

Octave Instrument Control Toolkit 0.9.5

Low level instrumentation functions for GNU Octave.

John Donoghue

Copyright © 2019-2025 John Donoghue

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the same conditions as for modified versions.

Distribution

The GNU Octave Instrument Control Toolkit is *free* software. Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. This means that everyone is free to use it and free to redistribute it on certain conditions. The GNU Octave Instrument Control toolkit is not, however, in the public domain. It is copyrighted and there are restrictions on its distribution, but the restrictions are designed to ensure that others will have the same freedom to use and redistribute Octave that you have. The precise conditions can be found in the GNU General Public License that comes with the GNU Octave Instrument Control toolkit and that also appears in [Appendix A \[Copying\]](#), page 65.

To download a copy of the GNU Octave Instrument Control Toolkit, please visit <http://octave.sourceforge.net/instrument-control/>.

Table of Contents

1	Installing and loading	1
1.1	Requirements	1
1.2	Windows install	1
1.3	Online Direct install	1
1.4	Off-line install	1
1.5	Loading	1
2	Basic Usage Overview	2
2.1	Authors	2
2.2	Available Interfaces	2
2.3	Basic Serial	2
2.3.1	Serial	2
2.3.2	SerialPort	3
2.4	Basic TCP	3
2.4.1	TCP	4
2.4.2	TCP Client	4
2.5	Basic UDP	5
2.5.1	UDP	5
2.5.2	UDP Port	5
3	Function Reference	7
3.1	Common Functions	7
3.1.1	flushinput	7
3.1.2	flushoutput	7
3.1.3	readbinblock	7
3.1.4	readline	7
3.1.5	writebinblock	8
3.1.6	writeline	8
3.1.7	writeread	8
3.2	General	9
3.2.1	instrhelp	9
3.2.2	instrhwinfo	9
3.2.3	resolvehost	10
3.3	GPIB	10
3.3.1	@octave_gpib/fclose	10
3.3.2	@octave_gpib/fopen	10
3.3.3	@octave_gpib/fprintf	10
3.3.4	@octave_gpib/fread	11
3.3.5	@octave_gpib/fscanf	11
3.3.6	@octave_gpib/fwrite	11
3.3.7	clrdevice	11
3.3.8	gpib	11
3.3.9	gpib_close	11
3.3.10	gpib_read	12
3.3.11	gpib_timeout	12
3.3.12	gpib_write	12
3.3.13	spoll	12

3.3.14	trigger	12
3.4	I2C	12
3.4.1	@octave_i2c/fclose	12
3.4.2	@octave_i2c/fopen	13
3.4.3	@octave_i2c/fread	13
3.4.4	@octave_i2c/fwrite	13
3.4.5	@octave_i2c/get	13
3.4.6	@octave_i2c/set	14
3.4.7	i2c	14
3.4.8	i2c_addr	15
3.4.9	i2c_close	15
3.4.10	i2c_read	15
3.4.11	i2c_write	15
3.5	Modbus	16
3.5.1	@octave_modbus/get	16
3.5.2	@octave_modbus/maskWrite	16
3.5.3	@octave_modbus/read	16
3.5.4	@octave_modbus/set	17
3.5.5	@octave_modbus/write	17
3.5.6	@octave_modbus/writeRead	18
3.5.7	modbus	18
3.6	Parallel	19
3.6.1	@octave_parallel/fclose	19
3.6.2	@octave_parallel/fopen	19
3.6.3	@octave_parallel/fread	20
3.6.4	@octave_parallel/fwrite	20
3.6.5	parallel	20
3.6.6	pp_close	20
3.6.7	pp_ctrl	21
3.6.8	pp_data	21
3.6.9	pp_datadir	21
3.6.10	pp_stat	22
3.7	Serial (Deprecated)	22
3.7.1	@octave_serial/fclose	22
3.7.2	@octave_serial/flushinput	22
3.7.3	@octave_serial/flushoutput	22
3.7.4	@octave_serial/fopen	22
3.7.5	@octave_serial/fprintf	22
3.7.6	@octave_serial/fread	23
3.7.7	@octave_serial/fwrite	23
3.7.8	@octave_serial/get	23
3.7.9	@octave_serial/serialbreak	24
3.7.10	@octave_serial/set	24
3.7.11	@octave_serial/srl_baudrate	25
3.7.12	@octave_serial/srl_bytesize	25
3.7.13	@octave_serial/srl_close	25
3.7.14	@octave_serial/srl_flush	26
3.7.15	@octave_serial/srl_parity	26
3.7.16	@octave_serial/srl_stopbits	26
3.7.17	@octave_serial/srl_timeout	27
3.7.18	serial	27
3.7.19	seriallist	28

3.7.20	srl_read	28
3.7.21	srl_write	28
3.8	Serial Port	28
3.8.1	@octave_serialport/configureTerminator	28
3.8.2	@octave_serialport/flush	29
3.8.3	@octave_serialport/fprintf	29
3.8.4	@octave_serialport/fread	29
3.8.5	@octave_serialport/fwrite	30
3.8.6	@octave_serialport/get	30
3.8.7	@octave_serialport/getpinstatus	30
3.8.8	@octave_serialport/read	31
3.8.9	@octave_serialport/serialbreak	31
3.8.10	@octave_serialport/set	31
3.8.11	@octave_serialport/setDTR	32
3.8.12	@octave_serialport/setRTS	32
3.8.13	@octave_serialport/write	33
3.8.14	serialport	33
3.8.15	serialportlist	34
3.9	SPI	34
3.9.1	@octave_spi/fclose	34
3.9.2	@octave_spi/fopen	34
3.9.3	@octave_spi/fread	34
3.9.4	@octave_spi/fwrite	35
3.9.5	@octave_spi/get	35
3.9.6	@octave_spi/read	36
3.9.7	@octave_spi/set	36
3.9.8	@octave_spi/write	36
3.9.9	@octave_spi/writeAndRead	37
3.9.10	spi	37
3.9.11	spi_close	38
3.9.12	spi_read	38
3.9.13	spi_write	38
3.9.14	spi_writeAndRead	38
3.10	TCP (Deprecated)	39
3.10.1	@octave_tcp/fclose	39
3.10.2	@octave_tcp/flush	39
3.10.3	@octave_tcp/flushinput	39
3.10.4	@octave_tcp/flushoutput	39
3.10.5	@octave_tcp/fopen	40
3.10.6	@octave_tcp/fprintf	40
3.10.7	@octave_tcp/fread	40
3.10.8	@octave_tcp/fwrite	40
3.10.9	@octave_tcp/get	41
3.10.10	@octave_tcp/read	41
3.10.11	@octave_tcp/set	41
3.10.12	@octave_tcp/write	42
3.10.13	tcp	42
3.10.14	tcp_close	43
3.10.15	tcp_read	43
3.10.16	tcp_timeout	43
3.10.17	tcp_write	43
3.10.18	tcpip	44

3.11	TCP Client	44
3.11.1	@octave_tcpclient/configureTerminator	44
3.11.2	@octave_tcpclient/flush	44
3.11.3	@octave_tcpclient/get	45
3.11.4	@octave_tcpclient/read	45
3.11.5	@octave_tcpclient/set	45
3.11.6	@octave_tcpclient/write	46
3.11.7	tcpclient	46
3.12	TCP Server	47
3.12.1	@octave_tcpserver/configureTerminator	47
3.12.2	@octave_tcpserver/flush	48
3.12.3	@octave_tcpserver/get	48
3.12.4	@octave_tcpserver/read	48
3.12.5	@octave_tcpserver/set	49
3.12.6	@octave_tcpserver/write	49
3.12.7	tcpserver	49
3.13	UDP (Deprecated)	50
3.13.1	@octave_udp/fclose	50
3.13.2	@octave_udp/flush	50
3.13.3	@octave_udp/flushinput	51
3.13.4	@octave_udp/flushoutput	51
3.13.5	@octave_udp/fopen	51
3.13.6	@octave_udp/fprintf	51
3.13.7	@octave_udp/fread	51
3.13.8	@octave_udp/fwrite	52
3.13.9	@octave_udp/get	52
3.13.10	@octave_udp/read	53
3.13.11	@octave_udp/set	53
3.13.12	@octave_udp/write	53
3.13.13	udp	54
3.13.14	udp_close	54
3.13.15	udp_demo	55
3.13.16	udp_read	55
3.13.17	udp_timeout	55
3.13.18	udp_write	55
3.14	UDP Port	56
3.14.1	@octave_udpport/configureMulticast	56
3.14.2	@octave_udpport/configureTerminator	56
3.14.3	@octave_udpport/flush	56
3.14.4	@octave_udpport/fprintf	57
3.14.5	@octave_udpport/fread	57
3.14.6	@octave_udpport/fwrite	57
3.14.7	@octave_udpport/get	57
3.14.8	@octave_udpport/read	58
3.14.9	@octave_udpport/set	58
3.14.10	@octave_udpport/write	59
3.14.11	@octave_udpport/writeline	59
3.14.12	udpport	59
3.15	USBTMC	60
3.15.1	@octave_usbtmc/fclose	60
3.15.2	@octave_usbtmc/fopen	61
3.15.3	@octave_usbtmc/fread	61

3.15.4	@octave_usbtmc/fwrite	61
3.15.5	usbtmc	61
3.15.6	usbtmc_close	62
3.15.7	usbtmc_read	62
3.15.8	usbtmc_write	62
3.16	VXI11	62
3.16.1	@octave_vxi11/fclose	62
3.16.2	@octave_vxi11/fopen	62
3.16.3	@octave_vxi11/fread	63
3.16.4	@octave_vxi11/fwrite	63
3.16.5	vxi11	63
3.16.6	vxi11_close	63
3.16.7	vxi11_read	64
3.16.8	vxi11_write	64
Appendix A GNU General Public License		65
Index		75

1 Installing and loading

The Instrument Control toolkit must be installed and then loaded to be used.

It can be installed in GNU Octave directly from octave-forge, or can be installed in an off-line mode via a downloaded tarball.

The toolkit must be then be loaded once per each GNU Octave session in order to use its functionality.

1.1 Requirements

For GPIB support (Linux only), linux-gpib must be installed before installing instrument-control. GPIB support is also available for windows by following the information from the wiki: https://wiki.octave.org/Instrument_control_package#Requirements

For VXI11 support, rpcgen, and libtirpc-devel must be installed before installing instrument-control.

For MODBUS support, the libmodbus-devel must be installed before installing instrument-control.

1.2 Windows install

If using the GNU Octave installer in Windows, the toolkit will have already been installed, and does not need to be re-installed unless a newer version is available.

Run the following command to verify if the toolkit is available:

```
pkg list instrument-control
```

1.3 Online Direct install

With an internet connection available, toolkit can be installed from octave-forge using the following command within GNU Octave:

```
pkg install -forge instrument-control
```

The latest released version of the toolkit will be downloaded, compiled and installed.

1.4 Off-line install

With the toolkit package already downloaded, and in the current directory when running GNU Octave, the package can be installed using the following command within GNU Octave:

```
pkg install instrument-control-0.9.5.tar.gz
```

1.5 Loading

Regardless of the method of installing the toolkit, in order to use its functions, the toolkit must be loaded using the pkg load command:

```
pkg load instrument-control
```

The toolkit must be loaded on each GNU Octave session.

2 Basic Usage Overview

2.1 Authors

The Instrument control package provides low level I/O functions for serial, i2c, spi, parallel, tcp, gpib, vx11, udp and usbtmc interfaces.

It was written mainly by the following developers:

- Andrius Sutas <andrius.sutasg at mail.com>
- Stefan Mahr <dac922 at gmx.de>
- John Donoghue <john.donoghue at ieee.org>

2.2 Available Interfaces

The ability to use each interface is dependent on OS and what libraries were available during the toolkit install.

To verify the available interfaces, run the following command in octave:

```
instrhwinfo
```

The function will return information on the supported interfaces that are available, similar to below:

```
ToolboxVersion = 0.7.0
ToolboxName = octave instrument control package
SupportedInterfaces =
{
    [1,1] = gpib
    [1,2] = i2c
    [1,3] = parallel
    [1,4] = serial
    [1,5] = serialport
    [1,6] = tcp
    [1,7] = tcpclient
    [1,8] = udp
    [1,9] = udpport
    [1,10] = usbtmc
    [1,11] = vx11
}
```

Most interfaces have two types of functions:

- somewhat compatible matlab functions such as fread, fwrite
- interface specific lower level functions such as udp_read, udp_write

2.3 Basic Serial

2.3.1 Serial

NOTE: The serial object has been deprecated and may not appear in newer versions of the instrument-control toolbox. Instead new code should use the serialport object.

The serial port can be opened using the serial function:

```
s = serial("/dev/ttyUSB1", 115200)
```

The first parameter is the device name and is OS specific. The second parameter is the baudrate.

A list of available serial ports can be retrieved using the function:

```
seriallist
```

After creating the interface object, properties of the device can be set or retrieved using get or set functions or as property access.

```
s = serial("/dev/ttyUSB1", 115200)
br = get(s, "baudrate") # gets the baudrate
br = s.baudrate # also gets the baudrate

set(s, "baudrate", 9600) # set the baudrate
s.baudrate = 9600 # also sets the baudrate
```

The device can be written and read from using fread, fwrite and srl_read and srl_write functions.

```
srl_write(s, "hello world") # write hello world
fprintf(s, "hello again")

val = srl_read(s, 10) # attempt to read
val = fread(s, 10)
```

The device can be closed using fclose or srl_close.

```
fclose(s)
```

2.3.2 SerialPort

The recommended method of accessing serial ports is through the serialport object.

The serial port can be opened using the serialport function:

```
s = serialport("/dev/ttyUSB1", 115200)
```

The first parameter is the device name and is OS specific. The second parameter is the baudrate.

A list of available serial ports can be retrieved using the function:

```
serialportlist
```

After creating the interface object, properties of the device can be set or retrieved using get or set functions or as property access.

```
s = serialport("/dev/ttyUSB1", 115200)
br = get(s, "BaudRate") # gets the baudrate
br = s.BaudRate # also gets the baudrate

set(s, "BaudRate", 9600) # set the baudrate
s.BaudRate = 9600 # also sets the baudrate
```

The device can be written and read from using read and write functions.

```
write(s, "hello world") # write hello world

val = read(s, 10)
```

The device can be closed by clearing the serialport object.

```
clear s
```

2.4 Basic TCP

2.4.1 TCP

NOTE: The TCP object has been deprecated and may not appear in newer versions of the instrument-control toolbox. Instead new code should use the tcpclient object.

A TCP connection can be opened using the tcp or tcpip function:

```
s = tcp("127.0.0.1", 80)
```

The first parameter is the IP address to connect to. The second parameter is the port number. And optional timeout value can be also be provided.

A more matlab compatible function is available as tcpip to also open a tcp port:

```
s = tcpip("gnu.org", 80)
```

The first parameter is a hostname or ip address, the second the port number. Additional parameter/value pairs can be provided after the port.

After creating the interface object, properties of the device can be set or retrieved using get or set functions or as property access.

```
s = tcp("127.0.0.1", 80)
oldtimeout = get(s, "timeout") # get timeout
```

```
set(s, "timeout", 10) # set the timeout
s.timeout = oldtimeout # also sets the timeout
```

The device can be written and read from using fread, fwrite and tcp_read and tcp_write functions.

```
tcp_write(s, "HEAD / HTTP/1.1\r\n\r\n")
```

```
val = tcp_read(s, 100, 500) # attempt to read 100 bytes
```

The device can be closed using fclose or tcp_close.

```
fclose(s)
```

2.4.2 TCP Client

The recommended method of creating a tcp connection is through the tcpclient object.

A TCP connection can be opened using the tcpclient function:

```
s = tcpclient("127.0.0.1", 80)
```

The first parameter is the IP address or hostname to connect to. The second parameter is the port number.

Additional parameter/value pairs can be provided after the port.

After creating the interface object, properties of the device can be set or retrieved using get or set functions or as property access.

```
s = tcpclient("127.0.0.1", 80)
oldtimeout = get(s, "Timeout") # get timeout
```

```
set(s, "Timeout", 10) # set the timeout
s.Timeout = oldtimeout # also sets the timeout
```

The device can be written and read from using read and write functions.

```
write(s, "HEAD / HTTP/1.1\r\n\r\n")
```

```
val = read(s, 100) # attempt to read 100 bytes
```

The device can be closed by clearing the object variable.

```
clear s
```

2.5 Basic UDP

2.5.1 UDP

NOTE: The UDP object has been deprecated and may not appear in newer versions of the instrument-control toolbox. Instead new code should use the `udpport` object.

A UDP connection can be opened using the `udp` function:

```
s = udp("127.0.0.1", 80)
```

The first parameter is the IP address data will be to. The second parameter is the port number. If an IP address and port is not provided, it will default to "127.0.0.1" and 23.

The address and port can be changed after creation using the `remotehost` and `remoteport` properties.

```
s = udp()
s.remotehost = "127.0.0.1";
s.remoteport = 100;
```

After creating the interface object, other properties of the device can be set or retrieved using `get` or `set` functions or as property access.

```
s = udp("127.0.0.1", 80)
oldtimeout = get(s, "timeout") # get timeout

set(s, "timeout", 10) # set the timeout
s.timeout = oldtimeout # also sets the timeout
```

The device can be written and read from using `fread`, `fwrite` and `udp_read` and `udp_write` functions.

```
udp_write(s, "test")

val = udp_read(s, 5)
```

The device can be closed using `fclose` or `udp_close`.

```
fclose(s)
```

2.5.2 UDP Port

The recommended method of creating a UDP socket is through the `udpport` object.

A `udpport` object can be created using the `udpport` function:

```
s = udpport()
```

Additional parameter/value pairs can be provided during creation of the object.

After creating the interface object, properties of the device can be set or retrieved using `get` or `set` functions or as property access.

```
s = udpport()
oldtimeout = get(s, "Timeout") # get timeout

set(s, "Timeout", 10) # set the timeout
s.Timeout = oldtimeout # also sets the timeout
```

The device can be written and read from using `read` and `write` functions.

The destination address and port to send data to must be specified at least on the first time `write` is used.

```
write(s, "test", "127.0.0.1", s.LocalPort)
```

```
val = read(s)
```

The device can be closed by clearing the object variable.

```
clear s
```

3 Function Reference

The functions currently available in the toolkit are described below.

3.1 Common Functions

3.1.1 flushinput

`flushinput (dev)`

Flush the instruments input buffers

Inputs

dev - connected device or array of devices

Outputs

None

See also: flushoutput.

3.1.2 flushoutput

`flushoutput (dev)`

Flush the instruments output buffers

Inputs

dev - connected device or array of devices

Outputs

None

See also: flushinput.

3.1.3 readbinblock

`data = readbinblock (dev)`

`data = readbinblock (dev, datatype)`

read a binblock of data from a instrument device

Inputs

dev - connected device

datatype - optional data type to read data as (default 'uint8')

Outputs

data - data read

See also: flushoutput.

3.1.4 readline

`data = readline (dev)`

read data from a instrument device excluding terminator value

Inputs

dev - connected device

Outputs

data - ASCII data read

See also: flushoutput.

3.1.5 writebinblock

writebinblock (*dev*, *data*, *datatype*)

Write a IEEE 488.2 binblock of data to a instrument device

binblock formatted data is defined as:

#<A><C>

where: <A> ASCII number containing the length of part

 ASCII number containing the number of bytes of <C>

<C> Binary data block

Inputs

dev - connected device

data - binary data to send

datatype - datatype to send data as

Outputs

None

See also: flushoutput.

3.1.6 writeline

writeline (*dev*, *data*)

Write data to a instrument device including terminator value

Inputs

dev - connected device

data - ASCII data to write

Outputs

None

See also: flushoutput.

3.1.7 writeread

data = writeread (*dev*, *command*)

write a ASCII command and read data from a instrument device.

Inputs

dev - connected device

command - ASCII command

Outputs

data - ASCII data read

See also: readline, writeline.

3.2 General

3.2.1 instrhelp

```
instrhelp ()
instrhelp (funcname)
instrhelp (obj)
    Display instrument help
```

Inputs

funcname - function to display help about.
obj - object to display help about.

If no input is provided, the function will display an overview of the package functionality.

Outputs

None

3.2.2 instrhwinfo

```
[list] = instrhwinfo () [Function File]
list = instrhwinfo (interface) [Function File]
    Query available hardware for instrument-control
```

When run without any input parameters, `instrhwinfo` will provide the toolbox information and a list of supported interfaces.

Inputs

interface is the instrument interface to query. When provided, `instrhwinfo` will provide information on the specified interface.

Currently only interface "serialport", "i2c" and "spi" are supported, which will provide a list of available serial ports or i2c ports.

Outputs

If an output variable is provided, the function will store the information to the variable, otherwise it will be displayed to the screen.

Example

```
instrhwinfo
    scalar structure containing the fields:
```

```
ToolboxVersion = 0.4.0
ToolboxName = octave instrument control package
SupportedInterfaces =
{
    [1,1] = i2c
    [1,2] = parallel
    [1,3] = serialport
    [1,4] = tcp
    [1,5] = udp
    [1,6] = usbtmc
    [1,7] = vx11
```

```
}
```

3.2.3 resolvehost

```
name = resolvehost (host) [Loadable Function]
[name, address] = resolvehost (host) [Loadable Function]
out = resolvehost (host, returntype) [Loadable Function]
```

Resolve a network host name or address to network name and address

Inputs

host - Host name or IP address string to resolve.
name - Resolved IP host name.
returntype - 'name' to get host name, 'address' to get IP address.

Outputs

name - Resolved IP host name.
address - Resolved IP host address.
out - host name if *returntype* is 'name', ipaddress if *returntype* is 'address'

Example

```
%% get resolved ip name and address of www.gnu.org
[name, address] = resolvehost ('www.gnu.org');

%% get ip address of www.gnu.org
ipaddress = resolvehost ('www.gnu.org', 'address');
```

See also: tcp, udp.

3.3 GPIB

3.3.1 @octave_gpib/fclose

```
res = fclose (obj) [Function File]
```

Closes connection to GPIB device *obj*

3.3.2 @octave_gpib/fopen

```
res = fopen (obj) (dummy) [Function File]
```

Opens connection to GPIB device *obj* This currently is a dummy function to improve compatibility to MATLAB

3.3.3 @octave_gpib/fprintf

```
fprintf (obj, cmd) [Function File]
fprintf (obj, format, cmd) [Function File]
fprintf (obj, cmd, mode) [Function File]
fprintf (obj, format, cmd, mode) [Function File]
```

Writes string *cmd* to GPIB instrument
obj is a GPIB object
cmd String *format* Format specifier *mode* sync

3.3.4 @octave_gpib/fread

`data = fread (obj)` [Function File]
`data = fread (obj, size)` [Function File]
`data = fread (obj, size, precision)` [Function File]
`[data,count] = fread (obj, ...)` [Function File]
`[data,count,errmsg] = fread (obj, ...)` [Function File]
 Reads *data* from GPIB instrument
obj is a GPIB object
size Number of values to read. (Default: 100) *precision* precision of data
count values read *errmsg* read operation error message

3.3.5 @octave_gpib/fscanf

`res = fscanf (obj)` [Function File]
`res = fscanf (obj, format)` [Function File]
`res = fscanf (obj, format, size)` [Function File]
`[res,count] = fscanf (obj, ...)` [Function File]
`[res,count,errmsg] = fscanf (obj, ...)` [Function File]
 Reads data *res* from GPIB instrument
obj is a GPIB object
format Format specifier *size* number of values
count values read *errmsg* read operation error message

3.3.6 @octave_gpib/fwrite

`fwrite (obj, data)` [Function File]
`fwrite (obj, data, precision)` [Function File]
`fwrite (obj, data, mode)` [Function File]
`fwrite (obj, data, precision, mode)` [Function File]
 Writes *data* to GPIB instrument
obj is a GPIB object
data data to write *precision* precision of data *mode* sync

3.3.7 clrdevice

`clrdevice (obj)` [Function File]
 Send clear command to Clear GPIB instrument.
obj is a GPIB object

3.3.8 gpib

`gpib = gpib ([gpibid], [timeout])` [Loadable Function]
 Open gpib interface.
gpibid - the interface number.
timeout - the interface timeout value. If omitted defaults to blocking call.
 The `gpib()` shall return instance of *octave_gpib* class as the result *gpib*.

3.3.9 gpib_close

`gpib_close (gpib)` [Loadable Function]
 Close the interface and release a file descriptor.
gpib - instance of *octave_gpib* class.

3.3.10 gpib_read

`[data, count, eoi] = gpib_read (gpib, n)` [Loadable Function]

Read from gpib interface.

gpib - instance of *octave_gpib* class.

n - number of bytes to attempt to read of type Integer.

The `gpib_read()` shall return number of bytes successfully read in *count* as Integer and the bytes themselves in *data* as uint8 array. *eoi* indicates read operation complete

3.3.11 gpib_timeout

`gpib_timeout (gpib, timeout)` [Loadable Function]

`t = gpib_timeout (gpib)` [Loadable Function]

Set new or get existing gpib interface timeout parameter. The timeout value is valid from 0 to 17.

gpib - instance of *octave_gpib* class.

timeout - Value of 0 means never timeout, 11 means one second and 17 means 1000 seconds (see GPIB documentation (ibtm0) for further details)

If *timeout* parameter is omitted, the `gpib_timeout()` shall return current timeout value as the result *t*.

3.3.12 gpib_write

`n = gpib_write (gpib, data)` [Loadable Function]

Write data to a gpib interface.

gpib - instance of *octave_gpib* class.

data - data to be written to the gpib interface. Can be either of String or uint8 type.

Upon successful completion, `gpib_write()` shall return the number of bytes written as the result *n*.

3.3.13 spoll

`out = spoll (obj)` [Function File]

`[out, statusByte] = spoll (obj)` [Function File]

Serial polls GPIB instruments.

obj is a GPIB object or a cell array of GPIB objects

out GPIB objects ready for service *statusByte* status Byte

3.3.14 trigger

`trigger (obj)` [Function File]

Triggers GPIB instrument.

obj is a GPIB object

3.4 I2C

3.4.1 @octave_i2c/fclose

`res = fclose (obj)` [Function File]

Closes I2C connection *obj*

3.4.2 @octave_i2c/fopen

`res = fopen (obj) (dummy)` [Function File]
 Opens I2C connection *obj*

This currently is a dummy function to improve compatibility to MATLAB

3.4.3 @octave_i2c/fread

`data = fread (obj)` [Function File]
`data = fread (obj, size)` [Function File]
`data = fread (obj, size, precision)` [Function File]
`[data,count] = fread (obj, ...)` [Function File]
`[data,count,errmsg] = fread (obj, ...)` [Function File]
 Reads *data* from I2C instrument

Inputs

obj is a I2C object.
size Number of values to read. (Default: 100).
precision precision of data.

Outputs

data data values.
count number of values read.
errmsg read operation error message.

3.4.4 @octave_i2c/fwrite

`numbytes = fwrite (obj, data)` [Function File]
`numbytes = fwrite (obj, data, precision)` [Function File]
 Writes *data* to I2C instrument

Inputs

obj is a I2C object.
data data to write.
precision precision of data.

Outputs

returns number of bytes written.

3.4.5 @octave_i2c/get

`struct = get (i2c)` [Function File]
`field = get (i2c, property)` [Function File]
 Get the properties of i2c object.

Inputs

i2c - instance of *octave_i2c* class.

property - name of property.

Outputs

When *property* was specified, return the value of that property.
otherwise return the values of all properties as a structure.

See also: @octave_i2c/set.

3.4.6 @octave_i2c/set

<code>set (obj, property,value)</code>	[Function File]
<code>set (obj, property,value,...)</code>	[Function File]

Set the properties of i2c object.

Inputs

obj - instance of *octave_i2c* class.
property - name of property.

If *property* is a cell so must be *value*, it sets the values of all matching properties.
The function also accepts property-value pairs.

Properties

'name' Set the name for the i2c socket.
'remoteaddress'
 Set the remote address for the i2c socket.

Outputs

None

See also: @octave_i2c/get.

3.4.7 i2c

<code>i2c = i2c ([port_path], [address])</code>	[Loadable Function]
---	---------------------

Open i2c interface.

Inputs

port_path - the interface device port/path of type String. If omitted defaults to '/dev/i2c-0'.
address - the slave device address. If omitted must be set using `i2c_addr()` call.

Outputs

i2c - An instance of *octave_i2c* class.

Properties

The i2c object has the following properties:

name	Name of the object
remoteaddress	the slave device address
port	The interface driver port (readonly)

3.4.8 i2c_addr

`i2c_addr (i2c, address)` [Loadable Function]
`addr = i2c_addr (i2c)` [Loadable Function]

Set new or get existing i2c slave device address.

Inputs

i2c - instance of *octave_i2c* class.

address - i2c slave device address of type Integer. The address is passed in the 7 or 10 lower bits of the argument.

Outputs

addr - If *address* parameter is omitted, the `i2c_addr()` shall return current i2c slave device address.

3.4.9 i2c_close

`i2c_close (i2c)` [Loadable Function]

Close the interface and release a file descriptor.

Inputs

i2c - instance of *octave_i2c* class.

Outputs

None

3.4.10 i2c_read

`[data, count] = i2c_read (i2c, n)` [Loadable Function]

Read from i2c slave device.

Inputs

i2c - instance of *octave_i2c* class.

n - number of bytes to attempt to read of type Integer.

Outputs

The `i2c_read()` shall return number of bytes successfully read in *count* as Integer and the bytes themselves in *data* as uint8 array.

3.4.11 i2c_write

`n = i2c_write (i2c, data)` [Loadable Function]

Write data to a i2c slave device.

Inputs

i2c - instance of *octave_i2c* class.

data - data, of type uint8, to be written to the slave device.

Outputs

Upon successful completion, `i2c_write()` shall return the number of bytes written as the result *n*.

3.5 Modbus

3.5.1 @octave_modbus/get

```
struct = get (dev) [Function File]
field = get (dev, property) [Function File]
```

Get the properties of modbus object.

Inputs

dev - instance of *octave_modbus* class.
property - name of property.

Outputs

When *property* was specified, return the value of that property.
 otherwise return the values of all properties as a structure.

See also: @octave_modbus/set.

3.5.2 @octave_modbus/maskWrite

```
data = maskWrite (dev, address, andmask, ormask)
data = maskWrite (dev, address, andmask, ormask, serverid)
```

Read holding register at *address* from modbus device *dev* apply masking and write the change data.

writeregister value = (readregister value AND andMask) OR (orMask AND (NOT andMask))

Inputs

dev - connected modbus device
address - address to read from.
andmask - AND mask to apply to the register
ormask - OR mask to apply to the register
serverId - address to send to (0-247). Default of 1 is used if not specified.

Outputs

data - data read from the device

See also: modbus.

3.5.3 @octave_modbus/read

```
data = read (dev, target, address)
data = read (dev, target, address, count)
data = read (dev, target, address, count, serverId, precision)
```

Read data from modbus device *dev* target *target* starting at address *address*.

Inputs

dev - connected modbus device
target - target type to read. One of 'coils', 'inputs', 'inputregs' or 'holdingregs'
address - address to start reading from.
count - number of elements to read. If not provided, count is 1.

serverId - address to send to (0-247). Default of 1 is used if not specified.

precision - Optional precision for how to interpret the read data. Currently known precision values are uint16 (default), int16, uint32, int32, uint64, int64, single, double.

Outputs

data - data read from the device

See also: `modbus`.

3.5.4 @octave_modbus/set

`set (obj, property, value)` [Function File]

`set (obj, property, value, ...)` [Function File]

Set the properties of modbus object.

Inputs

obj - instance of *octave_modbus* class.

property - name of property.

If *property* is a cell so must be *value*, it sets the values of all matching properties.

The function also accepts property-value pairs.

Properties

'Name' Set the stored string name of the object.

'Timeout' Set the timeout value.

'Numretries'
Set the numretries value.

'ByteOrder'
Set the byteorder value

'WordOrder'
Set the wordorder value

'UserData'
Set the userdata value

Outputs

None

See also: `@octave_modbus/get`.

3.5.5 @octave_modbus/write

`write (dev, target, address, values)`

`read (dev, target, address, values, serverId, precision)`

Write data *data* to modbus device *dev* target *target* starting at address *address*.

Inputs

dev - connected modbus device

target - target type to read. One of 'coils' or 'holdingregs'

address - address to start reading from.

data - data to write.

serverId - address to send to (0-247). Default of 1 is used if not specified.

precision - Optional precision for how to interpret the write data. Currently known precision values are uint16 (default), int16, uint32, int32, uint64, uint64, single, double.

Outputs

None

See also: modbus.

3.5.6 @octave_modbus/writeRead

```
data = writeRead (dev, writeAddress, values, readAddress, readcount)
```

```
data = writeRead (dev, writeAddress, values, readAddress, readcount,  
                  serverId)
```

```
data = writeRead (dev, writeAddress, values, writePrecision,  
                  readAddress, readCount, readPrecision)
```

Write data *values* to the modbus device *dev* holding registers starting at address *writeAddress* and then read *readCount* register values starting at address *readAddress*.

Inputs

dev - connected modbus device

writeAddress - address to start writing to.

values - data to write to the device.

readAddress - address to start reading from.

readCount - number of elements to read.

serverId - address to send to (0-247). Default of 1 is used if not specified.

precision - Optional precision for how to interpret the read data. Currently known precision values are uint16 (default), int16, uint32, int32, uint64, uint64, single, double.

Outputs

data - data read from the device

See also: modbus.

3.5.7 modbus

```
dev = modbus ('tcpip', deviceaddress) [Loadable Function]
```

```
dev = modbus ('tcpip', deviceaddress, remoteport) [Loadable Function]
```

```
dev = modbus ('tcpip', deviceaddress, name, value) [Loadable Function]
```

```
dev = modbus ('serialrtu', serialport) [Loadable Function]
```

```
dev = modbus ('serialrtu', serialport, name, value) [Loadable Function]
```

Open modbus interface using a specified transport of 'tcpip' or 'serialrtu'.

Inputs

deviceaddress - the device ip address of type String.

remoteport - the device remote port number. If not specified, a default of 502 will be used.

name, value - Optional name value pairs for setting properties of the object.

serialport - the name of the serial port to connect to. It must be specified when transport is 'serialrtu'.

Common Input Name, Value pairs

Timeout timeout value used for waiting for data

NumRetries number of retries after a timeout

UserData Additional data to attach to the object

Serial RTU Input Name, Value pairs

BaudRate Baudrate for the serial port

DataBits number of databits for serial port

Parity Parity for serial port ('odd', 'even' or 'none')

StopBits number of stopbits for serial port

Outputs

The `modbus()` shall return instance of *octave_modbus* class as the result *modbus*.

Properties

The *modbus* object has the following public properties:

Name name assigned to the *modbus* object

Type instrument type 'modbus' (readonly)

Port Remote port number or serial port name (readonly)

DeviceAddress Device address if transport was 'tcpip' (readonly)

Status status of the object 'open' or 'closed' (readonly)

Timeout timeout value used for waiting for data

NumRetries number of retries after a timeout

UserData Additional data to attach to the object

3.6 Parallel

3.6.1 @octave_parallel/fclose

`res = fclose (obj)` [Function File]
 Closes parallel connection *obj*

3.6.2 @octave_parallel/fopen

`res = fopen (obj) (dummy)` [Function File]
 Opens parallel interface *obj*

This currently is a dummy function to improve compatibility to MATLAB

3.6.3 @octave_parallel/fread

<code>data = fread (obj)</code>	[Function File]
<code>data = fread (obj, size)</code>	[Function File]
<code>data = fread (obj, size, precision)</code>	[Function File]
<code>[data,count] = fread (obj, ...)</code>	[Function File]
<code>[data,count,errmsg] = fread (obj, ...)</code>	[Function File]

Reads *data* from parallel instrument

Inputs

obj is a parallel object.
size Number of values to read. (Default: 1).
precision precision of data.

Outputs

data The read data.
count values read.
errmsg read operation error message.

3.6.4 @octave_parallel/fwrite

<code>numbytes = fwrite (obj, data)</code>	[Function File]
<code>numbytes = fwrite (obj, data, precision)</code>	[Function File]

Writes *data* to parallel instrument

Inputs

obj is a parallel object.
data data to write.
precision precision of data.

Outputs

returns number of bytes written.

3.6.5 parallel

<code>parallel = parallel ([path], [direction])</code>	[Loadable Function]
--	---------------------

Open Parallel interface.

Inputs

path - the interface path of type String. If omitted defaults to `'/dev/parport0'`.
direction - the direction of interface drivers of type Integer, see: `PP_DATADIR` for more info.
 If omitted defaults to 1 (Input).

Outputs

The `parallel()` shall return instance of *octave-parallel* class as the result *parallel*.

3.6.6 pp_close

<code>pp_close (parallel)</code>	[Loadable Function]
----------------------------------	---------------------

Close the interface and release a file descriptor.

Inputs

parallel - instance of *octave_serial* class.

Outputs

None

3.6.7 pp_ctrl

`pp_ctrl (parallel, ctrl)` [Loadable Function]

`c = pp_ctrl (parallel)` [Loadable Function]

Sets or Read the Control lines.

Inputs

parallel - instance of *octave_parallel* class.

ctrl - control parameter to be set of type Byte.

Outputs

If *ctrl* parameter is omitted, the `pp_ctrl()` shall return current Control lines state as the result *c*.

3.6.8 pp_data

`pp_data (parallel, data)` [Loadable Function]

`d = pp_data (parallel)` [Loadable Function]

Sets or Read the Data lines.

Inputs

parallel - instance of *octave_parallel* class.

data - data parameter to be set of type Byte.

Outputs

If *data* parameter is omitted, the `pp_data()` shall return current Data lines state as the result *d*.

3.6.9 pp_datadir

`pp_datadir (parallel, direction)` [Loadable Function]

`dir = pp_datadir (parallel)` [Loadable Function]

Controls the Data line drivers.

Normally the computer's parallel port will drive the data lines, but for byte-wide transfers from the peripheral to the host it is useful to turn off those drivers and let the peripheral drive the signals. (If the drivers on the computer's parallel port are left on when this happens, the port might be damaged.)

Inputs

parallel - instance of *octave_parallel* class.

direction - direction parameter of type Integer. Supported values: 0 - the drivers are turned on (Output/Forward direction); 1 - the drivers are turned off (Input/Reverse direction).

Outputs

If *direction* parameter is omitted, the `pp_datadir()` shall return current Data direction as the result *dir*.

3.6.10 pp_stat

`stat = pp_stat (parallel)` [Loadable Function]
 Reads the Status lines.

Inputs

parallel - instance of *octave_parallel* class.

Outputs

The `pp_stat()` shall return current Status lines state as the result *stat*.

3.7 Serial (Deprecated)

3.7.1 @octave_serial/fclose

`res = fclose (obj)` [Function File]
 Closes SERIAL connection *obj*

3.7.2 @octave_serial/flushinput

`flushinput (serial)` [Loadable Function]
 Flush the pending input, which will also make the BytesAvailable property be 0.

Inputs

serial - instance of *octave_serial* class.

Outputs

None

See also: `srl_flush`, `flushoutput`.

3.7.3 @octave_serial/flushoutput

`flushoutput (serial)` [Loadable Function]
 Flush the output buffer.

Inputs

serial - instance of *octave_serial* class.

Outputs

None

See also: `srl_flush`, `flushinput`.

3.7.4 @octave_serial/fopen

`res = fopen (obj) (dummy)` [Function File]
 Opens SERIAL interface *obj*
 This currently is a dummy function to improve compatibility to MATLAB

3.7.5 @octave_serial/fprintf

`numbytes = fprintf (obj, template ...)` [Function File]
 Writes formatted string *template* using optional parameters to serial instrument

Inputs

obj is a serial object.

template Format template string

Outputs

numbytes - number of bytes written to the serial device.

3.7.6 @octave_serial/fread

<code>data = fread (obj)</code>	[Function File]
<code>data = fread (obj, size)</code>	[Function File]
<code>data = fread (obj, size, precision)</code>	[Function File]
<code>[data,count] = fread (obj, ...)</code>	[Function File]
<code>[data,count,errmsg] = fread (obj, ...)</code>	[Function File]

Reads *data* from serial instrument

Inputs

obj is a serial object.

size Number of values to read. (Default: 100).

precision precision of data.

Outputs

data The read data.

count values read.

errmsg read operation error message.

3.7.7 @octave_serial/fwrite

<code>numbytes = fwrite (obj, data)</code>	[Function File]
<code>numbytes = fwrite (obj, data, precision)</code>	[Function File]

Writes *data* to serial instrument

Inputs

obj is a serial object.

data data to write.

precision precision of data.

Outputs

returns number of bytes written.

3.7.8 @octave_serial/get

<code>struct = get (serial)</code>	[Function File]
<code>field = get (serial, property)</code>	[Function File]

Get the properties of serial object.

Inputs

serial - instance of *octave_serial* class.

property - name of property.

Outputs

When *property* was specified, return the value of that property.
otherwise return the values of all properties as a structure.

See also: @octave_serial/set.

3.7.9 @octave_serial/serialbreak

`serialbreak (serial)` [Function File]

`serialbreak (serial, time)` [Function File]

Send a break to the serial port

Inputs

serial - serial object

time - number of milliseconds to break for. If not specified a value of 10 will be used.

Outputs

None

See also: serial.

3.7.10 @octave_serial/set

`set (obj, property,value)` [Function File]

`set (obj, property,value,...)` [Function File]

Set the properties of serial object.

Inputs

serial - instance of *octave_serial* class.

property - name of property.

If *property* is a cell so must be *value*, it sets the values of all matching properties.

The function also accepts property-value pairs.

Properties

'baudrate' Set the baudrate of serial port. Supported values by instrument-control: 0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 and 230400. The supported baudrate of your serial port may be different.

'bytesize' Set the bytesize. Supported values: 5, 6, 7 and 8.

'name' Set the stored string name of the serial object.

'parity' Set the parity value. Supported values: Even/Odd/None. This Parameter must be of type string. It is case insensitive and can be abbreviated to the first letter only

'stopbits' Set the number of stopbits. Supported values: 1, 2.

'timeout' Set the timeout value in tenths of a second. Value of -1 means a blocking call. Maximum value of 255 (i.e. 25.5 seconds).

'requesttosend'
Set the requesttosend (RTS) line.

`'dataterminalready'`

Set the dataterminalready (DTR) line.

Outputs

None

See also: `@octave_serial/get`.

3.7.11 `@octave_serial/srl_baudrate`

`srl_baudrate (serial, baudrate)\` [Loadable Function]

`br = srl_baudrate (serial)` [Loadable Function]

Set new or get existing serial interface baudrate parameter. Only standard values are supported.

Inputs

serial - instance of *octave_serial* class.

baudrate - the baudrate value used. Supported values: 0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600 19200, 38400, 57600, 115200 and 230400.

If *baudrate* parameter is omitted, the `srl_baudrate()` shall return current baudrate value as the result *br*.

Outputs

br - The currently set baudrate

This function is obsolete. Use get and set method instead.

3.7.12 `@octave_serial/srl_bytesize`

`srl_bytesize (serial, bsize)` [Loadable Function]

`bs = srl_bytesize (serial)` [Loadable Function]

Set new or get existing serial interface byte size parameter.

Inputs

serial - instance of *octave_serial* class.

bsize - byte size of type Integer. Supported values: 5/6/7/8.

If *bsize* parameter is omitted, the `srl_bytesize()` shall return current byte size value or in case of unsupported setting -1, as the result *bs*.

This function is obsolete. Use get and set method instead.

Outputs

bs -the currently set byte size.

3.7.13 `@octave_serial/srl_close`

`srl_close (serial)` [Loadable Function]

Close the interface and release a file descriptor.

Inputs

serial - instance of *octave_serial* class.

This function is obsolete. Use `fclose()` method instead.

Outputs

None

3.7.14 @octave_serial/srl_flush

`srl_flush (serial, [q])` [Loadable Function]

Flush the pending input/output.

Inputs

serial - instance of *octave_serial* class.

q - queue selector of type Integer. Supported values:

- 0 flush untransmitted output
- 1 flush pending input
- 2 flush both pending input and untransmitted output.

If *q* parameter is omitted, the `srl_flush()` shall flush both, input and output buffers.

Outputs

None

3.7.15 @octave_serial/srl_parity

`srl_parity (serial, parity)` [Loadable Function]

`p = srl_parity (serial)` [Loadable Function]

Set new or get existing serial interface parity parameter. Even/Odd/None values are supported.

Inputs

serial - instance of *octave_serial* class.

parity - parity value of type String. Supported values: Even/Odd/None (case insensitive, can be abbreviated to the first letter only)

If *parity* parameter is omitted, the `srl_parity()` shall return current parity value as the result *p*.

This function is obsolete. Use get and set method instead.

Outputs

p - The currently set parity

3.7.16 @octave_serial/srl_stopbits

`srl_stopbits (serial, stopb)` [Loadable Function]

`sb = srl_stopbits (serial)` [Loadable Function]

Set new or get existing serial interface stop bits parameter. Only 1 or 2 stop bits are supported.

Inputs

serial - instance of *octave_serial* class.

stopb - number of stop bits used. Supported values: 1, 2.

Outputs

If *stopb* parameter is omitted, the `srl_stopbits()` shall return current stop bits value as the result *sb*.

This function is obsolete. Use `get` and `set` method instead.

3.7.17 @octave_serial/srl_timeout

`srl_timeout (serial, timeout)` [Loadable Function]

`t = srl_timeout (serial)` [Loadable Function]

Set new or get existing serial interface timeout parameter used for `srl_read()` requests. The timeout value is specified in tenths of a second.

Inputs

serial - instance of *octave_serial* class.

timeout - `srl_read()` timeout value in tenths of a second. A value of -1 means a blocking call. Maximum value of 255 (i.e. 25.5 seconds).

Outputs

If *timeout* parameter is omitted, the `srl_timeout()` shall return current timeout value as the result *t*.

This function is obsolete. Use `get` and `set` method instead.

3.7.18 serial

`serial = serial ([path], [baudrate], [timeout])` [Loadable Function]

Open serial interface.

Inputs

path - the interface path of type String.

baudrate - the baudrate of interface. If omitted defaults to 115200.

timeout - the interface timeout value. If omitted defaults to blocking call.

Outputs

The `serial()` shall return an instance of *octave_serial* class as the result *serial*.

Properties

The `serial` object has the following public properties:

<code>name</code>	name assigned to the object
<code>type</code>	instrument type 'serial' (readonly)
<code>port</code>	OS specific port name (readonly)
<code>status</code>	status of the object 'open' or 'closed' (readonly)
<code>timeout</code>	timeout value used for waiting for data
<code>bytesavailable</code>	number of bytes currently available to read (readonly)
<code>stopbits</code>	number of stopbits to use
<code>requesttosend</code>	request to send state - 'on' or 'off'

parity Parity setting 'none', 'even', 'odd'
 bytesize Number of bits to a byte (7 or 8)
 baudrate Baudrate setting
 dataterminalready
 state of dataterminal ready - 'on' or 'off'
 pinstatus current state of pins (readonly)

3.7.19 seriallist

`list = seriallist ()` [Function File]
 Returns a list of all serial ports detected in the system.

Inputs

None

Outputs

list is a string cell array of serial ports names detected in the system.

See also: `instrhwinfo("serial")`.

3.7.20 srl_read

`[data, count] = srl_read (serial, n)` [Loadable Function]
 Read from serial interface.

Inputs

serial - instance of *octave_serial* class.

n - number of bytes to attempt to read of type Integer.

Outputs

The `srl_read()` shall return number of bytes successfully read in *count* as Integer and the bytes themselves in *data* as uint8 array.

3.7.21 srl_write

`n = srl_write (serial, data)` [Loadable Function]
 Write data to a serial interface.

Inputs

serial - instance of *octave_serial* class.

data - data to be written to the serial interface. Can be either of String or uint8 type.

Outputs

Upon successful completion, `srl_write()` shall return the number of bytes written as the result *n*.

3.8 Serial Port

3.8.1 @octave_serialport/configureTerminator

`configureTerminator (serial, term)` [Function File]
`configureTerminator (serial, readterm, writeterm)` [Function File]
 Set terminator for ASCII string manipulation

Inputs

serial - serialport object

term - terminal value for both read and write

readterm = terminal value type for read data

writeterm = terminal value for written data

The terminal can be either strings "cr", "lf" (default), "lf/cr" or an integer between 0 to 255.

Outputs

None

See also: serialport.

3.8.2 @octave_serialport/flush

data = flush (*dev*)

data = flush (*dev*, "input")

data = flush (*dev*, "output")

Flush the serial port buffers

Inputs

dev - connected serialport device

If an additional parameter is provided of "input" or "output", then only the input or output buffer will be flushed

Outputs

None

See also: serialport.

3.8.3 @octave_serialport/fprintf

numbytes = fprintf (*obj*, *template* ...) [Function File]

Writes formatted string *template* using optional parameters to serialport instrument

Inputs

obj is a serialport object.

template Format template string

Outputs

numbytes - number of bytes written to the serial device.

3.8.4 @octave_serialport/fread

data = fread (*obj*) [Function File]

data = fread (*obj*, *size*) [Function File]

data = fread (*obj*, *size*, *precision*) [Function File]

[*data*,*count*] = fread (*obj*, ...) [Function File]

[*data*,*count*,*errmsg*] = fread (*obj*, ...) [Function File]

Reads *data* from serial port instrument

Inputs

obj is a serialport object.

size Number of values to read.

precision precision of data.

Outputs

data The read data.

count number of values read.

errmsg read operation error message.

3.8.5 @octave_serialport/fwrite

numbytes = fwrite (*obj*, *data*)

[Function File]

numbytes = fwrite (*obj*, *data*, *precision*)

[Function File]

Writes *data* to serial port instrument

Inputs

obj is a serial port object.

data data to write.

precision precision of data.

Outputs

returns number of bytes written.

3.8.6 @octave_serialport/get

struct = get (*serial*)

[Function File]

field = get (*serial*, *property*)

[Function File]

Get the properties of serialport object.

Inputs

serial - instance of *octave_serialport* class.

property - name of property.

Outputs

When *property* was specified, return the value of that property.
otherwise return the values of all properties as a structure.

See also: @octave_serial/set.

3.8.7 @octave_serialport/getpinstatus

status getpinstatus (*serial*)

[Function File]

Get status of serial pins

Inputs

serial - serial object

Outputs

status - a structure with the logic names of ClearToSend, DataSetReady, CarrierDetect, and RingIndicator

See also: serialport.

3.8.8 @octave_serialport/read

```
data = read (dev, count)
data = read (dev, count, precision)
```

Read a specified number of values from a serialport using optional precision for valuesize.

Inputs

dev - connected serialport device

count - number of elements to read

precision - Optional precision for the output data read data. Currently known precision values are uint8 (default), int8, uint16, int16, uint32, int32, uint64, uint64

Outputs

data - data read from the device

See also: serialport.

3.8.9 @octave_serialport/serialbreak

```
serialbreak (serial) [Function File]
serialbreak (serial, time) [Function File]
```

Send a break to the serial port

Inputs

serial - serialport object

time - number of milliseconds to break for. If not specified a value of 10 will be used.

Outputs

None

See also: serial.

3.8.10 @octave_serialport/set

```
set (obj, property,value) [Function File]
set (obj, property,value,...) [Function File]
```

Set the properties of serialport object.

Inputs

serial - instance of *octave_serialport* class.

property - name of property.

If *property* is a cell so must be *value*, it sets the values of all matching properties.

The function also accepts property-value pairs.

Properties

- '*baudrate*' Set the baudrate of serial port. Supported values by instrument-control: 0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 and 230400. The supported baudrate of your serial port may be different.
- '*bytesize*' Set the bytesize. Supported values: 5, 6, 7 and 8.
- '*name*' Set the stored string name of the serial object.
- '*parity*' Set the parity value. Supported values: Even/Odd/None. This Parameter must be of type string. It is case insensitive and can be abbreviated to the first letter only
- '*stopbits*' Set the number of stopbits. Supported values: 1, 2.
- '*timeout*' Set the timeout value in tenths of a second. Value of -1 means a blocking call. Maximum value of 255 (i.e. 25.5 seconds).
- '*requesttosend*'
Set the requesttosend (RTS) line.
- '*dataterminalready*'
Set the dataterminalready (DTR) line.

Outputs

None

See also: @octave_serialport/-get.

3.8.11 @octave_serialport/setDTR

setDTR (*dev*, *true_false*)
Set the state of the DTR line

Inputs

dev - connected serial device.
true_false - state to set the line.

Outputs

None

See also: serialport, getpinstatus, setRTS.

3.8.12 @octave_serialport/setRTS

setRTS (*dev*, *true_false*)
Set the state of the RTS line

Inputs

dev - connected serial device.
true_false - state to set the line.

Outputs

None

See also: serialport, getpinstatus.

3.8.13 @octave_serialport/write

```
numbytes = write(obj, data) [Function File]
numbytes = write(obj, data, precision) [Function File]
```

Writes *data* to serialport instrument

Inputs

obj is a serialport object.
data data to write.
precision precision of data.

Outputs

returns number of bytes written.

3.8.14 serialport

```
serial = serialport([path], [baudrate]) [Loadable Function]
serial = serialport([path], [propname, propvalue]) [Loadable Function]
```

Open serial port interface.

Inputs

path - the interface path of type String.
baudrate - the baudrate of interface.
propname,propvalue - property name/value pairs.

Known input properties:

BaudRate	Numeric baudrate value
Timeout	Numeric timeout value in seconds or -1 to wait forever
StopBits	number of stopbits to use
Parity	Parity setting 'none', 'even', 'odd'
DataBits	Number of bits to a byte (5 to 8)
FlowControl	Number of bits to a byte 'none', 'hardware', 'software'
Tag	User settable string to identify the port.

Outputs

The serialport() shall return an instance of *octave_serialport* class as the result *serial*.

Properties

The serial object has the following public properties:

Name	name assigned to the object
Type	instrument type 'serial' (readonly)
Port	OS specific port name (readonly)
Status	status of the object 'open' or 'closed' (readonly)
Timeout	timeout value used for waiting for data

NumBytesAvailable	number of bytes currently available to read (readonly)
NumBytesWritten	number of bytes written (readonly)
StopBits	number of stopbits to use
Parity	Parity setting 'none', 'even', 'odd'
DataBits	Number of bits to a byte (5 to 8)
BaudRate	Baudrate setting
FlowControl	Number of bits to a byte 'none', 'hardware', 'software'
PinStatus	current state of pins (readonly)
UserData	user defined data
Tag	user defined tag name

3.8.15 serialportlist

```
list = serialportlist () [Function File]
list = serialportlist ("all") [Function File]
list = serialportlist ("available") [Function File]
```

Returns a list of all serial ports detected in the system.

Inputs

'all' - show all serial ports (same as providing no arguments) 'available' - show only serial ports that are available for use

Outputs

list is a string cell array of serial ports names detected in the system.

See also: instrhwinfo("serialport").

3.9 SPI

3.9.1 @octave_spi/fclose

```
res = fclose (obj) [Function File]
```

Closes SPI connection *obj*

3.9.2 @octave_spi/fopen

```
res = fopen (obj) (dummy) [Function File]
```

Opens SPI connection *obj*

This currently is a dummy function to improve compatibility to MATLAB

3.9.3 @octave_spi/fread

```
data = fread (obj) [Function File]
data = fread (obj, size) [Function File]
data = fread (obj, size, precision) [Function File]
[data,count] = fread (obj, ...) [Function File]
[data,count,errmsg] = fread (obj, ...) [Function File]
```

Reads *data* from a SPI instrument

Inputs

obj is a SPI object.

size Number of values to read. (Default: 10).

precision precision of data.

Outputs

data data values.

count number of values read.

errmsg read operation error message.

3.9.4 @octave_spi/fwrite

numbytes = `fwrite(obj, data)`

[Function File]

numbytes = `fwrite(obj, data, precision)`

[Function File]

Writes *data* to SPI instrument

Inputs

obj is a SPI object.

data data to write.

precision precision of data.

Outputs

returns number of bytes written.

3.9.5 @octave_spi/get

struct = `get(spi)`

[Function File]

field = `get(spi, property)`

[Function File]

Get the properties of spi object.

Inputs

spi - instance of *octave_spi* class.

property - name of property.

Properties

'*name*' Name for the spi socket.

'*bitrate*' The bitrate for the spi object.

'*clockpolarity*'
The clock polarity for the spi object of 'idlehigh' or 'idlelow'.

'*clockphase*'
The clock phase for the spi object of 'firstedge' or 'secondedge'.

'*port*' The device port name.

'*status*' The device status of 'open' or 'closed'

Outputs

When *property* was specified, return the value of that property.
otherwise return the values of all properties as a structure.

See also: @octave_spi/set.

3.9.6 @octave_spi/read

<code>data = read (obj)</code>	[Function File]
<code>data = read (obj, size)</code>	[Function File]

Reads *data* from SPI instrument

Inputs

obj is a SPI object.
size Number of values to read. (Default: 10).

Outputs

data data values.

3.9.7 @octave_spi/set

<code>set (obj, property,value)</code>	[Function File]
<code>set (obj, property,value,...)</code>	[Function File]

Set the properties of spi object.

Inputs

obj - instance of *octave_spi* class.
property - name of property.

If *property* is a cell so must be *value*, it sets the values of all matching properties.
The function also accepts property-value pairs.

Properties

'name' Set the name for the spi socket.
'bitrate' Set the bitrate for the spi object.
'clockpolarity'
 Set the clock polarity for the spi object of 'idlehigh' or 'idlelow'.
'clockphase'
 Set the clock phase for the spi object of 'firstedge' or 'secondedge'.

Outputs

None

See also: @octave_spi/get.

3.9.8 @octave_spi/write

<code>numbytes = fwrite (obj, data)</code>	[Function File]
--	-----------------

Writes *data* to SPI instrument

Inputs

obj is a SPI object.
data data to write.

Outputs

returns number of bytes written.

3.9.9 @octave_spi/writeAndRead

data = writeAndRead (*obj*, *wrdata*) [Function File]
 Writes and reads *data* from SPI instrument

Inputs

obj is a SPI object.
wrdata Data to write.

Outputs

data data values read.

3.9.10 spi

spi = spi ([*port_path*]) [Loadable Function]
spi = spi ([*port_path*], [*propname*, *propvalue*]) [Loadable Function]
 Open a spi interface.

Inputs

port_path - the interface device port/path of type String. If omitted defaults to '/dev/spi-0'.
propname,propvalue - property name/value pairs.

Known input properties:

name	Name of the object
bitrate	Numeric bitrate value
clockpolarity	Clock polarity: idlehigh or idlelow.
clockphase	Clock phase value: firstedge or secondedge

Outputs

spi - An instance of *octave_spi* class.

Properties

The spi object has the following properties:

name	Name of the object
status	Open or closed status of object (readonly).
bitrate	Numeric bitrate value

clockpolarity
Clock polarity: idlehigh or idlelow.

clockphase
Clock phase value: firstedge or secondedge

port
The interface driver port (readonly)

3.9.11 spi_close

spi_close (*spi*) [Loadable Function]
Close the interface and release a file descriptor.

Inputs

spi - instance of *octave_spi* class.

Outputs

None

3.9.12 spi_read

[data, count] = spi_read (*spi*, *n*) [Loadable Function]
Read from spi slave device.

Inputs

spi - instance of *octave_spi* class.
n - number of bytes to attempt to read of type Integer.

Outputs

The spi_read() shall return number of bytes successfully read in *count* as Integer and the bytes themselves in *data* as uint8 array.

3.9.13 spi_write

n = spi_write (*spi*, *data*) [Loadable Function]
Write data to a spi slave device.

Inputs

spi - instance of *octave_spi* class.
data - data, of type uint8, to be written to the slave device.

Outputs

Upon successful completion, spi_write() shall return the number of bytes written as the result *n*.

3.9.14 spi_writeAndRead

rddata = spi_writeAndRead (*spi*, *wrdata*) [Loadable Function]
Write data to a spi slave device and then read same number of values.

Inputs

spi - instance of *octave_spi* class.
wrdata - data, of type uint8, to be written to the slave device.

Outputs

Upon successful completion, `spi_writeAndRead()` shall return the bytes read.

3.10 TCP (Deprecated)

3.10.1 @octave_tcp/fclose

`res = fclose (obj)` [Function File]
Closes TCP connection *obj*

3.10.2 @octave_tcp/flush

`data = flush (dev)`
`data = flush (dev, "input")`
`data = flush (dev, "output")`
Flush the tcp socket buffers

Inputs

dev - connected tcp device

If an additional parameter is provided of "input" or "output", then only the input or output buffer will be flushed

Outputs

None

See also: `serialport`.

3.10.3 @octave_tcp/flushinput

`flushinput (tcp)` [Loadable Function]
Flush the pending input, which will also make the BytesAvailable property be 0.

Inputs

tcp - instance of *octave_tcp* class.

Outputs

None.

See also: `flushoutput`.

3.10.4 @octave_tcp/flushoutput

`flushoutput (tcp)` [Loadable Function]
Flush the output buffer.

Inputs

tcp - instance of *octave_tcp* class.

Outputs

None.

See also: `flushinput`.

3.10.5 @octave_tcp/fopen

`res = fopen (obj) (dummy)` [Function File]
 Opens TCP connection *obj*

This currently is a dummy function to improve compatibility to MATLAB

3.10.6 @octave_tcp/fprintf

`numbytes = fprintf (obj, template ...)` [Function File]
 Writes formatted string *template* using optional parameters to TCP instrument

Inputs

obj is a TCP object.

template Format template string

Outputs

Number of characters written

3.10.7 @octave_tcp/fread

`data = fread (obj)` [Function File]
`data = fread (obj, size)` [Function File]
`data = fread (obj, size, precision)` [Function File]
`[data,count] = fread (obj, ...)` [Function File]
`[data,count,errmsg] = fread (obj, ...)` [Function File]
 Reads *data* from TCP instrument

Inputs

obj is a TCP object.

size Number of values to read. (Default: 100).

precision precision of data.

Outputs

data data read.

count values read.

errmsg read operation error message.

3.10.8 @octave_tcp/fwrite

`numbytes = fwrite (obj, data)` [Function File]
`numbytes = fwrite (obj, data, precision)` [Function File]
 Writes *data* to TCP instrument

Inputs

obj is a TCP object.

data data to write.

precision precision of data.

Outputs

returns number of bytes written.

3.10.9 @octave_tcp/get

```
struct = get (tcp) [Function File]
field = get (tcp, property) [Function File]
```

Get the properties of *tcp* object.

Inputs

tcp - instance of *octave_tcp* class.
property - name of property.

Outputs

When *property* was specified, return the value of that property.
 otherwise return the values of all properties as a structure.

See also: @octave_tcp/set.

3.10.10 @octave_tcp/read

```
data = read (obj) [Function File]
data = read (obj, size) [Function File]
data = read (obj, size, datatype) [Function File]
```

Reads *data* from TCP instrument

Inputs

obj is a TCP object.
size Number of values to read. (Default: 100).
datatype datatype of data.

Outputs

data data read.

3.10.11 @octave_tcp/set

```
set (obj, property, value) [Function File]
set (obj, property, value, ...) [Function File]
```

Set the properties of *tcp* object.

Inputs

If *property* is a cell so must be *value*, it sets the values of all matching properties.
 The function also accepts property-value pairs.

Properties

'name' Set the name for the tcp socket.
 'remotehost' Set the remote host name for the tcp socket.
 'remoteport' Set the remote port for the tcp socket.
 'timeout' Set the timeout value in seconds. Value of -1 means a blocking call.

Outputs

None

See also: @octave_tcp/get.

3.10.12 @octave_tcp/write

```
numbytes = write(obj, data) [Function File]
numbytes = write(obj, data, datatype) [Function File]
```

Writes *data* to TCP instrument

Inputs

obj is a TCP object.

data data to write.

datatype datatype of data. If not specified, it defaults to "uint8".

Outputs

returns number of bytes written.

3.10.13 tcp

```
tcp = tcp () [Loadable Function]
tcp = tcp (ipaddress) [Loadable Function]
tcp = tcp (ipaddress, port) [Loadable Function]
tcp = tcp (ipaddress, port, timeout) [Loadable Function]
tcp = tcp (ipaddress, [propertyname, propertyvalue]) [Loadable Function]
tcp = tcp (ipaddress, port, [propertyname, [Loadable Function]
    propertyvalue])
```

Open tcp interface.

Inputs

ipaddress - the ip address of type String. If omitted defaults to '127.0.0.1'.

port - the port number to connect. If omitted defaults to 23.

timeout - the interface timeout value. If omitted defaults to blocking call.

propname,propvalue - property name/value pairs.

Known input properties:

name name value

timeout Numeric timeout value or -1 to wait forever

Outputs

The tcp() shall return instance of *octave_tcp* class as the result *tcp*.

Properties

The tcp object has the following public properties:

name name assigned to the tcp object

type instrument type 'tcp' (readonly)

localport local port number (readonly)

remoteport
 remote port number

`remotehost`
 remote host
`status` status of the object 'open' or 'closed' (readonly)
`timeout` timeout value in seconds used for waiting for data
`bytesavailable`
 number of bytes currently available to read (readonly)

3.10.14 tcp_close

`tcp_close (tcp)` [Loadable Function]
 Close the interface and release a file descriptor.

Inputs

tcp - instance of *octave_tcp* class.

Outputs

None

3.10.15 tcp_read

`[data, count] = tcp_read (tcp, n, timeout)` [Loadable Function]
 Read from tcp interface.

Inputs

tcp - instance of *octave_tcp* class.
n - number of bytes to attempt to read of type Integer
timeout - timeout in ms if different from default of type Integer

Outputs

count - number of bytes successfully read as an Integer
data - data bytes themselves as uint8 array.

3.10.16 tcp_timeout

`tcp_timeout (tcp, timeout)` [Loadable Function]
`t = tcp_timeout (tcp)` [Loadable Function]
 Set new or get existing tcp interface timeout parameter used for `tcp_read()` requests. The timeout value is specified in milliseconds.

Inputs

tcp - instance of *octave_tcp* class.
timeout - `tcp_read()` timeout value in milliseconds. Value of -1 means a blocking call.

Outputs

If *timeout* parameter is omitted, the `tcp_timeout()` shall return current timeout value as the result *t*.

3.10.17 tcp_write

`n = tcp_write (tcp, data)` [Loadable Function]
 Write data to a tcp interface.

Inputs

tcp - instance of *octave_tcp* class.

data - data to be written to the tcp interface. Can be either of String or uint8 type.

Outputs

Upon successful completion, *tcp_write()* shall return the number of bytes written as the result *n*.

3.10.18 tcpip

tcp = *tcpip* (*host*, [*port*], [*PropertyName*,
 PropertyValue...]) [Function File]

Matlab compatible wrapper to the tcp interface.

NOTE: *tcpip* has been deprecated. Use *tcpclient* instead

Inputs

host - the host name or ip.

port - the port number to connect. If omitted defaults to 80.

PropertyName, *PropertyValue* - Optional property name, value pairs to set on the tcp object.

Properties

Currently the only known properties are "timeout" and "name".

Outputs

tcpip will return an instance of *octave_tcp* class as the result.

3.11 TCP Client

3.11.1 @octave_tcpclient/configureTerminator

configureTerminator (*tcp*, *term*) [Function File]

configureTerminator (*tcp*, *readterm*, *writeterm*) [Function File]

Set terminator on a *tcpclient* object for ASCII string manipulation

Inputs

tcp - *tcpclient* object

term - terminal value for both read and write

readterm = terminal value type for read data

writeterm = terminal value for written data

The terminal can be either strings "cr", "lf" (default), "lf/cr" or an integer between 0 to 255.

Outputs

None

See also: *tcpport*.

3.11.2 @octave_tcpclient/flush

data = *flush* (*dev*)

data = *flush* (*dev*, "input")

```
data = flush (dev, "output")
```

Flush the tcpclient socket buffers

Inputs

dev - connected tcpclient device

If an additional parameter is provided of "input" or "output", then only the input or output buffer will be flushed

Outputs

None

See also: serialport.

3.11.3 @octave_tcpclient/get

```
struct = get (tcpclient) [Function File]
```

```
field = get (tcpclient, property) [Function File]
```

Get the properties of tcpclient object.

Inputs

tcpclient - instance of *octave_tcpclient* class.

property - name of property.

Outputs

When *property* was specified, return the value of that property.
otherwise return the values of all properties as a structure.

See also: @octave_tcpclient/set.

3.11.4 @octave_tcpclient/read

```
data = read (obj) [Function File]
```

```
data = read (obj, size) [Function File]
```

```
data = read (obj, size, datatype) [Function File]
```

Reads *data* from TCP instrument

Inputs

obj is a TCP object.

size Number of values to read. (Default: NumBytesAvailable).

datatype datatype of data.

Outputs

data data read.

3.11.5 @octave_tcpclient/set

```
set (obj, property, value) [Function File]
```

```
set (obj, property, value, ...) [Function File]
```

Set the properties of tcpclient object.

Inputs

If *property* is a cell so must be *value*, it sets the values of all matching properties.

The function also accepts property-value pairs.

Properties

'Name' Set the name for the tcpclient socket.

'UserData' Set user data for the tcpclient socket.

'Timeout' Set the timeout value in seconds. Value of -1 means a blocking call.

'Tag' Set user tag to identify the port

Outputs

None

See also: @octave_tcpclient/get.

3.11.6 @octave_tcpclient/write

`numbytes = write(obj, data)` [Function File]

`numbytes = write(obj, data, datatype)` [Function File]

Writes *data* to TCP instrument

Inputs

obj is a TCPClient object.

data data to write.

datatype datatype of data. If not specified, it defaults to "uint8".

Outputs

returns number of bytes written.

3.11.7 tcpclient

`tcpclient = tcpclient(ipaddress, port)` [Loadable Function]

`tcpclient = tcpclient(ipaddress, port, [propertyname, propertyvalue])` [Loadable Function]

Open tcpclient interface.

Inputs

ipaddress - the ip address of type String.

port - the port number to connect.

propname,propvalue - property name/value pairs.

Known input properties:

Name name value

Tag tag value

Timeout Numeric timeout value or -1 to wait forever

EnableTransferDelay

Boolean to enable or disable the nagle algorithm for delay transfer.

UserData User data value.

Outputs

The `tcpclient()` shall return instance of *octave_tcpclient* class as the result *tcpclient*.

Properties

The `tcpclient` object has the following public properties:

Name	name assigned to the <code>tcpclient</code> object
Tag	user tag assigned to the <code>tcpclient</code> object
Type	instrument type 'tcpclient' (readonly)
Port	remote port number (Readonly)
Address	remote host address (Readonly)
Status	status of the object 'open' or 'closed' (readonly)
Timeout	timeout value in seconds used for waiting for data
NumBytesAvailable	number of bytes currently available to read (readonly)
NumBytesWritten	number of bytes currently available to read (readonly)
ByteOrder	Byte order for data (currently not used)
Terminator	Terminator value used for string data (currently not used)
UserData	User data
EnableTransferDelay	Bool for whether transfer delay is enabled. (Read only)

3.12 TCP Server

3.12.1 @octave_tcpserver/configureTerminator

`configureTerminator (tcp, term)` [Function File]
`configureTerminator (tcp, readterm, writeterm)` [Function File]
 Set terminator on a `tcpserver` object for ASCII string manipulation

Inputs

tcp - `tcpserver` object
term - terminal value for both read and write
readterm = terminal value type for read data
writeterm = terminal value for written data

The terminal can be either strings "cr", "lf" (default), "lf/cr" or an integer between 0 to 255.

Outputs

None

See also: `tcpport`.

3.12.2 @octave_tcpserver/flush

```
data = flush (dev)
data = flush (dev, "input")
data = flush (dev, "output")
```

Flush the tcpserver socket buffers

Inputs

dev - connected tcpserver device

If an additional parameter is provided of "input" or "output", then only the input or output buffer will be flushed

Outputs

None

See also: serialport.

3.12.3 @octave_tcpserver/get

```
struct = get (tcpserver) [Function File]
field = get (tcpserver, property) [Function File]
```

Get the properties of tcpserver object.

Inputs

tcpserver - instance of *octave_tcpserver* class.

property - name of property.

Outputs

When *property* was specified, return the value of that property.
otherwise return the values of all properties as a structure.

See also: @octave_tcpserver/set.

3.12.4 @octave_tcpserver/read

```
data = read (obj) [Function File]
data = read (obj, size) [Function File]
data = read (obj, size, datatype) [Function File]
```

Reads *data* from TCP instrument

Inputs

obj is a TCP Server object.

size Number of values to read. (Default: NumBytesAvailable).

datatype datatype of data.

Outputs

data data read.

3.12.5 @octave_tcpserver/set

`set (obj, property,value)` [Function File]
`set (obj, property,value,...)` [Function File]
 Set the properties of tcpserver object.

Inputs

If *property* is a cell so must be *value*, it sets the values of all matching properties.
 The function also accepts property-value pairs.

Properties

'Name' Set the name for the tcpserver socket.
 'UserData' Set user data for the tcpserver socket.
 'Timeout' Set the timeout value in seconds. Value of -1 means a blocking call.

Outputs

None

See also: @octave_tcpserver/get.

3.12.6 @octave_tcpserver/write

`numbytes = write (obj, data)` [Function File]
`numbytes = write (obj, data, datatype)` [Function File]
 Writes *data* to TCP instrument

Inputs

obj is a TCPServer object.
data data to write.
datatype datatype of data. If not specified, it defaults to "uint8".

Outputs

returns number of bytes written.

3.12.7 tcpserver

`tcpserver = tcpserver (ipaddress, port)` [Loadable Function]
`tcpserver = tcpserver (port)` [Loadable Function]
`tcpserver = tcpserver (... , [propertyname, propertyvalue])` [Loadable Function]
 Open tcpserver interface.

Inputs

ipaddress - the ip address of type String.
port - the port number to bind.
propname,propvalue - property name/value pairs.

Known input properties:

Name name value
 Timeout Numeric timeout value or -1 to wait forever
 UserData User data value.

Outputs

The `tcpserver()` shall return instance of `octave_tcpserver` class as the result `tcpserver`.

Properties

The `tcpserver` object has the following public properties:

`Connected` boolean flag for when connected to a client (Readonly)

`ClientPort` connected client port number (Readonly)

`ClientAddress`
connected client address (Readonly)

`Name` name assigned to the `tcpserver` object

`Type` instrument type 'tcpserver' (readonly)

`ServerPort`
server port number (Readonly)

`ServerAddress`
server address (Readonly)

`Status` status of the object 'open' or 'closed' (readonly)

`Timeout` timeout value in seconds used for waiting for data

`NumBytesAvailable`
number of bytes currently available to read (readonly)

`NumBytesWritten`
number of bytes currently available to read (readonly)

`ByteOrder`
Byte order for data (currently not used)

`Terminator`
Terminator value used for string data (currently not used)

`UserData` User data

3.13 UDP (Deprecated)

3.13.1 @octave_udp/fclose

`res = fclose (obj)` [Function File]
Closes UDP connection `obj`

3.13.2 @octave_udp/flush

`data = flush (dev)`
`data = flush (dev, "input")`
`data = flush (dev, "output")`
Flush the udp socket buffers

Inputs

`dev` - open udp device

If an additional parameter is provided of "input" or "output", then only the input or output buffer will be flushed

Outputs

None

See also: `udp`.

3.13.3 @octave_udp/flushinput

`flushinput (udp)` [Loadable Function]

Flush the pending input, which will also make the BytesAvailable property be 0.

Inputs

udp - instance of *octave_udp* class.

Outputs

None

See also: `flushoutput`.

3.13.4 @octave_udp/flushoutput

`flushoutput (udp)` [Loadable Function]

Flush the output buffer.

Inputs

udp - instance of *octave_udp* class.

Outputs

None

See also: `flushinput`.

3.13.5 @octave_udp/fopen

`res = fopen (obj) (dummy)` [Function File]

Opens UDP connection *obj* This currently is a dummy function to improve compatibility to MATLAB

3.13.6 @octave_udp/fprintf

`numbytes = fprintf (obj, template ...)` [Function File]

Writes formatted string *template* using optional parameters to UDP instrument

Inputs

obj is a UDP object.

template Format template string.

Outputs

numbytes is the number of bytes written to the device

3.13.7 @octave_udp/fread

`data = fread (obj)` [Function File]

`data = fread (obj, size)` [Function File]

`data = fread (obj, size, precision)` [Function File]

<code>[data,count] = fread (obj, ...)</code>	[Function File]
<code>[data,count,errmsg] = fread (obj, ...)</code>	[Function File]

Reads *data* from UDP instrument

Inputs

obj is a UDP object.
size Number of values to read. (Default: 100).
precision precision of data.

Outputs

data data values.
count number of values read.
errmsg read operation error message.

3.13.8 @octave_udp/fwrite

<code>numbytes = fwrite (obj, data)</code>	[Function File]
<code>numbytes = fwrite (obj, data, precision)</code>	[Function File]

Writes *data* to UDP instrument

Inputs

obj is a UDP object.
data data to write.
precision precision of data.

Outputs

returns number of bytes written.

3.13.9 @octave_udp/get

<code>struct = get (udp)</code>	[Function File]
<code>field = get (udp, property)</code>	[Function File]

Get the properties of udp object.

Inputs

udp - instance of *octave_udp* class.

property - name of property.

Outputs

When *property* was specified, return the value of that property.
 otherwise return the values of all properties as a structure.

See also: @octave_udp/set.

3.13.10 @octave_udp/read

`data = read (obj)` [Function File]
`data = read (obj, size)` [Function File]
`data = read (obj, size, datatype)` [Function File]
 Reads *data* from UDP instrument

Inputs

obj is a UDP object.
size Number of values to read. (Default: BytesAvailable).
datatype datatype of data.

Outputs

data data read.

3.13.11 @octave_udp/set

`set (obj, property,value)` [Function File]
`set (obj, property,value,...)` [Function File]
 Set the properties of udp object.

Inputs

obj - instance of *octave_udp* class.
property - name of property.

If *property* is a cell so must be *value*, it sets the values of all matching properties.
 The function also accepts property-value pairs.

Properties

'name' Set the name for the udp socket.
'remotehost' Set the remote host name for the udp socket.
'remoteport' Set the remote port for the udp socket.
'timeout' Set the timeout value in seconds. Value of -1 means a blocking call.

Outputs

None

See also: @octave_udp/get.

3.13.12 @octave_udp/write

`numbytes = write (obj, data)` [Function File]
`numbytes = write (obj, data, destinationAddress, destinationPort)` [Function File]
`numbytes = write (obj, data, datatype)` [Function File]
`numbytes = write (obj, data, datatype, destinationAddress, destinationPort)` [Function File]
 Writes *data* to UDP instrument

Inputs

obj is a UDP object.

data data to write.

datatype datatype of data. If not specified defaults to uint8.

destinationAddress ipaddress to send to. If not specified, use the remote address.

destinationPort port to send to. If not specified, use the remote port.

Outputs

returns number of bytes written.

3.13.13 udp

`udp = udp ()` [Loadable Function]

`udp = udp (remoteipaddress, remoteport)` [Loadable Function]

`udp = udp (remoteipaddress, remoteport, [propertyname, propertyvalue ...])` [Loadable Function]

Open udp interface.

Inputs

remoteipaddress - the ip address of type String. If omitted defaults to '127.0.0.1'.

remoteport - the port number to connect. If omitted defaults to 23.

localport - the local port number to bind. If omitted defaults to 0

propertyname, propertyvalue - property name/value pair

Outputs

The `udp()` shall return instance of *octave_udp* class as the result *udp*.

Properties

The *udp* object has the following public properties:

name name assigned to the *udp* object

type instrument type 'udp' (readonly)

localport local port number (readonly)

localhost local host address (readonly)

remoteport
remote port number

remotehost
remote host

status status of the object 'open' or 'closed' (readonly)

timeout timeout value in seconds used for waiting for data

bytesavailable
number of bytes currently available to read (readonly)

3.13.14 udp_close

`udp_close (udp)` [Loadable Function]

Close the interface and release a file descriptor.

Inputs

udp - instance of *octave_udp* class.

Inputs

None

3.13.15 udp_demo

`result = udp_demo ()` [Function File]

Run test SNTP demonstration for udp class

See also: `udp`.

3.13.16 udp_read

`[data, count] = udp_read (udp, n, timeout)` [Loadable Function]

Read from udp interface.

Inputs

udp - instance of *octave_udp* class.

n - number of bytes to attempt to read of type Integer

timeout - timeout in ms if different from default of type Integer

Outputs

The `udp_read()` shall return number of bytes successfully read in *count* as Integer and the bytes themselves in *data* as uint8 array.

3.13.17 udp_timeout

`udp_timeout (udp, timeout)` [Loadable Function]

`t = udp_timeout (udp)` [Loadable Function]

Set new or get existing udp interface timeout parameter used for `udp_read()` requests. The timeout value is specified in milliseconds.

Inputs

udp - instance of *octave_udp* class.

timeout - `udp_read()` timeout value in milliseconds. Value of -1 means a blocking call.

Outputs

If *timeout* parameter is omitted, the `udp_timeout()` shall return current timeout value as the result *t*.

3.13.18 udp_write

`n = udp_write (udp, data)` [Loadable Function]

Write data to a udp interface.

Inputs

udp - instance of *octave_udp* class.

data - data to be written to the udp interface. Can be either of String or uint8 type.

Outputs

Upon successful completion, `udp_write()` shall return the number of bytes written as the result *n*.

3.14 UDP Port

3.14.1 @octave_udpport/configureMulticast

```
data = configureMulticast((dev, address)
data = configureMulticast((dev, "off")
    Configure udpport device to receive multicast data
```

Inputs

dev - open udpport device

If *address* is 'off' disable udp multicast. Otherwise it is the multicast address to use.

Outputs

None

See also: udpport.

3.14.2 @octave_udpport/configureTerminator

```
configureTerminator (udp, term) [Function File]
configureTerminator (udp, readterm, writeterm) [Function File]
    Set terminator for ASCII string manipulation
```

Inputs

udp - udpport object

term - terminal value for both read and write

readterm = terminal value type for read data

writeterm = terminal value for written data

The terminal can be either strings "cr", "lf" (default), "lf/cr" or an integer between 0 to 255.

Outputs

None

See also: udpport.

3.14.3 @octave_udpport/flush

```
data = flush (dev)
data = flush (dev, "input")
data = flush (dev, "output")
    Flush the udpport socket buffers
```

Inputs

dev - open udpport device

If an additional parameter is provided of "input" or "output", then only the input or output buffer will be flushed

Outputs

None

See also: udpport.

3.14.4 @octave_udpport/fprintf

numbytes = fprintf (*obj*, *template* ...) [Function File]

Writes formatted string *template* using optional parameters to UDP instrument

Inputs

obj is a UDPPort object.

template Format template string.

Outputs

numbytes is the number of bytes written to the device

3.14.5 @octave_udpport/fread

data = fread (*obj*) [Function File]

data = fread (*obj*, *size*) [Function File]

data = fread (*obj*, *size*, *precision*) [Function File]

[*data*,*count*] = fread (*obj*, ...) [Function File]

[*data*,*count*,*errmsg*] = fread (*obj*, ...) [Function File]

Reads *data* from UDP instrument

Inputs

obj is a UDP port object.

size Number of values to read. (Default: 100).

precision precision of data.

Outputs

data data values.

count number of values read.

errmsg read operation error message.

3.14.6 @octave_udpport/fwrite

numbytes = fwrite (*obj*, *data*) [Function File]

numbytes = fwrite (*obj*, *data*, *precision*) [Function File]

Writes *data* to UDP instrument

Inputs

obj is a UDP port object.

data data to write.

precision precision of data.

Outputs

returns number of bytes written.

3.14.7 @octave_udpport/get

struct = get (*udpport*) [Function File]

field = get (*udpport*, *property*) [Function File]

Get the properties of *udpport* object.

Inputs

udpport - instance of *octave_udpport* class.

property - name of property.

Outputs

When *property* was specified, return the value of that property.
otherwise return the values of all properties as a structure.

See also: @octave_udpport/set.

3.14.8 @octave_udpport/read

<code>data = read (obj)</code>	[Function File]
<code>data = read (obj, size)</code>	[Function File]
<code>data = read (obj, size, datatype)</code>	[Function File]

Reads *data* from UDP instrument

Inputs

obj is a UDP object.

size Number of values to read. (Default: BytesAvailable).

datatype datatype of data.

Outputs

data data read.

3.14.9 @octave_udpport/set

<code>set (obj, property,value)</code>	[Function File]
<code>set (obj, property,value,...)</code>	[Function File]

Set the properties of udpport object.

Inputs

obj - instance of *octave_udpport* class.

property - name of property.

If *property* is a cell so must be *value*, it sets the values of all matching properties.

The function also accepts property-value pairs.

Properties

'Name' Set the name for the udpport socket.

'UserData'
Set the user data of the object.

'Tag' Set the user tag to identify the port.

'Timeout' Set the timeout value in seconds. Value of -1 means a blocking call.

Outputs

None

See also: @octave_udpport/get.

3.14.10 @octave_udpport/write

`numbytes = write(obj, data)` [Function File]

`numbytes = write(obj, data, destinationAddress, destinationPort)` [Function File]

`numbytes = write(obj, data, datatype)` [Function File]

`numbytes = write(obj, data, datatype, destinationAddress, destinationPort)` [Function File]

Writes *data* to UDP instrument

Inputs

obj is a UDPPort object.

data data to write.

datatype datatype of data. If not specified defaults to uint8.

destinationAddress ipaddress to send to. If not specified, use the previously used remote address.

destinationPort port to send to. If not specified, use the remote port.

Outputs

returns number of bytes written.

3.14.11 @octave_udpport/writeline

`writeline(dev, data)`

`writeline(dev, data, destaddr, destport)`

Write data to a udpport including terminator value

Inputs

dev - connected device

data - ASCII data to write

destaddr - Destination address

destport - Destination port

Where the address and port is not specified, the previously used address and port is used.

Outputs

None

See also: flushoutput.

3.14.12 udpport

`udp = udpport()` [Loadable Function]

`udp = udpport(propertyname, propertyvalue ...)` [Loadable Function]

Open udpport interface.

Inputs

propertyname, propertyvalue - property name/value pair

Known input properties:

Name name assigned to the udp object
 LocalPort local port number
 LocalHost local host address
 Timeout timeout value in seconds used for waiting for data
 EnablePortSharing
 Boolean if the socket has port sharing enabled (readonly)

Outputs

The `udpport()` shall return instance of *octave_udp* class as the result *udp*.

Properties

The *udp* object has the following public properties:

Name name assigned to the udp object
 Tag user tag assigned to the udp object
 Type instrument type 'udpport' (readonly)
 LocalPort local port number (readonly)
 LocalHost local host address (readonly)
 Status status of the object 'open' or 'closed' (readonly)
 Timeout timeout value in seconds used for waiting for data
 NumBytesAvailable
 number of bytes currently available to read (readonly)
 MulticastGroup
 multicast group socket is subscribed to (readonly)
 EnableMulticast
 Boolean if the socket has any multicast group it is subscribed to (readonly)
 EnablePortSharing
 Boolean if the socket has port sharing enabled (readonly)
 Terminator
 Terminator value used for string data (currently not used)

3.15 USBTMC

3.15.1 @octave_usbtmc/fclose

res = `fclose` (*obj*)

[Function File]

Closes USBTMC connection *obj*

Inputs

obj is a *usbtmc* object.

3.15.2 @octave_usbtmc/fopen

`res = fopen (obj) (dummy)` [Function File]
 Opens USBTMC connection *obj* This currently is a dummy function to improve compatibility to MATLAB

3.15.3 @octave_usbtmc/fread

`data = fread (obj)` [Function File]
`data = fread (obj, size)` [Function File]
`data = fread (obj, size, precision)` [Function File]
`[data,count] = fread (obj, ...)` [Function File]
`[data,count,errmsg] = fread (obj, ...)` [Function File]
 Reads *data* from usbtmc instrument

Inputs

obj is a usbtmc object.
size Number of values to read. (Default: 100).
precision precision of data.

Outputs

data The read data.
count values read.
errmsg read operation error message.

3.15.4 @octave_usbtmc/fwrite

`numbytes = fwrite (obj, data)` [Function File]
`numbytes = fwrite (obj, data, precision)` [Function File]
 Writes *data* to an usbtmc instrument

Inputs

obj is a usbtmc object.
data data to write.
precision precision of data.

Outputs

returns number of bytes written.

3.15.5 usbtmc

`usbtmc = usbtmc (path)` [Loadable Function]
 Open usbtmc interface.

Inputs

path - the interface path of type String. If omitted defaults to `'/dev/usbtmc0'`.

Outputs

The `usbtmc()` shall return instance of *octave_usbtmc* class as the result *usbtmc*.

3.15.6 usbtmc_close

`usbtmc_close (usbtmc)` [Loadable Function]
 Close the interface and release a file descriptor.

Inputs

usbtmc - instance of *octave_usbtmc* class.

Outputs

None

3.15.7 usbtmc_read

`[data, count] = usbtmc_read (usbtmc, n)` [Loadable Function]
 Read from usbtmc slave device.

Inputs

usbtmc - instance of *octave_usbtmc* class.

n - number of bytes to attempt to read of type Integer.

Outputs

count - the number of bytes successfully read as an Integer.

data - the read bytes as a uint8 array.

3.15.8 usbtmc_write

`n = usbtmc_write (usbtmc, data)` [Loadable Function]
 Write data to a usbtmc slave device.

Inputs

usbtmc - instance of *octave_usbtmc* class.

data - data, of type uint8, to be written to the slave device.

Outputs

Upon successful completion, `usbtmc_write()` shall return the number of bytes written as the result *n*.

3.16 VXI11

3.16.1 @octave_vxi11/fclose

`res = fclose (obj)` [Function File]
 Closes VXI11 connection *obj*

3.16.2 @octave_vxi11/fopen

`res = fopen (obj) (dummy)` [Function File]
 Opens VXI11 connection *obj* This currently is a dummy function to improve compatibility to MATLAB

3.16.3 @octave_vxi11/fread

<code>data = fread (obj)</code>	[Function File]
<code>data = fread (obj, size)</code>	[Function File]
<code>data = fread (obj, size, precision)</code>	[Function File]
<code>[data,count] = fread (obj, ...)</code>	[Function File]
<code>[data,count,errmsg] = fread (obj, ...)</code>	[Function File]

Reads *data* from vxi11 instrument

Inputs

obj is a vxi11 object.
size Number of values to read. (Default: 100).
precision precision of data.

Outputs

data The read data.
count values read.
errmsg read operation error message.

3.16.4 @octave_vxi11/fwrite

<code>numbytes = fwrite (obj, data)</code>	[Function File]
<code>numbytes = fwrite (obj, data, precision)</code>	[Function File]

Writes *data* to vxi11 instrument

Inputs

obj is a vxi11 object.
data data to write.
precision precision of data.

Outputs

returns number of bytes written.

3.16.5 vxi11

<code>vxi11 = vxi11 (ip,instr)</code>	[Loadable Function]
---------------------------------------	---------------------

Open vxi11 interface.

ip - the ip address of type String. If omitted defaults to '127.0.0.1'. *instr* - the instrument name of type String. If omitted defaults to 'inst0'.

The `vxi11()` shall return instance of *octave_vxi11* class as the result *vxi11*.

3.16.6 vxi11_close

<code>vxi11_close (vxi11)</code>	[Loadable Function]
----------------------------------	---------------------

Close the interface and release a file descriptor.

vxi11 - instance of *octave_vxi11* class.

3.16.7 `vx11_read`

`[data, count] = vx11_read (vx11, n)` [Loadable Function]

Read from vx11 slave device.

vx11 - instance of *octave_vx11* class.

n - number of bytes to attempt to read of type Integer.

The `vx11_read()` shall return number of bytes successfully read in *count* as Integer and the bytes themselves in *data* as uint8 array.

3.16.8 `vx11_write`

`n = vx11_write (vx11, data)` [Loadable Function]

Write data to a vx11 slave device.

vx11 - instance of *octave_vx11* class.

data - data to be written to the slave device. Can be either of String or uint8 type.

Upon successful completion, `vx11_write()` shall return the number of bytes written as the result *n*.

Appendix A GNU General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works. The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable

section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

- d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything

designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

- c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRIT-

ING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

program Copyright (C) year name of author

```
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.  
This is free software, and you are welcome to redistribute it  
under certain conditions; type 'show c' for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

Index

B

Basic Usage Overview 2

C

clrdevice 11
 Common Functions 7
 configureMulticast 56
 configureTerminator 28, 44, 47, 56
 copyright 65

F

fclose 10, 12, 19, 22, 34, 39, 50, 60, 62
 flush 29, 39, 44, 48, 50, 56
 flushinput 7, 22, 39, 51
 flushoutput 7, 22, 39, 51
 fopen 10, 13, 19, 22, 34, 40, 51, 61, 62
 fprintf 10, 22, 29, 40, 51, 57
 fread 11, 13, 20, 23, 29, 34, 40, 51, 57, 61, 63
 fscanf 11
 Function Reference 7
 fwrite 11, 13, 20, 23, 30, 35, 40, 52, 57, 61, 63

G

General 9
 get 13, 16, 23, 30, 35, 41, 45, 48, 52, 57
 getpinstatus 30
 gpib 11
 gpib_close 11
 gpib_read 12
 gpib_timeout 12
 gpib_write 12
 GPIB 10

I

i2c 14
 i2c_addr 15
 i2c_close 15
 i2c_read 15
 i2c_write 15
 I2C 12
 Installing and loading 1
 instrhelp 9
 instrhwinfo 9

L

Loading 1

M

maskWrite 16
 Modbus 16
 modbus 18

O

Off-line install 1
 Online install 1

P

parallel 20
 Parallel 19
 pp_close 20
 pp_ctrl 21
 pp_data 21
 pp_datadir 21
 pp_stat 22

R

read 16, 31, 36, 41, 45, 48, 53, 58
 readbinblock 7
 readline 7
 Requirements 1
 resolvehost 10

S

serial 27
 Serial (Deprecated) 22
 Serial Port 28
 serialbreak 24, 31
 seriallist 28
 serialport 33
 serialportlist 34
 set 14, 17, 24, 31, 36, 41, 45, 49, 53, 58
 setDTR 32
 setRTS 32
 spi 37
 spi_close 38
 spi_read 38
 spi_write 38
 spi_writeAndRead 38
 SPI 34
 spoll 12
 srl_baudrate 25
 srl_bytesize 25
 srl_close 25
 srl_flush 26
 srl_parity 26
 srl_read 28
 srl_stopbits 26
 srl_timeout 27
 srl_write 28

T

tcp.....	42
tcp_close.....	43
tcp_read.....	43
tcp_timeout.....	43
tcp_write.....	43
TCP (Deprecated).....	39
TCP Client.....	44
TCP Server.....	47
tcpclient.....	46
tcpip.....	44
tcpserver.....	49
trigger.....	12

U

udp.....	54
udp_close.....	54
udp_demo.....	55
udp_read.....	55
udp_timeout.....	55
udp_write.....	55
UDP (Deprecated).....	50
UDP Port.....	56

udpport.....	59
usbtmc.....	61
usbtmc_close.....	62
usbtmc_read.....	62
usbtmc_write.....	62
USBTCM.....	60

V

vxi11.....	63
vxi11_close.....	63
vxi11_read.....	64
vxi11_write.....	64
VXI11.....	62

W

warranty.....	65
Windows install.....	1
write.....	17, 33, 36, 42, 46, 49, 53, 59
writeAndRead.....	37
writebinblock.....	8
writeline.....	8, 59
writeread.....	8
writeRead.....	18