SerialPort Scripting Addition

AppleScript Commands for the Serial Port

Version 1.1

Blake Ward

Blake.Ward@alumni.cs.cmu.edu
or
bward@kagi.com

1. Introduction

The SerialPort Scripting Addition is a collection of commands that make it possible to open the serial port from within AppleScript and send and receive data. This collection is shareware, if you decide to continue using it you need to submit the $10 registration fee. See the section **How To Register** for more information. Now you can easily register on-line!

With these scripting commands you can open the serial port with any combination of baud rate, data bits, stop bits, etc. You can then read or write bytes, short integers (2 bytes), long integers, strings or data. (The SerialPort Scripting Addition uses the Macintosh's Communications Toolbox and the Serial Port tool, so it is capable of any settings supported by the Serial Port tool.)

2. AppleScript

If you're new to AppleScript, don't be intimidated. It's easy to learn and there are lots of resources available to help get you started. Along with this scripting addition, AppleScript is an ideal environment for creating applications that will control other devices via the serial port. If you haven't used AppleScript before, you can probably learn enough just by looking at the sample scripts included with the SerialPort Scripting Addition, but the best place to start is a good AppleScript book from your local bookstore. Many of these books will take you from never having seen a script to writing fairly sophisticated scripts in a few easy lessons. Just take your time and start with simple scripts and you'll be sending and receiving data from your Macintosh in no time...

Finding AppleScript and the Script Editor

AppleScript is included with every Macintosh running System 7 or later. The Script Editor is the application that you'll likely use to edit scripts, including the samples ones that are enclosed. There are also a few of commercial applications available that work with AppleScript, including FaceSpan which also lets you build fancy graphical interfaces to control your scripts. The Script Editor included for free with System 7.5 or later (it's automatically installed in the "Apple Extras" folder). You can purchase it separately as part of an AppleScript programming kit from Apple if you're using an earlier version of the MacOS (although I would recommend just upgrading to System 7.5). The Script Editor is also included with several demo applications that are available from Info-Mac on the Internet (a list of these applications is part of the FAQ mentioned below).

Finding AppleScript and the Script Editor

Once you've got a copy of AppleScript and the Script Editor and an AppleScript book, you should be set. However, there are lots of other information sources if you get interested in writing more sophisticated scripts. You can find additional information about AppleScript, archives of other useful Scripting Additions, lists of fully scriptable applications, etc. on the world wide web at the following location:

`http://www.scriptweb.com/`

If you have questions about AppleScript, this site at Apple is a great starting point:

`http://www.applescript.apple.com/`

Note: If you prefer to use HyperCard, these SerialPort scripting commands can also be used from HyperCard.

These scripting commands also work from Frontier provided you call them from AppleScript code within Frontier.

These scripting commands can also been used from FileMaker Pro.

4. The SerialPort Scripting Commands

This section describes all of the SerialPort commands in detail, including any arguments they take and the values they return.

To install this Scripting Addition, all you have to do is put the "SerialPort Scripting Commands" file into the folder:

```
System Folder:
    Extensions:
        Scripting Additions:
```

There's no need to restart. Then just open some of the sample scripts in the Script Editor and give them a try.

All of these commands check carefully for error conditions and will signal an error if anything goes wrong (e.g. the device on the other end of the serial port isn't responding, the Serial Tool isn't installed, the script runs low on memory, etc.). Once you have successfully opened the serial port, it is essential that you catch any errors with your script and close the port again before continuing. See the **Error Handling** section for an example of how to use the **try** and **on error** AppleScript functionality.

If you don't have the **Serial Tool** installed you'll get an error message the first time you try to open the serial port. You can get a copy from the Apple server "ftp.support.apple.com" on the internet in the following folder:

```
http://swupdates.info.apple.com/cgi-
bin/pointer.pl?Apple.Support.Area/Apple.Software.Updates/US/Macintosh/Networking-
Communications/Comm_Toolbox/Serial_Tool_1.0.2.sea.hqx
```

After downloading the serial tool, place it in the extensions folder and **make sure that it's named "Serial Tool"**.

serial port list

Get a list of the names of all of the existing serial ports.

serial port list

Result: list of string

When you open a serial port for reading/writing you will need to provide the name of the port to be opened. Most desktop MacOS computers have two serial ports, named "Modem Port" and "Printer Port" so you can simply use the name of the port you've plugged your device into. However, some computers will have a single port with a different name (e.g. Powerbooks have a port called "Internal Modem") and if you've added additional hardware like a PC Card with a serial port, it might have a different name. This command scans the available hardware and returns a list of the names of all of the serial ports that it can find. If you're writing a script that includes a user interface, then you might want to take this list and present it as a set of choices for the end user. Even if your script has no user interface, you might still want to get a list of available ports and make sure that the one your script plans to use is included in the list.

Important: If you're using the SerialPort scripting addition on a localized version of the MacOS (e.g. a German, Swedish, etc. version of the OS), then it is especially helpful to use this command to find out exactly what the ports are called in the localized language. For instance, on a German version of the MacOS, the modem and printer ports are called "Modemanschluß" and "Druckeranschluß".

open serial port

Open a serial port with the given settings.

open serial port

string

Name of the port that should be opened. Typical desktop Macintoshes have two ports called "Modem Port" and "Printer Port". Most Powerbooks will have a single port called "Internal Modem". You can also use the 'serial port list' command to get the currently valid port names for this computer. The default is "Modem Port".

[**baud rate** integer]

Baud rate for the connection (default is 9600). This scripting command doesn't enforce any particular baud rate setting, but most devices require a specific baud rate (e.g. 123 is probably not a valid baud rate for any device).

[**data bits** integer]

number of data bits for the connection (5, 6, 7, or 8, default is 8).

[**handshake** none/XOnOff/DTRandCTS/DTR only/CTS only]

Type of handshaking for the connection (defaults to 'none').

[**hold connection** boolean]

If true, then don't drop DTR when closing the connection (defaults to false).

[**parity** none/odd/even]

Type of parity on the connection (defaults to 'none').

[**stop bits** one/half/two]

Number of stop bits for the connection (default is 'one').

[**remind disconnect** boolean]

When true and 'hold connection' is also true, then the Serial Tool will remind the user that DTR is being held high (defaults to false).

[**input buffer** integer]

Size of input buffer to be allocated (defaults to 1024).

[**output buffer** integer]

Size of the output buffer to be allocated (defaults to 1024).

Result: string *-- returns a special reference value for the open port that must be used with each of the other serial port commands.*

Before you can do anything else, you'll need to open the serial port using this command. Except for the buffer sizes, all of the settings are passed directly through to the Communication Toolbox Serial Tool and aren't interpreted by the command. The input and output buffer sizes determine how much space the Serial Tool has to store data that is being sent and received via the serial port. If you typically send large amounts of data back and forth, you'll want to increase these values. On the other hand, if you're communicating with a device using only a few bytes at a time, you might want to reduce the size of these buffers to save memory (the memory is allocated from overall system memory and is returned when you close the port). Note that when data is received on the serial port it must be stored in the input buffer until you ask for it by calling 'serial port read'. If you won't be calling the read command frequently, make sure that the input buffer is large enough to hold all of the pending data until you're ready for it (when the buffer is full, any excess data is discarded by the Serial Tool).

open serial port returns a special string that refers to the open port and must be passed as a parameter for all of the other serial port scripting commands. This string shouldn't be modified.

Important: You must close the serial port when you're finished with it or it will be unavailable to other scripts or applications and the memory allocated for the input and output buffers won't be returned. Since an error could cause your script to stop running before closing the port, see the section on Error Handling for an example of how to catch errors and close the port before proceeding.

To determine if a particular serial port is available or currently in use by another application, you can try to open it and watch for an -6114 error, which indicates that the port couldn't be opened. For instance:

```
set sPort to ""
try
   set sPort to open serial port "Modem Port" baud rate 600
                        data bits 8 parity none stop bits 1

   -- Do some reading and writing...


   close serial port sPort
   set sPort to ""

   on error theErr number errNum
      if sPort is not "" then
         close serial port sPort
      end if
      if errNum is -6114 then
         -- We couldn't open the port and this is almost always
         -- because it's already in use, so report this to the user
         display dialog "Serial port is already in use." buttons "OK"
      end if
   end try
```

close serial port

Close the serial port that was opened earlier.

close serial port

  string

    The special reference returned by a previous 'open serial port' command. After this command, the reference string is no longer valid and can be discarded.

  [**with/without waiting**]

    Should the command wait for any data still being written before closing the port. Defaults to closing the port immediately and discarding any pending unwritten bytes.

This command closes a previously opened serial port.

Important: You must close the serial port when you're finished with it or it will be unavailable to other scripts or applications and the memory allocated for the input and output buffers won't be returned. Since an error could easily cause your script to stop running before closing the port, see the section on Error Handling for an example of how to catch errors and close the port before proceeding. **It is very important that you always catch errors and close the serial port before proceeding!**

serial port read

<u>Read some data from the already opened serial port.</u>

serial port read

  string

  Special connection reference returned by 'serial port open'

  [**for** integer]

  Number of bytes to try to read from the connection. If not specified, then return as many bytes as are currently available or required for the data type being read.

  [**timeout** integer]

  Timeout in ticks (60 ticks = 1 second). If timeout isn't specified or is zero, then read doesn't wait and only returns those bytes that are immediately available (returning an error if there aren't enough bytes for the requested data type).

  [**as** type class]

  Type the data should be returned as (currently only 'integer', 'string' , 'real', 'short', 'boolean' and 'data' are supported). Defaults to 'string' .

Result: anything *-- returns a value of the requested type, read from the open serial port connection.*

serial port read reads some data from a currently open serial port. The number of bytes of data to read is determined as follows:

- if you've requested a particular type of data, then the number of bytes required for that data type are read (4 bytes for integer, 8 bytes for real, 2 bytes for short, 1 byte for boolean). If a type other than 'string' or 'data' is specified, then the **for** parameter is ignored.
- if the **for** parameter is provided, then the number of bytes specified is returned.
- if the **for** parameter is absent, and the data type is 'string' or 'data', then all of the currently available bytes are returned.

If serial port read can't get enough bytes for a data type that requires a specific number of bytes (e.g. integer, real or short) before the timeout, then it will return any bytes it was able to read as type 'data'.

If you want to read all of the bytes available within a certain period of time, specify a type of 'string' or 'data' and a large value for the number of bytes to read and a timeout. The command will return after the timeout with whatever it could read. Since the command has to temporarily set aside memory to hold all of the bytes specified in the 'for' parameter, don't make this value too big (e.g. 10,000,000) unless you really expect to get that many bytes.

Since AppleScript doesn't have a short data type, the 2 byte value read you specify 'as short' are automatically converted to an AppleScript integer. For the boolean type, if the byte read is zero, then a 'false' value is returned, otherwise a 'true' value is returned.

If you expect several bytes of data from some device, you can easily read the data as 'string' and then break the string apart into bytes (see the StringToBytes() function in the sample scripts) and then manipulate those bytes using the various commands in the **Bytes & Bits Scripting Addition**.

If you don't know how much data to expect over the serial port, you can set up a loop in your script and repeatedly call **serial port read** with no timeout value and a data type of 'string'. It will return a string with any available bytes or an empty string if none are available.

serial port write

Write some data to the already open serial port

serial port write

anything

Value to write (currently only 'string', 'integer', 'data' and 'boolean' classes are supported)

to string

Special connection reference returned by 'serial port open'

[**for** integer]

Number of bytes to write. The value being written must be at least this long. Defaults to the size of the item being written.

[**timeout** integer]

Timeout in ticks (60 ticks = 1 second). If timeout isn't specified or is zero, then write will wait forever for the bytes to be written successfully.

This command writes the given value to the serial port. If the **for** parameter is specified, then the value must be at least that large. You can use this command to write short or byte values by providing an AppleScript integer as the value and then specifying 2 or 1 for the **for** parameter. If you don't specify a length with the **for** parameter, then the **serial port write** command just asks AppleScript how big the value is and then writes that number of bytes.

If the **timeout** parameter is specified, then this command will only wait that amount of time for the write to succeed. If no **timeout** is provided, then this command will wait as long as necessary for the bytes to be written to the serial port. Note that normally bytes are written to the output buffer, the command returns, and then the bytes are actually transfered from the buffer to the serial port as quickly as possible.

serial port available bytes

Returns the number of bytes currently available for reading on the port.

serial port available bytes

    string

        Special connection reference returned by 'serial port open'

Result: integer

5. Error Handling

All of the SerialPort scripting commands do extensive error checking and can signal a variety of errors if a problem occurs. Normally, an error would be reported to the user in an error dialog. However, since we need to be careful to leave the port in a closed state, you should always set your script up to catch the error and handle it directly. For instance:

```
on serialPortTest()
   -- We have to make sure that we close the port if anything goes wrong,
   -- so always wrap the open/close in a try block!!
   set sPort to ""
   try
      set sPort to open serial port modem port baud rate 600 data bits 8
parity none stop bits 1

      -- Read all of the data that shows up in the next 20 seconds
      set someInput to serial port read sPort as string timeout 1200

      close serial port sPort with waiting

      -- We've successfully closed it, so get rid of the reference
      set sPort to ""

   on error theErr number errNum
      if sPort is not "" then
         -- If the port is still open, close it before proceeding
         close serial port sPort
         set sPort to ""
      end if
      return "ERROR: " & theErr & " --- " & errNum
   end try
end serialPortTest
```

Some of the errors that could be returned by the SerialPort commands include:

-108	Ran out of memory.
-1700	Data could not be coerced to the requested descriptor type (probably used a number for a string parameter or a string for a number).
-1701	Descriptor not found (message will actually indicate which parameter is missing).
-6101	Error: Illegal data bits parameter.
-6102	Error: Unsupported data type for 'as' parameter.
-6103	Error: Command failed for unknown reason.
-6104	Error: Data value not large enough for the number of requested write bytes.
-6105	Error: Serial Tool not found.
-6106	Error: Read or write timed out.
-6107	Error: Special reference string has been corrupted.
-6108	Error: Illegal write size.
-6111	Error: Serial Tool 'Rejected' error.
-6112	Error: Serial Tool 'Failed' error.
-6113	Error: Serial Tool 'Timeout' error.
-6114	Error: Couldn't open serial port (probably in use by another application).

6. Sample Scripts

The folder distributed with the SerialPort Scripting Commands contains several sample script files that illustrate how to use all of the commands.

The file **Sample CP290 Script** illustrates sending a couple of different commands to a CP290 X-10 interface. This script is for illustration only – if you're actually interested in doing home automation using the CP290, you should download the X10 Scripting Addition that from the same place you downloaded this scripting addition. It makes controlling the CP290 interface very easy.

Other scripts in the folder show you how to send a command to an ActiveHome interface, wait on input, etc. The Serial Port scripting addition has been used to control test equipment in labs, a scanning electron microscope, UPS’s, lighting controllers, etc.
The hardest part of controlling a device using the serial port is often setting up a cable for the device and figuring out the protocol that it expects. The following web site provides a wide variety of advice for using the Mac serial port with assorted lab equipment:

http://www.mindspring.com/~jc1/serial/main.html

If you have written any interesting scripts that illustrate using how to use this scripting addition, please send them to me for inclusion with future versions.

7. How To Register

These SerialPort scripting commands are shareware -- this gives you the chance to try them for 15 days without risk to ensure that they are useful to you. However, if you will continue using them beyond the 15 day trial period, you are required to either pay for them or remove all copies from your hard disk. These scripting commands are commercial quality, have seen extensive beta testing and are fully supported (if you find any problems, send email and they will be fixed promptly). This level of quality, testing and documentation takes a great deal of time and I don't think that $10 is much to ask for in return. Your registration fee covers the current version and any future versions. All registered users (with an email address) will be notified of future versions when they become available.

You can register by sending $10 US by check (US. bank only) or money order directly to:

Blake Ward
18620 Favre Ridge Road
Los Gatos, CA 95030

If you're ordering from another country, or would prefer to pay by credit card or purchase order, you can also use the Kagi shareware registration service. Use the enclosed **Register** program to fill out a registration form and then send it with your payment to:

Kagi Shareware
1442-A Walnut Street #392-WB
Berkeley, California, 94709-1405
USA

For credit card payments, you can also send the form via Email to shareware@kagi.com or by FAX to +1 510 652 6589. Credit card numbers are scrambled by Register for better security. Checks should be made out to "Kagi Shareware" and should be in US dollars.

You can now register On-line!

You can also register on-line by going to " http://order.kagi.com/?WB" or launching the **Register Online** application, which will launch your browser and take you to that page.

A site license for the SerialPort Scripting Addition is also available for $250 for one geographic area or $750 world-wide. Contact the author via email or at the above address for more information.

Note that your shareware registration fee entitles you to use the SerialPort Scripting Addition on one computer at a time. **If you want to redistribute the SerialPort Scripting Addition with another application or script that you've created, contact the author to negotiate a very reasonable license for redistribution.**

Appendix A: Software License Agreement

1. **License**

The software accompanying this manual is licensed, not sold, to you. This non-exclusive license allows you to use the software on only one CPU at a time.

2. **Restrictions**

When you pay the shareware fee for this software, you are only licensing the software for your own individual use. **You may not modify, adapt, translate, rent, lease, resell for profit, distribute, network or create derivative works based upon this software or any part thereof. In particular, you may not embed any of these scripting commands in another program for the purpose of redistribution.** If you wish to create end-user home automation applications that use this one or more of these SerialPort scripting commands, contact the author to negotiate a very reasonable redistribution agreement.

3. **Warranty Disclaimer, Limitation of Damages and Remedies**

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OR MERCHANTABILITY. THIS PROGRAM IS SOLD WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES WHATSOEVER. BECAUSE OF THE DIVERSITY OF CONDITIONS AND HARDWARE UNDER WHICH THIS PROGRAM MAY BE USED, NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS OFFERED. THE USER MUST ASSUME THE ENTIRE RISK OF USING THE PROGRAM. ANY LIABILITY OF SELLER OR AUTHOR WILL BE LIMITED EXCLUSIVELY TO REPLACEMENT OR REFUND OF THE PURCHASE PRICE.

4. **Government End Users**

If you are a United States Government end user, this license of the software conveys only “RESTRICTED RIGHTS”, and its use, disclosure, and duplication are subject to DFARS 52.227-7013. This software was developed at private expense, and no part of it was developed with government funds. The software is a trade secret of Blake Ward for all purpose of the Freedom of Information act, and is “commercial computer software” subject to the limited utilization as provided in the contract between the vendor and the governmental entity, and in all respects is proprietary data belonging solely to Blake Ward. Government personnel using the software are hereby on notice that the use of this software is subject to restrictions that are the same as, or similar to, those specified above.

4. **Trademarks**

Macintosh and AppleScript are registered trademarks of Apple Computer, Inc.