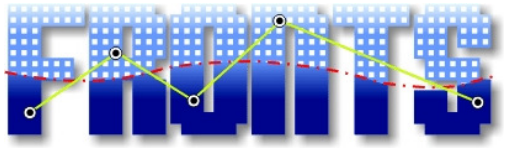


UZL-Testbed User Guide

Document history

Version	Date	Changes
1.0	28.06.2009	Initial version

Contents

1. About this User Guide

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 the iSense standard firmware. For further information on theses 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. IShellIntepreter API Description

The `IShellInterpreter` class in the `isense` namespace represents an interpreter between iShell and iSense and contains software functionality for communicating via uart 0. It provides methods to

- enable and disable iSeraerial and SerialRouting functionality on the connected node which is explained below,
- let the connected node demand for the iShell time,
- send a buffer with data to iShell

The `IShellInterpreter` class is defined in src/isense/util/ishell_interpreter.h in the iSense directory.

While iSeraerial functionality means data forwarding from uart to radio and radio to uart, SerialRouting means data forwarding from the uart to a routing protocol, i.e. a packet from iShell is sent out via the specified outgoing interface, e.g. the tree routing or flooding protocol.

SerAerialFilters decide if an incoming packet is forwarded. The default is that a packet is not forwarded. If iSeraerial is enabled, all registered filters are asked if the forwarding is required. If no filter returns true, the packet is not forwarded. Like this the number of forwarded packets is reduced to a minimum.

The API description also indicates the make targets for which the different functions are available.

2.1. Constructor

```
IShellInterpreter::IShellInterpreter(Os &os);
```

Description:

Generates an instance of the `IShellInterpreter` class and initializes it.

Parameters:

`os` Reference to the operating system class `Os`

Required modules:

IShellInterpreter* `#define ISENSE_ENABLE_ISHELL_INTERPRETER`

Available for the targets:

all

2.2. get_time_from_ishell

```
bool IShellInterpreter::get_time_from_ishell( TimeHandler* th);
```

Description:

This method demands the TimeProvider-Plugin of iShell for the time of the connected computer. The time is sent back to the connected node as UTC time.

Note, that this can fail (even though true was returned) if there is no connection or if the TimeProvider-Plugin is inactive or the device is sleeping. In these cases, the handle_time method of the given TimeHandler is never called. Hence, make sure that the device is not allowed to sleep when calling this method.

If called, the handle_time method of the TimeHandler is called in the interrupt context.

Parameters:

`bh` Pointer to the TimeHandler that should be called upon reception of the iShell time.

`return` `false if the IShellInterpreter failed to register for handling UART packets, true otherweise`

Required modules:

IShellInterpreter* `#define ISENSE_ENABLE_ISHELL_INTERPRETER`
`#define ISENSE_ENABLE_ISI_TIMEHANDLING`

Available for the targets:

all

2.3.send_jpeg

```
bool IShellInterpreter::send_jpeg(uint8* image, uint16 size);
```

Description:

This method sends a jpeg image of size bytes to iShell. Devices equipped with a Security Module and a camera can send such an image for instance.

Parameters:

`image` Image pointer to the JPEG image.

`size` Size of the image.

Required modules:

IShellInterpreter* `#define ISENSE_ENABLE_ISHELL_INTERPRETER`
`#define ISENSE_ENABLE_ISI_JPEG_SENDING`

Available for the targets:

all

2.4.send_buffer_data

```
bool IShellInterpreter::send_buffer_data (BufferData* data, uint16 id,
uint16 interval);
```

Description:

This method sends a BufferData to iShell with type MESSAGE_TYPE_INT_BUFFER. The accelerometer of the Security Module generates such data for example.

Parameters:

`data`	The buffer with the data to be sent to iShell.
`id`	The id of the corresponding device.
`interval`	The time in milliseconds between two values.

Required modules:

IShellInterpreter*	`#define ISENSE_ENABLE_ISHELL_INTERPRETER`
	`#define ISENSE_ENABLE_ISI_BUFFERSENDING`

Available for the targets:

all

2.5. add_seraerial_filter

```
bool IShellInterpreter::add_seraerial_filter( SerAerialFilter* filter);
```

Description:

Adds a `SerAerialFilter` to a list of filters which are all called on reception of a (radio or uart) packet.

Parameters:

`filter`	Pointer to the `SerAerialFilter` that should be asked if a packet should be forwarded.
`return`	`false` if the `IShellInterpreter` failed to register for handling UART packets or the maximum number of filters was already reached, `true` otherwise

Required modules:

IShellInterpreter*	`#define ISENSE_ENABLE_ISHELL_INTERPRETER`
	`#define ISENSE_ENABLE_ISI_SERAERIAL`

Available for the targets:

all

3. SerAerialFilter API Description

The `SerAerialFilter` class is defined in src/isense/util/seraerial_filter.h in the iSense directory.

Instances of classes which inherit from `SerAerialFilter` can be registerd at the `IShellInterpreter` in order to decide if packets shall be forwarded to another interface. They must implement the inherited method `forward_message`.

3.1.forward_message

```
bool SerAerialFilter::forward_message(const uint8* buf, uint8 len, bool
from_radio) = 0;
```

Description:

Can permit the forwarding of the actual message in the buffer.

Parameters:

`buf`	Pointer to the buffer containing the incoming packet
`len`	Length of the packet
`from_radio`	`false` if the incoming interface is the uart, otherwise `true`

Required modules:

IShellInterpreter*	`#define ISENSE_ENABLE_ISHELL_INTERPRETER`
	`#define ISENSE_ENABLE_ISI_SERAERIAL`

Available for the targets:

all

4. DataExchanger API Description

The `DataExchanger` class is defined in src/isense/util/data_exchanger.h in the iSense directory.

The data exchange protocol was designed to collect data from the sensor network easily and independent from the content by iShell via a connected node. The user can trigger a presence detection process and then select which nodes should be asked for data. When the user triggers the collection process (via button click) the connected node asks one selected node after the other (either via broadcast or via tree routing protocol) for data. The `DataExchanger` of each node calls each registered `DataProvider`. It asks first how many entries it provides and then transmits these entries (also via broadcast or tree routing protocol) to the requesting node.

Hence, a `DataExchanger` must be constructed by the application if the device is supposed to send data of `DataProviders` to another (requesting) node.

4.1. Constructor

```
DataExchanger::DataExchanger(Os &os);
```

Description:

Generates an instance of the `DataExchanger` class and initializes it.

Parameters:

`os` Reference to the operating system class `Os`

Required modules:

DataExchanger* `#define ISENSE_ENABLE_DATA_EXCHANGER`

`optional:`

`#define ISENSE_ENABLE_DATA_EXCHANGER_MULTIHOP`

Available for the targets:

all

4.2. add_provider

```
bool DataExchanger::add_provider( isense::DataProvider* d_provider );
```

Description:

Call this method with each `DataProvider` providing data you want to get, e.g. the `isense::NeighborhoodMonitor`.

Parameters:

`d_provider` `DataProvider` which should be registered at the `DataExchanger`.

Required modules:

DataExchanger* `#define ISENSE_ENABLE_DATA_EXCHANGER`

Available for the targets:

all

4.3.flush

```
bool DataExchanger::flush( uint16 dest_addr, bool multihop );
```

Description:

This method is called on reception of a FLUSH_REQUEST packet, but it can also be called by the application for sending the data actively. The DataExchanger asks one DataProvider after the other how many entries it provides and sends them either per broadcast or via the tree routing protocol.

Parameters:

`dest_addr` id of the destination node (sink)

`multihop` determines if the data is send via the tree routing protocol or directly to the sink node

Required modules:

DataExchanger* `#define ISENSE_ENABLE_DATA_EXCHANGER`

if the parameter `multihop` is set to `true`:

`#define ISENSE_ENABLE_DATA_EXCHANGER_MULTIHOP`

Available for the targets:

all

4.4.stop_flushing

```
void DataExchanger::stop_flushing( bool success );
```

Description:

This method is called by the DataExchanger itself with “success = true” when all data of all providers has successfully be sent. It is called with “success = false” if the sending process has been stopped ahead of time, e.g. if the maximum number of tries to send one packet has been reached. The application can also stop the sending process by calling this method.

Parameters:

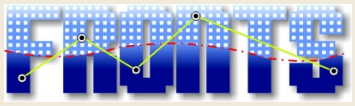
`success` indicates if all data could be sent or if an error occured

Required modules:

DataExchanger* `#define ISENSE_ENABLE_DATA_EXCHANGER`

Available for the targets:

all



5. DataProvider API Description

The `DataExchanger` class is defined in src/isense/data_provider.h in the iSense directory.

If a class implements the interface `DataProvider`, it can be registered at the `DataExchanger` for being called when the `DataExchanger` sends all data to a requesting node.

```
namespace isense
{
	class DataProvider :
		public iSenseObject
	{
		public:
			static char separator_[];

			virtual uint32 dataset_count() = 0;
			virtual bool data_string(uint32 index, uint8* data,
			    uint8 max_str_len, uint8* chars_written,
			    bool* serialized) = 0;

	};

}
```

5.1. separator_

The variable `separator_` is static and public. It is ";" as default, but can be set by the application. All iSense data providers use the variable as separator in order to have a unified sign for all providers.

5.2. dataset_count

```
uint32 DataProvider::dataset_count() = 0;
```

Description:

You must overwrite this method if your class inherits from `DataProvider` and return how many entries your provider provides. For example, let this number of available data sets be 5. Then, the `DataExchanger` will call the data_string method of your `DataProvider` 5 times whereas the index goes from 0 to 4.

Parameters:

return `Number of available data sets`

Required modules:

none

Available for the targets:

all

5.3.data_string

```
bool DataProvider::data_string(uint32 index, uint8* data, uint8
                max_str_len, uint8* chars_written, bool* serialized) = 0;
```

Description:

You must overwrite this method if your class inherits from `DataProvider`. Write the data to be sent into the buffer called `data` and the number of written characters in `chars_written`. Do not write more than `max_str_len` characters into the buffer. The index runs from 0 to `dataset_count()-1` over several calls. When this method returns false, it won't be called by the `DataExchanger` anymore for the actual sending process even if `dataset_count()-1` was not yet reached. Hence, the method should return `true` for normal operation.

Parameters:

`index`	index of the requested data set
`data`	data pointer to the string buffer to be written to
`max_str_len`	maximum number of characters that may be written to the buffer
`chars_written`	number of characters written to data within this method
`serialized`	can be used to adapt the interpretation in iShell (not used by iSense data providers)
return	`true if data was available, false otherwise`

Required modules:

none

Available for the targets:

all

6. References

[1] coalesenses Development Environment Setup User Guide, online available at http://www.coalesenses.com/download/UG_development_environment_setup_v1.9_web.pdf

[2] coalesenses iShell User Guide, online available at http://www.coalesenses.com/download/UG_ishell_v1.3.pdf

[3] coalesenses Writing iSense Applications User Guide, online available at http://www.coalesenses.com/download/UG_writing_isense_applications_1v1.pdf

[4] UZL Testbed Documentation at http://fronts.cti.gr/index.php/testbed