buffer. If `BytesAvailableFcnMode` is `terminator`, then the callback function executes every time the character specified by the `Terminator` property is read.

This event can be generated only during an asynchronous read operation.

Error Event. An error event is generated immediately after an error occurs.

This event executes the callback function specified for the `ErrorFcn` property. It can be generated only during an asynchronous read or write operation.

An error event is generated when a timeout occurs. A timeout occurs if a read or write operation does not successfully complete within the time specified by the `Timeout` property. An error event is not generated for configuration errors such as setting an invalid property value.

Output-Empty Event. An output-empty event is generated immediately after the output buffer is empty.

This event executes the callback function specified for the `OutputEmptyFcn` property. It can be generated only during an asynchronous write operation.

Pin Status Event. A pin status event is generated immediately after the state (pin value) changes for the CD, CTS, DSR, or RI pins. Refer to "Serial Port Signals and Pin Assignments" on page 9-6 for a description of these pins.

This event executes the callback function specified for the `PinStatusFcn` property. It can be generated for both synchronous and asynchronous read and write operations.

Timer Event. A timer event is generated when the time specified by the `TimerPeriod` property passes. Time is measured relative to when the serial port object is connected to the device.

This event executes the callback function specified for the `TimerFcn` property. Note that some timer events might not be processed if your system is significantly slowed or if the `TimerPeriod` value is too small.

Storing Event Information

You can store event information in a callback function or in a record file. Event information is stored in a callback function using two fields: `Type` and `Data`. The `Type` field contains the event type, while the `Data` field contains event-specific information. As described in "Creating and Executing Callback Functions" on

Ex.1049 / Page 664 of 748
Intel Corporation et al

page 9-52, these two fields are associated with a structure that you define in the callback function header. Refer to "Debugging: Recording Information to Disk" on page 9-62 to learn about recording data and event information to a record file.

The event types and the values for the `Type` and `Data` fields are given below.

Table 9-10: Event Information

Event Type	Field	Field Value
Break interrupt	`Type`	`BreakInterrupt`
	`Data.AbsTime`	day-month-year hour:minute:second
Bytes available	`Type`	`BytesAvailable`
	`Data.AbsTime`	day-month-year hour:minute:second
Error	`Type`	`Error`
	`Data.AbsTime`	day-month-year hour:minute:second
	`Data.Message`	An error string
Output empty	`Type`	`OutputEmpty`
	`Data.AbsTime`	day-month-year hour:minute:second
Pin status	`Type`	`PinStatus`
	`Data.AbsTime`	day-month-year hour:minute:second
	`Data.Pin`	`CarrierDetect`, `ClearToSend`, `DataSetReady`, or `RingIndicator`
	`Data.PinValue`	`on` or `off`
Timer	`Type`	`Timer`
	`Data.AbsTime`	day-month-year hour:minute:second

The `Data` field values are described below.

Ex.1049 / Page 665 of 748
Intel Corporation et al

The AbsTime Field. `AbsTime` is defined for all events, and indicates the absolute time the event occurred. The absolute time is returned using the `clock` format.

day-month-year hour:minute:second

The Pin Field. `Pin` is used by the pin status event to indicate if the CD, CTS, DSR, or RI pins changed state. Refer to "Serial Port Signals and Pin Assignments" on page 9-6 for a description of these pins.

The PinValue Field. `PinValue` is used by the pin status event to indicate the state of the CD, CTS, DSR, or RI pins. Possible values are `on` or `off`.

The Message Field. `Message` is used by the error event to store the descriptive message that is generated when an error occurs.

Creating and Executing Callback Functions

You can specify the callback function to be executed when a specific event type occurs by including the name of the M-file as the value for the associated callback property. You can specify the callback function as a function handle or as a string cell array element. Function handles are described in the `function_handle` reference pages.

For example, to execute the callback function `mycallback` every time the terminator is read from your device

```
s.BytesAvailableFcnMode = 'terminator';
s.BytesAvailableFcn = @mycallback;
```

Alternatively, you can specify the callback function as a cell array.

```
s.BytesAvailableFcn = {'mycallback'};
```

M-file callback functions require at least two input arguments. The first argument is the serial port object. The second argument is a variable that captures the event information given in Table 9-10, Event Information, on page 9-51. This event information pertains only to the event that caused the callback function to execute. The function header for `mycallback` is shown below.

```
function mycallback(obj,event)
```


Ex.1049 / Page 666 of 748
Intel Corporation et al

You pass additional parameters to the callback function by including both the callback function and the parameters as elements of a cell array. For example, to pass the MATLAB variable `time` to `mycallback`

```
time = datestr(now,0);
s.BytesAvailableFcnMode = 'terminator';
s.BytesAvailableFcn = {@mycallback,time};
```

Alternatively, you can specify the callback function as a string in the cell array.

```
s.BytesAvailableFcn = {'mycallback',time};
```

The corresponding function header is

```
function mycallback(obj,event,time)
```

If you pass additional parameters to the callback function, then they must be included in the function header after the two required arguments.

Note You can also specify the callback function as a string. In this case, the callback is evaluated in the MATLAB workspace and no requirements are made on the input arguments of the callback function.

Enabling Callback Functions After They Error

If an error occurs while a callback function is executing, then:

- The callback function is automatically disabled.
- A warning is displayed at the command line, indicating that the callback function is disabled.

If you want to enable the same callback function, you can set the callback property to the same value or you can disconnect the object with the `fclose` function. If you want to use a different callback function, the callback will be enabled when you configure the callback property to the new value.

Example: Using Events and Callbacks

This example uses the M-file callback function `instrcallback` to display event-related information to the command line when a bytes-available event or an output-empty event occurs.

1 **Create a serial port object –** Create the serial port object `s` associated with serial port COM1.

```
s = serial('COM1');
```

2 **Connect to the device –** Connect `s` to the Tektronix TDS 210 oscilloscope. Because the default value for the `ReadAsyncMode` property is `continuous`, data is asynchronously returned to the input buffer as soon as it is available from the instrument.

```
fopen(s)
```

3 **Configure properties –** Configure `s` to execute the callback function `instrcallback` when a bytes-available event or an output-empty event occurs. Because `instrcallback` requires the serial port object and event information to be passed as input arguments, the callback function is specified as a function handle.

```
s.BytesAvailableFcnMode = 'terminator';
s.BytesAvailableFcn = @instrcallback;
s.OutputEmptyFcn = @instrcallback;
```

4 **Write and read data –** Write the `RS232?` command asynchronously to the oscilloscope. This command queries the RS-232 settings and returns the baud rate, the software flow control setting, the hardware flow control setting, the parity type, and the terminator.

```
fprintf(s,'RS232?','async')
```

`instrcallback` is called after the `RS232?` command is sent, and when the terminator is read. The resulting displays are shown below.

```
OutputEmpty event occurred at 17:37:21 for the object:
Serial-COM1.

BytesAvailable event occurred at 17:37:21 for the object:
Serial-COM1.
```

Read the data from the input buffer.

```
out = fscanf(s)
out =
9600;0;0;NONE;LF
```

5 **Disconnect and clean up –** When you no longer need `s`, you should disconnect it from the instrument, and remove it from memory and from the MATLAB workspace.

```
fclose(s)
delete(s)
clear s
```


Ex.1049 / Page 669 of 748
Intel Corporation et al

Using Control Pins

As described in "Serial Port Signals and Pin Assignments" on page 9-6, 9-pin serial ports include six control pins. These control pins allow you to:

- Signal the presence of connected devices
- Control the flow of data

The properties associated with the serial port control pins are given below.

Table 9-11: Control Pin Properties

Property Name	Description
`DataTerminalReady`	Specify the state of the DTR pin
`FlowControl`	Specify the data flow control method to use
`PinStatus`	Indicate the state of the CD, CTS, DSR, and RI pins
`RequestToSend`	Specify the state of the RTS pin

Signaling the Presence of Connected Devices

DTE's and DCE's often use the CD, DSR, RI, and DTR pins to indicate whether a connection is established between serial port devices. Once the connection is established, you can begin to write or read data.

You can monitor the state of the CD, DSR, and RI pins with the `PinStatus` property. You can specify or monitor the state of the DTR pin with the `DataTerminalReady` property.

The following example illustrates how these pins are used when two modems are connected to each other.

Example: Connecting Two Modems

This example connects two modems to each other via the same computer, and illustrates how you can monitor the communication status for the computer-modem connections, and for the modem-modem connection. The first modem is connected to COM1, while the second modem is connected to COM2.

Ex.1049 / Page 670 of 748
Intel Corporation et al

1 **Create the serial port objects –** After the modems are powered on, the serial port object `s1` is created for the first modem, and the serial port object `s2` is created for the second modem.

```
s1 = serial('COM1');
s2 = serial('COM2');
```

2 **Connect to the devices –** `s1` and `s2` are connected to the modems. Because the default value for the `ReadAsyncMode` property is `continuous`, data is asynchronously returned to the input buffers as soon as it is available from the modems.

```
fopen(s1)
fopen(s2)
```

Because the default `DataTerminalReady` property value is `on`, the computer (data terminal) is now ready to exchange data with the modems. You can verify that the modems (data sets) can communicate with the computer by examining the value of the Data Set Ready pin with the `PinStatus` property.

```
s1.Pinstatus
ans =
    CarrierDetect: 'off'
      ClearToSend: 'on'
     DataSetReady: 'on'
    RingIndicator: 'off'
```

The value of the `DataSetReady` field is `on` because both modems were powered on before they were connected to the objects.

3 **Configure properties –** Both modems are configured for a baud rate of 2400 bits per second and a carriage return (CR) terminator.

```
s1.BaudRate = 2400;
s1.Terminator = 'CR';
s2.BaudRate = 2400;
s2.Terminator = 'CR';
```

4 **Write and read data –** Write the `atd` command to the first modem. This command puts the modem “off the hook,” which is equivalent to manually lifting a phone receiver.

```
fprintf(s1,'atd')
```

Write the `ata` command to the second modem. This command puts the modem in "answer mode," which forces it to connect to the first modem.

```
fprintf(s2,'ata')
```

After the two modems negotiate their connection, you can verify the connection status by examining the value of the Carrier Detect pin using the `PinStatus` property.

```
s1.PinStatus
ans =
    CarrierDetect: 'on'
      ClearToSend: 'on'
     DataSetReady: 'on'
    RingIndicator: 'off'
```

You can also verify the modem-modem connection by reading the descriptive message returned by the second modem.

```
s2.BytesAvailable
ans =
    25
out = fread(s2,25);
char(out)'
ans =
ata
CONNECT 2400/NONE
```

Now break the connection between the two modems by configuring the `DataTerminalReady` property to `off`. You can verify that the modems are disconnected by examining the Carrier Detect pin value.

```
s1.DataTerminalReady = 'off';
s1.PinStatus
ans =
    CarrierDetect: 'off'
      ClearToSend: 'on'
     DataSetReady: 'on'
    RingIndicator: 'off'
```

5 **Disconnect and clean up –** Disconnect the objects from the modems, and remove the objects from memory and from the MATLAB workspace.

```
fclose([s1 s2])
delete([s1 s2])
clear s1 s2
```

Controlling the Flow of Data: Handshaking

Data flow control or *handshaking* is a method used for communicating between a DCE and a DTE to prevent data loss during transmission. For example, suppose your computer can receive only a limited amount of data before it must be processed. As this limit is reached, a handshaking signal is transmitted to the DCE to stop sending data. When the computer can accept more data, another handshaking signal is transmitted to the DCE to resume sending data.

If supported by your device, you can control data flow using one of these methods:

- Hardware handshaking
- Software handshaking

Note Although you might be able to configure your device for both hardware handshaking and software handshaking at the same time, MATLAB does not support this behavior.

You can specify the data flow control method with the `FlowControl` property. If `FlowControl` is `hardware`, then hardware handshaking is used to control data flow. If `FlowControl` is `software`, then software handshaking is used to control data flow. If `FlowControl` is `none`, then no handshaking is used.

Hardware Handshaking

Hardware handshaking uses specific serial port pins to control data flow. In most cases, these are the RTS and CTS pins. Hardware handshaking using these pins is described in “The RTS and CTS Pins” on page 9-10.

If `FlowControl` is `hardware`, then the RTS and CTS pins are automatically managed by the DTE and DCE. You can return the CTS pin value with the

Ex.1049 / Page 673 of 748
Intel Corporation et al

`PinStatus` property. You can configure or return the RTS pin value with the `RequestToSend` property.

Note Some devices also use the DTR and DSR pins for handshaking. However, these pins are typically used to indicate that the system is ready for communication, and are not used to control data transmission. In MATLAB, hardware handshaking always uses the RTS and CTS pins.

If your device does not use hardware handshaking in the standard way, then you might need to manually configure the `RequestToSend` property. In this case, you should configure `FlowControl` to `none`. If `FlowControl` is `hardware`, then the `RequestToSend` value that you specify might not be honored. Refer to the device documentation to determine its specific pin behavior.

Software Handshaking

Software handshaking uses specific ASCII characters to control data flow. These characters, known as Xon and Xoff (or XON and XOFF), are described below.

Table 9-12: Software Handshaking Characters

Character	Integer Value	Description
Xon	17	Resume data transmission
Xoff	19	Pause data transmission

When using software handshaking, the control characters are sent over the transmission line the same way as regular data. Therefore you need only the TD, RD, and GND pins.

The main disadvantage of software handshaking is that you cannot write the Xon or Xoff characters while numerical data is being written to the device. This is because numerical data might contain a 17 or 19, which makes it impossible to distinguish between the control characters and the data. However, you can write Xon or Xoff while data is being asynchronously read from the device because you are using both the TD and RD pins.

Ex.1049 / Page 674 of 748
Intel Corporation et al

Example: Using Software Handshaking

Suppose you want to use software flow control with the example described in “Example: Reading Binary Data” on page 9-45. To do this, you must configure the oscilloscope and serial port object for software flow control.

```
fprintf(s,'RS232:SOFTF ON')
s.FlowControl = 'software';
```

To pause data transfer, you write the numerical value `19` to the device.

```
fwrite(s,19)
```

To resume data transfer, you write the numerical value `17` to the device.

```
fwrite(s,17)
```


Ex.1049 / Page 675 of 748
Intel Corporation et al

Debugging: Recording Information to Disk

While the serial port object is connected to the device, you can record this information to a disk file:

- The number of values written to the device, the number of values read from the device, and the data type of the values
- Data written to the device, and data read from the device
- Event information

Recording information to disk provides a permanent record of your serial port session, and is an easy way to debug your application.

You record information to a disk file with the `record` function. The properties associated with recording information to disk are given below.

Table 9-13: Recording Properties

Property Name	Description
`RecordDetail`	Specify the amount of information saved to a record file
`RecordMode`	Specify whether data and event information is saved to one record file or to multiple record files
`RecordName`	Specify the name of the record file
`RecordStatus`	Indicate if data and event information are saved to a record file

Example: Introduction to Recording Information

This example records the number of values written to and read from the device, and stores the information to the file `myfile.txt`

```
s = serial('COM1');
fopen(s)
s.RecordName = 'myfile.txt';
record(s)
fprintf(s,'*IDN?')
idn = fscanf(s);
```


Ex.1049 / Page 676 of 748
Intel Corporation et al

```
fprintf(s,'RS232?')
rs232 = fscanf(s);
```

End the serial port session.

```
fclose(s)
delete(s)
clear s
```

You can use the `type` command to display `myfile.txt` at the command line.

Creating Multiple Record Files

When you initiate recording with the `record` function, the `RecordMode` property determines if a new record file is created or if new information is appended to an existing record file.

You can configure `RecordMode` to `overwrite`, `append`, or `index`. If `RecordMode` is `overwrite`, then the record file is overwritten each time recording is initiated. If `RecordMode` is `append`, then the new information is appended to the file specified by `RecordName`. If `RecordMode` is `index`, a different disk file is created each time recording is initiated. The rules for specifying a record filename are discussed in the next section.

Specifying a Filename

You specify the name of the record file with the `RecordName` property. You can specify any value for `RecordName` – including a directory path – provided the filename is supported by your operating system. Additionally, if `RecordMode` is `index`, then the filename follows these rules:

- Indexed filenames are identified by a number. This number precedes the filename extension and is increased by 1 for successive record files.
- If no number is specified as part of the initial filename, then the first record file does not have a number associated with it. For example, if `RecordName` is `myfile.txt`, then `myfile.txt` is the name of the first record file, `myfile01.txt` is the name of the second record file, and so on.
- `RecordName` is updated after the record file is closed.
- If the specified filename already exists, then the existing file is overwritten.

The Record File Format

The record file is an ASCII file that contains a record of one or more serial port sessions. You specify the amount of information saved to a record file with the `RecordDetail` property.

`RecordDetail` can be `compact` or `verbose`. A compact record file contains the number of values written to the device, the number of values read from the device, the data type of the values, and event information. A verbose record file contains the preceding information as well as the data transferred to and from the device.

Binary data with precision given by `uchar`, `schar`, `(u)int8`, `(u)int16` or `(u)int32` is recorded using hexadecimal format. For example, if the integer value 255 is read from the instrument as a 16-bit integer, the hexadecimal value 00FF is saved in the record file. Single- and double-precision floating-point numbers are recorded as decimal values using the `%g` format, and as hexadecimal values using the format specified by the IEEE Standard 754-1985 for Binary Floating-Point Arithmetic.

The IEEE floating-point format includes three components: the sign bit, the exponent field, and the significand field. Single-precision floating-point values consist of 32 bits. The value is given by

$$\text{value} = (-1)^{\text{sign}}(2^{\text{exp-127}})(1.\text{significand})$$

Double-precision floating-point values consist of 64 bits. The value is given by

$$\text{value} = (-1)^{\text{sign}}(2^{\text{exp-1023}})(1.\text{significand})$$

The floating-point format component, and the associated single-precision and double-precision bits are given below.

Component	Single-Precision Bits	Double-Precision Bits
`sign`	1	1
`exp`	2-9	2-12
`significand`	10-32	13-64

Bit 1 is the left-most bit as stored in the record file.

Ex.1049 / Page 678 of 748
Intel Corporation et al

Example: Recording Information to Disk

This example illustrates how to record information transferred between a serial port object and a Tektronix TDS 210 oscilloscope. Additionally, the structure of the resulting record file is presented.

1 **Create the serial port object –** Create the serial port object `s` associated with the serial port COM1.

```
s = serial('COM1');
```

2 **Connect to the device –** Connect `s` to the oscilloscope. Because the default value for the `ReadAsyncMode` property is `continuous`, data is asynchronously returned the input buffer as soon as it is available from the instrument.

```
fopen(s)
```

3 **Configure property values –** Configure `s` to record information to multiple disk files using the verbose format. Recording is then initiated with the first disk file defined as `WaveForm1.txt`.

```
s.RecordMode = 'index';
s.RecordDetail = 'verbose';
s.RecordName = 'WaveForm1.txt';
record(s)
```

4 **Write and read data –** The commands written to the instrument, and the data read from the instrument are recorded in the record file. Refer to “Example: Writing and Reading Text Data” on page 9-42 for an explanation of the oscilloscope commands.

```
fprintf(s,'*IDN?')
idn = fscanf(s);
fprintf(s,'MEASUREMENT:IMMED:SOURCE CH2')
fprintf(s,'MEASUREMENT:IMMED:SOURCE?')
source = fscanf(s);
```

Read the peak-to-peak voltage with the `fread` function. Note that the data returned by `fread` is recorded using hex format.

```
fprintf(s,'MEASUREMENT:MEAS1:TYPE PK2PK')
fprintf(s,'MEASUREMENT:MEAS1:VALUE?')
ptop = fread(s,s.BytesAvailable);
```

Convert the peak-to-peak voltage to a character array.

```
char(ptop)'
ans =
2.0199999809E0
```

The recording state is toggled from `on` to `off`. Because the `RecordMode` value is `index`, the record filename is automatically updated.

```
record(s)
s.RecordStatus
ans =
off
s.RecordName
ans =
WaveForm2.txt
```

5 **Disconnect and clean up –** When you no longer need `s`, you should disconnect it from the instrument, and remove it from memory and from the MATLAB workspace.

```
fclose(s)
delete(s)
clear s
```

The Record File Contents

The contents of the WaveForm1.txt record file are shown below. Because the `RecordDetail` property was `verbose`, the number of values, commands, and data were recorded. Note that data returned by the `fread` function is in hex format.

```
type WaveForm1.txt

Legend:
  * - An event occurred.
  > - A write operation occurred.
  < - A read operation occurred.

1     Recording on 22-Jan-2000 at 11:21:21.575. Binary data in...
2    > 6 ascii values.
       *IDN?
3    < 56 ascii values.
       TEKTRONIX,TDS 210,0,CF:91.1CT FV:v1.16 TDS2CM:CMV:v1.04
4    > 29 ascii values.
       MEASUREMENT:IMMED:SOURCE CH2
5    > 26 ascii values.
       MEASUREMENT:IMMED:SOURCE?
6    < 4 ascii values.
       CH2
7    > 27 ascii values.
       MEASUREMENT:MEAS1:TYPE PK2PK
8    > 25 ascii values.
       MEASUREMENT:MEAS1:VALUE?
9    < 15 uchar values.
       32 2e 30 31 39 39 39 39 39 38 30 39 45 30 0a
10     Recording off.
```

Saving and Loading

You can save serial port objects to a MAT-file just as you would any workspace variable – using the `save` command. For example, suppose you create the serial port object `s` associated with the serial port COM1, configure several property values, and perform a write and read operation.

```
s = serial('COM1');
s.BaudRate = 1920O;
s.Tag = 'My serial object';
fopen(s)
fprintf(s, '*IDN?')
out = fscanf(s);
```

To save the serial port object and the data read from the device to the MAT-file `myserial.mat`

```
save myserial s out
```

Note You can save data and event information as text to a disk file with the `record` function.

You can recreate `s` and `out` in the workspace using the `load` command.

```
load myserial
```

Values for read-only properties are restored to their default values upon loading. For example, the `Status` property is restored to `closed`. Therefore, to use `s`, you must connect it to the device with the `fopen` function. To determine if a property is read-only, examine its reference pages.

Using Serial Port Objects on Different Platforms

If you save a serial port object from one platform, and then load that object on a different platform having different serial port names, then you will need to modify the `Port` property value. For example, suppose you create the serial port object `s` associated with the serial port COM1 on a Windows platform. If you want to save `s` for eventual use on a Linux platform, you should configure `Port` to an appropriate value such as `ttyS0` after the object is loaded.

Disconnecting and Cleaning Up

When you no longer need your serial port object, you should disconnect it from the device, and clean up your MATLAB environment by removing the object from memory and from the workspace. These are the steps you take to end a serial port session.

Disconnecting a Serial Port Object

When you no longer need to communicate with the device, you should disconnect it from the serial port object with the `fclose` function.

```
fclose(s)
```

You can examine the `Status` property to verify that the serial port object and the device are disconnected.

```
s.Status
ans =
closed
```

After `fclose` is issued, the serial port associated with `s` is available. You can now connect another serial port object to it using `fopen`.

Cleaning Up the MATLAB Environment

When you no longer need the serial port object, you should remove it from memory with the `delete` function.

```
delete(s)
```

Before using `delete`, you must disconnect the serial port object from the device with the `fclose` function.

A deleted serial port object is *invalid*, which means that you cannot connect it to the device. In this case, you should remove the object from the MATLAB workspace. To remove serial port objects and other variables from the MATLAB workspace, use the `clear` command.

```
clear s
```

If you use `clear` on a serial port object that is still connected to a device, the object is removed from the workspace but remains connected to the device. You can restore cleared objects to MATLAB with the `instrfind` function.

Property Reference

This section includes information to help you learn about serial port properties. The information is organized using these topics:

- "The Property Reference Page Format" describes the organization of the property reference pages.
- "Serial Port Object Properties" summarizes the properties using several categories based on how they are used. Following this section, the properties are listed alphabetically and described in detail.

The Property Reference Page Format

Each serial port property description contains some or all of this information:

- The property name
- A description of the property
- The property characteristics, including:
  - Read only – the condition under which the property is read only

    A property can be read only always, never, while the serial port object is open, or while the serial port object is recording. You can configure a property value using the `set` function or dot notation. You can return the current property value using the `get` function or dot notation.
  - Data type – the property data type

    This is the data type you use when specifying a property value.
- Valid property values including the default value

  When property values are given by a predefined list, the default value is usually indicated by `{}`.
- An example using the property
- Related properties and functions

Serial Port Object Properties

The serial port object properties are briefly described below, and organized into categories based on how they are used. Following this section, the properties are listed alphabetically and described in detail.

Communications Properties	
`BaudRate`	Specify the rate at which bits are transmitted
`DataBits`	Specify the number of data bits to transmit
`Parity`	Specify the type of parity checking
`StopBits`	Specify the number of bits used to indicate the end of a byte
`Terminator`	Specify the terminator character

Write Properties	
`BytesToOutput`	Indicate the number of bytes currently in the output buffer
`OutputBufferSize`	Specify the size of the output buffer in bytes
`Timeout`	Specify the waiting time to complete a read or write operation
`TransferStatus`	Indicate if an asynchronous read or write operation is in progress
`ValuesSent`	Indicate the total number of values written to the device

Ex.1049 / Page 685 of 748
Intel Corporation et al

Read Properties	
BytesAvailable	Indicate the number of bytes available in the input buffer
InputBufferSize	Specify the size of the input buffer in bytes
ReadAsyncMode	Specify whether an asynchronous read operation is continuous or manual
Timeout	Specify the waiting time to complete a read or write operation
TransferStatus	Indicate if an asynchronous read or write operation is in progress
ValuesReceived	Indicate the total number of values read from the device

Callback Properties	
BreakInterruptFcn	Specify the M-file callback function to execute when a break-interrupt event occurs
BytesAvailableFcn	Specify the M-file callback function to execute when a specified number of bytes is available in the input buffer, or a terminator is read
BytesAvailableFcnCount	Specify the number of bytes that must be available in the input buffer to generate a bytes-available event
BytesAvailableFcnMode	Specify if the bytes-available event is generated after a specified number of bytes is available in the input buffer, or after a terminator is read
ErrorFcn	Specify the M-file callback function to execute when an error event occurs

Callback Properties (Continued)	
`OutputEmptyFcn`	Specify the M-file callback function to execute when the output buffer is empty
`PinStatusFcn`	Specify the M-file callback function to execute when the CD, CTS, DSR, or RI pins change state
`TimerFcn`	Specify the M-file callback function to execute when a predefined period of time passes
`TimerPeriod`	Specify the period of time between timer events

Control Pin Properties	
`DataTerminalReady`	Specify the state of the DTR pin
`FlowControl`	Specify the data flow control method to use
`PinStatus`	Indicate the state of the CD, CTS, DSR, and RI pins
`RequestToSend`	Specify the state of the RTS pin

Recording Properties	
`RecordDetail`	Specify the amount of information saved to a record file
`RecordMode`	Specify whether data and event information are saved to one record file or to multiple record files
`RecordName`	Specify the name of the record file
`RecordStatus`	Indicate if data and event information are saved to a record file

General Purpose Properties	
`ByteOrder`	Specify the order in which the device stores bytes
`Name`	Specify a descriptive name for the serial port object
`Port`	Indicate the platform-specific serial port name
`Status`	Indicate if the serial port object is connected to the device
`Tag`	Specify a label to associate with a serial port object
`Type`	Indicate the object type
`UserData`	Specify data that you want to associate with a serial port object

Ex.1049 / Page 688 of 748
Intel Corporation et al

Purpose

Specify the rate at which bits are transmitted

Description

You configure `BaudRate` as bits per second. The transferred bits include the start bit, the data bits, the parity bit (if used), and the stop bits. However, only the data bits are stored.

The baud rate is the rate at which information is transferred in a communication channel. In the serial port context, "9600 baud" means that the serial port is capable of transferring a maximum of 9600 bits per second. If the information unit is one baud (one bit), then the bit rate and the baud rate are identical. If one baud is given as 10 bits, (for example, eight data bits plus two framing bits), the bit rate is still 9600 but the baud rate is 9600/10, or 960. You always configure `BaudRate` as bits per second. Therefore, in the above example, set `BaudRate` to 9600.

Note Both the computer and the peripheral device must be configured to the same baud rate before you can successfully read or write data.

Standard baud rates include 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000 and 256000 bits per second. To display the supported baud rates for the serial ports on your platform, refer to “Finding Serial Port Information for Your Platform” on page 9-15.

Characteristics

Read only	Never
Data type	Double

Values

The default value is `9600`.

See Also

Properties

`DataBits`, `Parity`, `StopBits`

Ex.1049 / Page 689 of 748
Intel Corporation et al

BreakInterruptFcn

Purpose

Specify the M-file callback function to execute when a break-interrupt event occurs

Description

You configure `BreakInterruptFcn` to execute an M-file callback function when a break-interrupt event occurs. A break-interrupt event is generated by the serial port when the received data is in an off (space) state longer than the transmission time for one byte.

Note A break-interrupt event can be generated at any time during the serial port session.

If the `RecordStatus` property value is `on`, and a break-interrupt event occurs, the record file records this information:

- The event type as `BreakInterrupt`
- The time the event occurred using the format day-month-year hour:minute:second:millisecond

Refer to “Creating and Executing Callback Functions” on page 9-52 to learn how to create a callback function.

Characteristics

Read only	Never
Data type	Callback function

Values

The default value is an empty string.

See Also

Functions

`record`

Properties

`RecordStatus`

Purpose

Specify the byte order of the device

Description

You configure `ByteOrder` to be `littleEndian` or `bigEndian`. If `ByteOrder` is `littleEndian`, then the device stores the first byte in the first memory address. If `ByteOrder` is `bigEndian`, then the device stores the last byte in the first memory address.

For example, suppose the hexadecimal value 4F52 is to be stored in device memory. Because this value consists of two bytes, 4F and 52, two memory locations are used. Using big-endian format, 4F is stored first in the lower storage address. Using little-endian format, 52 is stored first in the lower storage address.

Note You should configure `ByteOrder` to the appropriate value for your device before performing a read or write operation. Refer to your device documentation for information about the order in which it stores bytes.

Characteristics

Read only	Never
Data type	String

Values

`{littleEndian}`	The byte order of the device is little-endian.
`bigEndian`	The byte order of the device is big-endian.

See Also

Properties

`Status`

Ex.1049 / Page 691 of 748
Intel Corporation et al

BytesAvailable

Purpose

Indicate the number of bytes available in the input buffer

Description

`BytesAvailable` indicates the number of bytes currently available to be read from the input buffer. The property value is continuously updated as the input buffer is filled, and is set to `0` after the `fopen` function is issued.

You can make use of `BytesAvailable` only when reading data asynchronously. This is because when reading data synchronously, control is returned to the MATLAB command line only after the input buffer is empty. Therefore, the `BytesAvailable` value is always 0. Refer to "Reading Text Data" on page 9-39 to learn how to read data asynchronously.

The `BytesAvailable` value can range from zero to the size of the input buffer. Use the `InputBufferSize` property to specify the size of the input buffer. Use the `ValuesReceived` property to return the total number of values read.

Characteristics

Read only	Always
Data type	Double

Values

The default value is `0`.

See Also

Functions

`fopen`

Properties

`InputBufferSize`, `TransferStatus`, `ValuesReceived`

Ex.1049 / Page 692 of 748
Intel Corporation et al

Purpose

Specify the M-file callback function to execute when a specified number of bytes is available in the input buffer, or a terminator is read

Description

You configure `BytesAvailableFcn` to execute an M-file callback function when a bytes-available event occurs. A bytes-available event occurs when the number of bytes specified by the `BytesAvailableFcnCount` property is available in the input buffer, or after a terminator is read, as determined by the the `BytesAvailableFcnMode` property.

Note A bytes-available event can be generated only for asynchronous read operations.

If the `RecordStatus` property value is `on`, and a bytes-available event occurs, the record file records this information:

- The event type as `BytesAvailable`
- The time the event occurred using the format day-month-year hour:minute:second:millisecond

Refer to "Creating and Executing Callback Functions" on page 9-52 to learn how to create a callback function.

Characteristics

Read only	Never
Data type	Callback function

Values

The default value is an empty string.

Example

Create the serial port object `s` for a Tektronix TDS 210 two-channel oscilloscope connected to the serial port COM1.

```
s = serial('COM1');
```


Ex.1049 / Page 693 of 748
Intel Corporation et al

Configure s to execute the M-file callback function instrcallback when 40 bytes are available in the input buffer.

```
s.BytesAvailableFcnCount = 40;
s.BytesAvailableFcnMode = 'byte';
s.BytesAvailableFcn = @instrcallback;
```

Connect s to the oscilloscope.

```
fopen(s)
```

Write the *IDN? command, which instructs the scope to return identification information. Because the default value for the ReadAsyncMode property is continuous, data is read as soon as it is available from the instrument.

```
fprintf(s,'*IDN?')
```

The resulting output from instrcallback is shown below.

```
BytesAvailable event occurred at 18:33:35 for the object:
Serial-COM1.
```

56 bytes are read and instrcallback is called once. The resulting display is shown above.

```
s.BytesAvailable
ans =
    56
```

Suppose you remove 25 bytes from the input buffer and then issue the MEASUREMENT? command, which instructs the scope to return its measurement settings.

```
out = fscanf(s,'%c',25);
fprintf(s,'MEASUREMENT?')
```

The resulting output from instrcallback is shown below.

```
BytesAvailable event occurred at 18:33:48 for the object:
Serial-COM1.

BytesAvailable event occurred at 18:33:48 for the object:
Serial-COM1.
```

There are now 102 bytes in the input buffer, 31 of which are left over from the `*IDN?` command. `instrcallback` is called twice; once when 40 bytes are available and once when 80 bytes are available.

```
s.BytesAvailable
ans =
   102
```

See Also

Functions

`record`

Properties

`BytesAvailableFcnCount`, `BytesAvailableFcnMode`, `RecordStatus`, `Terminator`, `TransferStatus`

BytesAvailableFcnCount

Purpose

Specify the number of bytes that must be available in the input buffer to generate a bytes-available event

Description

You configure `BytesAvailableFcnCount` to the number of bytes that must be available in the input buffer before a bytes-available event is generated.

Use the `BytesAvailableFcnMode` property to specify whether the bytes-available event occurs after a certain number of bytes are available or after a terminator is read.

The bytes-available event executes the M-file callback function specified for the `BytesAvailableFcn` property.

You can configure `BytesAvailableFcnCount` only when the object is disconnected from the device. You disconnect an object with the `fclose` function. A disconnected object has a `Status` property value of `closed`.

Characteristics

Read only	While open
Data type	Double

Values

The default value is `48`.

See Also

Functions

`fclose`

Properties

`BytesAvailableFcn`, `BytesAvailableFcnMode`, `Status`

Purpose

Specify if the bytes-available event is generated after a specified number of bytes is available in the input buffer, or after a terminator is read

Description

You can configure `BytesAvailableFcnMode` to be `terminator` or `byte`. If `BytesAvailableFcnMode` is `terminator`, a bytes-available event occurs when the terminator specified by the `Terminator` property is reached. If `BytesAvailableFcnMode` is `byte`, a bytes-available event occurs when the number of bytes specified by the `BytesAvailableFcnCount` property is available.

The bytes-available event executes the M-file callback function specified for the `BytesAvailableFcn` property.

You can configure `BytesAvailableFcnMode` only when the object is disconnected from the device. You disconnect an object with the `fclose` function. A disconnected object has a `Status` property value of `closed`.

Characteristics

Read only	While open
Data type	String

Values

`{terminator}`	A bytes-available event is generated when the terminator is read.
`byte`	A bytes-available event is generated when the specified number of bytes are available.

See Also

Functions

`fclose`

Properties

`BytesAvailableFcn`, `BytesAvailableFcnCount`, `Status`, `Terminator`

BytesToOutput

Purpose

Indicate the number of bytes currently in the output buffer

Description

`BytesToOutput` indicates the number of bytes currently in the output buffer waiting to be written to the device. The property value is continuously updated as the output buffer is filled and emptied, and is set to `0` after the `fopen` function is issued.

You can make use of `BytesToOutput` only when writing data asynchronously. This is because when writing data synchronously, control is returned to the MATLAB command line only after the output buffer is empty. Therefore, the `BytesToOutput` value is always `0`. Refer to "Writing Text Data" on page 9-34 to learn how to write data asynchronously.

Use the `ValuesSent` property to return the total number of values written to the device.

Note If you attempt to write out more data than can fit in the output buffer, then an error is returned and `BytesToOutput` is `0`. You specify the size of the output buffer with the `OutputBufferSize` property.

Characteristics

Read only	Always
Data type	Double

Values

The default value is `0`.

See Also

Functions

`fopen`

Properties

`OutputBufferSize`, `TransferStatus`, `ValuesSent`

Purpose

Specify the number of data bits to transmit

Description

You can configure `DataBits` to be `5`, `6`, `7`, or `8`. Data is transmitted as a series of five, six, seven, or eight bits with the least significant bit sent first. At least seven data bits are required to transmit ASCII characters. Eight bits are required to transmit binary data. Five and six bit data formats are used for specialized communications equipment.

Note Both the computer and the peripheral device must be configured to transmit the same number of data bits.

In addition to the data bits, the serial data format consists of a start bit, one or two stop bits, and possibly a parity bit. You specify the number of stop bits with the `StopBits` property, and the type of parity checking with the `Parity` property.

To display the supported number of data bits for the serial ports on your platform, refer to “Finding Serial Port Information for Your Platform” on page 9-15.

Characteristics

Read only	Never
Data type	Double

Values

`DataBits` can be `5`, `6`, `7`, or `8`. The default value is `8`.

See Also

Properties

`Parity`, `StopBits`

DataTerminalReady

Purpose

Specify the state of the DTR pin

Description

You can configure `DataTerminalReady` to be `on` or `off`. If `DataTerminalReady` is `on`, the Data Terminal Ready (DTR) pin is asserted. If `DataTerminalReady` is `off`, the DTR pin is unasserted.

In normal usage, the DTR and Data Set Ready (DSR) pins work together, and are used to signal if devices are connected and powered. However, there is nothing in the RS-232 standard that states the DTR pin must be used in any specific way. For example, DTR and DSR might be used for handshaking. You should refer to your device documentation to determine its specific pin behavior.

You can return the value of the DSR pin with the `PinStatus` property. Handshaking is described in “Controlling the Flow of Data: Handshaking” on page 9-59.

Characteristics

Read only	Never
Data type	String

Values

`{on}`	The DTR pin is asserted.
`off`	The DTR pin is unasserted.

See Also

Properties

`FlowControl`, `PinStatus`

Purpose

Specify the M-file callback function to execute when an error event occurs

Description

You configure `ErrorFcn` to execute an M-file callback function when an error event occurs.

Note An error event is generated only for asynchronous read and write operations.

An error event is generated when a timeout occurs. A timeout occurs if a read or write operation does not successfully complete within the time specified by the `Timeout` property. An error event is not generated for configuration errors such as setting an invalid property value.

If the `RecordStatus` property value is `on`, and an error event occurs, the record file records this information:

- The event type as `Error`
- The error message
- The time the event occurred using the format day-month-year hour:minute:second:millisecond

Refer to "Creating and Executing Callback Functions" on page 9-52 to learn how to create a callback function.

Characteristics

Read only	Never
Data type	Callback function

Values

The default value is an empty string.

See Also

Functions

`record`

Properties

`RecordStatus`, `Timeout`

FlowControl

Purpose

Specify the data flow control method to use

Description

You can configure `FlowControl` to be `none`, `hardware`, or `software`. If `FlowControl` is `none`, then data flow control (handshaking) is not used. If `FlowControl` is `hardware`, then hardware handshaking is used to control data flow. If `FlowControl` is `software`, then software handshaking is used to control data flow.

Hardware handshaking typically utilizes the Request to Send (RTS) and Clear to Send (CTS) pins to control data flow. Software handshaking uses control characters (Xon and Xoff) to control data flow. Refer to “Controlling the Flow of Data: Handshaking” on page 9-59 for more information about handshaking.

You can return the value of the CTS pin with the `PinStatus` property. You can specify the value of the RTS pin with the `RequestToSend` property. However, if `FlowControl` is `hardware`, and you specify a value for `RequestToSend`, then that value might not be honored.

Note Although you might be able to configure your device for both hardware handshaking and software handshaking at the same time, MATLAB does not support this behavior.

Characteristics

Read only	Never
Data type	String

Values

`{none}`	No flow control is used.
`hardware`	Hardware flow control is used.
`software`	Software flow control is used.

See Also

Properties

`PinStatus`, `RequestToSend`

Purpose

Specify the size of the input buffer in bytes

Description

You configure `InputBufferSize` as the total number of bytes that can be stored in the input buffer during a read operation.

A read operation is terminated if the amount of data stored in the input buffer equals the `InputBufferSize` value. You can read text data with the `fgetl`, `fgets`, or `fscanf` functions. You can read binary data with the `fread` function.

You can configure `InputBufferSize` only when the serial port object is disconnected from the device. You disconnect an object with the `fclose` function. A disconnected object has a `Status` property value of `closed`.

If you configure `InputBufferSize` while there is data in the input buffer, then that data is flushed.

Characteristics

Read only	While open
Data type	Double

Values

The default value is `512`.

See Also

Functions

`fclose`, `fgetl`, `fgets`, `fopen`, `fread`, `fscanf`

Properties

`Status`

Name

Purpose

Specify a descriptive name for the serial port object

Description

You configure `Name` to be a descriptive name for the serial port object.

When you create a serial port object, a descriptive name is automatically generated and stored in `Name`. This name is given by concatenating the word "Serial" with the serial port specified in the `serial` function. However, you can change the value of `Name` at any time.

The serial port is given by the `Port` property. If you modify this property value, then `Name` is automatically updated to reflect that change.

Characteristics

Read only	Never
Data type	String

Values

`Name` is automatically defined when the serial port object is created.

Example

Suppose you create a serial port object associated with the serial port COM1.

```
s = serial('COM1');
```

`s` is automatically assigned a descriptive name.

```
s.Name
ans =
Serial-COM1
```

See Also

Functions

`serial`

Ex.1049 / Page 704 of 748
Intel Corporation et al

Purpose

Specify the size of the output buffer in bytes

Description

You configure `OutputBufferSize` as the total number of bytes that can be stored in the output buffer during a write operation.

An error occurs if the output buffer cannot hold all the data to be written. You write text data with the `fprintf` function. You write binary data with the `fwrite` function.

You can configure `OutputBufferSize` only when the serial port object is disconnected from the device. You disconnect an object with the `fclose` function. A disconnected object has a `Status` property value of `closed`.

Characteristics

Read only	While open
Data type	Double

Values

The default value is `512`.

See Also

Functions

`fprintf`, `fwrite`

Properties

`Status`

OutputEmptyFcn

Purpose

Specify the M-file callback function to execute when the output buffer is empty

Description

You configure `OutputEmptyFcn` to execute an M-file callback function when an output-empty event occurs. An output-empty event is generated when the last byte is sent from the output buffer to the device.

Note An output-empty event can be generated only for asynchronous write operations.

If the `RecordStatus` property value is `on`, and an output-empty event occurs, the record file records this information:

- The event type as `OutputEmpty`
- The time the event occurred using the format day-month-year hour:minute:second:millisecond

Refer to "Creating and Executing Callback Functions" on page 9-52 to learn how to create a callback function.

Characteristics

Read only	Never
Data type	Callback function

Values

The default value is an empty string.

See Also

Functions

`record`

Properties

`RecordStatus`

Ex.1049 / Page 706 of 748
Intel Corporation et al

Purpose

Specify the type of parity checking

Description

You can configure `Parity` to be `none`, `odd`, `even`, `mark`, or `space`. If `Parity` is `none`, parity checking is not performed and the parity bit is not transmitted. If `Parity` is `odd`, the number of mark bits (1's) in the data is counted, and the parity bit is asserted or unasserted to obtain an odd number of mark bits. If `Parity` is `even`, the number of mark bits in the data is counted, and the parity bit is asserted or unasserted to obtain an even number of mark bits. If `Parity` is `mark`, the parity bit is asserted. If `Parity` is `space`, the parity bit is unasserted.

Parity checking can detect errors of one bit only. An error in two bits might cause the data to have a seemingly valid parity, when in fact it is incorrect. Refer to "The Parity Bit" on page 9-14 for more information about parity checking.

In addition to the parity bit, the serial data format consists of a start bit, between five and eight data bits, and one or two stop bits. You specify the number of data bits with the `DataBits` property, and the number of stop bits with the `StopBits` property.

Characteristics

Read only	Never
Data type	String

Values

`{none}`	No parity checking
`odd`	Odd parity checking
`even`	Even parity checking
`mark`	Mark parity checking
`space`	Space parity checking

See Also

Properties

`DataBits`, `StopBits`

Ex.1049 / Page 707 of 748
Intel Corporation et al

PinStatus

Purpose

Indicate the state of the CD, CTS, DSR, and RI pins

Description

`PinStatus` is a structure array that contains the fields `CarrierDetect`, `ClearToSend`, `DataSetReady` and `RingIndicator`. These fields indicate the state of the Carrier Detect (CD), Clear to Send (CTS), Data Set Ready (DSR) and Ring Indicator (RI) pins, respectively. Refer to "Serial Port Signals and Pin Assignments" on page 9-6 for more information about these pins.

`PinStatus` can be `on` or `off` for any of these fields. A value of `on` indicates the associated pin is asserted. A value of `off` indicates the associated pin is unasserted. A pin status event occurs when any of these pins changes its state. A pin status event executes the M-file specified by `PinStatusFcn`.

In normal usage, the Data Terminal Ready (DTR) and DSR pins work together, while the Request to Send (RTS) and CTS pins work together. You can specify the state of the DTR pin with the `DataTerminalReady` property. You can specify the state of the RTS pin with the `RequestToSend` property.

Refer to "Example: Connecting Two Modems" on page 9-56 for an example that uses `PinStatus`.

Characteristics

Read only	Always
Data type	Structure

Values

`off`	The associated pin is asserted.
`on`	The associated pin is asserted.

The default value is device dependent.

See Also

Properties

`DataTerminalReady`, `PinStatusFcn`, `RequestToSend`

Ex.1049 / Page 708 of 748
Intel Corporation et al

Purpose

Specify the M-file callback function to execute when the CD, CTS, DSR, or RI pins change state

Description

You configure `PinStatusFcn` to execute an M-file callback function when a pin status event occurs. A pin status event occurs when the Carrier Detect (CD), Clear to Send (CTS), Data Set Ready (DSR) or Ring Indicator (RI) pin changes state. A serial port pin changes state when it is asserted or unasserted. Information about the state of these pins is recorded in the `PinStatus` property.

Note A pin status event can be generated at any time during the serial port session.

If the `RecordStatus` property value is `on`, and a pin status event occurs, the record file records this information:

- The event type as `PinStatus`
- The pin that changed its state, and the pin state as either `on` or `off`
- The time the event occurred using the format day-month-year hour:minute:second:millisecond

Refer to “Creating and Executing Callback Functions” on page 9-52 to learn how to create a callback function.

Characteristics

Read only	Never
Data type	Callback function

Values

The default value is an empty string.

See Also

Functions

`record`

Properties

`PinStatus`, `RecordStatus`

Port

Purpose

Specify the platform-specific serial port name

Description

You configure `Port` to be the name of a serial port on your platform. `Port` specifies the physical port associated with the object and the device.

When you create a serial port object, `Port` is automatically assigned the port name specified for the `serial` function.

You can configure `Port` only when the object is disconnected from the device. You disconnect an object with the `fclose` function. A disconnected object has a `Status` property value of `closed`.

Characteristics

Read only	While open
Data type	String

Values

The `Port` value is determined when the serial port object is created.

Example

Suppose you create a serial port object associated with serial port COM1.

```
s = serial('COM1');
```

The value of the `Port` property is `COM1`.

```
s.Port
ans =
COM1
```

See Also

Functions

`fclose`, `serial`

Properties

`Name`, `Status`

Purpose

Specify whether an asynchronous read operation is continuous or manual

Description

You can configure `ReadAsyncMode` to be `continuous` or `manual`. If `ReadAsyncMode` is `continuous`, the serial port object continuously queries the device to determine if data is available to be read. If data is available, it is automatically read and stored in the input buffer. If issued, the `readasync` function is ignored.

If `ReadAsyncMode` is `manual`, the object will not query the device to determine if data is available to be read. Instead, you must manually issue the `readasync` function to perform an asynchronous read operation. Because `readasync` checks for the terminator, this function can be slow. To increase speed, you should configure `ReadAsyncMode` to `continuous`.

Note If the device is ready to transmit data, then it will do so regardless of the `ReadAsyncMode` value. Therefore, if `ReadAsyncMode` is `manual` and a read operation is not in progress, then data might be lost. To guarantee that all transmitted data is stored in the input buffer, you should configure `ReadAsyncMode` to `continuous`.

You can determine the amount of data available in the input buffer with the `BytesAvailable` property. For either `ReadAsyncMode` value, you can bring data into the MATLAB workspace with one of the synchronous read functions such as `fscanf`, `fgetl`, `fgets`, or `fread`.

Characteristics

Read only	Never
Data type	String

Values

`{continuous}`	Continuously query the device to determine if data is available to be read.
`manual`	Manually read data from the device using the `readasync` function.

Ex.1049 / Page 711 of 748
Intel Corporation et al

ReadAsyncMode

See Also

Functions

`fgetl`, `fgets`, `fread`, `fscanf`, `readasync`

Properties

`BytesAvailable`, `InputBufferSize`

Ex.1049 / Page 712 of 748
Intel Corporation et al

Purpose

Specify the amount of information saved to a record file

Description

You can configure `RecordDetail` to be `compact` or `verbose`. If `RecordDetail` is `compact`, the number of values written to the device, the number of values read from the device, the data type of the values, and event information are saved to the record file. If `RecordDetail` is `verbose`, the data written to the device, and the data read from the device are also saved to the record file.

The event information saved to a record file is shown in Table 9-10, Event Information, on page 9-51. The verbose record file structure is shown in "Example: Recording Information to Disk" on page 9-65.

Characteristics

Read only	Never
Data type	String

Values

`{compact}`	The number of values written to the device, the number of values read from the device, the data type of the values, and event information are saved to the record file.
`verbose`	The data written to the device, and the data read from the device are also saved to the record file.

See Also

Functions

`record`

Properties

`RecordMode`, `RecordName`, `RecordStatus`

RecordMode

Purpose

Specify whether data and event information are saved to one record file or to multiple record files

Description

You can configure `RecordMode` to be `overwrite`, `append`, or `index`. If `RecordMode` is `overwrite`, then the record file is overwritten each time recording is initiated. If `RecordMode` is `append`, then data is appended to the record file each time recording is initiated. If `RecordMode` is `index`, a different record file is created each time recording is initiated, each with an indexed filename.

You can configure `RecordMode` only when the object is not recording. You terminate recording with the `record` function. A object that is not recording has a `RecordStatus` property value of `off`.

You specify the record filename with the `RecordName` property. The indexed filename follows a prescribed set of rules. Refer to “Specifying a Filename” on page 9-63 for a description of these rules.

Characteristics

Read only	While recording
Data type	String

Values

`{overwrite}`	The record file is overwritten.
`append`	Data is appended to an existing record file.
`index`	A different record file is created, each with an indexed filename.

Example

Suppose you create the serial port object `s` associated with the serial port COM1.

```
s = serial('COM1');
fopen(s)
```

Specify the record filename with the `RecordName` property, configure `RecordMode` to `index`, and initiate recording.

```
s.RecordName = 'MyRecord.txt';
s.RecordMode = 'index';
record(s)
```

The record filename is automatically updated with an indexed filename after recording is turned off.

```
record(s,'off')
s.RecordName
ans =
MyRecord01.txt
```

Disconnect s from the peripheral device, remove s from memory, and remove s from the MATLAB workspace.

```
fclose(s)
delete(s)
clear s
```

See Also

Functions

`record`

Properties

`RecordDetail`, `RecordName`, `RecordStatus`

RecordName

Purpose

Specify the name of the record file

Description

You configure `RecordName` to be the name of the record file. You can specify any value for `RecordName` – including a directory path – provided the filename is supported by your operating system.

MATLAB supports any filename supported by your operating system. However, if you access the file through MATLAB, you might need to specify the filename using single quotes. For example, suppose you name the record file `My Record.txt`. To type this file at the MATLAB command line, you must include the name in quotes.

```
type('My Record.txt')
```

You can specify whether data and event information are saved to one disk file or to multiple disk files with the `RecordMode` property. If `RecordMode` is `index`, then the filename follows a prescribed set of rules. Refer to “Specifying a Filename” on page 9-63 for a description of these rules.

You can configure `RecordName` only when the object is not recording. You terminate recording with the `record` function. An object that is not recording has a `RecordStatus` property value of `off`.

Characteristics

Read only	While recording
Data type	String

Values

The default record filename is `record.txt`.

See Also

Functions

`record`

Properties

`RecordDetail`, `RecordMode`, `RecordStatus`

Purpose

Indicate if data and event information are saved to a record file

Description

You can configure `RecordStatus` to be `off` or `on` with the `record` function. If `RecordStatus` is `off`, then data and event information are not saved to a record file. If `RecordStatus` is `on`, then data and event information are saved to the record file specified by `RecordName`.

Use the `record` function to initiate or complete recording. `RecordStatus` is automatically configured to reflect the recording state.

For more information about recording to a disk file, refer to "Debugging: Recording Information to Disk" on page 9-62.

Characteristics

Read only	Always
Data type	String

Values

`{off}`	Data and event information are not written to a record file.
`on`	Data and event information are written to a record file.

See Also

Functions

`record`

Properties

`RecordDetail`, `RecordMode`, `RecordName`

Ex.1049 / Page 717 of 748
Intel Corporation et al

RequestToSend

Purpose

Specify the state of the RTS pin

Description

You can configure `RequestToSend` to be `on` or `off`. If `RequestToSend` is `on`, the Request to Send (RTS) pin is asserted. If `RequestToSend` is `off`, the RTS pin is unasserted.

In normal usage, the RTS and Clear to Send (CTS) pins work together, and are used as standard handshaking pins for data transfer. In this case, RTS and CTS are automatically managed by the DTE and DCE. However, there is nothing in the RS-232 standard that requires the RTS pin must be used in any specific way. Therefore, if you manually configure the `RequestToSend` value, it is probably for nonstandard operations.

If your device does not use hardware handshaking in the standard way, and you need to manually configure `RequestToSend`, then you should configure the `FlowControl` property to `none`. Otherwise, the `RequestToSend` value that you specify might not be honored. Refer to your device documentation to determine its specific pin behavior.

You can return the value of the CTS pin with the `PinStatus` property. Handshaking is described in “Controlling the Flow of Data: Handshaking” on page 9-59.

Characteristics

Read only	Never
Data type	String

Values

`{on}`	The RTS pin is asserted.
`off`	The RTS pin is unasserted.

See Also

Properties

`FlowControl`, `PinStatus`

Purpose

Indicate if the serial port object is connected to the device

Description

`Status` can be `open` or `closed`. If `Status` is `closed`, the serial port object is not connected to the device. If `Status` is `open`, the serial port object is connected to the device.

Before you can write or read data, you must connect the serial port object to the device with the `fopen` function. You use the `fclose` function to disconnect a serial port object from the device.

Characteristics

Read only	Always
Data type	String

Values

`{closed}`	The serial port object is not connected to the device.
`open`	The serial port object is connected to the device.

See Also

Functions

`fclose`, `fopen`

StopBits

Purpose

Specify the number of bits used to indicate the end of a byte

Description

You can configure `StopBits` to be `1`, `1.5`, or `2`. If `StopBits` is `1`, one stop bit is used to indicate the end of data transmission. If `StopBits` is `2`, two stop bits are used to indicate the end of data transmission. If `StopBits` is `1.5`, the stop bit is transferred for 150% of the normal time used to transfer one bit.

Note Both the computer and the peripheral device must be configured to transmit the same the number of stop bits.

In addition to the stop bits, the serial data format consists of a start bit, between five and eight data bits, and possibly a parity bit. You specify the number of data bits with the `DataBits` property, and the type of parity checking with the `Parity` property.

Characteristics

Read only	Never
Data type	Double

Values

`{1}`	One stop bit is transmitted to indicate the end of a byte.
`1.5`	The stop bit is transferred for 150% of the normal time used to transfer one bit.
`2`	Two stop bits are transmitted to indicate the end of a byte.

See Also

Properties

`DataBits`, `Parity`

Purpose

Specify a label to associate with a serial port object

Description

You configure `Tag` to be a string value that uniquely identifies a serial port object.

`Tag` is particularly useful when constructing programs that would otherwise need to define the serial port object as a global variable, or pass the object as an argument between callback routines.

You can return the serial port object with the `instrfind` function by specifying the `Tag` property value.

Characteristics

Read only	Never
Data type	String

Values

The default value is an empty string.

Example

Suppose you create a serial port object associated with the serial port COM1.

```
s = serial('COM1');
fopen(s)
```

You can assign `s` a unique label using `Tag`.

```
set(s,'Tag','MySerialObj')
```

You can access `s` in the MATLAB workspace or in an M-file using the `instrfind` function and the `Tag` property value.

```
s1 = instrfind('Tag','MySerialObj');
```

See Also

Functions

`instrfind`

Terminator

Purpose

Specify the terminator character

Description

You can configure `Terminator` to an integer value ranging from 0 to 127, which represents the ASCII code for the character, or you can configure `Terminator` to the ASCII character. For example, to configure `Terminator` to a carriage return, you specify the value to be `CR` or `13`. To configure `Terminator` to a line feed, you specify the value to be `LF` or `10`. You can also set `Terminator` to `CR/LF` or `LF/CR`. If `Terminator` is `CR/LF`, the terminator is a carriage return followed by a line feed. If Terminator is `LF/CR`, the terminator is a line feed followed by a carriage return. Note that there are no integer equivalents for these two values. Additionally, you can set `Terminator` to a 1-by-2 cell array. The first element of the cell is the read terminator and the second element of the cell array is the write terminator

When performing a write operation using the `fprintf` function, all occurrences of `\n` are replaced with the `Terminator` property value. Note that `%s\n` is the default format for `fprintf`. A read operation with `fgetl`, `fgets`, or `fscanf` completes when the `Terminator` value is read. The terminator is ignored for binary operations.

You can also use the terminator to generate a bytes-available event when the `BytesAvailableFcnMode` is set to `terminator`.

Characteristics

Read only	Never
Data type	String

Values

An integer value ranging from 0 to 127, or the equivalent ASCII character. `CR/LF` and `LF/CR` are also accepted values. You specify different read and write terminators as a 1-by-2 cell array.

See Also

Functions

`fgetl`, `fgets`, `fprintf`, `fscanf`

Properties

`BytesAvailableFcnMode`

Ex.1049 / Page 722 of 748
Intel Corporation et al

Purpose

Specify the waiting time to complete a read or write operation

Description

You configure `Timeout` to be the maximum time (in seconds) to wait to complete a read or write operation.

If a timeout occurs, then the read or write operation aborts. Additionally, if a timeout occurs during an asynchronous read or write operation, then:

- An error event is generated.
- The M-file callback function specified for `ErrorFcn` is executed.

Characteristics

Read only	Never
Data type	Double

Values

The default value is `10` seconds.

See Also

Properties

`ErrorFcn`

Ex.1049 / Page 723 of 748
Intel Corporation et al

TimerFcn

Purpose

Specify the M-file callback function to execute when a predefined period of time passes.

Description

You configure `TimerFcn` to execute an M-file callback function when a timer event occurs. A timer event occurs when the time specified by the `TimerPeriod` property passes. Time is measured relative to when the serial port object is connected to the device with `fopen`.

Note A timer event can be generated at any time during the serial port session.

If the `RecordStatus` property value is `on`, and a timer event occurs, the record file records this information:

- The event type as `Timer`
- The time the event occurred using the format day-month-year hour:minute:second:millisecond

Some timer events might not be processed if your system is significantly slowed or if the `TimerPeriod` value is too small.

Refer to "Creating and Executing Callback Functions" on page 9-52 to learn how to create a callback function.

Characteristics

Read only	Never
Data type	Callback function

Values

The default value is an empty string.

See Also

Functions

`fopen`, `record`

Properties

`RecordStatus`, `TimerPeriod`

Purpose

Specify the period of time between timer events

Description

`TimerPeriod` specifies the time, in seconds, that must pass before the callback function specified for `TimerFcn` is called. Time is measured relative to when the serial port object is connected to the device with `fopen`.

Some timer events might not be processed if your system is significantly slowed or if the `TimerPeriod` value is too small.

Characteristics

Read only	Never
Data type	Callback function

Values

The default value is 1 second. The minimum value is 0.01 second.

See Also

Functions

`fopen`

Properties

`TimerFcn`

Ex.1049 / Page 725 of 748
Intel Corporation et al

TransferStatus

Purpose

Indicate if an asynchronous read or write operation is in progress

Description

`TransferStatus` can be `idle`, `read`, `write`, or `read&write`. If `TransferStatus` is `idle`, then no asynchronous read or write operations are in progress. If `TransferStatus` is `read`, then an asynchronous read operation is in progress. If `TransferStatus` is `write`, then an asynchronous write operation is in progress. If `TransferStatus` is `read&write`, then both an asynchronous read and an asynchronous write operation are in progress.

You can write data asynchronously using the `fprintf` or `fwrite` functions. You can read data asynchronously using the `readasync` function, or by configuring the `ReadAsyncMode` property to `continuous`. While `readasync` is executing, `TransferStatus` might indicate that data is being read even though data is not filling the input buffer. If `ReadAsyncMode` is `continuous`, `TransferStatus` indicates that data is being read only when data is actually filling the input buffer.

You can execute an asynchronous read and an asynchronous write operation simultaneously because serial ports have separate read and write pins. Refer to “Writing and Reading Data” on page 9-29 for more information about synchronous and asynchronous read and write operations.

Characteristics

Read only	Always
Data type	String

Values

`{idle}`	No asynchronous operations are in progress.
`read`	An asynchronous read operation is in progress.
`write`	An asynchronous write operation is in progress.
`read&write`	Asynchronous read and write operations are in progress.

See Also

Functions

`fprintf`, `fwrite`, `readasync`

Properties

`ReadAsyncMode`

Ex.1049 / Page 726 of 748
Intel Corporation et al

Purpose

Indicate the object type

Description

`Type` indicates the type of the object. `Type` is automatically defined after the serial port object is created with the `serial` function. The `Type` value is always `serial`.

Characteristics

Read only	Always
Data type	String

Values

`Type` is always `serial`. This value is automatically defined when the serial port object is created.

Example

Suppose you create a serial port object associated with the serial port COM1.

```
s = serial('COM1');
```

The value of the `Type` property is `serial`, which is the object class.

```
s.Type
ans =
serial
```

You can also display the object class with the `whos` command.

```
Name      Size         Bytes  Class

  s         1x1            644  serial object

Grand total is 18 elements using 644 bytes
```

See Also

Functions

`serial`

Ex.1049 / Page 727 of 748
Intel Corporation et al

UserData

Purpose

Specify data that you want to associate with a serial port object

Description

You configure `UserData` to store data that you want to associate with a serial port object. The object does not use this data directly, but you can access it using the `get` function or the dot notation.

Characteristics

Read only	Never
Data type	Any type

Values

The default value is an empty vector.

Example

Suppose you create the serial port object associated with the serial port COM1.

```
s = serial('COM1');
```

You can associate data with `s` by storing it in `UserData`.

```
coeff.a = 1.0;
coeff.b = -1.25;
s.UserData = coeff;
```

Purpose

Indicate the total number of values read from the device

Description

`ValuesReceived` indicates the total number of values read from the device. The value is updated after each successful read operation, and is set to `0` after the `fopen` function is issued. If the terminator is read from the device, then this value is reflected by `ValuesReceived`.

If you are reading data asynchronously, use the `BytesAvailable` property to return the number of bytes currently available in the input buffer.

When performing a read operation, the received data is represented by values rather than bytes. A value consists of one or more bytes. For example, one `uint32` value consists of four bytes. Refer to "Bytes Versus Values" on page 9-11 for more information about bytes and values.

Characteristics

Read only	Always
Data type	Double

Values

The default value is `0`.

Example

Suppose you create a serial port object associated with the serial port COM1.

```
s = serial('COM1');
fopen(s)
```

If you write the `RS232?` command, and then read back the response using `fscanf`, `ValuesReceived` is `17` because the instrument is configured to send the `LF` terminator.

```
fprintf(s,'RS232?')
out = fscanf(s)
out =
9600;0;0;NONE;LF
s.ValuesReceived
ans =
    17
```

See Also

Functions

`fopen`

ValuesReceived

Properties

`BytesAvailable`

Purpose

Indicate the total number of values written to the device

Description

`ValuesSent` indicates the total number of values written to the device. The value is updated after each successful write operation, and is set to `0` after the `fopen` function is issued. If you are writing the terminator, then `ValuesSent` reflects this value.

If you are writing data asynchronously, use the `BytesToOutput` property to return the number of bytes currently in the output buffer.

When performing a write operation, the transmitted data is represented by values rather than bytes. A value consists of one or more bytes. For example, one `uint32` value consists of four bytes. Refer to "Bytes Versus Values" on page 9-11 for more information about bytes and values.

Characteristics

Read only	Always
Data type	Double

Values

The default value is `0`.

Example

Suppose you create a serial port object associated with the serial port COM1.

```
s = serial('COM1');
fopen(s)
```

If you write the `*IDN?` command using the `fprintf` function, then `ValuesSent` is `6` because the default data format is `%s\n`, and the terminator was written.

```
fprintf(s,'*IDN?')
s.ValuesSent
ans =
    6
```

See Also

Functions

`fopen`

Properties

`BytesToOutput`

ValuesSent

Index

Symbols

%val 5-7
  allocating memory 5-25
  DIGITAL Visual Fortran 5-7

A

add-in
  MATLAB for Visual Studio 3-28
API
  access methods 3-3
  memory management 3-36
argument checking 4-8
argument passing, from Java methods
  data conversion 7-56
    built-in data types 7-57
    conversions you can perform 7-57
    Java objects 7-57
argument passing, to Java methods
  data conversion 7-46
    built-in arrays 7-48
    built-in data types 7-48
    Java object arrays 7-52
    Java object cell arrays 7-53
    Java objects 7-50
    objects of Object class 7-51
    string arrays 7-50
    string data types 7-49
  effect of dimension on 7-53
argument type, Java
  effect on method dispatching 7-54
array access methods 5-6
  mat 1-2
arrays
  cell 3-7
  empty 3-8
  hybrid 4-39
  MATLAB 3-5
  multidimensional 3-7
  persistent 4-38
  serial port object 9-26
  sparse 4-29
  temporary 4-37
arrays, Java
  accessing elements of 7-35
  assigning
    the empty matrix 7-41
    values to 7-38
    with single subscripts 7-39
  comparison with MATLAB arrays 7-29
  concatenation of 7-42
  creating a copy 7-44
  creating a reference 7-43
  creating in MATLAB 7-33
  creating with javaArray 7-33
  dimensionality of 7-29
  dimensions 7-32
  indexing 7-31
    with colon operator 7-36
    with single subscripts 7-35
  linear arrays 7-40
  passed by reference 7-49
  representing in MATLAB 7-29
  sizing 7-31
  subscripted deletion 7-41
  using the end subscript 7-38
ASCII file mode 1-5
ASCII flat file 1-3
automation
  client 8-30
  controller 8-6, 8-36
  server 8-36

Ex.1049 / Page 733 of 748
Intel Corporation et al

B

BaudRate 9-75
bcc53engmatopts.bat 3-17
bcc53opts.bat 3-16
bcc54engmatopts.bat 3-17
bcc54opts.bat 3-16
bcc55engmatopts.bat 3-17
bcc55opts.bat 3-16
bccengmatopts.bat 3-17
bccopts.bat 3-16
binary data
  reading from a device 9-41
  writing to a device 9-36
binary file mode 1-5
BLAS and LAPACK functions 4-40
  building C MEX files for 4-44
  building Fortran MEX files for 4-46
  example of 4-45
  handling complex numbers 4-41
  passing arguments 4-40
  specifying the function name 4-40
BreakInterruptFcn 9-76
BSTR 8-38
buffer
  input, serial port object 9-38
  output, serial port object 9-32
ByteOrder 9-77
BytesAvailable 9-78
BytesAvailableFcn 9-79
BytesAvailableFcnCount 9-82
BytesAvailableFcnMode 9-83

C

C example
  convec.c 4-20
  doubleelem.c 4-23
  findnz.c 4-25
  fulltosparse.c 4-29
  phonebook.c 4-16
  revord.c 4-10
  sincall.c 4-33
  timestwo.c 4-7
  timestwoalt.c 4-9
  xtimesy.c 4-14
C language
  data types 3-8
  debugging 4-48
  MEX-files 4-1
C language example
  basic 4-7
  calling MATLAB functions 4-33
  calling user-defined functions 4-33
  handling 8-, 16-, 32-bit data 4-23
  handling arrays 4-25
  handling complex data 4-20
  handling sparse arrays 4-29
  passing multiple values 4-13
  persistent array 4-38
  strings 4-10
callback
  serial port object 9-48
    functions 9-52
    properties 9-49
caller workspace 4-37
cat
  using with Java arrays 7-42
  using with Java objects 7-11
cell 3-8
  using with Java objects 7-60
cell arrays 3-7, 4-16
  converting from Java object 7-60
char 3-8
  overloading toChar in Java 7-58

Ex.1049 / Page 734 of 748
Intel Corporation et al

`class`
  using in Java 7-17
classes, Java 7-4
  built-in 7-4
  defining 7-4
  identifying using `which` 7-24
  importing 7-7
  loading into workspace 7-6
  making available to MATLAB 7-5
  sources for 7-4
  third-party 7-4
  user-defined 7-4
`classpath.txt`
  finding and editing 7-6
  using with Java archive files 7-5
  using with Java classes 7-5
  using with Java packages 7-5
client (DDE) 8-42
collections 8-32
colon
  using in Java array access 7-36
  using in Java array assignment 7-40
COM
  automation server 8-36
  collections 8-32
  concepts 8-2
  controller 8-36
  `Count` property 8-32
  event handler function 8-26
  `Item` method 8-32
  launching server 8-40
  limitations of MATLAB support 8-35
  MATLAB as automation client 8-30
  ProgID 8-8
  server 8-36
  use in the MATLAB Engine 6-4
commands. *See* individual commands.
Compaq Visual Fortran compiler
  debugging 5-38
compiler
  changing on UNIX 3-11
  debugging
    Compaq Visual Fortran 5-38
    Microsoft 4-49
    Watcom 4-50
  preconfigured options file 3-16
  selecting on Windows 3-13
  supported 3-10
compiling
  engine application
    UNIX 6-17
    Windows 6-18
  MAT-file application
    UNIX 1-29
    Windows 1-30
complex data
  in Fortran 5-21
`compopts.bat` 3-22
concatenation
  of Java arrays 7-42
  of Java objects 7-11
configuration 3-10
  problems 3-34
  UNIX 3-11
  Windows 3-13, 3-15
control pins
  serial port object, using 9-56
`convec.c` 4-20
`convec.f` 5-21
conversation (DDE) 8-42
conversion, data
  in Java method arguments 7-46
copying a Java array 7-44
`Count` property 8-32

D

data access
  within Java objects 7-15
data bits 9-13
data format
  serial port 9-11
data storage 3-5
data type 4-6
  C language 3-8
  cell arrays 3-7
  checking 4-8
  complex double-precision nonsparse matrix 3-6
  empty arrays 3-8
  Fortran language 3-8
  MAT-file 1-5
  MATLAB 3-8
  MATLAB string 3-6
  multidimensional arrays 3-7
  numeric matrix 3-6
  objects 3-7
  sparse arrays 4-29
  sparse matrices 3-6
  structures 3-7
data, MATLAB 3-5
  exporting from 1-3
  importing to 1-2
`DataBits` 9-85
`DataTerminalReady` 9-86
`DataToOutput` 9-84
`dblmat.f` 5-25
`dbmex` 5-37
DCE 9-5
DCOM (distributed component object model) 8-41
  using MATLAB as a server 8-41
DDE (dynamic data exchange) 8-42
  accessing MATLAB as server 8-44
  advisory links 8-51
  client 8-42
  conversation 8-42
  hot link 8-51
  item 8-43
  MATLAB
    requesting data from 8-47
    sending commands to 8-46
    sending data to 8-48
    using as client 8-50
  name hierarchy 8-45
  notifying when data changes 8-51
  server 8-42
  service name 8-43
  topic 8-43, 8-45
    engine 8-46
    system 8-45
  warm link 8-51
  Windows clipboard formats 8-43
debugging C language MEX-files 4-48
  UNIX 4-48
  Windows 4-49
debugging Fortran language MEX-files
  UNIX 5-37
  Windows 5-38
DEC Alpha
  declaring pointers 5-4
`df50engmatopts.bat` 3-18
`df50opts.bat` 3-17
`df60engmatopts.bat` 3-18
`df61opts.bat` 3-17
`df66opts.bat` 3-17
`diary` 1-3
diary file 1-3

Ex.1049 / Page 736 of 748
Intel Corporation et al

DIGITAL Visual Fortran compiler
  debugging 5-38
directory
  eng_mat 3-46
  mex 3-46
  mx 3-46
  refbook 3-45
directory organization
  MAT-file application 1-8
  Microsoft Windows 3-43
  UNIX 3-41
directory path
  convention 3-8
display
  serial port object 9-25
display function
  overloading toString in Java 7-25
distributed component object model. *See* DCOM.
dll extension 3-3
dll libraries
  data conversion 2-11
    enumerated types 2-15
    primitive types 2-11
    reference pointers 2-25
    references 2-21
    strings 2-14
    structures 2-15
  library functions
    getting information about 2-4
    invoking functions 2-7
    passing arguments 2-8
    passing libstruct objects 2-18
    passing references 2-10
    passing structures 2-17
  loading the library 2-3
  MATLAB interface to 2-1
  unloading the library 2-3
DLLs 3-10
  locating 3-31
double 3-8
  overloading toDouble in Java 7-58
doubleelem.c 4-23
DTE 9-5
dynamic data exchange. *See* DDE.
dynamic memory allocation
  in Fortran 5-25
  mxCalloc 4-11
dynamically linked subroutines 3-2

E

empty arrays 3-8
empty matrix
  conversion to Java NULL 7-54
  in Java array assignment 7-41
empty string
  conversion to Java object 7-54
end
  use with Java arrays 7-38
eng_mat directory 3-46
engClose
  C library 6-2
  Fortran library 6-3
engdemo.c 6-5
engEvalString
  C library 6-3
  Fortran library 6-3
engGetVariable
  C library 6-2
  Fortran library 6-3
engGetVisible 6-3
engine
  compiling 6-16
  linking 6-16

Ex.1049 / Page 737 of 748
Intel Corporation et al

UNIX 6-17
windows 6-18
engine application
Windows 6-18
engine example
calling MATLAB
from C program 6-5
from Fortran program 6-10
engine functions 6-2
engine library 6-1
communicating with MATLAB
UNIX 6-4
Windows 6-4
`engOpen`
C library 6-2
Fortran library 6-3
`engOpenSingleUse` 6-3
`engopts.sh` 3-18
`engOutputBuffer`
C library 6-3
Fortran library 6-3
`engPutVariable`
C library 6-3
Fortran library 6-3
`engSetVisible` 6-3
`engwindemo.c` 6-5
`ErrorFcn` 9-87
event handler
function 8-26
writing 8-26
events
serial port object 9-48
storing information 9-50
types 9-49
examples, Java programming
communicating through a serial port 7-68
creating and using a phone book 7-73
finding an internet protocol address 7-66
reading a URL 7-63
exceptions, Java
handling 7-27
exceptions, masking
engine applications 6-16
MAT-file applications 1-28
`explore` example 3-8
extension
MEX-file 3-2

F

`-f` option 3-16
`fengdemo.f` 6-10
`fieldnames`
using with Java objects 7-14
file mode
ASCII 1-5
binary 1-5
files
flat 1-3
linking multiple
C 4-36
fortran 5-35
`findnz.c` 4-25
floating-point exceptions
masking 1-28, 6-16
`FlowControlHardware` 9-88
`fopen` 1-3, 1-4
Fortran
case in 5-5
data types 3-8
pointers
concept 5-4, 5-16
declaring 5-4

Ex.1049 / Page 738 of 748
Intel Corporation et al

Fortran examples
  convec.f 5-21
  dblmat.f 5-25
  fulltosparse.f 5-27
  matsq.f 5-16
  passstr.f 5-14
  revord.f 5-11
  sincall.f 5-31
  timestwo.f 5-9
  xtimesy.f 5-19
Fortran language examples
  calling MATLAB functions 5-31
  handling complex data 5-21
  handling sparse matrices 5-27
  passing arrays of strings 5-13
  passing matrices 5-16
  passing multiple values 5-18
  passing scalar 4-7, 5-9
  passing strings 5-11
Fortran language MEX-files 5-2
  components 5-2
fread 1-3
fulltosparse.c 4-29
fulltosparse.f 5-27
function handles
  serial port object callback 9-52
fwrite 1-4

G

-g option
  C 4-48
gateway routine
  accessing mxArray data 4-2
gccopts.sh 3-18

H

handshaking
  serial port object 9-59
help files 5-35
  C 4-36
  Fortran 5-35
hybrid arrays 4-39

I

IDE
  building MEX-files 3-19
IEEE routines 3-3
import
  using with Java classes 7-7
include directory 1-8
indexing Java arrays
  using single colon subscripting 7-37
  using single subscripting 7-35
InputBufferSize 9-89
internet protocol address
  Java example 7-66
ir 3-6
isa
  using with Java objects 7-17
isjava
  using with Java objects 7-17
Item method 8-32

J

Java
  API class packages 7-2
  archive (JAR) files 7-5
  development kit 7-4
  Java Virtual Machine (JVM) 7-2
  packages 7-5

Ex.1049 / Page 739 of 748
Intel Corporation et al

Java, MATLAB interface to
  arguments passed to Java methods 7-46
  arguments returned from Java methods 7-56
  arrays, working with 7-28
  benefits of 7-2
  classes, using 7-4
  examples 7-62
  methods, invoking 7-18
  objects, creating and using 7-9
  overview 7-2
`javaArray` function 7-33
`jc` 3-6

L

LAPACK and BLAS functions 4-40
  building C MEX files for 4-44
  building Fortran MEX files for 4-46
  example of 4-45
  handling complex numbers 4-41
  passing arguments 4-40
  specifying the function name 4-40
`lccengmatopts.bat` 3-17
`lccopts.bat` 3-16
library path
  setting on UNIX 1-29, 6-17
linking DLLs to MEX-files 3-27
linking multiple files 4-36, 5-35
`load` 1-5
  using with Java objects 7-13
loading
  serial port objects 9-68
locating DLLs 3-31
`logical` 3-8

M

`mat.h` 1-8
`matClose`
  C library 1-6
  Fortran library 1-7
`matDeleteVariable`
  C library 1-6
  Fortran library 1-7
`matdemo1.f` 1-19
`matdemo2.f` 1-24
MAT-file
  C language
    reading 1-15
  compiling 1-28
  data types 1-5
  examples 1-9
  Fortran language
    creating 1-19
    reading 1-24
  linking 1-28
  overview 1-1
  subroutines 1-5
  UNIX libraries 1-9
  using 1-2
  Windows libraries 1-8
MAT-file application
  UNIX 1-29
  Windows 1-30
MAT-file example
  creating
    C language 1-10
    Fortran language 1-19
  reading
    C language 1-15
    Fortran language 1-24
MAT-functions 1-6

Ex.1049 / Page 740 of 748
Intel Corporation et al

matGetDir
  C library 1-6
  Fortran library 1-7
matGetFp 1-6
matGetNextVariable
  C library 1-6
  Fortran library 1-7
matGetNextVariableInfo
  C library 1-6
  Fortran library 1-7
matGetVariable
  C library 1-6
  Fortran library 1-7
matGetVariableInfo
  C library 1-6
  Fortran library 1-7
MATLAB
  arrays 3-5
  as DCOM server client 8-34
  data 3-5
  data file format 1-2
  data storage 3-5
  data type 3-8
  engine 6-1
  exporting data 1-2
  importing data 1-2
  MAT-file 1-5
    reading arrays from 1-5
    saving arrays to 1-5
  moving data between platforms 1-4
  stand-alone applications 1-2
  string 3-6
  using as a computation engine 6-1
  variables 3-5
MATLAB for Visual Studio add-in 3-28
matOpen
  C library 1-6
  Fortran library 1-7
matopts.sh 3-18
matPutArray
  C library 1-6
  Fortran library 1-7
matPutArrayAsGlobal
  C library 1-6
  Fortran library 1-7
matrix
  complex double-precision nonsparse 3-6
  numeric 3-6
  sparse 3-6
matrix.h 1-8
matsq.f 5-16
memory
  allocation 4-11
  leak 3-39, 4-38
memory management
  API 3-36
  C 4-37
  compatibility 3-36
  Fortran 5-36
  routines 3-3
  special considerations 4-37
methods
  using with Java methods 7-23
methods, Java
  calling syntax 7-18
  converting input arguments 7-46
  displaying 7-23
  displaying information about 7-21
  finding the defining class 7-24
  overloading 7-54
  passing data to 7-46
  static 7-20

Ex.1049 / Page 741 of 748
Intel Corporation et al

undefined 7-26
methodsview 7-21
output fields 7-23
mex
<ENV_VAR>#<val> 3-21
-g
C 4-48
Fortran 5-37
mex build script 3-19
default options file, UNIX 3-21
default options file, Windows 3-22
switches 3-20
<name>#<def> 3-21
<name>=<def> 3-21
@<rsp_file> 3-20
-argcheck 3-20
-c 3-20
-D<name>[#<def>] 3-20
-f <file> 3-20
-fortran 5-35
-g 3-20
-h[elp] 3-20
-I<pathname> 3-20
-inline 3-20
-l<file> 3-20
-L<pathname> 3-20
-O 3-21
-outdir <name> 3-21
-output <name> 3-21
-setup 3-13, 3-21
-U<name> 3-21
-v 3-21
-V4 3-21
mex directory 3-46
mex.bat 4-9
mex.m 4-9
mex.sh 4-9
mexAtExit 4-38
register a function 4-38
mexaxp extension 3-2
mexCallMATLAB
calling from C code 4-33
calling from Fortran code 5-31
generating mxUNKNOWN_CLASS
C 4-35
Fortran 5-34
temporary array cleanup 4-37
mexErrMsgTxt
C library 4-37
Fortran library 5-6
mexEvalString
C library 4-37
Fortran library 5-35
MEX-file 3-2
advanced topics 4-36
Fortran 5-35
applications of 3-2
arguments 4-4
C
help 4-36
C language 4-1
calling 3-3
compiling
Microsoft Visual C++ 3-27
UNIX 3-11, 3-23
Windows 3-15
components 4-2
C computational routine 4-2
C gateway routine 4-2
Fortran computational routine 5-2
Fortran gateway routine 5-2
computation error 3-36
configuration problem 3-34
creating C language 4-2

Ex.1049 / Page 742 of 748
Intel Corporation et al

creating Fortran language 5-2
custom building 3-19
Windows 3-25
debugging C language 4-48
debugging Fortran language 5-37
DLL linking 3-27
documenting 5-35
dynamically allocated memory 4-37
examples
C 4-6
Fortran 5-8
extensions 3-2
Fortran
help 5-35
load error 3-34
overview 3-2
passing cell arrays 4-16
passing structures 4-16
problems 3-33
segmentation error 3-35
syntax errors 3-34
temporary array 4-37
using 3-2
versioning 3-27
mexFunction
altered name 5-38
C 4-2
Fortran 5-2
parameters
C 4-2
Fortran 5-2
mexGetMatrix 5-35
mexGetVariable 4-37
mexglx extension 3-2
mexhp7 extension 3-2
mexhpux extension 3-2
mexMakeArrayPersistent 4-38
mexMakeMemoryPersistent 4-38
mexopts.bat 3-22
mexopts.sh 3-18
mexPutMatrix 5-35
mexPutVariable 4-37
mexrs6 extension 3-2
mexSetTrapFlag 4-37
mexsg extension 3-2
mexsol extension 3-3
mexversion.rc 3-27
M-file
creating data 1-2
Microsoft compiler
debugging 4-49
Microsoft Windows
directory organization 3-43
msvc50engmatopts.bat 3-17
msvc50opts.bat 3-16
msvc60engmatopts.bat 3-17
msvc60opts.bat 3-16
multidimensional arrays 3-7
mx directory 3-46
mxArray 3-5, 5-6
accessing data 4-2
contents 3-5
improperly destroying 3-37
ir 3-7
jc 3-7
nzmax 3-7
pi 3-7
pr 3-7
temporary with improper data 3-38
type 3-5

Ex.1049 / Page 743 of 748
Intel Corporation et al