



## *iSense Core Module 3 User Guide*

## *Document history*

<i>Version</i>	<i>Date</i>	<i>Changes</i>
1.0	26.08.2011	Initial version

## Contents

1.	About this User Guide .....	4
2.	General Description .....	5
2.1.	Handling and Security Instructions .....	5
2.2.	Using the switch.....	7
2.3.	Operating the Voltage Regulator .....	7
2.4.	Identifying the chip version of the Core Module.....	<b>Fehler! Textmarke nicht definiert.</b>
3.	CoreModule API Description .....	8
3.1.	Constructor.....	8
3.2.	led_on.....	9
3.3.	led_off.....	9
3.4.	regulator_on .....	10
3.5.	regulator_off .....	10
3.6.	set_regulator_mode.....	11
3.7.	supply_voltage .....	11
4.	Core Module Demo Application .....	12
4.1.	Obtaining the Core Module Demo Application.....	12
4.2.	Compiling the Core Module Demo Application.....	12
4.2.1.	Using Eclipse .....	12
4.2.2.	Using the Command Line .....	15
4.3.	Flashing the Core Module Demo Application .....	16
4.4.	Core Module Demo Application functionality .....	16
5.	References .....	26

## *1. About this User Guide*

In this user guide,

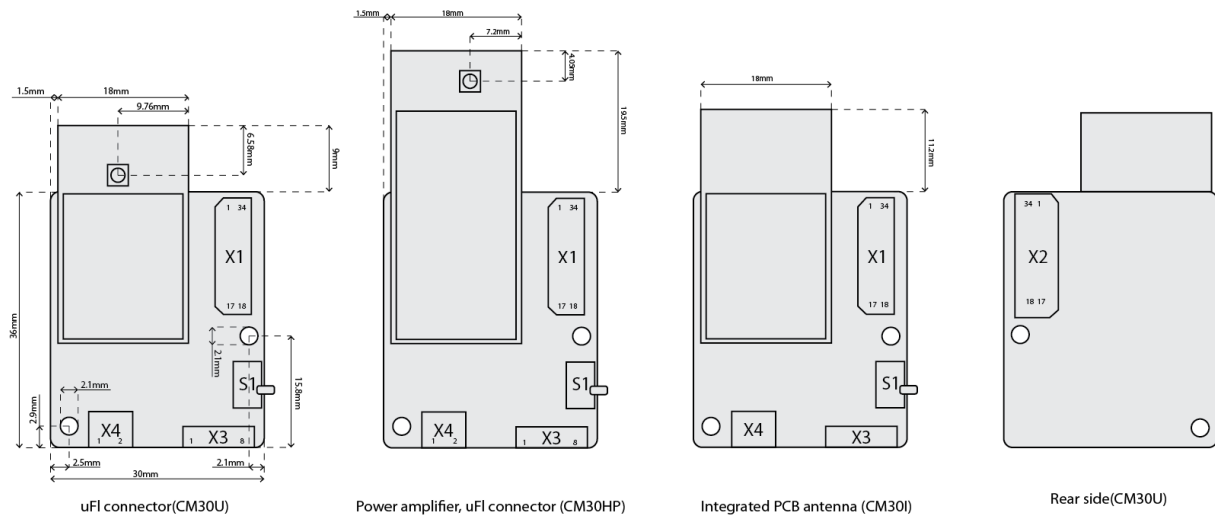
- files and folders are represented in the `Arial` typeface,
- code fragments, function names etc. are represented in the `Courier New` typeface,
- GUI elements such as button descriptions etc. are represented in “quotation marks”,
- titles of other documents are presented in *Italic* type.

This manual assumes that the reader has successfully installed the iSense development environment, and obtained the iSense standard firmware. For further information on 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. General Description



**Figure 1: Mechanical information.**

The iSense Core Module is based on a Jennic JN5148 wireless microcontroller [4], a chip that combines the controller and the wireless communication transceiver in a single housing.

The controller provides 32 bit RISC computation and runs at a software-scalable frequency between 4 and 32 MHz. It comprises 128kbytes of memory that are shared by program code and data. The advantage of this choice is that memory consumption of program code and data can be traded. Opposite to other controllers where the user is limited to a certain amount of data and code memory, free choices that are only bounded by the sum of both become possible here.

The radio part complies with the IEEE 802.15.4 standard [5]. It achieves a data rate of 250kBit/s, provides hardware AES encryption and is ZigBee-ready. As the world's first IEEE 802.15.4 radio, it supports distance measurements to neighboring devices using time of flight ranging. Besides IEEE 802.15.4 standard compliant operation, the radio transceiver provides two additional modes of operation, offering increased data rates of 500kBit/s and 667kBit/s.

Apart from the CM30U version that is equipped with a  $\mu$ Fl antenna connector, the CM20I with an integrated PCB antenna for especially compact systems is available. Both provide a receive sensitivity of -95dBm (at 250kBit/s) and a transmit power tunable between -60dBm and +2.5dBm.

In addition, a Core Module version (CM30HP) with a power amplification stage for transmitting and receiving is available. It is equipped with a  $\mu$ Fl antenna connector, reaches a receive sensitivity of -98dBm (at 250kBit/s) and a transmit power of up to 10dBm.

A common quandary in design is whether or not to use a voltage regulator. It has the advantage that operation with voltages lower than the required one is possible, but the regulator inherently wastes energy. This is especially bad as it also wastes current if the voltage would be high enough and the regulator would not be required. To resolve this problem, we decided to combine the measurement of the supply voltage with the possibility to bypass the regulator by a software switch. Like this, the regulator usage can be omitted when not required but is available when the supply voltage drops.

To enable long, but still synchronous sleep and wakeup cycles, the module is equipped with a high precision clock (error < 30ppm). The Core Module also features a software-switchable LED for debugging purposes.

There is a 34 pin connector (X1, X2) on both sides of the module where other modules can be attached to the Core Module. It can supply up to 500mA to other modules.

The controller can be programmed in various ways. While over-the-air programming (OTAP) is possible and considered to be the standard procedure, the program can also be transferred via the gateway module, or using a special programming adapter that mates with corresponding pads on the module.

The module can be powered by a wall mount adapter or a standard battery holder, by one of the power modules or via the USB interface of the gateway module.

### *2.1. Handling and Security Instructions*

Note that all iSense Modules are sensitive to electro-static discharge. Appropriate protection measures have to be taken.

If not protected by an appropriate housing, all iSense components must be protected from humidity, mechanical impact, and short circuiting by contact with conductive materials.

Note that all cable plugs and headers are designed to be extremely compact, and are not intended for frequent plugging and unplugging. Never pull the cables to remove the plugs from the corresponding modules, always pull the plug itself. If required, use an appropriate tool (or the pulling cord if available). Even though all plugs/headers are coded, be sure to pay attention to the proper orientation. Never apply force. Never use cables if their insulation is damaged.

Note that iSense modules are not intended for hot-plugging, (dis-)connecting modules from or to other modules in operation can result in a reset of the Core Module and other undesired effects.

All delivered components are intended for use in research application. For using these components within other application domains, consult coalesenses prior to use.

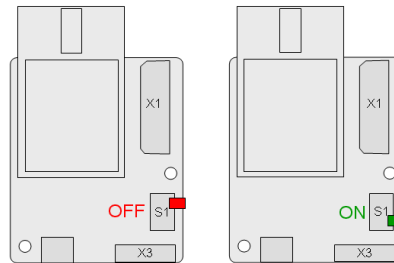
Coalesenses products are not intended for use in life support systems, appliances or systems where malfunction of these products can reasonably be expected to result in personal injury, death or severe property or environmental damage. Coalesenses customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify coalesenses for any damages resulting from such use.

Do not connect components or devices to any iSense component that are not manufactured or distributed by coalesenses without prior consultation of coalesenses. Note that doing so regardless of consultation will void the warranty and the declaration of CE conformity. Coalesenses customers connecting third party devices do so at their own risk and agree to fully indemnify coalesenses for any damages or injury resulting from such use.

Note that the SMA connector is sensitive to mechanical stress. Be sure not to apply forces to the connector or a connected antenna. Protect the module from shock and vibration.

If a wall mount adapter is connected to the Core Module, or a voltage is applied to pins 18 and/or 19 (V+\_USB) of the system connector X1/X2 (e.g. because an iSense Gateway Module that is connected to an active USB port is attached to the Core Module), currents may flow back into devices or components connected to X3, X4 or the programming pads.

### 2.2. Using the switch



The Core Module is switched off if the switch is oriented towards the 34 pin connector X1 and switched on if the switch is oriented towards the 8 pin connector X3.

Note that S1 switches the power supply of the Core Module and all connected modules (except modules that have their own power supply, such as the iSense Gateway Module, the power modules and the Solar Power Management Module), regardless of how the Core Module is powered. This in particular means that the Core Module must be switched on to program the module via the iSense Gateway Module.

### 2.3. Operating the Voltage Regulator

The Core Module voltage regulator can be switched off and bypassed via an API call (see Section 3.5). However, great care must be taken if switching off the regulator. By default, the regulator is switched on.

#### *Warning!*

To prevent damage or malfunction, be sure that the following conditions are met before switching off the regulator:

1. The voltage applied to the Core Module must not exceed or under-run the input voltage requirements of all connected modules. To ensure this, the Core Module can measure its supply voltage (see Section 3.7).

The regulator supports two modes of operation that can be set via an API call (see Section 3.6). The power save mode works particularly efficient if the current drawn by the sensor node is small, but induces a slightly higher supply voltage ripple. The continuous mode is less efficient for small currents, but guarantees a smoother supply voltage.

As a rule of thumb, the power save mode is more efficient when the Core Module is sleeping, or when the controller is running or dozing with the radio switched off, while the continuous is more efficient if the radio is switched on. In addition, the continuous mode should be used when the analogue peripherals are in use, as the reduced ripple increases the sample accuracy.

### 3. CoreModule API Description

The `CoreModule` class in the `isense` namespace contains all software functionality for operating the Core Module-specific features, including

- switching the LED,
- managing the voltage regulator,
- measuring the regulator input voltage.

The `CoreModule` class is defined in `src/isense/modules/core_module/core_module.h` in the `iSense` directory.

The iSense firmware is structured into a large number of software modules that can be (de-)activated separately. The below API description for each function specifies the required modules that must be activated to use the functions, and names both the web compilation module name as well as the according code define. An asterisk (\*) indicates that the respective module is activated in the iSense standard firmware.

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

#### 3.1. Constructor

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

##### Description:

Initializes the `CoreModule` class. By default, the regulator is enabled and working in the power save mode, the LED is switched off.

##### Parameters:

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

##### Required modules:

Core Module\*            `#define ISENSE_ENABLE_CORE_MODULE`

##### Available for the targets:

JN5139R, JN5139R1, JN5148



### *3.2.led\_on*

```
void CoreModule::led_on();
```

*Description:*

Switches on the Core Module LED (regardless of prior state).

*Parameters:*

none

*Required modules:*

Core Module\*                    #define ISENSE\_ENABLE\_CORE\_MODULE

*Available for the targets:*

JN5139R, JN5139R1, JN5148

### *3.3.led\_off*

```
void CoreModule::led_off();
```

*Description:*

Switches off the Core Module LED (regardless of prior state).

*Parameters:*

none

*Required modules:*

Core Module\*                    #define ISENSE\_ENABLE\_CORE\_MODULE

*Available for the targets:*

JN5139R, JN5139R1, JN5148

### 3.4.regulator\_on

```
void CoreModule::regulator_on();
```

#### Description:

Switches on the voltage regulator (regardless of prior state).

#### Parameters:

none

#### Required modules:

Core Module*	#define ISENSE_ENABLE_CORE_MODULE
Regulator management*	#define ISENSE_ENABLE_CORE_MODULE_REGULATOR_MANAGEMENT

#### Available for the targets:

JN5139R, JN5139R1, JN5148

### 3.5.regulator\_off

```
void CoreModule::regulator_off();
```

#### Description:

Switches off the voltage regulator (regardless of prior state).

#### Warning:

Switching off the regulator must be done with great care. Please consider Section 2.3 prior to use.

#### Parameters:

none

#### Required modules:

Core Module*	#define ISENSE_ENABLE_CORE_MODULE
Regulator management*	#define ISENSE_ENABLE_CORE_MODULE_REGULATOR_MANAGEMENT

#### Available for the targets:

JN5139R, JN5139R1, JN5148

### 3.6.set\_regulator\_mode

```
void CoreModule::set_regulator_mode(regulator_mode mode);
```

#### Description:

Sets the voltage regulator mode

#### Parameters:

mode    CoreModule::continuous\_mode    Sets the regulator to the continuous mode  
CoreModule::power\_save\_mode    Sets the regulator to the power save mode

#### Required modules:

Core Module*	#define ISENSE_ENABLE_CORE_MODULE
Regulator management*	#define ISENSE_ENABLE_CORE_MODULE_REGULATOR_MANAGEMENT

#### Available for the targets:

JN5139R, JN5139R1, JN5148

### 3.7.supply\_voltage

```
uint16 CoreModule::supply_voltage();
```

#### Description:

Measures the voltage that is fed into the voltage regulator. Note that if the regulator is disabled, the same voltage is put through to the controller.

This measurement can be used to decide whether the voltage regulator can be switched off safely. However, keep in mind that the battery voltage might rise or fall over time "without prior notice".

This operation takes more than a millisecond, and should hence not be called from the interrupt context.

#### Returns:

Voltage in millivolt

#### Required modules:

Core Module*	#define ISENSE_ENABLE_CORE_MODULE
Pre-regulator voltage measurement*	#define ISENSE_ENABLE_CORE_MODULE_VOLTAGE_MEASUREMENT

#### Available for the targets:

JN5139R, JN5139R1, JN5148

## *4. Core Module Demo Application*

The Core Module demo application exemplifies how an application can make use of the CoreModule API. It shows how to

- switch on and off the LED,
- set the voltage regulator operation mode,
- measure the regulator input voltage.

Note that the demo application does not make use of the voltage regulator bypass feature, as this is currently only possible in conjunction with the iSense 1/2AA Battery Module.

### *4.1. Obtaining the Core Module Demo Application*

Go to coalesenses web site, click on “Downloads”, and then on “iSense Hardware”. In the “Core Module” section, click on the icon at the right in the “Core Module Demo Application” row, and download CoreModuleDemoApplication.zip.

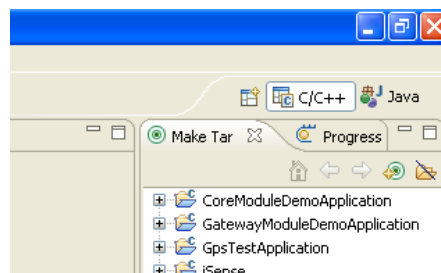
Extract the content of CoreModuleDemoApplication.zip to the iApps directory.

### *4.2. Compiling the Core Module Demo Application*

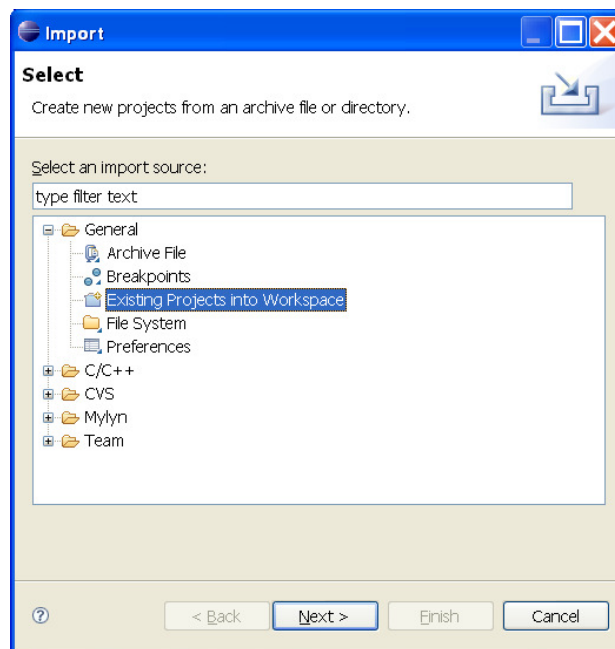
The next step is compiling the Core Module Demo Application.

#### *4.2.1. Using Eclipse*

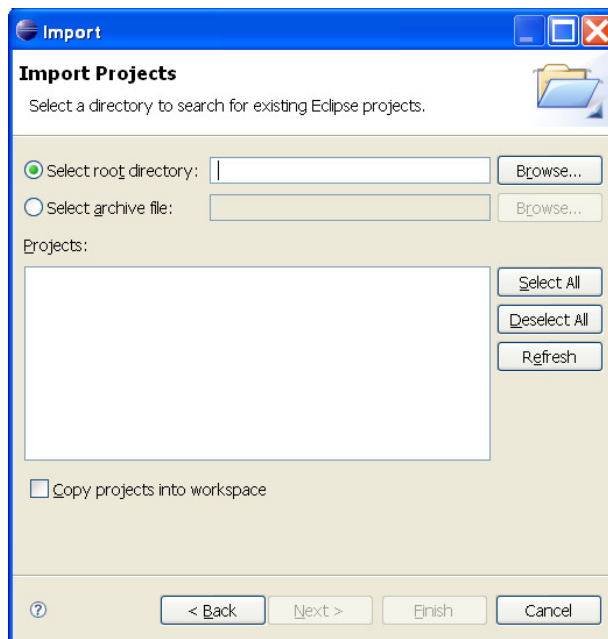
If you want to use Eclipse for compiling the Core Module Demo Application, open Eclipse, and change to the “C++” perspective.



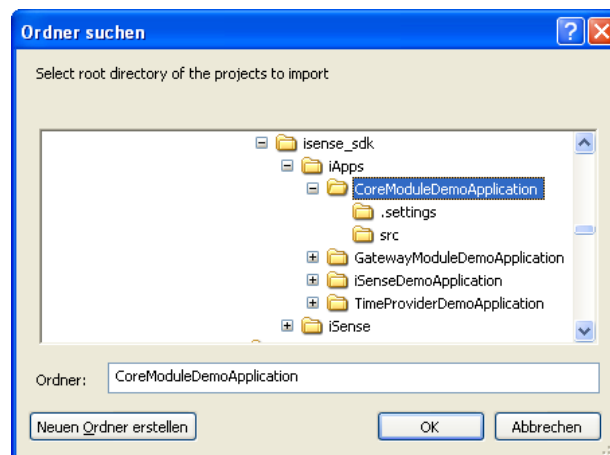
You can now import the application into Eclipse. Choose “File” → “Import” from the menu bar to open the “Import” dialog.



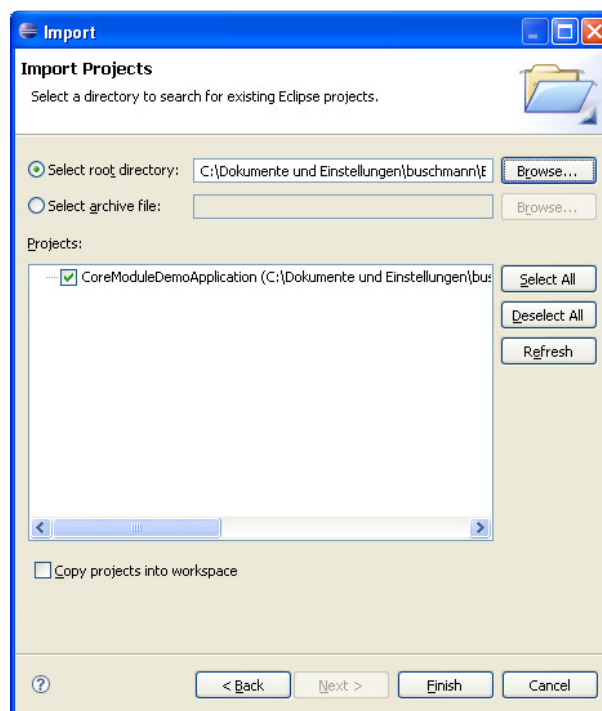
Choose “Existing projects into workspace”, and click on “Next”.



Click on the “Browse...”, and select the directory of the Core Module demo application, i.e. CoreModuleDemoApplication in the iApps directory.



After doing so, the “CoreModuleDemoApplication” project should appear in the projects list of the “Import” dialog.

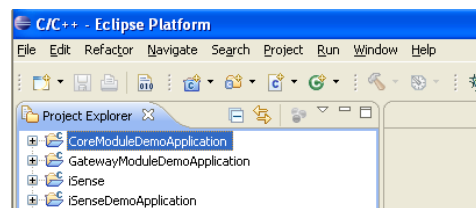


If it doesn't, the typical reason is that there is already a project called “CoreModuleDemoApplication” in Eclipse, i.e.

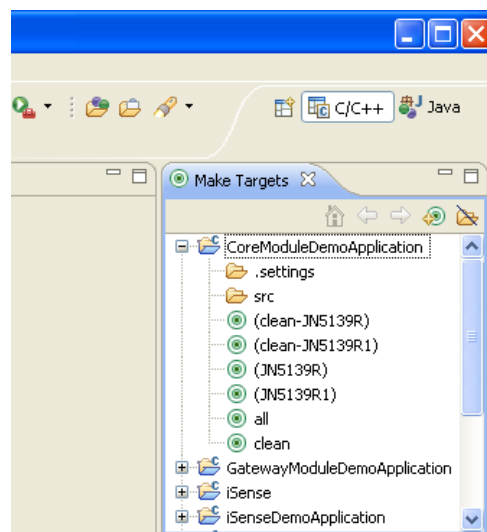
- you imported the CoreModuleDemoApplication before or
- you created or imported a project with the same name before.

Be sure not to check “Copy projects into workspace” in the above dialog. Finish the import by clicking on the “Finish” button.

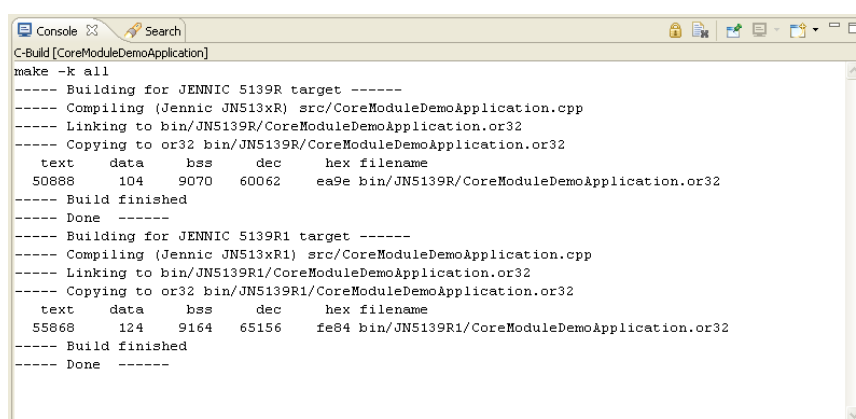
As a result, you will find the “CoreModuleDemoApplication” project in the “Project Explorer”.



In addition, the “CoreModuleDemoApplication” will appear in the “Make Targets” view. Double click on “all” to build the demo application for all targets.

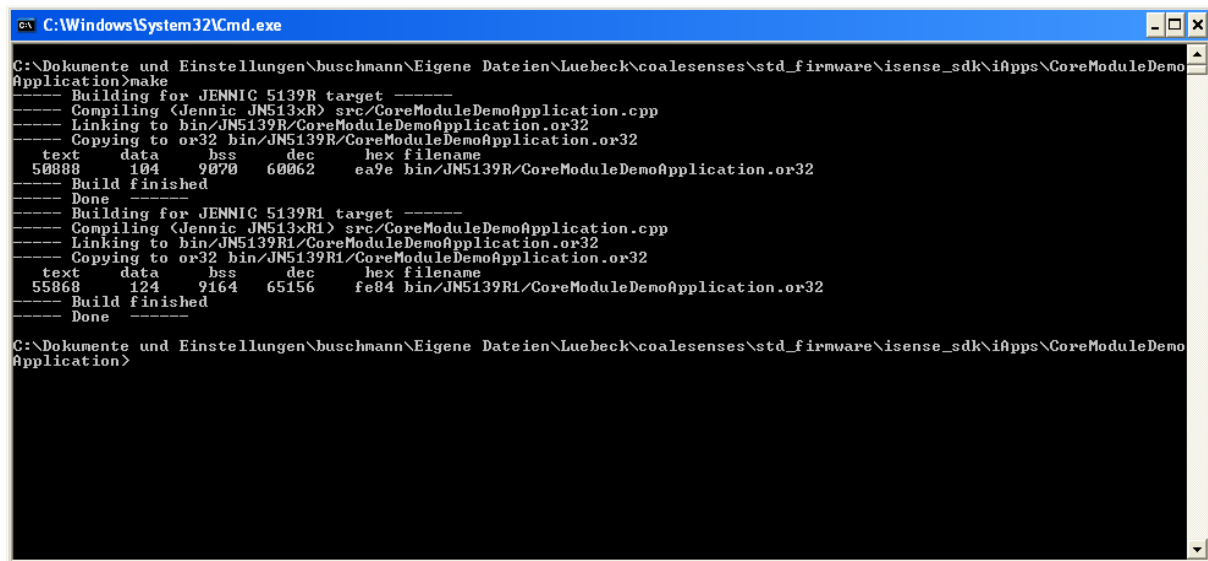


As a result, the “Console” view displays the compiler output during the build process. After that, it should look similar to the output depicted below.



### 4.2.2. Using the Command Line

If you want to use the command line tools for building the Core Module Demo Application, open a console window, change to the CoreModuleDemoApplication directory in your iApps directory, and type “make”.



```
C:\Windows\System32\Cmd.exe
C:\Dokumente und Einstellungen\buschmann\Eigene Dateien\Luebeck\coalesenses\std_firmware\isense_sdk\iApps\CoreModuleDemo
Application>make
----- Building for JENNIC 5139R target -----
----- Compiling (Jennic JN513xR) src/CoreModuleDemoApplication.cpp
----- Linking to bin/JN5139R/CoreModuleDemoApplication.or32
----- Copying to or32 bin/JN5139R/CoreModuleDemoApplication.or32
text    data    bss    dec    hex filename
50888    104     9070    60062    ea9e bin/JN5139R/CoreModuleDemoApplication.or32
----- Build finished
----- Done
----- Building for JENNIC 5139R1 target -----
----- Compiling (Jennic JN513xR1) src/CoreModuleDemoApplication.cpp
----- Linking to bin/JN5139R1/CoreModuleDemoApplication.or32
----- Copying to or32 bin/JN5139R1/CoreModuleDemoApplication.or32
text    data    bss    dec    hex filename
55868    124     9164    65156    fe84 bin/JN5139R1/CoreModuleDemoApplication.or32
----- Build finished
----- Done
C:\Dokumente und Einstellungen\buschmann\Eigene Dateien\Luebeck\coalesenses\std_firmware\isense_sdk\iApps\CoreModuleDemo
Application>
```

After the build finished, the console output should look similar as shown above.

### 4.3. Flashing the Core Module Demo Application

Connect

- an iSense Core Module and
- an iSense Gateway Module with a USB cable

and plug the cable to the PC. Be sure that the Core Module is switched on.

Start iShell, configure the correct communication port, and change to the “Flash Loader” view. Select the appropriate binary according to the table below.

Chip	Binary file
------	-------------

JN5148	iApps/CoreModuleDemoApplication/bin/JN5148/CoreModuleDemoApplication.bin
--------	--

For details regarding the identification of the chip version, refer to Section **Fehler! Verweisquelle konnte nicht gefunden werden..** Then flash the binary to the connected iSense sensor node.

### 4.4. Core Module Demo Application functionality

The source file of the Core Module demo application is located at `CoreModuleDemoApplication/src/CoreModuleDemoApplication.cpp`. The class `CoreModuleApplication` is derived from `isense::Application`, and additionally inherits from `isense::Task` and `isense::SleepHandler`.

```
class CoreModuleDemoApplication :
public Application,
public Task,
```



```
    public SleepHandler
{
public:
    CoreModuleDemoApplication(Os &os);
    // inherited from application, called upon device boot
    void boot();
    // inherited from SleepHandler, called upon wake->sleep transition
    bool stand_by (void);
    // inherited from SleepHandler, called upon sleep->wake transition
    void wake_up (bool memory_held);
    // inherited from Task, called when registered task is due
    void execute( void* userdata );
private:
    //Pointer to CoreModule instance
    CoreModule* cm_;
};
```

CoreModuleApplication overwrites `isense::Application::boot()`, which is called upon the start up of the iSense sensor node.

```
void
CoreModuleDemoApplication::
boot ()
{
    os().debug("Booting Core Module Demo Application, id=%x", os().id());
    //create CoreModule instance
    cm_ = new CoreModule(os());
    if (cm_ != NULL) // new was successful
    {
        //switch on Core Module LED to show device is awake
        cm_>led_on();
        //set regulator to continuous mode
        cm_>set_regulator_mode(CoreModule::continuous_mode);
        // register application as sleep handler
        // --> wake_up and stand_by call upon sleep state changes
        os().add_sleep_handler(this);

        :
    }
}
```

Within the `boot()` method, a `CoreModule` object is created. If the allocation was successful, the Core Module LED is switched on, and the regulator is set into the continuous mode. Then, the application is registered at the `Os` as a `SleepHandler`. This is possible because the application implements the two pure virtual methods from `isense::SleepHandler`.

```
class SleepHandler :
    public iSenseObject
{
public:
```

```
virtual ~SleepHandler() {}

/** This method is called before the device goes to sleep.
 * In contrast to hibernate, it indicates the that memory
 * will be retained during sleep. Derived classes can do
 * here whatever is required before sleep. But: BE QUICK.
 *
 * Note, that tasks and timeouts registered here are NOT
 * called before sleep.
 *
 * \return true if application agrees to going to sleep
 * mode, false otherwise
 */
virtual bool stand_by (void) = 0;

/** This method is called by the os after wake up from sleep.
 * Derived classes can use it e.g. deserialize themselves from
 * non-volatile memory in case system memory was not retained
 * during sleep. However, note that this currently never happens.
 *
 * It is necessary that your application does not take longer
 * than 1ms/SLEEP_HANDLER_MAX_COUNT.
 *
 * \param memory_held Indicates whether the memory was retained
 * during sleep.
 */
virtual void wake_up (bool memory_held) = 0;

};
```

**void** wake\_up (bool memory\_held) is called upon wake up of the device from sleep mode, while **bool** stand\_by (**void**) is called when the sensor nodes goes into the sleep mode. Classes that implement this interface can overwrite these methods to take actions after wake up and before sleeping of the sensor node.

In these methods, the CoreModuleDemoApplication toggles the LED and the voltage regulator operation mode. It switches on the LED on wake up and set the regulator into the continuous mode, and switches off the LED on sleep and sets the regulator into the power save mode.

```
//-----  
void  
CoreModuleDemoApplication::  
wake_up (bool memory_held)  
{  
    //switch on Core Module LED to show device is awake  
    cm_>led_on();  
    //set regulator to continuous mode  
    cm_>set_regulator_mode(CoreModule::continous_mode);  
}  
  
//-----  
bool  
CoreModuleDemoApplication::  
stand_by () // Memory held  
{  
    //switch off Core Module LED to show device is sleeping  
    cm_>led_off();  
    //set regulator to power save mode  
    cm_>set_regulator_mode(CoreModule::power_save_mode);  
    return true;  
}
```

CoreModuleDemoApplication::boot() then prevents the sensor node from going to sleep immediately after booting by calling `os().allow_sleep(false);`

```
void  
CoreModuleDemoApplication::  
boot ()  
{  
    :  
    // device should stay awake for the first five seconds  
    // forbid sleep now  
    os().allow_sleep(false);  
    :  
}
```

The iSense software system features an automatic power management system. It manages the different power modes to achieve minimum energy consumption.

Although the system works automatically in the background, the different software components must indicate to the OS which power modes they can tolerate for the current operation.

Three power modes are distinguished:

- full operation: all hardware features are fully functional
- doze mode: all peripherals and interrupt sources are functional, radio packets can be received and sent (if the radio is not turned off manually)

- sleep mode: radio and controller are switched off, but IO interrupt sources, tasks and timeouts are still functional.

The OS holds a counter for sleep and doze mode. If that counters are 0, sleeping or dozing are allowed. Each software component can disallow entering sleep and doze mode by calling the according OS methods that increment or decrement the according counters.

```
//-----  
/** This method increments or decrements a counter for the permission  
 * of automatic sleep. If the counter is zero automatic sleep is allowed.  
 *  
 * \return Returns true if the operation has been successful, false  
 * if the variable had already the desired value.  
 */  
bool allow_sleep(bool allow);  
  
//-----  
/** This method increments or decrements a counter for the permission of  
 * automatic doze. If the counter is zero automatic doze is allowed.  
 *  
 * \return Returns true if the operation has been successful, false if the  
 * variable had already the desired value.  
 */  
bool allow_doze(bool allow);
```

After boot, sleeping and dozing are allowed by default. If the application wants to prevent sleeping, it must call

```
os().allow_sleep(false);
```

for example in its `boot` method. This prevents the system from sleeping until

```
os().allow_sleep(true);
```

is called again. The same holds for `allow_doze`.

For the application, sleeping and waking is transparent except for the fact that no radio packets can be received when sleeping. The application can operate as usual, as the system automatically wakes up in time when timeouts or tasks are due.

Note that allowing sleeping or dozing does not necessarily mean that the device actually goes to sleep or doze mode, as other software components can also prevent sleeping or dozing if their current operation cannot tolerate sleeping or dozing. An example for that is the TDMA radio that prevents sleeping during the active sending and receiving times (see doze documentation for the TDMA radio).

To be able to sleep later on, the demo application finally registers a task in the `CoreModuleDemoApplication::boot()` method to be called after 5 seconds with the userdata constant `TASK_SLEEP`, casted to a `void*` pointer to match the API signature of `Os::add_task_in` (see below).

```
void
CoreModuleDemoApplication::
boot ()
{
    :

    // register task in 5 seconds, to allow sleeping then
    os().add_task_in(Time(5000), this, (void*)TASK_SLEEP);
} else
    os().fatal("Could not allocate memory for CoreModule");
}
```

An application can register itself at the OS as a Task to receive a callback immediately, at a certain time, or after a certain interval. Task callbacks are called in the regular application context. They cannot interrupt other activities, and hence may be delayed. Task execution can take an arbitrary time, and is hence suited for longer activities such as complex computations.

```
class Task : public iSenseObject
{
public:
    /**
     * This method is called by the os in normal context.
     */
    virtual void execute( void* userdata ) = 0;
};
```

Once a class (such as the demo application) inherits the `isense::Task` interface and implements the corresponding method, it is a task and can be registered at the operating system for later callback.

For tasks, the OS provides three methods for registering tasks to be executed as soon as possible, at a certain time, or after a certain interval from now on.

```

//-----
/** This method enqueues the given task into the queue of tasks
 * Its execute method is called during the main control flow
 *
 * \param task object implementing the Task interface, whose execute
 * method is called
 * \param userdata an arbitrary pointer that is passed back again
 * to the callback
 *
 * \return true if task was enqueued successfully, false otherwise (e.g.
 * if the task queue is full)
 */
bool add_task(Task *task, void* userdata);

//-----
/** This method enqueues the given task in the queue of lower priority
 * tasks. Its execute method is called during the main control flow
 *
 * \param interval Point in time when the callback will be called
 * \param task object implementing the Task interface, whose execute
 * method is called
 * \param userdata an arbitrary pointer that is passed back again
 * to the callback
 *
 * \return true if task was enqueued successfully, false otherwise (e.g.
 * if the task queue is full)
 */
bool add_task_at(Time time, Task *task, void* userdata);

//-----
/** This method enqueues the given task in the queue of lower priority
 * tasks. Its execute method is called during the main control flow
 *
 * \param interval Time period after which the callback will be called
 * \param task object implementing the Task interface, whose execute
 * method is called
 * \param userdata an arbitrary pointer that is passed back again
 * to the callback
 *
 * \return true if task was enqueued successfully, false otherwise (e.g.
 * if the task queue is full)
 */
bool add_task_in(Time interval, Task *task, void* userdata);

```

All methods for registering tasks at the OS feature a so called userdata parameter. It can be used to attach additional information to a task, that is then passed to the execute method when it is called. If you don't use the feature, simply pass NULL.

As a result of the task registration in the `CoreModuleDemoApplication::boot()` method, `CoreModuleDemoApplication::execute(...)` is called by the operating system five seconds after the sensor node booted. As `Os::add_task_in` was called with the userdata constant `TASK_SLEEP`, the second if condition applies for the first call of execute.

```

void
CoreModuleDemoApplication::
execute(void *userdata )
{
    if ((uint32)userdata == TASK_WAKE)
    {
        os().debug("Staying awake");
        //forbid sleeping
        os().allow_sleep(false);
        // register Task to allow sleeping again in 5 seconds
        os().add_task_in(Time(5000), this, (void*)TASK_SLEEP);
        //measure and output Core Module supply voltage
        uint16 voltage = cm_>supply_voltage();
        os().debug("Core Module supply voltage is %d", voltage);
    }
    else if ((uint32)userdata == TASK_SLEEP)
    {
        //allow sleeping
        os().allow_sleep(true);
        os().debug("Allowing Sleep");
        // register task in order to wake up and then forbid sleeping
        //again
        os().add_task_in(Time(5000), this, (void*)TASK_WAKE);
    }
}

```

Hence, a call of `Os::allow_sleep(true)` allows sleeping again (was forbidden in boot), additionally the application is registered again as a task at the os, this time using the constant `TASK_WAKE`. After leaving the method `CoreModuleDemoApplication::execute(...)`, the sensor node will enter the sleep mode. Before going to sleep, the `CoreModuleDemoApplication::stand_by()` method is called. After another 5 seconds, the sensor node will wake up again, just in time for another call of `execute`, will then enter the first `if` clause, and forbid sleeping again. Additionally, the application is registered again as a task (now again with the constant `TASK_SLEEP`). It also measures the Core Module supply voltage, and outputs the result using `os().debug("Core Module supply voltage is %d", voltage);`

You can send character output to the outside world using the methods `Os::debug` and `Os::fatal`:

```

//-----
** Logs the given string depending on the log mode to a uart or radio.
*/
void debug( const char *format, ...);

//-----
** Logs the given string depending on the log mode to a uart or radio.
*/
void fatal( const char *format, ...);

```

They work similar to the well-known `sprintf` functions in regular C, i.e. integer values etc. can be printed using the `%` notation. For example

```
uint16 i = 128;
os().debug("value of i is %d, hex=%x", i, i);
```

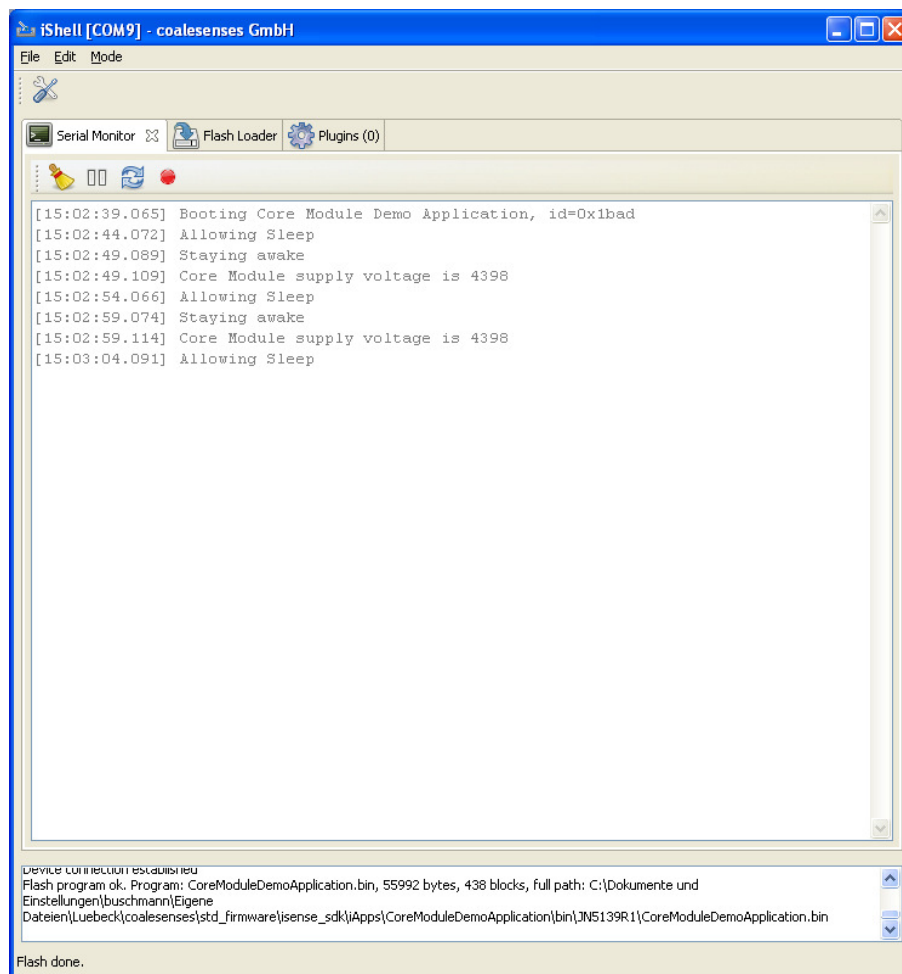
will output „value of i is 128, hex=0x80“. The destination of the debug output can be set using `Os::set_log_mode`, the corresponding constants can be found in `iSense/src/isense/os.h`.

```
//-----
/** Sets the log mode to the given value, e.g.
 * (ISENSE_LOG_MODE_UART0 | ISENSE_LOG_MODE_RADIO)
 */
void set_log_mode( uint8 mode ) {log_mode_ = mode; }
```

By default, the output destination is set to UART0, i.e. the output is sent to a PC via a connected iSense Gateway Module.

All in all, the `CoreModuleDemoApplication` makes the sensor node sleep and wake in a 5 second interval. The Core Module LED is on during the wake periods and the regulator is working in the continuous mode, whereas the LED is off and the regulator is in the power save mode while the sensor node is sleeping. At the beginning of each wake period, the Core Module measures its supply voltage and outputs the result.





Observing the output of the `CoreModuleDemoApplication` with iShell yields the above result.

## *5. References*

- [1] coalesenses Development Environment Setup User Guide, online available at [http://www.coalesenses.com/download/UG\\_development\\_environment\\_setup\\_v1.9\\_web.pdf](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](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\\_v1.pdf](http://www.coalesenses.com/download/UG_writing_isense_applications_v1.pdf)
- [4] Advanced Data Sheet – JN513x, online available at [http://www.jennic.com/support/view\\_file.php?fileID=0000000111](http://www.jennic.com/support/view_file.php?fileID=0000000111)
- [5] IEEE Computer Society, IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), <http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf>

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

[www.coalesenses.com](http://www.coalesenses.com)  
[sales@coalesenses.com](mailto:sales@coalesenses.com)