iSense Serial Data Transmission User Guide

Document history

Version	*Date*	*Changes*
1.0	10.03.2012	Initial version

Contents

1. *About this User Guide*

This guide shows how exchange data with the serial interface of iSense nodes both in the plain bytes modes, as well as in packet mode.

In this user guide,

- files and folders are represented in the Arial typeface,
- code fragments, function names etc. are represented in the `Courier New` typeface,
- GUI elements such as button descriptions etc. are represented in "quotation marks",
- titles of other documents are presented in *Italic* type.

This manual assumes that the reader has successfully installed the iSense development environment, and obtained either the iSense standard firmware or a custom firmware from the iSense firmware web configuration interface. For further information on these steps, consult the *Development Environment Setup User Guide* [1].

In addition, it is assumed that the user is familiar with the use of iShell. For further information on iShell, consult the *iShell User Guide* [2].

For further information on iSense firmware programming concepts and on application development, it is recommended to read the *Writing iSense Applications User Guide* [3].

2. Serial Data Transmission Hardware

The following iSense devices and modules can be used for serial data transmission:

Order code	*Description*	*RS232 serial connection*	*Virtual COM port (VCP) connection*
USB10x	Wireless IEEE 802.15.4 Gateway USB Stick		X
GM10U	iSense Gateway Module with USB cable (together with iSense Core Modules CM10x, CM20x and CM30x)		X
GM10S	iSense Gateway Module with serial RS232 cable (together with iSense Core Modules CM10x, CM20x and CM30x)	X	
GM20-2P	iSense Gateway Module 2 with USB cable and 2 Panasonic connectors (together with iSense Core Modules CM10x, CM20x and CM30x as well as NET10x)		X
GM20-1P	iSense Gateway Module 2 with USB cable and 1 Panasonic connector (together with iSense Core Modules CM10x, CM20x and CM30x)		X

For connections to external devices such as a PC, UART0 must be used, while UART1 is used for communication with other iSense components such as the serial camera, the GPS module etc.

In the remainder of the document, data transmission with an external devices is assumed, so all examples use UART0.

3. Serial Data Transmission Software

The iSense firmware API allows serial data transmission in two ways:

1. Sending and receiving of plain byte data
2. Sending and receiving data packets using the iSense serial packet format

Please be aware that iShell by default is set to operate in packet mode. It can be set into the plain text mode, where the "Serial Monitor" plugin will show all data received from the serial connection. However, this will render all plugins except the "Serial Monitor" dysfunctional, as they rely on packet mode transmission. The remainder of the document assumes that iShell is set to the packet mode.

To send and receive serial data, iSense devices use the class `Uart`. Applications must obtain a platform specific UART object reference by including

```
#include <isense/uart.h>
```

and calling

```
Uart& uart = os().uart(0);
```

By default, the UART is configured to operate at 115200 baud, using no flow control, 8 data bits, no parity bit, and 1 stop bit per character (8N1 encoding).

Be aware that all write operations to a UART, be it directly or through other API calls described later in this document, are blocking operations.

3.1.Plain Byte Data Transmission

To send plain byte data, applications can use the `write_buffer` method:

```
//-----------------------------------------------------------------------------
/**
 *
 * @param buf The data to be transmitted
 * @param length The number of bytes to transmit
 */
void write_buffer (const char *buf, size_t length);
```

The below example sends 5 bytes of data via UART0:

```
Uart& uart = os().uart(0);
uint8 buf[5];
buf[0] = 0;
buf[1] = 1;
buf[2] = 2;
buf[3] = 3;
buf[4] = 4;
uart.write_buffer(buf, 5);
```

Alternatively, the method `Uart::put` can be used to write single data bytes to the UART:

```
Uart& uart = os().uart(0);
uint8 b = 5;
uart.put(b);
```

To receive plain byte data, applications must inherit from the class `Uint8DataHandler` defined in `<isense/data_handlers.h>`

```
class Uint8DataHandler :
        public iSenseObject
{
public:
        #if defined (ISENSE_SHAWN) || not defined (ISENSE_AVOID_EMPTY_DESTRUCTORS)
                /* virtual Destructor to avoid compiler warnings */
                virtual ~Uint8DataHandler() {}
        #endif

        /* This method must be implemented by any Uint8DataHandler and
         * is called when a uint8 is available
         *
         * \param data the data to be handled
         *
         */

        virtual void handle_uint8_data( uint8 data ) = 0;

};
```

and implement the method `handle_uint8_data`. It will be called whenever UART data arrives. Be aware that it is called in the interrupt context, so no long running actions should be taken within that method.

Then, UART interrupts must be enabled and the application must set itself as the interrupt handler:

```
Uart& uart = os().uart(0);
uart.enable_interrupts(true);
uart.set_uint8_data_handler(this);
```

3.2.Packet Data Transmission

iSense devices use a particular data packet format to exchange data with the PC tool iShell. However, any application that can send and receive byte data to/from the serial connection can be used on the PC side to interact with iSense nodes.

3.2.1. Packet Data Format Specification

Data packets start and end with 2 special control characters: DLE and STX signal a packet start, DLE and ETX signal the end of a packet. The third character of a packet indicates its type (c.f. Section 0). Figure 1 shows the packet structure.

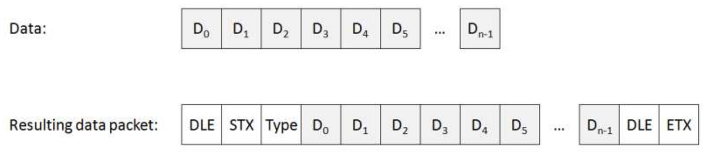
Data:
D_0 | D_1 | D_2 | D_3 | D_4 | D_5 | ... | D_{n-1}
Resulting data packet:
DLE | STX | Type | D_0 | D_1 | D_2 | D_3 | D_4 | D_5 | ... | D_{n-1} | DLE | ETX

Figure 1: Schematic depiction of the data exchange packet format

Symbol Name	*Description*
DLE	Delimiter, always set to 10h
STX	Start byte, always set to 2h
Type	Packet type (c.f. section 0)
D_0 - D_{n-1}	n data bytes
ETX	End byte, always set to 3h

Obviously, special precautions must be taken if type, length or data contain the DLE character. In such a case, byte stuffing is used, i.e. an extra DLE is added for each DLE in type or data, which must be removed again when receiving the data. The figure below shows an example of 8 data bytes to be sent as a packet of type 4.

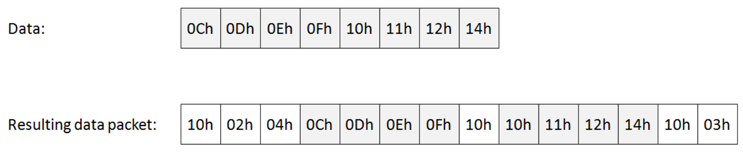
Data:
0Ch | 0Dh | 0Eh | 0Fh | 10h | 11h | 12h | 14h
Resulting data packet:
10h | 02h | 04h | 0Ch | 0Dh | 0Eh | 0Fh | 10h | 10h | 11h | 12h | 14h | 10h | 03h

Figure 2: Example of a data packet

3.2.2. *Packet Data Type Specification*

The different packet types exchanged between iSense devices and the PC software are distinguished by their packet types.

Packets from the PC to the iSense devices must have types ranging from 0 to 13, where most types are reserved for internal purposes, while types `MESSAGE_TYPE_CUSTOM_IN_1` (10), `MESSAGE_TYPE_CUSTOM_IN_2` (11) and `MESSAGE_TYPE_CUSTOM_IN_1` (12) are available for application use. The table below gives an overview of currently defined packet types.

Message Type Name	*Value*	*Application use*
MESSAGE_TYPE_CODE	0	No
MESSAGE_TYPE_RESET	1	No
MESSAGE_TYPE_SERAERIAL	2	No
MESSAGE_TYPE_TIME	3	No
MESSAGE_TYPE_CAMERA_APPLICATION	4	No
MESSAGE_TYPE_AMR_APPLICATION	5	No
MESSAGE_TYPE_ACC_APPLICATION	6	No
MESSAGE_TYPE_VIRTUAL_RADIO_IN	7	No
MESSAGE_TYPE_IN_RESERVED_2	8	No
MESSAGE_TYPE_IN_RESERVED_3	9	No
MESSAGE_TYPE_CUSTOM_IN_1	10	Yes
MESSAGE_TYPE_CUSTOM_IN_2	11	Yes
MESSAGE_TYPE_CUSTOM_IN_3	12	Yes
MESSAGE_TYPE_NETWORK_IN	13	No

Packets from iSense devices to the PC side must have types larger or equal to 100, where types up to 115 are reserved for internal purposes. When defining your own packets, you should stick to packet types larger than 115. The table below gives an overview of currently defined packet types.

Message Type Name	*Value*	*Application use*
MESSAGE_TYPE_OUT_RESERVED_1	100	No
MESSAGE_TYPE_OUT_RESERVED_1	101	No

MESSAGE_TYPE_OUT_RESERVED_1	102	No
MESSAGE_TYPE_MEM_DEBUG	103	No
MESSAGE_TYPE_LOG	104	No
MESSAGE_TYPE_PLOT	105	No
MESSAGE_TYPE_CUSTOM_OUT	107	No
MESSAGE_TYPE_JPEG	108	No
MESSAGE_TYPE_TIMEREQUEST	109	No
MESSAGE_TYPE_AUDIO	110	No
MESSAGE_TYPE_SPYGLASS	111	No
MESSAGE_TYPE_FLOAT_BUFFER	112	No
MESSAGE_TYPE_SQL	113	No
MESSAGE_TYPE_VIRTUAL_RADIO_OUT	114	No
MESSAGE_TYPE_NETWORK_OUT = 115	115	No

3.2.3. Sending Packet Data

To send arbitrary packet data, applications can use the `write_packet` method:

```
//----------------------------------------------------------------------------
/** Send a packet of a certain type. This should be used to transmit user-defined
 * data packets to the serial link.
 *
 * @param type The type of the packet
 * @param buf The data to be transmitted
 * @param length The number of bytes to transmit
 */
virtual void write_packet (unsigned char type, const char *buf, size_t length);
```

The below example sends a packet of type 127 with 5 bytes of data via UART0:

```
Uart& uart = os().uart(0);
uint8 buf[5];
buf[0] = 0;
buf[1] = 1;
buf[2] = 2;
buf[3] = 3;
buf[4] = 4;
uart.write_buffer(127, (char*)buf, 5);
```

However, for particular packet types, there are special ready-made functions in the iSense API. They are introduced in the sections below.

3.2.4. Sending Debug and Fatal Text Data

For sending text messages to iShell to be shown in the “Serial Monitor” plugin, you can use the methods `Os::debug` and `Os::fatal`.

```
// text shown as debug message (grey on white)
for (uint8 i = 1; i <= 3; i++)
        os().debug(“Hello World %u”, i);
// text shown as fatal message (white on red)
for (uint8 i = 1; i <= 3; i++)
        os().fatal(“Critical message no %u”, i);
```

3.2.5. Sending Multi Dimensional Sensor Data

For sending buffers with sensor data, applications can use the method `IShellInterpreter::send_buffer_data` (include `<isense/util/ishell_interpreter.h>` and `<isense/data_handlers.h>`):

```
/**
 * Sends the buffer_data to iShell as isense::Uart::MESSAGE_TYPE_INT_BUFFER
 *
 * \param buffer_data The buffer with the data to be sent to iShell.
 * \param id The id of the corresponding device.
 * \param interval The time in milliseconds between two values.
 */
bool send_buffer_data(BufferData* buffer_data, uint16 id, uint16 interval);
```

The example below shows how to send 10 triples of fake data to iShell. It will be shown in the “Curve Plotter” plugin of iShell if it is activated.

```
IShellInterpreter isi(os());
// get current sensor node time
Time t = os().time()
BufferData b;
// set time of the first sample
b.sec = t.sec();
b.ms = t.ms();
// set dimension to indicate 3 dimensional data
b.dim = 3;
// set number of triples to 10
b.count = 10;
// fill buffer with some bogus “sensor data”
for (uint8 count = 0; count < 10; count++)
{
        for (uint8 dim = 0; dim < 3; dim++)
        {
                b.buf[count*b.dim+dim] = os().rand(100);
        }
}
// send data to iShell with the local ID and a spacing of triples of 100ms
```

```
        isi.send_buffer_data(&b, (uint16)os().id(), 100);
```

3.2.6. Sending Audio Alarm Notifications

To send a packet to iShell that makes the iShell “Audio” plugin play a sound, applications can use the `write_packet` method. The example below shows how to send an according packet:

```
        Uart& uart = os().uart(0);
        // Types of sound
        // 0: play info sound “blib”
        // 1: play warning sound “ping”
        // 2: play engine sound
        // 3: play alert sound 1 (Windows alert sound)
        // 4: play alert sound 2 (Windows chord sound)
        // 5 play buzzer alarm sound
        uint8 buf = 0; // set sound to info
        uart.write_packet(MESSAGE_TYPE_AUDIO, &buf, 1);
```

3.2.7. Sending Audio Alarm Notifications

To send a packet to iShell that makes the iShell “Audio” plugin play a sound, applications can use the `write_packet` method. The example below shows how send an according packet:

```
        Uart& uart = os().uart(0);
        // Types of sound
        // 0: play info sound “blib”
        // 1: play warning sound “ping”
        // 2: play engine sound
        // 3: play alert sound 1 (Windows alert sound)
        // 4: play alert sound 2 (Windows chord sound)
        // 5 play buzzer alarm sound
        uint8 buf = 0; // set sound to info
        uart.write_buffer(MESSAGE_TYPE_AUDIO, &buf, 1);
```

3.2.8. Sending JPEG Image Data

For sending image data, applications can use the method `IShellInterpreter:: send_jpeg` (include `<isense/util/ishell_interpreter.h>`):

```
        /**
         * Sends a jpeg image that of size bytes to iShell
         *
         * \param image pointer to the JPEG image
         * \param size size of the image
         */
        void send_jpeg(uint8* image, uint16 size);
```

The example below shows how to send a JPEG image. It is assumed that the `image` pointer points to the buffer holding the JPEG image (e.g. because it was received from the camera) of size `size`. It will be shown in the "JPEG Illustrator" plugin of iShell if it is activated.

```
IShellInterpreter isi(os());
// get current sensor node time
uint8* image = NULL;
uint16 size = 0;

// set image and size here


// send image to iShell
isi.send_jpeg(image, size);
```

3.2.9. *Receiving Packet Data*

To receive packet data, applications must inherit from the class `UartPacketHandler` defined in `<isense/uart.h>`

```
//----------------------------------------------------------------------------
/** Callback handler for user-defined packets that arrive via the serial
 * interface. Implement this interface to receive these packets. See
 * Uart::register_packet_handler(uint8 type, UartPacketHandler* uph );
 * for detailed information (especially on how to use the type field).
 */
class UartPacketHandler : public iSenseObject
{
public:
        #if defined (ISENSE_SHAWN) || not defined (ISENSE_AVOID_EMPTY_DESTRUCTORS)
                virtual ~UartPacketHandler(){}
        #endif

        /** Called by the UART on packet reception
         * @param type The type of the received packet
         * @param buf The received content
         * @param length The number of uint8 elements in the buf array
         */
        virtual void handle_uart_packet( uint8 type, uint8* buf, uint8 length ) = 0;
};
```

and implement the method `handle_uart_packet`. It is called whenever a UART packet of the type the class is registered for arrives. Be aware that it is called in the interrupt context, so no long running actions should be taken within that method.

Then, UART interrupts must be enabled and the application must set itself as the packet handler for the packet type(s) it wants to receive:

```
// obtain uart reference
Uart& uart = os().uart(0);
// enable uart interrupts
uart.enable_interrupts(true);
// set this object to be the packet handler for type MESSAGE_TYPE_CUSTOM_IN_1
uart.set_packet_handler(MESSAGE_TYPE_CUSTOM_IN_1, this);
```

4. References

[1] coalesenses Development Environment Setup User Guide, online available at http://www.coalesenses.com/index.php?page=development-environent

[2] coalesenses iShell User Guide, online available at http://www.coalesenses.com/index.php?page=development-environent

[3] coalesenses Writing iSense Applications User Guide, online available at http://www.coalesenses.com/index.php?page=isense-applications

coalesenses GmbH
Maria-Goeppert-Str. 1
23562 Lübeck
Germany

www.coalesenses.com
sales@coalesenses.com