RS232 Blockset

For use with Simulink

version 1.2.5 of 24.Jul.2004

Matlab 5.3+: v. 1.2.2 (25.Feb.2004)

Matlab 6.0: v. 1.2.2 (25.Feb.2004)

Matlab 6.5: v. 1.2.5 (24.Jul.2004)

Index

Introduction

The RS232 Blockset is born in the context of the FODIAS project, to support the communication between real and simulated devices that constitutes the avionics of a simulated aircraft. The actual blockset, in any case, has been realized and expanded to satisfy a great number of simulation context that use real devices as support for the simulation, but it can be used also to allows the communication between Simulink simulations running on different computers.

The blockset has been realized only for the Windows platform and has been tested on a Win2000 and WinME OS with different speed processors.

The real-time support, often needed, can be provided by the RT Blockset, published in the Matlab Central web page.

The RS232 Blockset for Simulink

Each component of the blockset has been realized using a S-function written in C++ language (the relative DLLs are contained in the installation directory). The blockset is composed by some base blocks that can be composed to translate a complex serial protocol (ex.: a NMEA GPS protocol). The development of more blocks is foreseen in the case that one or more user of the blockset will signal a macro block needs.

The RS232 blocks description

The Block List

Initialization	RS232_Setup
Unbuffered Formatted Communication	RS232_Read_Format RS232_Write_Format RS232_Write_String RS232_Read_Fix_Format
Buffered Formatted Communication	RS232_Create_Buffer RS232_Read_Buffer RS232_Wait_Buffer_Synch
Bynary Formatted Communication	RS232_Write_Binary
Binary Buffered Formatted Communication	RS232_Read_Binary_Buffer RS232_Peek_Binary_Buffer RS232_Synch_Binary_Buffer

RS232_Setup

Purpose

Initialize and close the serial communication. The settings of the serial communication are selectable using the block panel

Parameters

- **Port**: COM1/2/3/4 (If needed, a port number greater then 4 can be used simply modifying the block's mask).
- **Baudrate**: 115200,...,9600,...,110 baud
- **Number of Databits**: 8 / 7 / 6 / 5 bits
- **Number of Stopbits**: 1 / 2
- **Parity**: None / Odd / Even

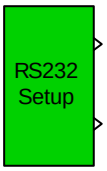
Input

None

Output

- **out1**: the first is used to transmit an handle for the opened communication. Therefore, it should be connected to all the blocks that use the port opened by this block. More then one RS232 Setup block can be present at the same time in the same simulation model, allowing the management of many serial port at the same time.
- **out2**: The second port is used to communicate the state of the port (0: not open, 1: open). This output is used to conditioning the others RS232 blocks to the actual status of the port.

Description



Even if this block is executed at each step, this block is active only in correspondence to the first and the last step of the simulation, step in which the block, respectively, open and close the communication. During the simulations, when the communication is open, the handle (**out1**) and the flag (**out2**) are constant, unless a major communications error occurs: in that case, **out1** and **out2** becomes 0.

RS232_Read_Format

Purpose

This block reads from the serial port a formatted string (function analogue to the standard-C s*scanf* function).

Parameters

- **Format String**: A format control string that defines how data is organized in the incoming string. The format specification follows the same rules as the C-standard s*scanf* function, as described in the string format page.
- **Number of char for msg terminator:** parameter that specifies the number of characters that should be considered as message terminator. Very often, the terminator char selected is the linefeed character (\n) that should be considered as one character.
- **Sample Time:** Indicates if the sample time should be inherited from the preceding blocks or should be defined independently (but it should be a multiple of the base sample time).

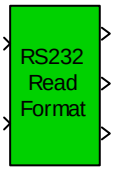
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the read operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its read operation.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag that indicated the completion (successful or not) of the read operation (0: not completed or not enabled, 1: read complete).
- **Output 3:** the output port that contains the data read from the input string. The width of the port is in the same number of the format string argument (i.e.: the number of '%' chars). All the output are held until the next data incomes from the port (zero order hold). All the data output are stored, in the output, as double, even if the formatted data contains integer or boolean values.

Description



Remarks:

The block starts its read operations when received a '1' flag from the input 2. The read operation remains active until the completion of the data read (that may request more than one simulation step). Once completed the read of the string, it modify the output 3 with new data and put the output 2 flag to '1' for only one simulation step.

Each block instance use an internal buffer to accumulate pieces of data read from the serial port but that compose (for example) the next message. The buffer is used in the next step to complete the read of the incoming message. The buffer is separated for each

instantiated block and resizes dynamically to manage eventual data quantity peak or, more generally, the multitasking of the OS.

When the head of the message (all the part of the format message that precedes the first '%' character) is detected into the buffer, all the buffer that precedes the head is simply discarded and removed from the buffer. If both the head and the end of the message are detected in the buffer, the message is interpreted and, after, all the chars that precede (and contain) the end of the message are discarded from the buffer. See the example in the RS232 Read Format Behavior page.

While the block is not active (the comm. handle is not valid (0) or the input flag (input 2) is not active) and no read operation is pending, the block doesn't performs any operation.

Cautions:

Putting two Read_Format block in cascade to read two different formatted messages may work but with unpredictable result. Each block, when activated to read its kind of message, may read from serial port part of the message addressed to another block. In this case, the other block, when activated in the next step, may skip a message destined to it, waiting until this message back in the input port in a successive simulation time. Go to the RS232 Read Format Behavior page to see a typical behavior of two Read_Format block configured in cascade.

RS232_Write_Format

Purpose

This block writes to the serial port a formatted string (function analogue to the standard-C s*printf* function).

Parameters

- **Format String**: A format control string that defines how data is organized in the incoming string. The format specification follows the same rules as the C-standard s*printf* function, as described in the string format page.
- **Sample Time:** Indicates if the sample time should be inherited from the preceding blocks or should be defined independently (but it should be a multiple of the base sample time).

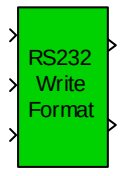
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the write operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its read operation. The block writes to the serial output each time the flag becomes 1.
- **Input 3:** the input port that contains the data to write to the output string. The width of the port is in the same number of the format string argument (i.e.: the number of '%' chars). All the data input are double, even if the formatted data contains integer or boolean values.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag that indicated the completion (successful or not) of the write operation (0: not completed or not enabled, 1: write complete).

Description



Remarks:

While the block is not active (the comm. handle is not valid (0) or the input flag (input 2) is not active) the block doesn't performs any operation.

RS232_Write_String

Purpose

This block writes to the serial port the string indicated as block parameter.

Parameters

- **String to write**: the string to send to the serial output

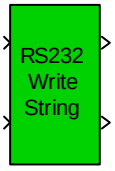
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the write operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its read operation. The block writes to the serial output each time the flag becomes 1.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag that indicated the completion (successful or not) of the write operation (0: not completed or not enabled, 1: write complete).

Description



Remarks:

While the block is not active (the comm. handle is not valid (0) or the input flag (input 2) is not active) the block doesn't performs any operation.

RS232_Read_Fix_Format

Purpose

Read fix formatted string from a buffer. This block has been generated specifically to read Flight Simulator's formatted data string.

Parameters

- **Format String**: The format string is structured as follows:
  - each field is contained into a couple of '%' chars.
  - each field is structured as follows: - the number of characters that is assigned to this field in the input string;
  - the type of the input data;
  - the size of the input data;
  - each field not contained in the '%' delimitators is considered as a simple string, that should be found in each input string;
- **Sample Time:** Indicates if the sample time should be inherited from the base sample time or should be defined independently (but it should be a multiple of the base sample time).

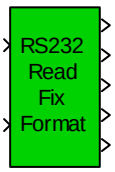
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the check operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its check operation.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag that indicated the completion of the check operation.
- **Output 3..n**: The output containing the data read, each data in a different port. The size of the output is specified in the format string.

Description



Remarks:

About the format string, for example, to read a string like this:

"#DATA:123.19,1010.1,23.1*"

you have to use a format string as follows:

"#DATA:%6d4%,%6d8%,%4d8%*"

In this example format string:

1. the format %6d4% means a field of 6 chars that should be translated into a single precision float (double of 4 bytes);
2. the format %6d8% means a field of 6 chars that should be translated into a double precision float (double of 8 bytes);
3. the format %4d8% means a field of 4 chars that should be

translated into a double precision float (double of 8 bytes);

The decimal point inside the field should be taken into account in the evaluation of the field length. In your case, the string to read is something like this: "_00594-003+000357000000_" The format string is the following: "_%5i4%%4i4%%4i4%%3i4%%3i4%%3i4%_"

RS232_Create_Buffer

Purpose

Initialize and allocate an input communication buffer.

Parameters

- **Buffer Minimum Size**: The size, in bytes, of the buffer to allocate.
- **Sample Time:** Indicates if the sample time should be inherited from the base sample time or should be defined independently (but it should be a multiple of the base sample time).

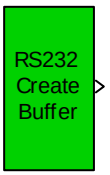
Input

None

Output

- **out1**: handle to the allocated buffer. This output should be propagated to all blocks that needs an input buffer for the same serial port.

Description



Remarks:

Even if this block is executed at each step, this block is active only in correspondence to the first and the last step of the simulation, step in which the block, respectively, create and destroy the buffer. During the simulations, the handle (**out1**) is constant.

The output of this block should be propagated to all Read_Buffer blocks that use the same serial port. Create another buffer to manage another serial port. In the case that the read of the actual message need more space, the Read_Buffer block may allocate more memory space itself.

RS232_Read_Buffer

Purpose

This block reads bytes from the serial port, add them to the input buffer and looks into the buffer for a formatted string (function analogue to the standard-C s*scanf* function), leaving into the buffer all the bytes that doesn't correspond to the searched string (except the ones that are before the found string).

Parameters

- **Format String**: A format control string that defines how data is organized in the incoming string. The format specification follows the same rules as the C-standard s*scanf* function, as described in the string format page.
- **Number of char for msg terminator:** parameter that specifies the number of characters that should be considered as message terminator. Very often, the terminator char selected is the linefeed character (\n) that should be considered as one character.
- **Sample Time:** Indicates if the sample time should be inherited from the preceding blocks or should be defined independently (but it should be a multiple of the base sample time).
- **Buffer Allocation Step:** Each time the read operation need more space than the one already allocated, this block may allocate more space. The allocation of new memory is performed in step, to avoid a great number of allocation/reallocation operations. The bytes to allocate at each step is given by this parameter.

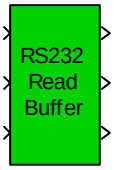
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the read operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its read operation.
- **Input 3**: The handle to the buffer in which the bytes read from the serial port is stored and added.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag that indicated the completion (successful or not) of the read operation (0: not completed or not enabled, 1: read complete).
- **Output 3:** the output port that contains the data read from the input string. The width of the port is in the same number of the format string argument (i.e.: the number of '%' chars). All the outputs are held until the next data incomes from the port (zero order hold). All the data output are stored, in the output, as double, even if the formatted data contains integer or boolean values.

Description



Remarks:

The block starts its read operations when received a '1' flag from the input 2. The read operation remains active until the completion of the data read (that may request more than one simulation step). Once completed the read of the string, it modify the output 3 with new data and put the output 2 flag to '1' for only one simulation step.

Each block instance use the buffer indicated by the handle (received as input 3) to accumulate pieces of data read from the serial port. The buffer is used in the next step from the same block or from another block to complete the read of the incoming message. The buffer is common for each instantiated block that receive the same handle and resizes dynamically to manage eventual data quantity peak or, more generally, the multitasking of the OS.

When the head of the message (all the part of the format message that precedes the first '%' character) is detected into the buffer, all the buffer that precedes the head is simply discarded and removed from the buffer. If both the head and the end of the message are detected in the buffer, the message is interpreted and, after, all the chars that precede (and contain) the end of the message are discarded from the buffer. See the example in the RS232 Read Buffer Behavior page.

While block is not active (the comm. handle is not valid (0) or the input flag (iput 2) is not active) and no read operation is pending, the block doesn't performs any operation.

RS232_Wait_Buffer_Synch

Purpose

Looks for messages headers into the buffer and returns the "code" of first message in the buffer. Read data from the serial port and add it to the buffer, but doesn't remove any byte from it.

Parameters

- **Synch String:** A list of message headers separated by comma or spaces.
- **Buffer Allocation Step:** Each time the read operation need more space than the one already allocated, this block may allocate more space. The allocation of new memory is performed in step, to avoid a great number of allocation/reallocation operations. The bytes to allocate at each step is given by this parameter.
- **Sample Time:** Indicates if the sample time should be inherited from the base sample time or should be defined independently (but it should be a multiple of the base sample time).

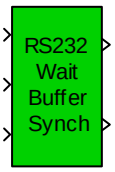
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the check operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its check operation.
- **Input 3**: The handle to the buffer in which the bytes read from the serial port is stored and added.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag vector that indicated the completion (successful or not) of the check operation (0: not completed or not enabled, 1: read complete). The size of the vector is equal to the number of token into the synch string.

Description



Remarks:

The block starts its check operations when received a '1' flag from the input 2. The check operation (that consists in the storing of the incoming bytes into the buffer until a message header is detected) remains active until the completion (that may request more than one simulation step). Once completed the read of the string, it modifies the output 2 with a flag '1' for only one simulation step into the component of the vector relative to the message header detected. All the flags output of this block must be managed by at least one block, or the read from serial port may block the read sequence.

Each block instance use the buffer indicated by the handle (received as input 3) to accumulate pieces of data read from the serial port. The buffer is used in the next step from the same block or from another block to complete the read of the incoming message. The

buffer is common for each instantiated block that receive the same handle and resizes dynamically to manage eventual data quantity peak or, more generally, the multitasking of the OS. See the example in the RS232 Read Buffer Behavior page.

While the block is not active (the comm. handle is not valid (0) or the input flag (input 2) is not active) and no read operation is pending, the block doesn't performs any operation.

The output 2 of this block should be propagated to all Read_Buffer blocks that needs a coordination in the read operation from the buffer.

RS232_Write_Binary

Purpose

Write a binary formatted data to the serial output.

Parameters

- **Format String**: The format of the data to send.
- **Sample Time:** Indicates if the sample time should be inherited from the base sample time or should be defined independently (but it should be a multiple of the base sample time).

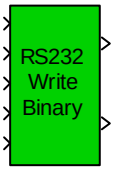
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the write operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its read operation. The block writes to the serial output each time the flag becomes 1.
- **Input 3..n:** the nth input port contains the data to write to the n-th format element of the output data. The width and the type of the input port is dependant of the format string argument, as reported in the Binary Format Page.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag that indicated the completion (successful or not) of the write operation (0: not completed or not enabled, 1: write complete).

Description



Remarks:

While the block is not active (the comm. handle is not valid (0) or the input flag (input 2) is not active) the block doesn't performs any operation.

RS232_Read_Binary_Buffer

Purpose

This block reads bytes from the serial port, add them to the input buffer and gets, starting at the start of the buffer, formatted data (the format is declared in the format string argument), leaving into the buffer all the bytes that exceeds the read operation.

Parameters

- **Format String**: A format control string that defines how data is organized in the incoming string. The format specification is contained in the Binary Format Page.
- **Sample Time:** Indicates if the sample time should be inherited from the preceding blocks or should be defined independently (but it should be a multiple of the base sample time).
- **Buffer Allocation Step:** Each time the read operation need more space than the one already allocated, this block may allocate more space. The allocation of new memory is performed in step, to avoid a great number of allocation/reallocation operations. The bytes to allocate at each step is given by this parameter.

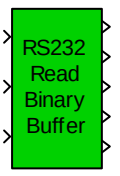
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the read operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its read operation.
- **Input 3**: The handle to the buffer in which the bytes read from the serial port is stored and added.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag that indicated the completion (successful or not) of the read operation (0: not completed or not enabled, 1: read complete).
- **Output 3..n:** the output port that contains the data read from the input data. The nth output port contains the read data relative to the n-th format element. The width and the type of the input port is dependant of the format string argument, as reported in the Binary Format Page. All the outputs are held until the next data incomes from the port (zero order hold).

Description



Remarks:

The block starts its read operations when received a '1' flag from the input 2. The read operation remains active until the completion of the data read (that may request more than one simulation step). Once completed the read of the data, it modify the output 3 with new data and put the output 2 flag to '1' for only one simulation step.

Each block instance use the buffer indicated by the handle (received

as input 3) to accumulate pieces of data read from the serial port. The buffer is used in the next step from the same block or from another block to complete the read of the incoming message. The buffer is common for each instantiated block that receive the same handle and resizes dynamically to manage eventual data quantity peak or, more generally, the multitasking of the OS.

When the input flag is "1", the buffer is read when and only when the buffer size is greater than the total size of the data to read. The total size is automatically calculated by the block. See the example in the RS232 Read Binary Buffer Behavior page. After the read is complete, the read portion of data is removed from the buffer.

While the block is not active (the comm. handle is not valid (0) or the input flag (iput 2) is not active) and no read operation is pending, the block doesn't performs any operation.

RS232_Peek_Binary_Buffer

Purpose

This block peeks bytes from the serial port, add them to the input buffer and gets, starting at the start of the buffer, formatted data (the format is declared in the format string argument), leaving unmodified the buffer (no bytes are removed from the buffer, eventually some data are added in the case that new data is in the input port).

The interface of this function is the same respect to the **RS232_Read_Binary_Buffer** block, except that after the read is complete, the read portion of data is NOT removed from the buffer.

Parameters

- **Format String**: A format control string that defines how data is organized in the incoming string. The format specification is contained in the Binary Format Page.
- **Sample Time:** Indicates if the sample time should be inherited from the preceding blocks or should be defined independently (but it should be a multiple of the base sample time).
- **Buffer Allocation Step:** Each time the read operation need more space than the one already allocated, this block may allocate more space. The allocation of new memory is performed in step, to avoid a great number of allocation/reallocation operations. The bytes to allocate at each step is given by this parameter.

Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the read operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its read operation.
- **Input 3**: The handle to the buffer in which the bytes read from the serial port is stored and added.

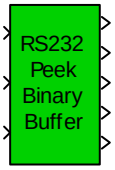
Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag that indicated the completion (successful or not) of the read operation (0: not completed or not enabled, 1: read complete).
- **Output 3..n:** the output port that contains the data read from the input data. The nth output port contains the read data relative to the n-th format element. The width and the type of the input port is dependant of the format string argument, as reported in the Binary Format Page. All the outputs are held until the next data incomes from the port (zero order hold).

Description

Remarks:

The block starts its read operations when received a '1' flag from the input 2. The read operation remains active until the completion of



the data read (that may request more than one simulation step). Once completed the read of the data, it modify the output 3 with new data and put the output 2 flag to '1' for only one simulation step.

Each block instance use the buffer indicated by the handle (received as input 3) to accumulate pieces of data read from the serial port. The buffer is used in the next step from the same block or from another block to complete the read of the incoming message. The buffer is common for each instantiated block that receive the same handle and resizes dynamically to manage eventual data quantity peak or, more generally, the multitasking of the OS.

When the input flag is "1", the buffer is read when and only when the buffer size is greater than the total size of the data to read. The total size is automatically calculated by the block. See the example in the RS232 Read Binary Buffer Behavior page. After the read is complete, the read portion of data is removed from the buffer.

While the block is not active (the comm. handle is not valid (0) or the input flag (iput 2) is not active) and no read operation is pending, the block doesn't performs any operation.

RS232_Synch_Binary_Buffer

Purpose

Looks for data header into the buffer and returns "1" at the output flag when the header is found.

Parameters

- **Synch Size (in Bytes):** The size of the synch header. The maximum size value is 4.
- **Mask:** The hexadecimal mask to use in the evaluation of the header.
- **Value:** The desired header value after that the header is filtered by the mask
- **Buffer Allocation Step:** Each time the read operation need more space than the one already allocated, this block may allocate more space. The allocation of new memory is performed in step, to avoid a great number of allocation/reallocation operations. The bytes to allocate at each step is given by this parameter.
- **Sample Time:** Indicates if the sample time should be inherited from the base sample time or should be defined independently (but it should be a multiple of the base sample time).

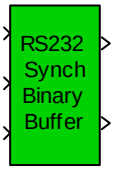
Input

- **Input 1**: the handle to an open communication link generated from a RS232 Setup block.
- **Input 2:** flag that enable the execution of the check operations (0: not enabled, 1: enabled). It is sufficient that the input flag becomes 1 once, to let the block executes its check operation.
- **Input 3**: The handle to the buffer in which the bytes read from the serial port is stored and added.

Output

- **Output 1**: A replication of the Input 1 handle
- **Output 2:** A flag vector that indicated the completion (successful or not) of the check operation (0: not completed or not enabled, 1: read complete). The size of the vector is equal to the number of token into the synch string.

Description



Remarks:

The block starts its check operations when received a '1' flag from the input 2. The check operation (that consists in the storing of the incoming bytes into the buffer until a new data header is detected) remains active until the completion (that may request more than one simulation step). Once found the data header, it modifies the output 2 with a flag '1' for only one simulation step. Once the data header is found, all the bytes before the data header are simply removed.

Each block instance use the buffer indicated by the handle (received as input 3) to accumulate pieces of data read from the serial port. The buffer is used in the next step from the same block or from

another block to complete the read of the incoming message. The buffer is common for each instantiated block that receive the same handle and resizes dynamically to manage eventual data quantity peak or, more generally, the multitasking of the OS. See the example in the RS232 Read Binary Buffer Behavior page.

While the block is not active (the comm. handle is not valid (0) or the input flag (input 2) is not active) and no read operation is pending, the block doesn't performs any operation.

The output 2 of this block should be propagated to the Read_Binary_Buffer block that needs a coordination in the read operation from the buffer.

RS232_Buffer_Size

Purpose

This block reads the status of the buffer in terms of buffer utilized space, buffer allocated space and input byte progressive count.

Parameters

- **Sample Time:** Indicates if the sample time should be inherited from the preceding blocks or should be defined independently (but it should be a multiple of the base sample time).

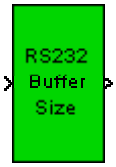
Input

- **Input 1**: The handle to the buffer in which the bytes read from the serial port is stored and added.

Output

- **Output 1**: A three element vector that output the following information:
  1. The actual utilized buffer;
  2. The actual allocated buffer space;
  3. The progressive count of the bytes stored in the buffer;

Description



Remarks:

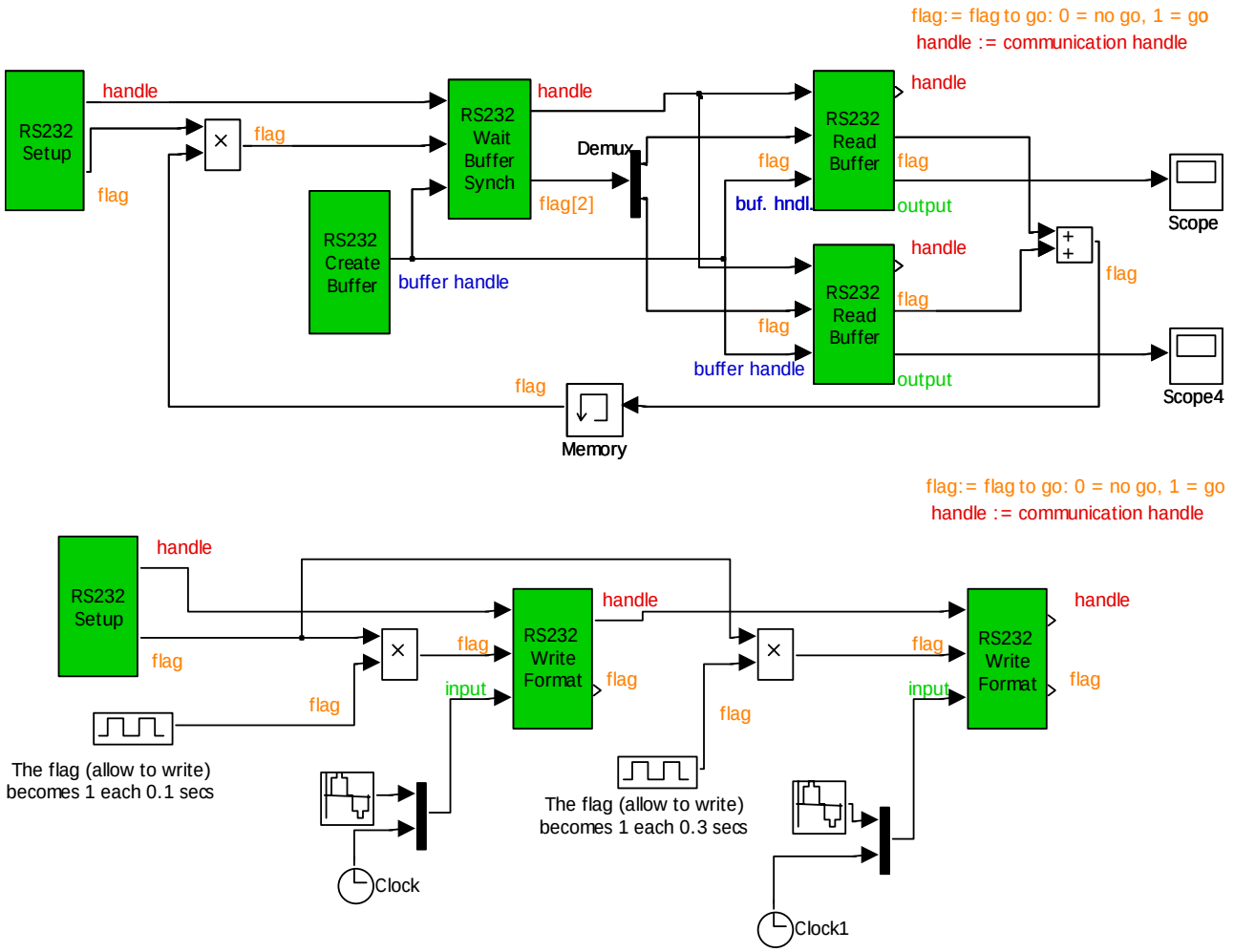
This block is always operative when a valid buffer handle is in the block input port. In the output vector, the following information are available:

1) The actual utilized buffer: the count of bytes actually contained in the buffer that may still be interpreted from the "Read" buffer functions (RS232_Read_Buffer, RS232_Wait_Buffer_Synch, RS232_Read_Binary_Buffer, RS232_Peek_Binary_Buffer, RS232_Synch_Binary_Buffer)
2) The actual allocated buffer space: the buffer space is allocated and deallocated in blocks and only when needed. The reallocation step can be fixed in each buffered function. Reallocation is performed only when needed to improve performances;
3) The progressive count of bytes stored in the buffer: this progressive count is update from all "read" buffered functions (not updated from no-buffered functions). Each byte read from the serial port, even if discarded, is taken into account in the progressive count.

RS232 blockset Application Examples

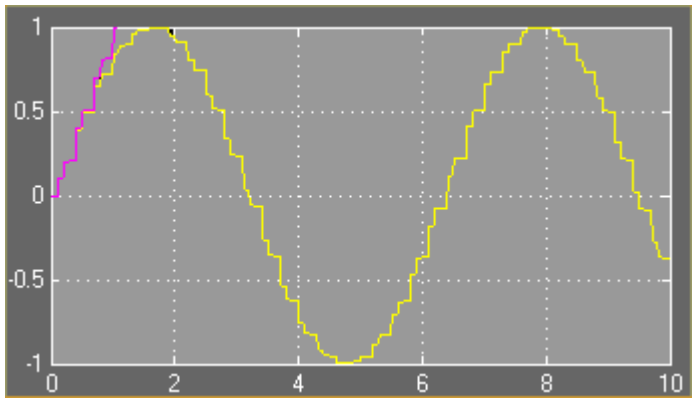
An application example is the one reported in the figure and contained in the installation pack of the blockset (RS232ex3.mdl).

To setup your computer for this example model, connect an inverting serial cable between the COM1 and COM2 ports.

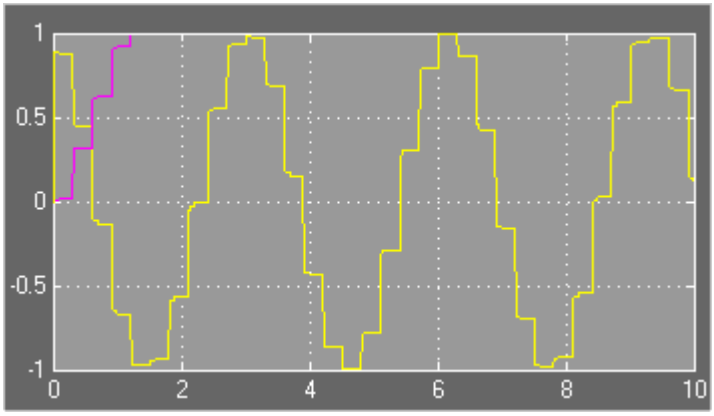


In this example, two asynchronous messages are sent from the Write Format block with different rates (the rate is generated by two pulse generator with different pulse period). A Wait Buffer Synch block is used to detect which message is first in the input queue.

Scope:



Scope4:



Information and Warranty for the End User

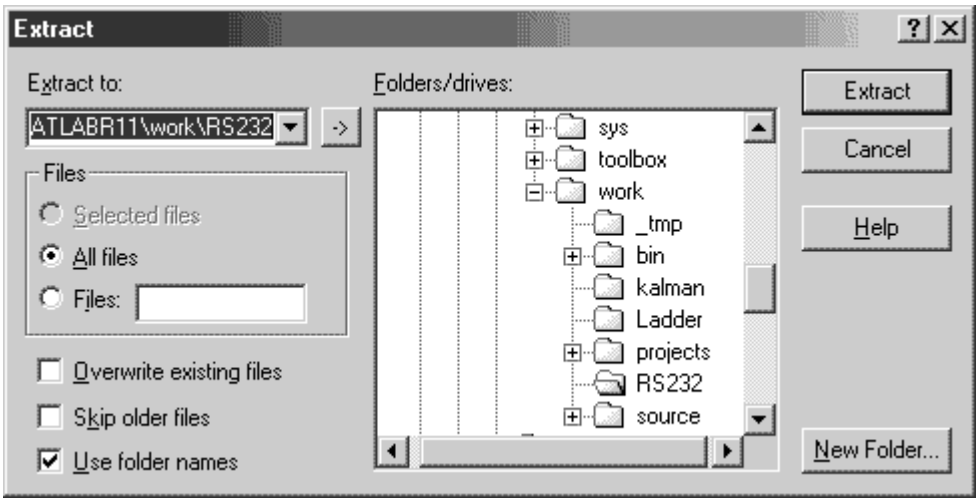
I must remember to all the users of this blockset that this is a product not covered by any warranty abut its working performances and characteristics, because it's completely free. All the material and the manual has been realized by myself independently from any contractual constraint with any University or company, then the intellectual property is exclusive of the author (me).

The present library is actually under development and more blocks will be added. The notice from the users of any problem will be appreciated and recorded and any collaboration will be mentioned in the final version of the blockset.

Installation of the RS232 blckset

At the start of this page, a link to the file RS232.zip allows the download of the RS232 blockset library. The zip file has been memorized without any reference to an installation directory, then the directory will be selected from the end user at the installation time.

After the start of the Winzip (or equivalent) program, press the button "Extract" to extract all files in the desired directory. Preferably, select a folder (or create a new folder) insied the <MatlabDir>\work directory.

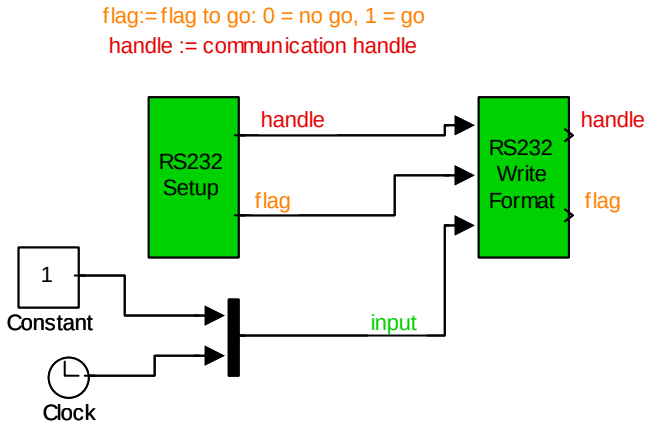


Installation Directory selection window

After the completion of the setup, add the path of the selected directory utilizing the menu item File-->Set Path...

Now the installation is complete. To open the RS232 Blockset library is sufficient to type in the Matlab command window the command "RS232". A window containing all the available blocks will be displayed.

Example of the Read Format block behavior.



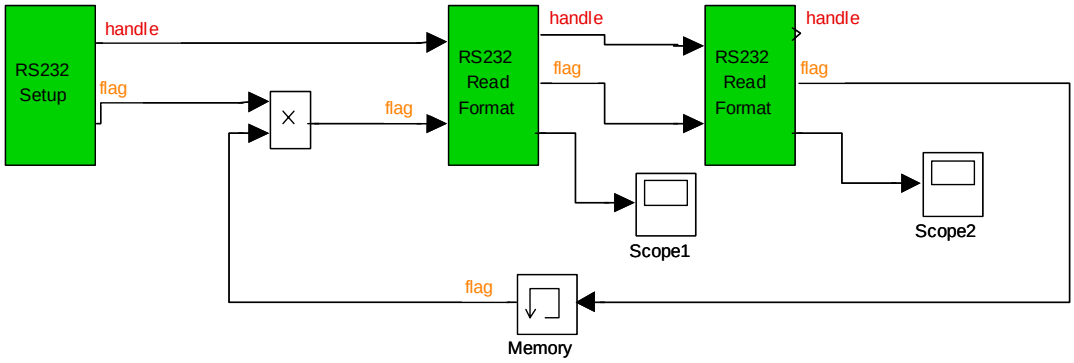
Incoming String	I1*	Buff1	O1*	Notes
	2	empty	0	Initial status
"4,end.#MSG1,12.5,"	2	"4,end.#MSG1,12.5,"	0	A not complete message with a previous not complete message is arrived
"end.#MSG1,23.5,end."	2	"4,end.**#MSG1,12.5,end.** #MSG1,23.5,end."	0	The rest of the message is arrived with a new complete msg. The block identifies the first msg head in the buffer and the first following message end.
""	0	"#MSG1,23.5,end."	1	the block outputs data. The control pass to the next block. The first message. with the part of the

				incomplete message, is removed from the internal buffer.
""	1	empty	1	the block outputs data. The control pass to the next block

Example of two Read Format block in cascade

All the block (that has an input flag) works only when the input flag is 1.
When the block proceeded the flag, it output 1 for only one step

flag:= flag to go: 0 = no go, 1 = go
handle := communication handle



Incoming String	I1*	Buff1	O1*	I2*	Buffer 2	O2*	Notes
	2	empty	0	0	empty	0	Initial status
"#MSG1,12.5,"	2	"#MSG1,12.5,"	0	0	empty	0	
"end.#MSG2,23.5,end."	2	"#MSG1,12.5,end. #MSG2,23.5,end."	0	0	empty	0	
""	0	"#MSG2,23.5,end."	1	0	empty	0	block 1 outputs data. The control pass to the 2nd block
"#MSG1,14.5,end."	0	"#MSG2,23.5,end."	0	1	"#MSG1,14.5,end."	0	block 2 take the control
""	0	"#MSG2,23.5,end."	0	2	"#MSG1,14.5,end."	0	
"#MSG2,26.3,end."	0	"#MSG2,23.5,end."	0	2	"#MSG1,14.5,end. #MSG2,26.3,end."	0	
""	0	"#MSG2,23.5,end."	0	0	empty	1	block 2 outputs data. The control pass to the 1st

							block
"#MSG1,14.5,end."	1	"#MSG2,23.5,end."	0	0	empty	0	
""	2	"#MSG2,23.5,end. #MSG1,14.5,end."	0	0	empty	0	
""	0	empty	1	0	empty	0	block 1 outputs data. The control pass to the 2nd block.

*Legenda for the status of the input ports:

0: Not enabled for write

1: Enabled for write

2: Wait for completion

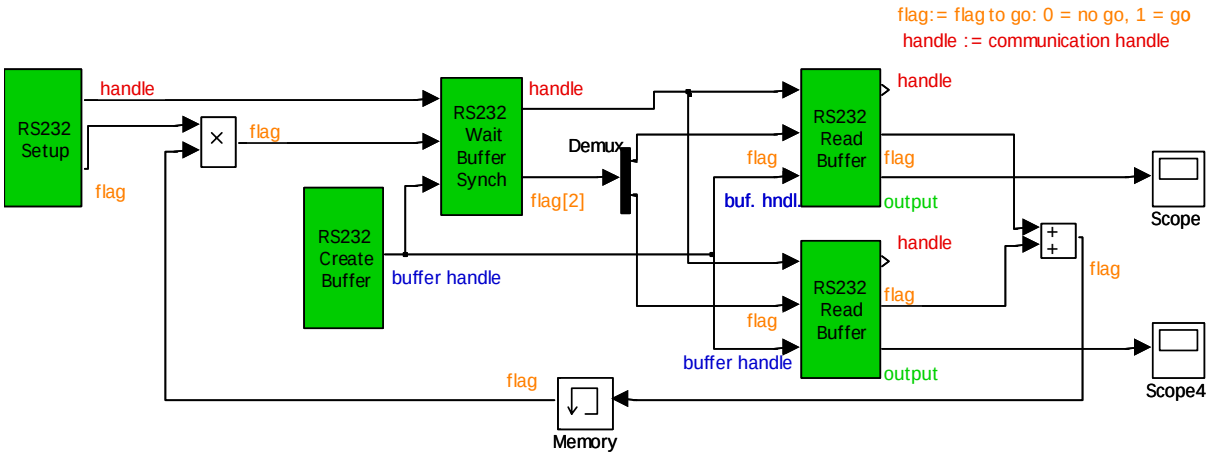
*Legenda for the status of the output ports:

0: Do Not Enable the next block

1: Enable the next block

Example of the Read Buffer and Synch Check blocks behavior.

In this example, two asynchronous msg are expected on the serial port. A buffer is used to detect which msg is first in the input queue. The wait buffer synch output flags are in the same number of the synch string indicated in the synch block panel. All the flags output of this block must be managed by at least one block, or the read from serial port may block the read sequence.



Incoming String from RS232	Common Buffer	In* CK	Out* CK		In* R1	Out* R1	In* R2	Out* R2	Notes
	empty	1	0	0	0	0	0	0	Initial status
"#MSG1,12.5,"	"#MSG1,12.5,"	2	1	0	0	0	0	0	Detected msg1.
""	"#MSG1,12.5,"	0	0	0	1	0	0	0	The control passed to the Read block 1.
"end.#MSG2,23.5,end."	"#MSG1,12.5,end. #MSG2,23.5,end."	0	0	0	2	1	0	0	Block 1 detected the message end. block 1 outputs data read from the serial port and remove the read portion from the buffer. The block left the control to the next block
"#MSG1,14.5,end."	"#MSG2,23.5,end. #MSG1,14.5,end."	1	0	1	0	0	0	0	The control back to the wait synch block. Detected msg1 and msg2. msg2 is first and then the control is left to the Read block2.
""	"#MSG2,23.5,end. #MSG1,14.5,end."	0	0	0	0	0	1	1	The control passed to the Read block 2.block 2 outputs data read from the serial port and remove the read portion from the buffer. The block left the control to the next block
"#MSG1,15.9,end."	"#MSG1,14.5,end. #MSG1,15.9,end."	1	1	0	0	0	0	0	The control back to the wait synch block. Detected msg1. the control is left to the Read block1.
""	"#MSG1,14.5,end. #MSG1,15.9,end."	0	0	0	1	1	0	0	The control passed to the Read block 1.block 1 outputs data read from the serial port and remove the read portion from the buffer. The block left the control to the next block
""	"#MSG1,15.9,end."	1	1	0	0	0	0	0	The control back to the wait synch block. Detected msg1. the control is left to the Read block1.

""	"#MSG1,15.9,end."	0	0	0	1	1	0	0	The control passed to the Read block 1. block 1 outputs data read from the serial port and remove the read portion from the buffer. The block left the control to the next block
""	empty	1	0	0	0	0	0	0	The control back to the wait synch block. Wait for a new message.
""	empty	2	0	0	0	0	0	0	Wait for a new message.

*Legenda for the status of the input ports:

0: Not enabled for write

1: Enabled for write

2: Wait for completion

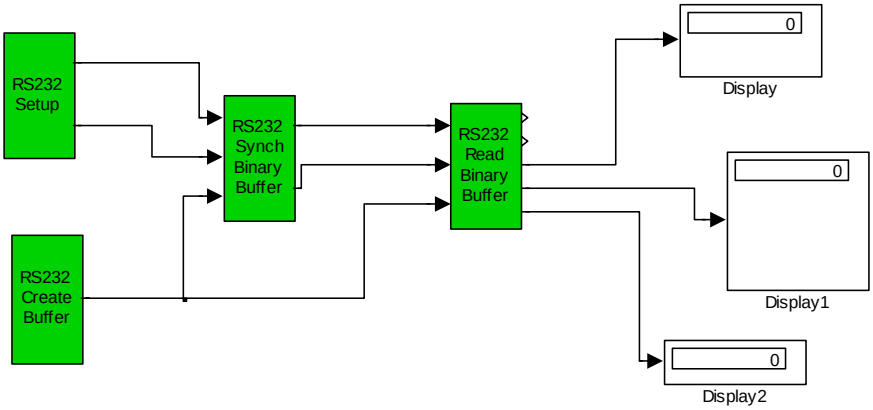
*Legenda for the status of the output ports:

0: Do Not Enable the next block

1: Enable the next block

Example of the Read Binary Buffer and Synch Check blocks behavior.

In this example, a data message, mixed with other spurious bytes, is expected on the serial port. A buffer is used in order to avoid the lose any data from the buffer. When the output flag of the synch block becomes 1, the read binary block is allowed to read data from buffer. Remember that the Read Binary Buffer block doesn't remove any byte from the buffer until the read of the data (the needed size of the buffer to contain all the desired data is present) is possible.



Behavior table:

- header mask: 0xffff
- header value: 0x0209

- header size: 2
- data format: 2i1,3u4,1d8 (2 int8, 3 unsigned int32, 1 double)

Incoming String from RS232 (hex)	Common Buffer	In* CK	Out+ CK	In* RB	Out+ RB	Notes
00 01 04 06 07	00 01 04 06 07	1	0	0	0	The header value searched is not in the buffer. All the bytes are
00 02 09 01 00 00 00	00 02 09 01 00 00 00	2	1	1	0	The header value searched is in the buffer. All the bytes before 02 09 are
06 00 00 00 03 00 00 00 00 00 00	02 09 01 00 00 00 06 00 00 00 03 00 00 00 00 00 00	1	1	1	0	The check block still detected the presence of the header. The buffer is not large enough to contain all data needed from the read operation. The buffer remains unchanged.
00 F0 3F 71 77 72 AB 3F	02 09 01 00 00 00 06 00 00 00 03 00 00 00 00 00 00	1	1	1	1	The check block still detected the presence of the header. The buffer is large enough to contain all data needed from the read operation. The data read is removed from the buffer.
03 00 00 00 00 00 00 00 00 F0 3F 71 77 72	71 77 72 AB 3F 03 00 00 00 00 00 00 00 00 F0 3F	1	0	0	0	The synch block didn't detected any header. All the data bytes will be removed at the next step (only the last header-size number of bytes is not removed from the buffer.
-	71 77 72	1	0	0	0	No data at input
04 06 07 02 09 01 00 00 00 06 00 00 00 03 00 00 00 00 00 00 00	71 77 72	1	1	1	0	The header value searched is in the buffer. All the bytes before 02 09 are discarded. The control is passed to the read block that has not enough bytes in the buffer input. The read block doesn't modify the buffer.

*****Legenda for the status of the input ports:

- 0: Not enabled for write
- 1: Enabled for write
- 2: Wait for completion

*****Legenda for the status of the output ports:

- 0: Do Not Enable the next block
- 1: Enable the next block

Format Specification Fields

A format specification has the following form:

% [width] [l] type

The *format* argument specifies the interpretation of the input and can contain one or more of the following:

- White-space characters: blank (' '); tab ('\t'); or newline ('\n'). A white-space character causes the block to read, but not store, all consecutive white-space characters in the input up to the next non–white-space character. One white-space character in the format matches any number (including 0) and combination of white-space characters in the input.
- Non–white-space characters, except for the percent sign (%). A non–white-space character causes the block to read, but not store, a matching non–white-space character. If the next character in the **input** buffer does not match, the block terminates the read, sending an error.
- Format specifications, introduced by the percent sign (%). A format specification causes the block to read and convert characters in the input into values of a specified type. The value is assigned to an argument in the argument list.

The format is read from left to right. Characters outside format specifications are expected to match the sequence of characters in **input buffer**; the matching characters in input buffer are scanned but not stored. If a character in input buffer conflicts with the format specification, **the block** terminates, and the character is left in input buffer as if it had not been read.

When the first format specification is encountered, the value of the first input field is converted according to this specification and stored in the location that is specified by the first *argument*. The second format specification causes the second input field to be converted and stored in the second *argument*, and so on through the end of the format string.

An input field is defined as all characters up to the first white-space character (space, tab, or newline), or up to the first character that cannot be converted according to the format specification, or until the field width (if specified) is reached. If there are too many arguments for the given specifications, the extra arguments are evaluated but ignored. The results are unpredictable if there are not enough arguments for the format specification.

Each field of the format specification is a single character or a number signifying a particular format option. The *type* character, which appears after the last optional format field, determines whether the input field is interpreted as a character, a double, or an integer.

The simplest format specification contains only the percent sign and a *type* character (for example, %d). The percent sing is always intended as a format-control character, except in the case that another percent sing follows. Then, to specify that a percent-sign character is to be input, use %%.

Format

The *type* character is the only required format field; it appears after any optional format fields. The *type* character determines whether the associated argument is interpreted as a character, string, or number.

Table 1 Type Characters for format scan functions

Character	Type of Input Expected
d	Decimal integer.
i	Decimal, hexadecimal, or octal integer.
o	Octal integer.
x	Hexadecimal integer.
e, **E, f**, **g**, **G**	Floating-point value consisting of optional sign (+ or –), series of one or more decimal digits containing decimal point, and optional exponent ("e" or "E") followed by an optionally signed integer value.

Format Width Specification

width is a positive decimal integer controlling the maximum number of characters to be read from **input buffer**. No more than *width* characters are converted and stored at the corresponding *argument*. Fewer than *width* characters may be read if a white-space character (space, tab, or newline) or a character that cannot be converted according to the given format occurs before *width* is reached.

The optional prefixe **l** indicate the "size" of the *argument* (long or short, single-byte character or wide character, depending upon the type character that they modify). This format-specification character is used with type characters in **format block** functions to specify interpretation of arguments as shown in the Table 2. The type characters and their meanings are described in Table 1

Table 2 Size Prefixes for format block Format-Type Specifiers

To Specify	Use Prefix	With Type Specifier
double	l	e, E, f, g, or G
long int	l	d, i, o, x, or X

Binary Format Specification Fields

A format specification has the following form:

[spec1],[spec2],...,[specn]

Each specification is associated to an output port that is created soon after the specification string is read and the block panel is closed.

Each specification is structured as follows:

[n][Type][bytes]

where:

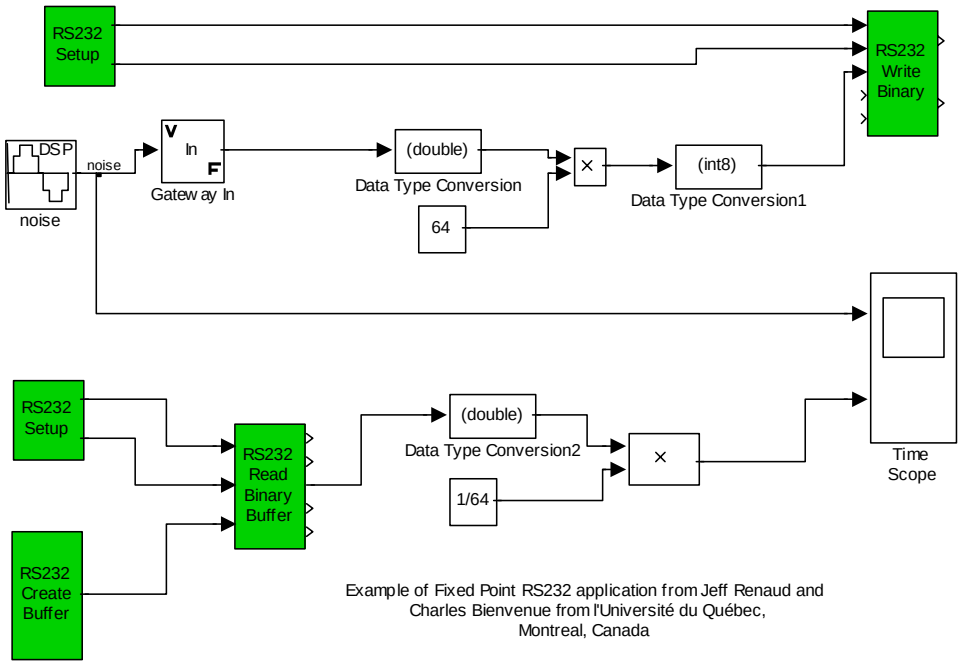
- **n** is the width of the relative port, i.e. the number of consecutive data to read from serial port
- **Type:** is one of the following**: d, i, u, b,** where**:**
  - **d:** indicates a double
  - **i:** integer
  - **u:** unsigned
  - **b:** boolean
- **bytes:** indicate the number of bytes that compose the incoming data

Example:

- **2i1,3u4,1d8** means: 3 ports, with respectively 2 signed bytes, 3 unsigned integers, 1 double
- **6i2,u2,d4,d8** means: 4 ports, with respectively 6 signed words, 1 single float, 1 double float

Application Examples from Users

An example of Fixed Point application from Jeff Renaud and Charles Bienvenue from l'Université du Québec, Montreal, Canada.



Acknowledgements

Many thanks for their suggestions and requests:

- Nelson Rojas Mendelewski, from Chile, for his suggestions and requests about many topics.
- Juan Francisco Blanes Noguera, from Valencia (Espana), for his requests about upgrade of the RS232_Read_Format block.
- Pramlia Rani, from Vanderbilt University (Nashville, Tennessee - USA) that suggested me the creation of the binary blocks.
- Francesco Nebula and Domenico Liguori from CIRA (Italy) for their update request.

Many thanks for their debugging activity:

- Hugo Montenegro, from Argentina, for his debugging activity on the RS232_Wait_Buffer_Synch block.
- Jens Haecker, from Institute for Statics and Dynamics of Aerospace Structures (ISD), University of Stuttgart (Germany), for his requests about the RS232_Format_Read for his debugging on the formatted write and read.
- Lutz Evert, from FH-Bielefeld, Germany, for his debugging activity.

Many thanks for their encouragement and contributes:

- Jeff Renaud and Charles Bienvenue from l'Université du Québec, Montreal, Canada
- Ciamak Abkai, from Germany
- Cord Elias, from Mathworks Gmbh, Germany
- Victor Polidoro, from MIT, Boston, USA
- Gordon Strickert, from Germany
- Stefan Thomas, from Wien, Austria

- Juan Francisco Blanes Noguera, from Escuela Universitaria de Informática, Valencia, Spain
- Peter How, from MIT, Boston, USA
- Tarif Erf, from Turkey
- Phaisit Chewputtanagul, from I don't know where
- Thorsten Knappenberger, from Germany
- Sithabiso Madlala, from Zaire
- Shu-An Lee, from Taiwan
- Wesley Yu, from Republic of China
- Jacy Montenegro Magalhaes Neto, from Instituto Militar de Engenharia, Rio de Janeiro, Brasil
- Costanzo Salvatore and Gullì Daniel, from Genova (Italy)