SerLib.Library

Garry Glendown

Version 3.0 Feb. 16th, 1991

Abstract

The Amiga shared library 'SerLib.Library' allows easy access to the serial device by doing the device-specific accesses itself and providing an interface for the programmer that is easy to use.

Preface

Today there are many people out in the world programming this nice machine, and many succeed. But even of those who do succeed in writing programs that keep the rules, probably everybody has problems with a couple pieces of the system software. There have been many good approaches to solve those problems, and there are quite a few good pieces of code you can use.

But until today, there hasn't been any help for people that want easy access to the serial port. So what do they do ? They take a language that includes that easy access, and program their (mostly fine) ideas in it. But as many use the thing called "GFA Basic" and "GFA Basic Compiler", it tends to go from bad to worse. Instead of having maybe 5 little bugs in their program, they get those 5 little bugs plus approximately 10 big ones and a lot of unwanted "features" that the compiler generates.

Some of the biggest deficiencies of GFA-programs are:
- no access using baud rates larger than 19200 Baud (HST-Modems can go up to 38400 Baud)
- carrier detect via accessing the hardware (which breaks when using multiple serial ports or internal modems)
- current versions of the interpreter and compiler are incompatible to OS 2.0 and Amiga 3000/3000T

Though the first two problems can be fixed by a simple workaround, only few people know about it. But what should be done ? The best thing would be to allow easy access to the serial.device, so that people can use their C-Compiler again.

As a result of my work, here's "`serlib.library`", a library that includes all routines needed for easy `serial.device`-access.

Garry Glendown, Bad Hersfeld, Feb. 16th, 91

Table of Contents

1. Installing

The original `SerLib.Library`-Archive includes the following files:

File	Function
`Antrag.lzh`	An example program with some nice routines
`LibSrcAztec.lzh`	Sources to create library stubs
`LibSrcAztecL.lzh`	for Aztec C
`makefile`	Makefile to compile `mini.c` and `sert.c`
`mini`	A minimal terminal-program
`mini.c`	with source
`OberonSerLib.lzh`	Oberon linker lib and examples
`serlib.autodocs`	Autdocs-file of serlib.library's functions
`serlib.doc`	The doc-file in pure ASCII
`serlib.dvi`	This document as TEX DVI-file
`serlib.fd`	FD-file for the library functions
`serlib.h`	C-header-file.
`serlib.i`	Same, for assembler
`serlib.library`	The library
`SerLib.Readme`	Short information file
`serlibbase.h`	Internal definitions file for the library
`serlibbase.i`	Same, assembler format
`serlib_lvos.asm`	Library-offsets for assembler
`serlib_SAS.lib`	Linker-stubs for SAS C 5.10a
`sert`	Test-program for the library
`sert.c`	including source

In order to install the library and additional files, copy the file "`serlib.library`" to your `LIBS:`-directory. Users of MANX' Aztec C will have to create the Library-Stubs from the sources included in the two `.lzh`-Files. (I don't use Aztec anymore, and didn't want to re-install it on my HD, sorry). Anyway, it shouldn't be a problem.

If you use Lattice/SAS C, you have to decide if you want to use pragmas to call the library functions, or want to use the .lib-File. If you use pragmas, you can forget about "`serlib_SAS.lib`" (created with V5.10a). Now, copy the `.h`- and `.i`-Files to your include-directory. Copy the rest of the files to any place you wish. That's it already.

If you should have a modem attached to your computer, just start 'sert' or 'mini'. The first is a small test-program that asks for some modem-help (`AT$`, at least it does on my HST...), the second is a minimal terminal-program that lets you input a line, sends the line to the modem, prints the modems answer, etc. Nothing great, just a test program.

Take a look at the archive 'antrag.lzh', it includes an example program using the the library. This program also has some nice routines built in for string handling. Please note that handling has changed from the first release version due to additional routines for asynchronous transfers. In order to add these functions, programs using the function WaitSer() will most likely break using version 3.0. Sorry... (take a look at the source of Antrag to see how to fix it, just a minor workaround...)

As I released a couple different versions and revisions in a rather short time (between Feb. 6th and Feb. 16th), here's some comments on those versions:

Version 1.0 not released to the public, just a few testers.

Version 1.1 first release version, no known bugs; only had synchronous transfer.

Version 2.0 new version with asynchronous support. Due to large changes and too little testing time, this version had some terrible bugs.

Version 2.1 most of the bigger bugs removed. Has some troubles when Wait()ing.

Version 2.2 removed the `WaitSer()`-bug, `OR`ed in wrong direction. This was hoped to be the 'final' version.

Version 3.0 as I still have some strange behavior with the `WaitSer()`- Function, some more testing revealed a `MOVEM`, that did too much and restored an old value in `D0` ... removed, hopefully this is the final version... (until some new ideas come around the corner...)

The next version will probably include easy access to XPR-libraries, plus some functions still missing now (any wishes ?). Also, I plan to add an extra chapter with some examples on how to use the library...

2. Using other languages

Additionally to the original implementation, which includes code for calling SerLib.Library from either C or Assembler, I've included some code from Frank Schummertz, that demonstrates the usage of the library from Oberon. The archive 'OberonSerLib.lzh' includes the files as I received them from Frank.

To use any of SerLib.Libraries routines, just link the file 'serlib.objs' to the other objects. Check the example program for some hints on how to use the library from Oberon. There shouldn't be any problems, though.

3. The library functions

Following are the document pages of the library's functions, as are found in the autodocs-file.

For Online-help, please take apart the autodoc-file supplied in the archive.

3.1. AbortIOSer

NAME

AbortIOSer – abort a previous `SendSer()` or `RecvSer()`

SYNOPSIS

AbortIOSer(serlibdata, which);
A0 D0:16

AbortIOSer(struct SerLibdata *, ULONG);

FUNCTION

AbortIOSer will perform an `AbortIO()` on a previous `SendSer()`- or `RecvSer()`-Command.

By setting the appropriate bit in '`which`', either the `SendSer()` or `RecvSer()` is aborted.

INPUTS

`serlibdata` – Pointer returned by OpenSerial
`which` – Either '`ABORT_SEND`' or '`ABORT_RECV`'

RESULT

None

SEE ALSO

`OpenSerial()`, `SendSer()`, `RecvSer()`

3.2. ChangeData

NAME

ChangeData – change the serial setup

SYNOPSIS

```
ChangeData(serlibdata, baud, bpc ,  stop , serFlags);
                  A0        D0  D1:16 D2:16    D3:16

ChangeData(struct SerLibData *, ULONG, UWORD, UWORD,
           UWORD);
```

FUNCTION

Changes the setup of the serial port.

INPUTS

`serlibdata` – Pointer returned by `OpenSerial()`
`baud` – the baud rate at which the port will be operated.
`bpc` – Bits per character (normally '8').
`stop` – number of stop-bits (normally '1', may be '0' through '2')
`serFlags` – Flags for the device. Entered directly in `io_SerFlags` of the `IOExtSer`-structure. See '`devices/serial.h`' for values to enter.

RESULT

None

NOTES

No sanity-check of the values given to ChangeDate. So you can use a baud rate of 27182 Baud, 20 Bits per character and 42 stop bits. There's no telling how serial.device will react to such a change-request...

SEE ALSO

`OpenSerial()`, `devices/serial.h`

3.3. CheckCD

NAME

CheckCD – check if the modem has a Carrier detected

SYNOPSIS

```
carrier = CheckCD(serlibdata)
  D0                      A0

ULONG CheckCD(struct SerLibData *);
```

FUNCTION

Checks if a carrier is detected.

INPUTS

`serlibdata` – pointer returned by `OpenSerial()`

RESULT

`carrier` – 0 if no carrier, 1 if carrier is detected

NOTES

If you need both carrier detect and number of bytes, a call to `GetStatus()` and checking in the `SerStatus`-structure will be faster as both `SerBuffer()` and `CheckCD()` call `GetStatus()` internally.

SEE ALSO

`OpenSerial()`, `GetStatus()`, `serlib.h`, `devices/serial.h`

3.4. CheckIOSer

NAME

CheckIOSer – check if a previous RecvSer is finished

SYNOPSIS

return = CheckIOSer(serlibdata);
D0 A0

BOOL CheckIOSer(struct SerLibdata *);

FUNCTION

CheckIOSer will perform a `CheckIO()` on a previous `RecvSer()`-Command.

INPUTS

`serlibdata` – pointer returned by OpenSerial

RESULT

`return` – Return-value of `CheckIO()`-call.

SEE ALSO

`OpenSerial()`, `SendSer()`, `RecvSer()`, `AbortIOSer()`

3.5. ClearSer

NAME

ClearSer – Clear the serial buffer

SYNOPSIS

ClearSer(serlibdata)
A0

ClearSer(struct SerLibData *);

FUNCTION

ClearSer empties out the serial receive buffer by doing a `CMD_CLEAR`.

INPUTS

`serlibdata` – Pointer returned by OpenSerial()

RESULT

None

SEE ALSO

`OpenSerial()`

3.6. CloseSerial

NAME

CloseSerial – Close the serial.device

SYNOPSIS

CloseSerial(serlibdata)
A0

CloseSerial(struct SerLibData *);

FUNCTION

After being finished, use this to close the device and free all memory allocated.

INPUTS

`serlibdata` – Pointer returned by OpenSerial.

RESULT

None

NOTES

No sanity check, better not get the idea to give this one a Null- pointer! If the `SerLibData`-structure should be trashed, it could lead to minor problems, too!

SEE ALSO

`OpenSerial()`, `serlib.h`

3.7. GetStatus

NAME

GetStatus – get the Status data from the serial port

SYNOPSIS

GetStatus(serlibdata, serstatus)
A0 A1

GetStatus(struct SerLibData *, struct SerStatus *);

FUNCTION

Gets the status data from the serial port. Additionally, the number of bytes in the system's serial buffer are also returned in the SerStatus-structure.

INPUTS

`serlibdata` – pointer returned by `OpenSerial()`
`serstatus` – address of a `SerStatus`-structure

RESULT

`serstatus` – filled structure

NOTES

As usual, no sanity check.

SEE ALSO

`OpenSerial()`, `serlib.h`, `devices/serial.h`

3.8. OpenSerial

NAME

OpenSerial – Open the serial device for access through serlib.library

SYNOPSIS

```
serlibdata = OpenSerial(device, unit , baud , bpc ,  stop , serFlags)
   D0                          A0   D0:16 D1:16 D2:16 D3:16   D4:16

struct SerLibData *OpenSerial(STRPTR, UWORD, ULONG,
                              UWORD, UWORD, UWORD);
```

FUNCTION

This routine attempts to open the serial.device for usage with serlib.library.

INPUTS

`device` – the device to be opened. Normally "serial.device". Change when using internal modems or multiple line serial cards.

`unit` – the unit to open. Normally '0', other when using a multiple line serial card.

`baud` – the baud rate at which the port will be opened.

`bpc` – Bits per character (normally '8').

`stop` – number of stop-bits (normally '1', may be '0' through '2')

`serFlags` – Flags for opening the device. Entered directly in `io_SerFlags` of the `IOExtSer`-structure. See '`devices/serial.h`' for values to enter.

RESULT

`serlibdata` – pointer to the `SerLibData`-structure containing the paramteres for working with the port. This pointer will be needed for every function call to `serlib.library`.

NOTES

No sanity-check of the values given to OpenSerial. So you can use a baud rate of 27182 Baud, 20 Bits per character and 42 stop bits. There's no telling how serial.device will react to such an open-request...

BUGS

Currently, the baud rate itself isn't set, as the OpenDevice-call doesn't set it. Some later release will probably correct this. Until then, use the ChangeData routine to actually set the baud rate needed.

SEE ALSO

`CloseSerial()`, `serlib.h`, `serlibbase.h`, `devices/serial.h`

3.9. ReadSer

NAME

ReadSer – read bytes from the serial port

SYNOPSIS

```
number = ReadSer(serlibdata, buffer, max)
  D0                     A0        A1     D0

ULONG ReadSer(struct SerLibdata *, STRPTR, ULONG);
```

FUNCTION

Upon calling, ReadSer first checks how many bytes there are still left to be read. If 0, it will return right away. Otherwise, ReadSer will send a `CMD_READ` to the serial device with `IO_SIZE` set to the number of bytes unread (or 'max' if max is smaller) If you get a return value that is equal to 'max', you should process the data received and then go back and get the rest.

INPUTS

`serlibdata` – pointer returned by OpenSerial
`buffer` – pointer to a block of memory to be filled with data from the serial port
`max` – size of the buffer. A maximum of 'max' bytes will be read.

RESULT

`len` – number of bytes read from the port.

NOTES

You might get in trouble if you set 'max' to zero..

SEE ALSO

`OpenSerial()`

3.10. RecvSer

NAME

RecvSer – read bytes from the serial port

SYNOPSIS

RecvSer(serlibdata, buffer, num)
A0 A1 D0

RecvSer(struct SerLibdata *, STRPTR, ULONG);

FUNCTION

RecvSer will initiate a `SendIO`-call to the serial port, trying to receive num bytes.

INPUTS

`serlibdata` – pointer returned by OpenSerial
`buffer` – pointer to a block of memory to be filled with data from the serial port
`max` – size of the buffer. '`num`' bytes will be read if the `IORequest` isn't aborted.

RESULT

None

NOTES

You might get in trouble if you set 'max' to zero..

SEE ALSO

`OpenSerial()`, `ReadSer()`

3.11. SendSer

NAME

SendSer – Send a string to the serial port without waiting

SYNOPSIS

```
SendSer(serlibdata, buffer, len)
            A0       A1    D0

SendSer(struct SerLibData *, STRPTR, ULONG);
```

FUNCTION

SendSer takes a buffer pointer and sends the specified number of bytes to the serial port using the `SendIO` function, so it doesn't wait for completion of the command.

INPUTS

`serlibdata` – Pointer returned by `OpenSerial()`
`buffer` – Pointer to a memory block containing the data for the serial port
`len` – number of bytes to be sent

RESULT

None

NOTES

No sanity check, as usual.

SEE ALSO

`OpenSerial()`, `WriteSer()`

3.12. SerBuffer

NAME

SerBuffer – get the number of bytes still in the buffer

SYNOPSIS

result = SerBuffer(serlibdata)
D0 A0

ULONG SerBuffer(struct SerLibData *)

FUNCTION

Returns the number of bytes still in the system's serial buffer.

INPUTS

`serlibdata` – pointer returned by `OpenSerial()`

RESULT

`result` – number of bytes in the buffer

NOTES

If you need both number of bytes and the carrier detect, a call to `GetStatus()` and checking in the `SerStatus`-structure will be faster as both `SerBuffer()` and `CheckCD()` call `GetStatus()` internally.

BUGS

Would be hard for this one to have any.

SEE ALSO

`OpenSerial()`, `GetStatus()`, `serlib.h`

3.13. WaitSer

NAME

WaitSer – wait for data from the serial port

SYNOPSIS

```
signal = WaitSer(serlibdata, mask)
  D0                     A0        D0

ULONG WaitSer(struct SerLibData *, ULONG);
```

FUNCTION

Waits for a signal bit set. Waits for the one of the serial port, plus the ones you specify (timer, break, etc.).

INPUTS

`serlibdata` – Pointer returned by `OpenSerial()`
`mask` – mask of signals waited for

RESULT

`signal` – set of signals that were active

NOTES

This function has changed slightly from release 1.1. It doesn't do a dummy-Read to the serial port any more, so in order to use it, add an extra 'RecvSer()' before calling WaitSer().

3.14. WriteSer

NAME

WriteSer – Send a string to the serial port

SYNOPSIS

WriteSer(serlibdata, buffer, len)
A0 A1 D0

WriteSer(struct SerLibData *, STRPTR, ULONG);

FUNCTION

WriteSer takes a buffer pointer and send the specified number of bytes to the serial port.

INPUTS

`serlibdata` – Pointer returned by `OpenSerial()`
`buffer` – Pointer to a memory block containing the data for the serial port
`len` – number of bytes to be sent

RESULT

None

NOTES

No sanity check, as usual.

SEE ALSO

`OpenSerial()`

4. Using serlib.library in programs

`SerLib.Library` is Shareware. You may copy it on non-profit-basis as you wish. Anyway, I do not consider the german PD-'Distributors' working on 'non-profit' basis (how can you make a living when working on non-profit basis?), so before selling SerLib on any disks, german dealers have to get written permission from me.

After using `SerLib.Library` for a while, you have to register by sending me the shareware fee of DM 20.– / US$ 15. This will get you a free copy of the next version (including other programs finished at that time). (Places other than Germany or the USA please add another DM/$ 5.– for additional shipping costs).

When using SerLib.Library in a PD or Shareware-program, include both the library and the file '`SerLibLibrary.Readme`' with the program. Anyway, if you do, I would appreciate a copy of the program (either via email or on disk). Additionially, you have to register as I presume you find SerLib.Library very useful!

Commercial programs using it need a special registration. Contact me for further details.

Remember, supporting shareware authors helps to keep up the standards of free and affordable software!

If you have problems or bug reports, need help or just want to send me the shareware fee, use one of the addresses below:

from any country:
Garry Glendown
Güldene Kammer 35
W-6430 Bad Hersfeld
Germany

from the US & Canada:
Garry Glendown
Box R
APO NY 09141

Telephone: 06621-77923 (HST/V32bis), give me a yell for voice call...

eMail may be sent to any of the following addresses:
..cbmvax!cbmger!inside!garry
Garry@DGIHRZ01.BITNET
Garry@fulmin.zer.sub.org
Garry@INSIDER.zer
Garry Glendown @ 2:243/43.999 (Fido)

5. Index