various stages and a more detailed description of how I installed it with each system. In each case you may be using a different system, but if you follow the general guidelines you should not have much difficulty.

I will also show the commands I used to compile and link the demo programs for each system.

```
      C Programmers        Assembly Programmers     Modula-2 Programmers
     ---------------      ----------------------   ----------------------

      Installation            Installation            Installation
         Usage                   Usage                   Usage
```

The documentation includes an Amigaguide file called prog_bar.guide. This contains a detailed description of each function and can be copied to your compiler's document directory, or can be linked into the Autodocs documentation if you have it installed. A plain text version of this file is also supplied. It is called prog_bar.doc.

Usage

General Description

Using the system should now be (almost) as simple as using any of the operating system libraries. When you write a program which needs a progress bar just include the relevant definition file at the top of your source and remember to link your object file to prog_bar.lib.

To actually create and handle a progress bar within your program you will need to follow the rough guide given below. However, every situation is different so you might need to vary your approach. All the available functions are documented in the function list.

N.B. Because all of the necessary functions will be linked into your final executable it is completely unnecessary to open the library at runtime.

Step 1.

Work out what size your bar has to be. Note that the size is the value represented by the full bar. It is not related to the bar's actual dimensions.

How you do this will depend on what you are using the bar for. For example, if you want to represent a percentage the size would be 100. If you wanted to use a progress bar while printing the size might be the number of lines to be printed.

Step 2.

Create a suitable progress bar using CreateProgBarA() or CreateProgBar().

Step 3.

Every time your program completes one unit of whatever it is doing, you should use UpdateProgBar() to display the new position. For the examples above this would mean updating the bar after one percent, or after printing each line.

If the value of the bar is decreasing and you have the text function activated, it will probably be necessary to call ClearText() immediately before updating the bar. This is not necessary if the text is centred within the bar, or if the length of the text never decreases.

Step 4.

When you have finished you should delete the progress bar and release the memory used by calling FreeProgBar().

Step 5.

When your window needs refreshed you should call RefreshProgBar().

1.4 C Installation

The C system is comprised of two files, "prog_bar.h" and "prog_bar.lib".

"prog_bar.h" should be copied to anywhere on your compiler's include file search path. Just make sure you know where you put it because you will need to #include it any time you want to use prog_bar.

For Dice I created a directory called "DINCLUDE:library" and copied it to there.

"prog_bar.lib" should be copied to anywhere on your compiler's library search path. Alternatively you could put it anywhere and then specify its full pathname when linking.

In Dice I copied it to "DCC:dlib/"

Using Prog_Bar in C

1.5 Using Prog_Bar from C

When writing your source you will need to include the file "prog_bar.h" before you use any of the functions, otherwise the compiler will complain about undefined structures, etc.

In Dice I use the line "#include <library/prog_bar.h>".

When compiling you will then need to tell the compiler to include the code for the prog_bar functions. This is done by linking your code to "prog_bar.lib". In some compilers, or with large multi-module programs, it will be necessary to compile your source to *.o files, and then link these and prog_bar.lib together to produce the executable. If your program is a single source file then you might be able to compile it and link to prog_bar.lib in one instruction. Your compiler documentation will contain more detailed information about this.

To compile the demo with Dice I was able to compile the source and perform the link in one operation. The command line to do this was "dcc -2.0 -// -Tt: -o Demo Demo.c prog_bar.lib".

Calling the prog_bar functions uses exactly the same notation as any other C function, e.g. "ClearProgBar(PBar);".

1.6 Assembler Installation

The Assembler system is comprised of two files, "prog_bar.i" and "prog_bar.lib".

"prog_bar.i" should be copied to anywhere on your assembler's include file search path. Just make sure you know where you put it because you will need to include it any time you want to use prog_bar.

For Devpac I copied it to the "Devpac/include/libraries" directory.

"prog_bar.lib" should be copied to anywhere on your assembler's library search path. Alternatively you could put it anywhere and then specify its full pathname when linking.

In Devpac I copied it to "Devpac/lib/"

Using Prog_Bar in Assembler

1.7 Using Prog_Bar from Assembler

When writing your source you will need to include the file "prog_bar.i" before you use any of the functions, otherwise the assembler will complain about undefined symbols, etc.

In Devpac I use the line "include libraries/prog_bar.i".

To produce your final executable you will need to assemble your source as a linkable file. The resulting *.o file will then need to be linked with Prog_Bar.

To create the demo program with Devpac I created my source in a directory called "Devpac/Progs". From here the command to link the executable was "BLINK demo.o TO demo LIB /lib/prog_bar.lib".

Your assembler documentation will contain more detailed information about assembling and linking programs.

Calling the prog_bar functions is different from all the operating system functions. The reason is because the library has been written to be used from C with the minimum of fuss, and as such each function expects its parameters to be placed on the stack, NOT in the registers.

Before calling a prog_bar function you must place its parameters on the stack in the reverse order, i.e. right to left. Each value you place on the stack MUST be a longword value, even if the parameter type is a word or byte value.

To actually call the function you just use a JSR instruction. The target of the jump will be the function name preceded by an underscore, e.g. to call CreateProgBar() you use the instruction "JSR _CreateProgBar".

When the function returns you are responsible for removing its parameters from the stack. Any return values will be in the register D0.

An example from the demo program is this small piece of code which calls CreateProgBarA() and stores the result in PBar_ptr.

```
   bar = CreateProgBarA ( Wnd, Left, Top, Width, Height, Size, taglist )


         *  create the progress bar
            pea.l    pbar_tags         ;  taglist
            move.l   max_size,-(sp)    ;  Size
            move.l   #30,-(sp)         ;  Height
            move.l   #300,-(sp)        ;  Width
            move.l   #65,-(sp)         ;  Top
            move.l   #100,-(sp)        ;  Left
            move.l   windowptr,-(sp)   ;  Wnd
            jsr      _CreateProgBarA   ;  Call CreateProgBarA()
            move.l   d0,PBar_ptr       ;  bar
            lea      28(sp),sp         ;  remove parameters from stack
```

```
                              ;  7 parameters * 4 bytes each = 28

pbar_tags
   dc.l     PB_BorderType,PBBT_RIDGE
   dc.l     PB_TextMode,PBTM_PERCENT
   dc.l     PB_TextPosition,PBTP_CENTRE
   dc.l     TAG_DONE

windowptr   dc.l     0
max_size    dc.l     200
PBar_ptr    dc.l     0
```

1.8 Modula-2 Installation

The Modula-2 system is comprised of three files, "prog_bar.def", "prog_bar.sym" and "prog_bar.lib".

"prog_bar.def" should be copied to anywhere on your compiler's include file search path. Just make sure you know where you put it because you will need to import it any time you want to use prog_bar.

For Turbo Modula-2 I copied it into the "Modula_2/amiga/" directory.

"prog_bar.sym" should be copied to your compiler's symbol file directory. If the "prog_bar.sym" file is not of the correct format for your compiler you will need to recreate it. This is done by compiling "prog_bar.def".

For Turbo Modula-2 I copied it into the "Modula_2/sym/" directory.

"prog_bar.lib" should be copied to anywhere on your compiler's library search path. Alternatively you could put it anywhere and then specify its full pathname when linking.

For Turbo Modula-2 I copied it into the "Modula_2/lib/" directory.

N.B. I have my copy of Turbo Modula-2 installed in the "Modula_2" directory.

Using Prog_Bar in Modula-2

1.9 Using Prog_Bar from Modula-2

When writing your source you will need to import the file "prog_bar.def" before you use any of the functions, otherwise the compiler will complain about undefined structures, etc.

In Turbo Modula-2 I use the line "IMPORT P := prog_bar".

When compiling you will then need to tell the compiler to include the code for the prog_bar functions. This is done by linking your code to "prog_bar.lib". Your compiler documentation will contain more detailed information about this.

To compile the demo with Turbo Modula-2 I just added "prog_bar.lib" to the end of my compile command. The resulting command line to achieve this was "m2b Demo prog_bar.lib".

Calling the prog_bar functions uses exactly the same notation as any other Modula-2 function, e.g. "P.ClearProgBar(PBar);".

1.10 Demonstration Programs

The Library can currently be used from Modula-2, C and Assembly Language. I have included small example programs in each language so that you can test the library with your compiler before writing anything of your own.

All Programmers

The demo program written in C is by far the largest of the demo programs. C is my preferred language so I used this program as a complete demonstration of everything the library can do. As such it is probably worth looking at even if you do not normally program in C.

Demo Program   Source

Assembly Programmers

This is a small program which creates a progress bar and then slowly fills it. When full, the bar is reset and the process repeats. Just click on the Window's Close gadget to end the program.

Demo Program   Source

Modula-2 Programmers

This is also a small program. It fills a progress bar in steps of 1, then changes the direction, border and text mode of the bar, and then fills it again in steps of 5. You just have to wait until it has finished.

Demo Program   Source

1.11 Distribution

Prog_Bar is freely distributable, providing that no commercial gain is made from its distribution, and no modification is made to the original archive.

Anyone wishing to include Prog_Bar on a magazine coverdisk or other similar collection, or use it in any application, commercial or otherwise, have my full permission. All I ask in return is to be acknowledged somewhere in the documentation and to be told about it (preferably by e-mail).

1.12 Acknowledgements

Acknowledgements

The Demonstration program was also compiled using DICE v2.07.56 R and its interface was designed using GadToolsBox v37.300. Thanks to Matthew Dillon and Jan van den Baard for these excellant programs.

Many thanks also to Roberto Bizzarri for his assistance in testing Prog_Bar v1.1 with SAS/C.

1.13 How to contact me

If you have any suggestions for improving Prog_Bar, bugs to report or queries about the program, please send them to me at one of the addresses below.

```
        E-Mail :  asavage@bitsmart.com         (preferred option)
                  asavage@enterprise.net

        Post   :  Mr. Allan Savage
                  2 Navar Drive
                  Gransha Road
                  Bangor
                  N. Ireland
                  BT19 7SW
```

1.14 Program History

Program History

```
V1.0   13/12/96   - Initial Release.  (Only worked with Dice)
```

```
V1.1   11/01/96   - Rewritten  in  Assembly  language.   Now  works with all
                    compilers (hopefully).
                  - Added definiton files for Assembler and Modula-2.
                  - Added Demo programs for Assembler and Modula-2.
```