ēKo PRO Series

ESB Developer's guide

Rev. 6, August 2009

NEXT GENERATION WIRELESS FOR PRECISION AGRICULTURE • by CROSSBOW TECHNOLOGY, INC.

www.xbow.com/eko

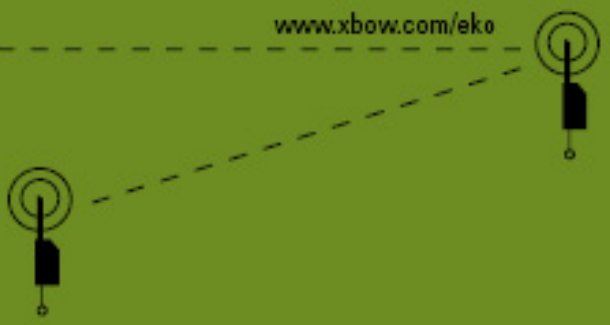


Table of Contents

About This Document

The following annotations have been used to provide additional information.

◂ NOTE

Note provides additional information about the topic.

☑ EXAMPLE

Examples are given throughout the manual to help the reader understand the terminology.

IMPORTANT

This symbol defines items that have significant meaning to the user

⚠ WARNING

The user should pay particular attention to this symbol. It means there is a chance that physical harm could happen to either the person or the equipment.

The following paragraph heading formatting is used in this manual:

1 Heading 1

1.1 Heading 2

1.1.1 Heading 3

This document also uses different body text fonts (listed in Table 0-1) to help you distinguish between names of files, commands to be typed, and output coming from the computer.

Table 0-1. Font types used in this document.

Font Type	Usage
`Courier New Normal`	Sample code and screen output
`Courier New Bold`	Commands to be typed by the user
Times New Roman Italic	Files names, directory names
Franklin Medium Condensed	Text labels in GUIs

1 Introduction

This manual describes how to interface sensors to the ēKo eN2100 node.

1.1 ēKo Sensors Types

The ēKo eN2100 node is designed to interface to many different types of sensors. Low power sensors that can be powered from battery voltage (typically 3.6V) and output signals of 3 volts can interface directly to the node. Other sensors that require more signal conditioning, power conditioning, or use digital communications require an external interface module or circuit.

There are three types of sensors that can be interfaced to the eN2100:

Type 1: Analog sensors that can be connected directly to the eN2100 node without any additional signal or power conditioning. The eN2100 can interface directly to:

- Two wire resistive sensors that need 10K ohm completion resistors such as thermistors or watermark soil moisture sensors.
- Three wire sensors (power, ground, and signal out) that are excited from 3 to 4 volts and output signals up to 3 volts. The eKo node can supply power from a regulated GPIO line (8 ma max) or directly from batteries (3.6V to 4.2V), 100mA max.

Type 1 sensors only need to be wired to a Switchcraft EN3C6F connector with a programmed Dallas EEPROM for self-identification. Crossbow's eS9000 consisting of the connector, EEPROM and cable can be used for this.

Type 2: Analog sensors requiring additional signal conditioning (amplification, level shifting, etc.) and/or power conditioning. Crossbow's eS9100 analog board interfaces between the eN2100 and the sensor; also the Dallas self-identification EEPROM is embedded in the Switchcraft EN3C6F connector.

Type 3: Digital sensors requiring serial or other communication protocols. These sensors use an external interface circuit (Crossbow eS9200 or other) between the eN2100 and the sensor. They do not require the Dallas EEPROM embedded in the cable as the self-identification information is contained in the microprocessor on the eS9200.

The table below summarizes the basic requirements of each ēKo sensor type:

Sensor Type	Needs EEPROM?	Needs analog signal conditioning module?	Needs digital interface?	eKo Interface
Type 1	√			eS9000
Type 2	√	√		eS9100
Type 3			√	eS9200

1.2 ēKo Power Requirements for Sensors

ēKo is a low power wireless networking system and users need to be aware of the power limitations when interfacing new sensors:

1. The eN2100 node can supply two different voltage levels as determined by the self-identification information in the embedded EEPROM:
    a. Regulated 3 V derived from a precision LDO and powered by the eN2100's NiMH batteries. The maximum current that can be supplied to any one of the four eN2100 ports is 8 mA at a given time. Software within the eN2100 applies power to one port at a time.
    b. Unregulated battery voltage outputted through a multiplexer circuit to the eN2100 port. The open circuit voltage is typically between 3.6 V and 4.2 V when using the standard solar version of the eN2100. The multiplexer circuit has on-resistance of 4 ohms plus the internal resistance of the batteries.
2. The average current consumption of an ēKo node is about 400uA not including the sensor current. Most sensors will have very little impact on this when sampling at a 15 minute interval. In general, on average, they will not contribute more than 10s of µAs. Crossbow does not recommend exceeding 900uA on average for an eN2100 node including the 400µA of the node. This includes all of the average currents of the attached sensors. If higher average currents are needed the eN2100 can run with a D cell (3) alkaline battery pack or an external DC voltage. An Excel spreadsheet is posted on the Crossbow website SensorCurrentCalculator.xls to compute the average current contribution of a sensor.

2 Type 1 Analog Sensors using eS9000

Type 1 sensor assemblies consist of the following:

1. One or two compatible sensors wired to a Switchcraft EN3C6F connector. The connector plugs into one of the eN2100 ports.
2. A Maxim DS2431 EEPROM programmed with the ēKo self-identification information and embedded in the EN3C6F connector. When the connector is plugged into an eN2100 port the eN2100 will read the EEPROM when the unit is turned on or reset.

Crossbow's eS9000 is available as a complete assembly consisting of a Switchcraft connector, embedded EEPROM and five feet of cable (shown in the picture below). The EEPROM is preprogrammed as a generic voltage sensor and can be reprogrammed for other sensors.

If you want to build your own cable assembly, the EEPROMs and Switchcraft connectors can be ordered from Digikey (www.digikey.com).

Component	Digikey Part Number
DS2431 EEPROM	DS2431+-ND
Switchcraft EN3C6F	SC1162-ND
Switchcraft EN3P6MP	EN3P6MP-ND

Constructing a Type 1 sensor assembly requires the following three steps

1. Programming the DS2431 self-identification EEPROM and installing it in the connector Switchcraft EN3C6F connector shell.
2. Wiring the sensor(s) to the connector shell.
3. Creating an XML file and loading it into the eG2100 gateway

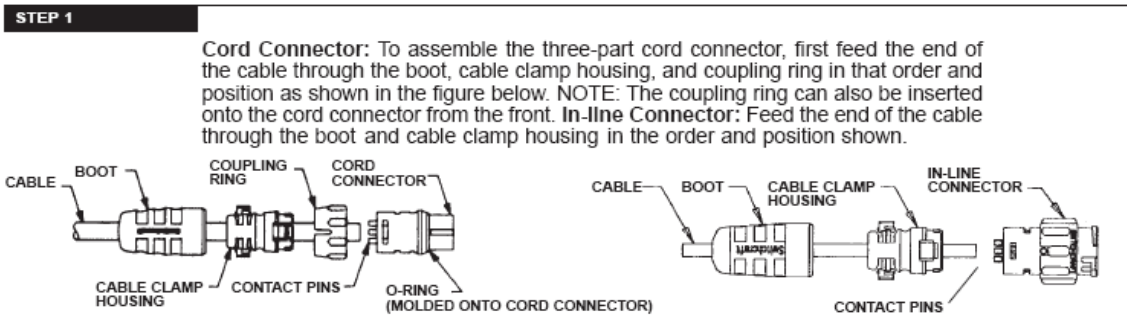
2.1 Step 1: Install & Program the self-identification EEPROM

Refer to Sections 5and 6

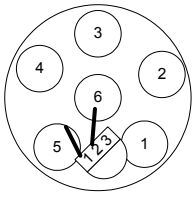
2.2 Step 2: Wire the Sensor(s) to the EN3C6F Connector

Each eN2100 port can support one or two sensors. These can be any combination of 2-wire (10k type) or 3-wire sensors.

1. Route the sensor cable(s) thru the EN3C6F connector backshell components as shown (Cord Connector)



2. Wire the sensors to the Switchcraft EN3C6F connector according to the table below.



Pin	Standard Wire Color	Two 2 Wire Sensors	Two 3 Wire Sensors	One 3 Wire and one 2-Wire Sensors
1	Red	N/U	Excitation [2]	Excitation [2]
2	Orange	Sensor 1+ [1]	Sensor 1 Out [3]	Sensor 1 Out [3]
3	Yellow	Sensor 1- [1]	Exc Monitor [4]	Exc Monitor [4]
4	Green	Sensor 2+ [1]	Sensor 2 Out [3]	Sensor 2 + [1]
5	Black	Sensor 2- [1]	Gnd [5]	Sensor 2 - [1]
6	Brown	ID (n/c)		

[1] Either wire of a 2-Wire sensor can be wired to either of the two Switchcraft pins.

[2] This is common excitation (power) for all 3-wire sensors on the connector. Solder the excitation input of all 3 Wire sensors to this pin.

[3] The output of the sensor

[4] Jumper this pin to Pin1 and it will monitor and report the actual excitation to the sensor.

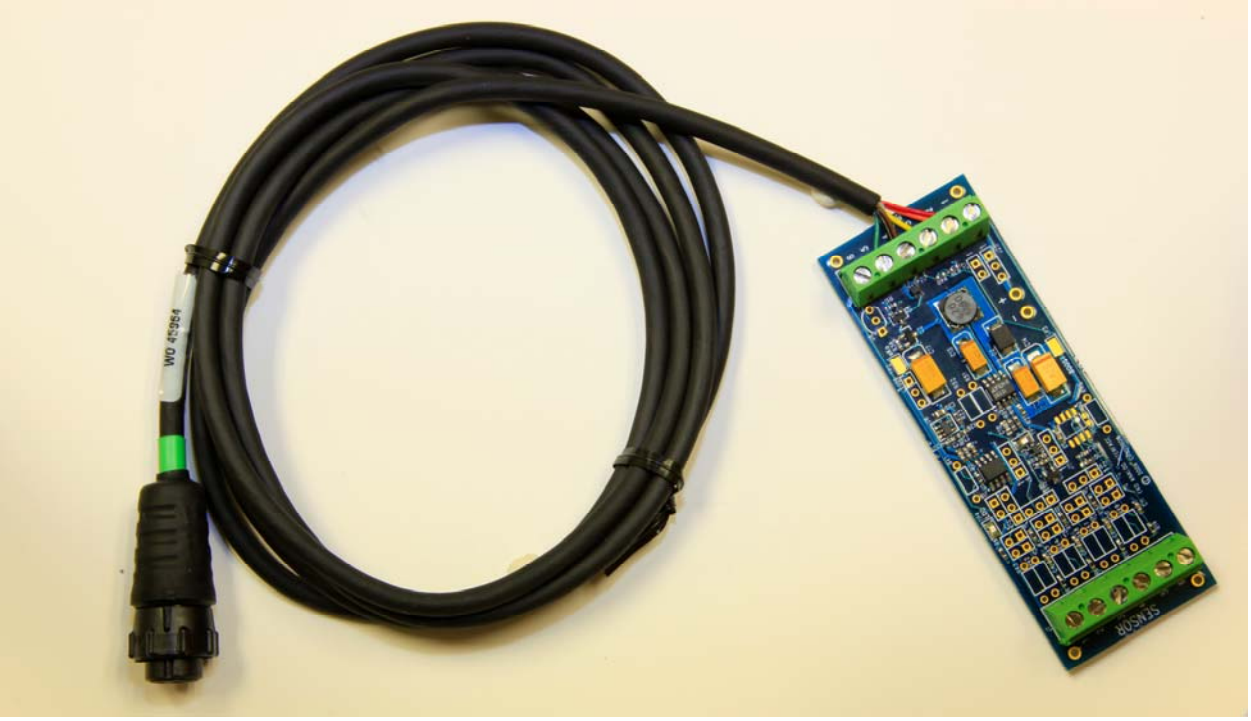
[5] Common ground for all 3-wire sensors

2.3 Step 3: Create the XML File

Refer to section 7.

3 Type 2 Analog Sensors using eS9100

Type 2 sensors are analog sensors that require additional signal conditioning and/or power conditioning than the ēKo node normally provides. Type 2 sensors use the eS9100 Analog Sensor Development platform. The eS9100 is a reference design board that contains flexible signal and power conditioning circuits to interface to many different types of sensors. All design documentation (schematics, gerbers, parts list) are available to users who wish to customize their own version of the eS9100.

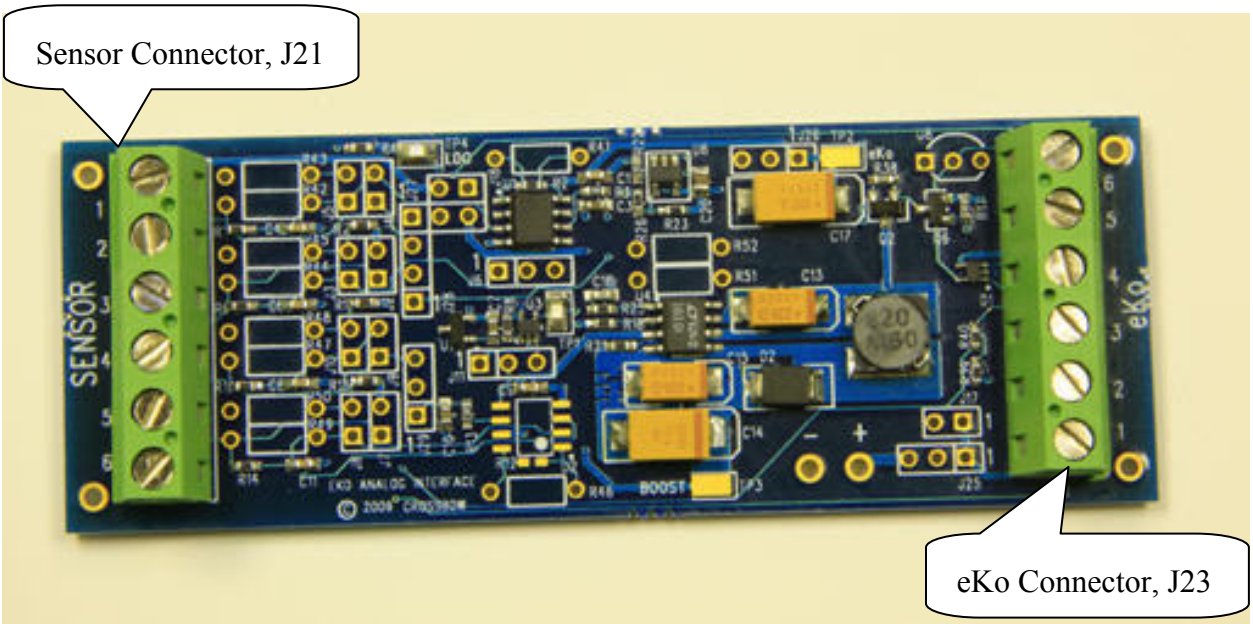


Features of this board include:

- Support for two analog sensors on one ēKo port.
- Each sensor input is:
  - Configurable (jumper) for single or differential input.
  - Selectable (resistor) attenuation
  - Selectable (resistor) gains of 1- 500 for the on-board AD623 amplifier.
  - Selectable (jumper) amplifier (AD623) offset of ground or 1.2 volts.
- Selectable (jumper) sensor power:
  - ēKo battery voltage (3.6-4.2 volts)
  - Voltage booster powered by ēKo battery or external battery. Resistor configurable.
  - Precision linear voltage regulator powered by ēKo battery or external battery. Resistor configurable.
- Two 6 pin screw terminals (one for external sensors and one for the ēKo cable)
- Five foot cable connecting with DS2431 eeprom and Switchcraft EN3C6F ēKo port connector.

NOTE: Only one sensor input channel on the eS9100 is loaded with an AD623 amplifier.

3.1 eS9100 Connections



Wire connections from eKo to eS9100, J23

J23 PIN	Color	Sensor Signal
1	Red	eKo battery voltage
2	Orange	Sensor channel 1 output
3	Yellow	Excitation Monitor to eKo
4	Green	Sensor channel 2 output
5	Black	Gnd return
6	Brown	ID (n/c), EEPROM pin

Wire connections to sensor from eS9100, J21

J1 PIN	Sensor Signal
1	Sensor Voltage
2	Sensor channel 1 positive input
3	Sensor channel 1 negative input
4	Sensor channel 2 positive input
5	Sensor channel 2 negative input
6	Ground

3.2 Configuring the eS9100 PCB

The eS9100 is configured for specific sensors by changing resistor values and wire jumpers. The factory configuration is:

- VCC_Sensor (sensor power) : Voltage booster enabled, 11 volts out
- Channel 1 gain set for 3.0, differential input
- Eeprom programmed for Voltage Sensor.

The last page of the schematic shows the standard jumpers and resistors. All resistors on the board are surface mount (0603 size). Since changing these resistors may be difficult for some users, parallel through-hole resistor pads have been placed on the board by each surface mount resistor. Users can either remove the surface mount resistors and insert through-hole resistors or just parallel the surface mount resistor with a through-hole resistor. Refer to section **Error! Reference source not found.** for more options.

3.3 Interfacing Sensors to the eS9100

Constructing a Type 2 sensor assembly requires the following four steps

1. Programming the DS2431 self-identification eeprom and installing it in the connector Switchcraft EN3C6F connector shell.
2. Configuring eS9100 jumpers and resistors for the analog and power conditioning circuits.
3. Connecting the ēKo and sensor cables to the eS9100.
4. Creating an XML file and loading it into the eG2100 gateway

3.4 Step 1: Install & Program the self-identification EEPROM

The eS9100 assembly is delivered with a 5ft cable, Switchcraft EN3C6F connector and DS2431 EEPROM already attached. To program the EEPROM refer to Sections 5 and 6.

3.5 Step 2: Configure eS9100 Jumpers and Resistors

Section 10 and the table in the eS9100 schematic shows all the jumper and resistor selections for the board. Some example files on the ēKo sensor integration page show how to configure the eS9100 for some different sensors. Most configuration options are jumper selectable but others such as amplifier gain or output booster voltage require changing surface mount resistors. The eS9100 amplifier (Analog Devices AD623) has a limited input full scale so users should refer to its datasheet.

3.6 Step 3: Connecting ēKo and Sensor Cables to the eS9100

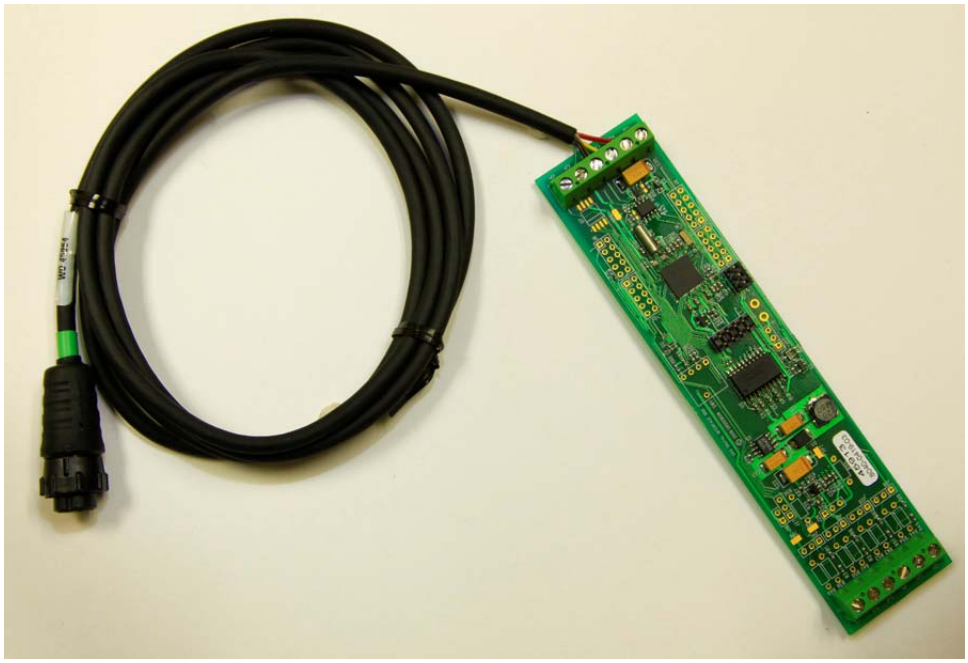
The eS9100 has two 6 pin screw type connectors for the ēKo and sensor cables. Refer to the schematic for the correct pin wiring.

3.7 Step 4: Creating an XML

Refer to Section 7.

4 Type 3 Digital Sensors using eS9200

Type 3 sensors are digital sensors such as the Decagon 5TE soil moisture/salinity sensor that use a digital interface instead of analog and require serial communication to transfer data to the ēKo node. The eS9200 is a reference design board that contains flexible communication and power conditioning. All design documentation (schematics, Gerbers, parts list, software code) are available to users who wish to customize their own version of the eS9200.



Hardware Features include:

- Programmable Atmega128L µP for ēKo communication and sensor interface.
  - JTAG connector for code development and debug
  - ISP connector for code loading and fuse settings.
- RS485 transceiver for half-duplex communication with ēKo node.
- RS232 transceiver for communication with intelligent sensors.
- Selectable (jumper) inputs from sensors for interrupt response, pulse counting, SPI and I2C port interfaces.
- Selectable (jumper) ēKo battery or external voltage for sensor power.
- Selectable (jumper) voltage boost and/or linear regulator output for sensor power.
- Two 6-pin screw terminals (one for external sensors and one for the ēKo cable). One eS9200 can communicate with multiple sensors. For example one eS9200 can communicate with an anemometer (wind speed and direction) and a rain bucket.
- Five foot cable with Switchcraft EN3C6F connector for the ēKo port.

Firmware Features include:

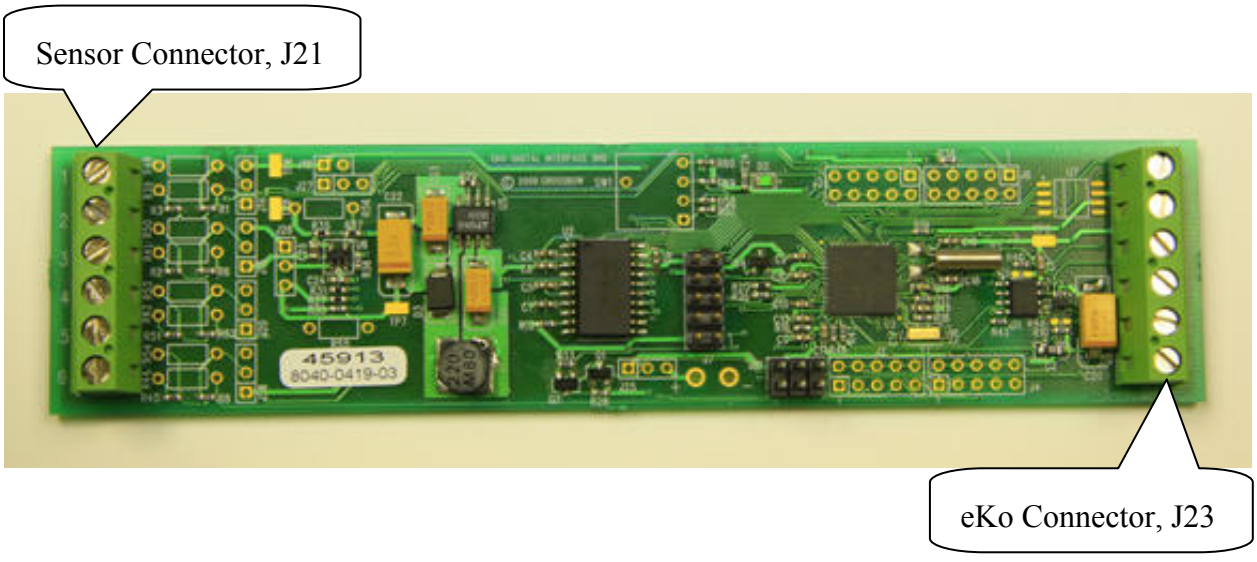
- Code Development:

- Language: C. All source code and executables for ēKo communication along with some sensor examples (such as Decagon 5TE and general weather sensor) are available on the Crossbow ēKo sensor integration page.
- Development Platform: IAR for Atmega128. (Free version available from IAR for limited code development (www.iar.com).
- Code debug using IAR tools and inexpensive JTAG ICE (In Circuit Emulator) such as Olimex, PGM-00012 (available from Sparkfun http://www.sparkfun.com)

- In circuit programming and fuse settings using inexpensive Atmel ATAVRISP2 pod (Digikey part ATAVRISP2-ND available from http://www.digikey.com/)

- Low power support. Allows the eS9200 to be powered continually from the ēKo node to support sensors such as anemometers and rain buckets. These sensors are continually updating versus other sensors that only power-on when a data sample is taken. The eS9200 runs in a very low power sleep mode, servicing sensor interrupts when required and sending a data packet every ēKo sample period.

4.1 eS9200 Connections



Wire connections from eKo to eS9200, J23

J23 PIN	Color	Sensor Signal
1	Red	eKo battery voltage
2	Orange	RS485 +
3	Yellow	IntComm (reserved)
4	Green	RS485-
5	Black	Gnd return
6	Brown	ID (n/c)

Wire connections to sensor from eS9200, J21

J1 PIN	Sensor Signal
1	Sensor Voltage
2	Sensor signal 1
3	Sensor signal 2
4	Sensor signal 3
5	Sensor signal 4
6	Ground

4.2 Configuring the eS9200 PCB

The eS9200 is configured for specific sensors by changing resistor values and wire jumpers. All resistors on the board are surface mount (0603 size). Since changing these resistors may be difficult for some users, parallel through-hole resistor pads have been placed on the board by each surface mount resistor. Users can either remove the surface mount resistors and insert through-hole resistors or just parallel the surface mount resistor with a through-hole resistor. Refer to section 11 for more options.

4.3 Interfacing Sensors to the eS9200

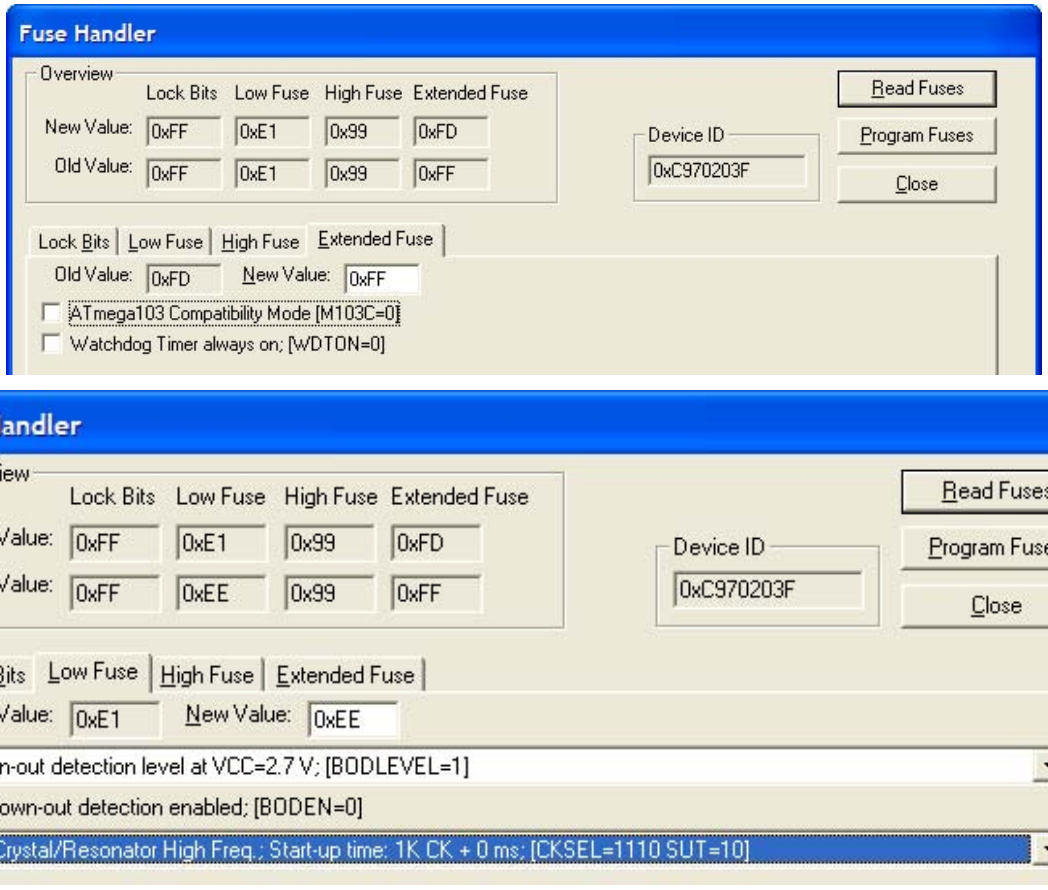
Constructing a Type 3 sensor assembly requires the following four steps

1. Programming the eS9200 Atmega128 processor.
2. Configuring jumpers and resistors for the analog and power conditioning circuits.
3. Connecting ēKo and sensor cables to the eS9200.
4. Creating an XML file and loading it into the eG2100 gateway

4.4 Step 1: Programming the eS9200

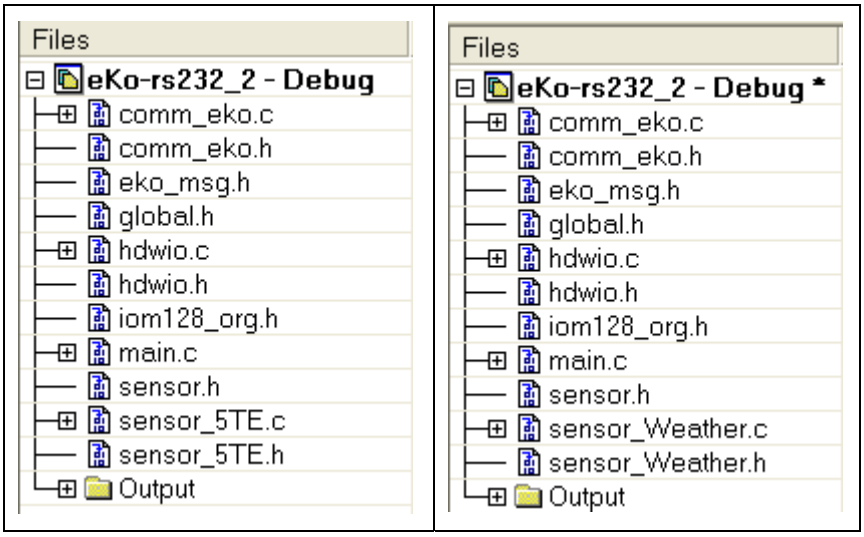
As new sensor applications are developed such as a general weather interface Crossbow will make both the code and executable available. Users who do not need to modify the code can just download the available executable through the ISP connector on the eS9200 board.

The eS9200's Atmega128L µP is programmed with the Atmega103 computability fuse disabled and the external oscillator, high frequency, enabled with a 1K clock startup.



All code for the eS9200 is available on the Crossbow ēKo sensor integration page. Code for all sensors is contained in one IAR project, DIM100.eww in the eS2900Firmware directory. To build different sensor versions:

- In the IAR project pane delete the present sensor_xx .c and .h files and replace them with the .c and .h of the sensor files of interest. For example to build code for the Davis weather sensor remove the sensor_5TE .c and .h files (left pane below) and replace with the sensor_Weather .c and .h files (right pane below)



- **Creating code for new sensors:**

  This requires writing a new sensor_xx.c file for the particular sensor. This file contains all code particular to the sensor and also some common procedures (see sensor.h) used for communication with the ēKo node:

  - sensor_get_SIS() : this procedure returns the sensor information structure (SIS) used by ēKo to identify the sensor. It is the same structure as defined in Section 5. Copy sensor_get_SIS() from sensor_5TE.c and only change the following parameters:

    pSisMsg->SensorId=SENSOR_ID

    SENSOR_ID is unique to the sensor. Refer to the Crossbow ēKo sensor integration page for the most recent sensor ids.

    pSisMsg->WarmUp=0x14

    WarmUp: Encoded time it takes for the on-board (eS9200 Atmega128) processor to boot-up and start receiving messages from the Node.

    pSisMsg->Sampling=0x14

    Sampling: Time it takes the eS9200 to return a data packet after receiving a data message from the ēKo node.

    pSisMsg->PwrCycle=0;

    A zero value tells the ēKo node to only turn on power to the eS9200 when taking a sample (every 15 minutes). A value of 0xffff can be used to always keep power on to the eS9200 after the first sample. This is required for sensors such as anemometers or rain buckets that need continual monitoring. To use this feature requires:

    1) An ēKo node firmware upgrade (contact Crossbow).
    2) The eS9200 must run in a very low power mode in order not to drain the ēKo batteries. Average current should not exceed 400uA. If using the eS9200 with the general weather sensor code users must disable the JTAG fuse on the Atmega128 or else the eS9200 will draw an average of 4 mA and drain the batteries within a few days.

- sensor_get_data(); this procedure performs a measurement on all sensors and returns a data structure to forward to the ēKo node. The data structure is defined by the struct Xm_msg. Users need to change the value of SENSOR_DATA_LEN in the structure which is the number of data bytes returned to ēKo. Users also need to fill-in the sensor data bytes in the structure. Any data can be returned, the maximum SENSOR_DATA_LEN should be kept below 48 bytes.
- sensor_init(): this procedure initializes all sensor unique I/O ports, interrupts, and parameters.

- **ēKo Communications**

  When the ēKo node is turned on it executes the following:

  - Interrogates each port to see if a sensor is attached. First it will assume that there is an analog sensor attached and try to read the eeprom embedded in the connector. If no eeprom is found then the node will see if a digital sensor is attached. It does this by sending a message to the sensor over the communications bus requesting the SIS structure (which calls sensor_get_SIS()). If a SIS data packet is returned the node knows an active digital sensor is present. At this point it turns off power to the sensor.
  - After sensor discovery the node starts sampling every 15 minutes. This means it will send a message to the digital sensor requesting return of a data packet (which calls sensor_get_data()). If the SIS PwrCycle parameter is set for zero the node will turn off power after receiving the data or a time-out occurs. If PwrCycl is set for 0xffff then the node keeps the power active.

- **Testing Code via JTAG**
  - Since in-circuit debugging via JTAG requires the processor to be continually powered an external power source (~4V) should be used. Disconnect the ēKo cable wire on J5(1) (power) and J5(6) (ground) and reconnect to an external source. If the JTAG fuse was disabled it must be re-enabled before starting using the ISP interface.
  - Connect the JTAG pod's connector to J7 and use IAR Embedded Workbench to download code and set breakpoints.
  - Start running the code then turn-on the ēKo node. When it scans the port connected to the eS9200 the node should display a green indicator. If a red indicator is shown then the node has not been able to communicate with the eS9200.
  - After the node finishes port scanning and connects to the network (blue flashing led; see ēKo user's guide) it will sample sensors at a 30 second interval for about one hour, after that it will sample every 15 minutes.
  - After code development is complete the JTAG fuse should be disabled if the eS9200 is set with PwrCycle for always on. If this isn't needed the JTAG fuse can be left enabled.

4.5 Step 2: Configure eS9200 Jumpers and Resistors

The table on sheet 4 of the eS9200 schematic shows all the jumper and resistor selections. The sensor files on the ēKo website for Type 3 sensors show how to configure the eS9200 for some different sensors.

4.6 Step 3: Connecting ēKo and Sensor Cables to the eS9200

The eS9200 has two 6 pin screw type connectors for the ēKo and sensor cables. Refer to the schematic for the correct pin wiring.

4.7 Step 4: Creating an XML

Refer to Section 7.

5 Sensor Identification Structure (SIS)

When the eN2100 powers-up it reads a data structure (SIS) from an embedded EEPROM in the sensor cable to auto-identify the attached sensor. The present SIS structure is defined by the table below.

Field	Bytes	Data Type	Required (Y/N)? If N then set = 0	Description
ByteTotal	1	Char	Y	Total number of bytes to follow not including crc
Header Block				
BytesHdr	1	Char	Y	Number of bytes in header block (1)
Version	1	Char	Y	Version number of table, incremented each time a change is made
Sensor Block				
BytesSensor	1	Char	Y	Number of bytes in sensor block (1)
Sensor Id	2	Uint	Y	Uniquely identifies the sensor. Only lower byte used for Version 1 & 2. Example: ēKo102 = 16. Refer to section 3.1
SN	6	Lint	N	Optional, serial number
Lot Code	2	Uint	N	Optional manufacturing lot code
NOM	1	Char	Y	Number of measurements (physical sensors) (1 or 2) attached to a eN2100 connector. For smart sensors always = 1. (One smart sensor can return many sensor measurements, but in one data packet).
Measurement Block for the first sensor on connector (wired to pins 1,2,3)				
BytesMeasure	1	Char	Y	Number of bytes in measurement block
SensorType	1	Char	Y	1: 2 wire, resistive, 10K ohm completion 2: Digital (smart, type 3) 3: 3 wire 4: Pulse count (not supported yet)
Boot-upTime (BT) [1]	1	Char	Y	Time required for sensor to boot-up or warm-up. See codes below [3]
MeasureTime (MT) [2]	1	Char	Y	Time required for sensor signal to be present See codes below. [3]

Interrupt	1	Char	Y	**Upper 4 bits** 0: no interrupt capability 1: interrupt on falling edge 2: interrupt on raising edge 3: interrupt on transition **Lower 4 bits** 0: do nothing on interrupt 1: forward data on interrupt 2: forward data on interrupt, also at standard sampling interval 3: accumulate interrupt counts; transmit counts at sample interval, zero counts 4: accumulate interrupt counts until max value of xx and forward data, zero counts
Power Source	1	Char	Y	For Analog Sensors: 0 : not used (2 wire, 10k ohm) 1 : GPIO (10 ma max @ 3.0V) 2: Battery (3.6V to 4.3V typical) 3: External For Digital Sensors (smart) 0 : not used (2 wire, 10k ohm) 1: Battery (3.6V to 4.3V typical) 2 : GPIO (10 ma max @ 3.0V) 3: External
Power Cycle	2	Uint	Y	0 : turn on power to sensors only when making measurement 1…0xFFFE : power on in msec 0xFFFF: always on
Measurement Block for second sensor on connector (wired to pins 3,4,5)				
Same bytes as #1 [4]				
CRC	2	Uint	Y (only for non-smart sensors)	CRC of all bytes in SIS (except crc); includes ByteTotal

[1] Boot-up Time (BT): The time between the eN2100 activating power to a smart sensor and then sending the command to make a measurement or return the SIS. The time must be long enough to allow the smart sensor uP to boot and be ready to receive commands. Minimum time allowed is 3 msec. For analog sensors Boot-up Time is not used and can be set to zero (as fast as possible)
[2] Measure Time (MT): The maximum time it takes for the sensor to return data to the eN2100. For analog sensors this is the warm-up time; for digital sensors it is the time after receiving the command to make a measurement. The eN2100 will turn-off power after receiving the data or if

no data is transmitted after the Measure Time expires so this interval should be long enough to cover the worse case measurement time. Minimum time allowed is 3 msec.

[3] Codes for Boot-up Time and Measure Time.

- This is a single byte value; the UN (upper nibble, i.e. upper 4 bits) specifies a time value between 1 to 9. The LN (lower nibble, i.e. lower 4 bits) is a multiplier.
- Time is based on a 32 kHz clock with 1024 clock tics per msec.
- Time is in units of 1/8 msec or 128 clock ticks.
- Time is computed as UN times the LN.
- The max allowed time is UN= 5 and LN = 7 (512 sec)

Upper Nibble Values		**Lower Nibble**	
		Value (hex)	**Multiplier**
0	As fast as possible	0	X1
1	X1	1	X5
2	X2	2	X10
3	X3	3	X50
4	X4	4	X100
5	X5	5	X500
6	X6	6	X1,000
7	X7	7	X5000
8	X8	8	X10,000
9	X9	9	X50000
A	0x10	A	X100,000
B	N/A	B	X500,000
C	N/A	C	X1,000,000
D	N/A	D	N/A
D	N/A	E	N/A
F	N/A	F	N/A

☑ EXAMPLE

UN (hex)	LN (hex)	Boot-up Time or Measure Time (decimal)	Time
0	0	0	As fast as possible
1	0	16	0.128 msec
2	0	32	0.256 msec
1	2	18	1.28 msec
1	4	20	12.8 msec
1	6	22	128 msec
1	8	24	1.28 sec
1	A	26	12.8 sec
1	C	28	128 sec

[4] This block is not used for intelligent sensors. An intelligent sensor is treated as a single sensor that can send one or multiple measurements.

5.1 Sensor Ids

Each sensor type requires a unique identification number (1..255). The Sensor ID is read by the eN2100 node and sent with every data packet. When received by the eG2100 gateway the sensor's data conversion, storage, and display properties are determined by an XML file with the same Sensor_Id. Different sensor types with the same Sensor_Id in the same ēKo network will not be interpreted correctly.

◂ **NOTE:** The most up to date version of this file is available on the Crossbow ēKo sensor integration page or contact Crossbow technical support.

Sensor Types 1-16 reserved for eN2100 variations	Sensor ID	Sensor ID (hex)
eN2100 internal sensors (voltages, temp)	?	?
Reserved	1	1
Reserved	2	2
Reserved	3	3
Reserved	4	4
Sensor Ids 16-127 reserved for Crossbow sensors		
eS1101,Crossbow, Watermark soil moisture and soil temperature	16	10
eS1201,Crossbow, Ambient Temperature & Humidity	17	11
eS1301, Decagon leaf wetness	18	12

eS1110, Decagon EC-5 soil content	20	13
eS140-1,Davis 6450 solar radiation sensor	21	15
Rain gauge	22	16
Anemometer	23	17
Sensor Ids reserved for Crossbow non-ēKo sensor boards	128-154	80-9A
Sensor Ids reserved for users	155-239	9B-EF
Reserved	240-255	F0-FF

6

Programming and Installing the EEPROM

NOTE: The EEPROM can be programmed directly in the Maxim fixture (see below) before installation in the Switchcraft connector however it's easiest to first install the EEPROM in the connector and use an EN3P6MP adapter then program the EEPROM.

6.1 Step 1: Create a SIS Text File

Create a text file for the SIS structure to be programmed into the EEPROM such as myfile.txt. See the MaxBotixTemp.txt file for an example. When filling out the file:

- Only change the **Value** fields and leave all other fields the same.
- Do not change the values in the first 5 rows.
- **Sensor ID**: This is a unique 8 bit number (0..255) that uniquely identifies the sensor. Contact Crossbow for a new number or make sure that no to other sensors use the same ID.
- **Serial Number L, Serial Number H, Lot Code**: optional sensor serial number and lot code. Presently these values are not transmitted to the base station (future enhancement)
- **NoElements**: This is one or two depending on the number of sensor attached to the port. If NoElements = 1 then all the rows from 19 to 24 are ignored.
- The crc check value (row 25) is automatically computed.

6.2 Step 2: Convert the SIS file to Bytes

After the SIS file has been created run the DOS utility ēKomake.exe (in EEPROM directory) to create an output file (example MaxBotixTemp_out.txt). The command is:

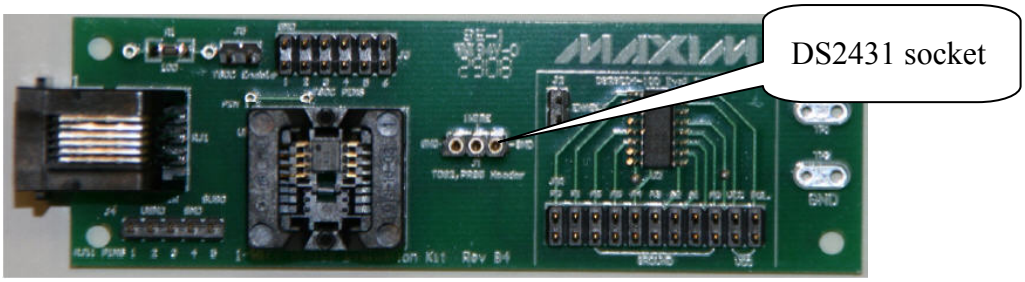
'run ēKomake infile.txt'

where infile.txt is the name of the file with the SIS structure. The ēKomake will create an output hex file called infile_out.txt, for example:

```
1F 01 02 0B 77 00 00 00 00 00 00 00 00 00 02 07
03 00 82 00 01 00 00 07 01 00 10 00 00 00 00 81
F6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

6.3 Step 3: Install and Program the EEPROM

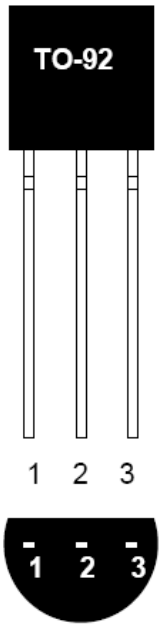
The DS2431P+ EEPROM can be programmed directly in the Maxim DS9090K# fixture (http://www.maxim-ic.com/products/1-wire/kits/9090K.cfm). Crossbow supplies a utility program **ēKomake.exe** to create a hex byte file that is copied into the DS2431 fixture using Maxim's supplied software. The DS2431P+ can be programmed directly via the Maxim fixture and then soldered into the EN3C6F connector.



Users can also solder the EN3P6M Switchcraft mate to the Maxim fixture to act as an extender (see picture below). This allows the EEPROM to be mounted in the Switchcraft connector before programming then connected to the Maxim fixture. Wire center pin of 3 pin terminal to Switchcraft pin6 (eeprom) and the other pin to Switchcraft pin 5 (gnd).

EEPROMs and Switchcraft connectors can be ordered from Digikey (www.digikey.com),

Component	Digikey Part Number
DS2431 EEPROM	DS2431+-ND
Switchcraft EN3C6F	SC1162-ND
Switchraft EN3P6MP	EN3P6MP-ND

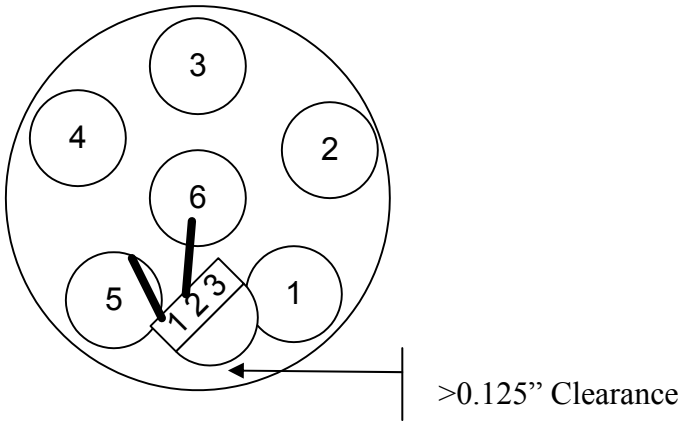


⚠ WARNING

This device is ESD sensitive

1. Cut off PIN#3
2. Form and shape DS2431 PIN# 1 and PIN# 2 for placement in the EN3C6F connector.

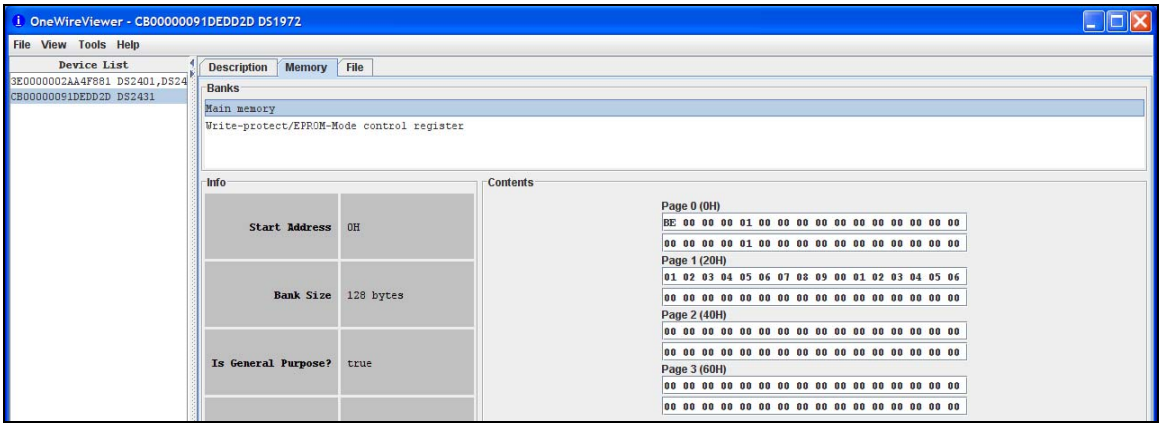
3. Install as shown in the picture below.
   - PIN #1 connects to #5.
   - PIN #2 connects to #6.



4. Maintain >0.125” clearance between component and connector wall. This is required to allow complete assembly of connector backshell.

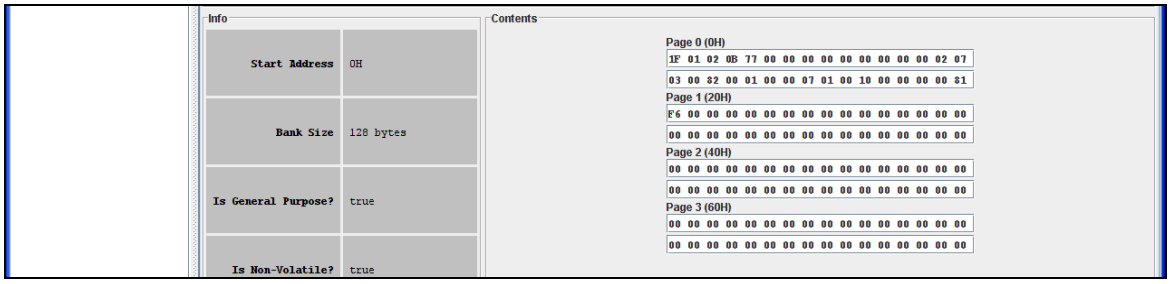
Program the DS2431

- Install the DS2431 on the DS9090K eval board or connect via the EN3P6MP connector.
- Start the Maxim ‘OneWireViewer’ (delivered with the Dallas DS9090K kit), select the DS2431 from the ‘Device List’ , select the ‘Memory’ tab and then select ‘main memory’



- Delete all entries on Page 0 and first line of Page 1

- Copy and paste each line from the ēKomake output file (i.e. myfile_out.txt) to the Page 0 and Page 1 entries



- Select ‘Commit Changes’ to program the device.

6.4 Step 4: Test the Programmed Assembly

Connect the completed assembly to an eN2100 port and reset the unit (press the On button)

If the eN2100 successfully reads the EEPROM it will flash green during the port scan.

7 Creating XML Files

Each ēKo sensor assembly needs a corresponding XML file that's loaded into the ēKo eG2100 gateway. When the gateway services start-up all the XML files are read. The XML files are used to:

1. Identify the incoming sensor data. Each sensor data packet contains an identifier that uniquely identifies the sensor type. Refer to Section 3
2. Convert the incoming data to final engineering units (ex: psi).
3. Assign labels to the data for display in ēKoView.

Users can view all Crossbow support XML files in the eG2100's /xserve/configxml directory.

After the xml file is created do the following:

- Upload to the /xserve/configxml directory. This can be done via SSH or mapping the gateway to a PC drive (see ēKo User's Manual)
- Restart the gateway using the Gateway services (see user manual)

An example is the ex9119_ET_119_Maxbotix_v1.xml file as shown below. The XML files are broken down into the following blocks:

In the first block the only value that should be changed is the name which can be any name that describes the sensor assembly. This name must be unique from any other xml file name.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE XServeConfig SYSTEM "./xserve_config.dtd">
<XServeConfig>
    <XFieldExtractor name="eX9119 MaxBotix Range Sensor v1" order="3">
        <XFields>

        <!-- Tos Hdr -->
            <XField name="amType" byteoffset="2" length="1" type="uint8"/>
            <XField name="group" byteoffset="3" length="1" type="uint8"/>

        <!-- XMesh Hdr -->
            <XField name="nodeId" byteoffset="7" length="2" type="uint16"
specialtype="nodeid"/>
            <XField name="socketId" byteoffset="11" length="1" type="uint8"/>

        <!-- XSensor Hdr -->
            <XField name="boardId" byteoffset="12" length="1" type="uint8"
specialtype="sensorboardid"/>
            <XField name="packetId" byteoffset="13" length="1" type="uint8"/>
            <XField name="ParentID" byteoffset="14" length="1" type="uint16"/>
```

The second block converts raw data in the data packet to engineering units. Analog data from the eN2100 node is converted through a 10 bit adc which results in zero volts having a value of 0 and 3.00 volts having a value of 1023. ADC values are always returned as 2 byte numbers.

The 'byteoffset' field is used to find the byte location of the data in the packet. The first adc value is always at byteoffset = 16. The data in the packet depends on the sensors attached to the port. All analog data from analog sensors (eS9000 and eS9100) are converted by the eko nodes to 2 bytes adc counts (0-1023).

- Two wire sensors (ex: watermark and thermistor) send only 2 bytes of adc data. These sensors are ratiometric and can be accurately measured without a known exact ADC full scale voltage.

byteoffset	Name	Description
16	RefCnts	ADC value of internal reference 1.22V
18	ExcCnts	ADC value of sensor excitation voltage

- Three wire sensors require an accurate, known full scale ADC voltage. Also they require a measurement of the excitation voltage in case the sensor is running from battery voltage. For three wire sensors the data packet contains a measurement of the eN2100's internal reference voltage (used to determine the full scale ADC voltage) and the excitation voltage when the sensor output was measured. The data and order of data in the packet depends on the configuration of sensors attached to the eko port.

3 wire sensor for Sensor #1 and for Sensor #2.

byteoffset	Name	Description
16	RefCnts	ADC value of internal reference 1.22V
18	ExcCnts	ADC value of sensor excitation voltage
20	Sensor1AdcCnts	ADC value of 3 wire sensor output voltage
22	RefCnts	ADC value of internal reference 1.22V
24	ExcCnts	ADC value of sensor excitation voltage
26	Sensor2AdcCnts	ADC value of 3 wire sensor output voltage

3 wire sensor for Sensor #1 and 2 wire sensor for Sensor #2.

byteoffset	Name	Description
16	RefCnts	ADC value of internal reference 1.22V
18	ExcCnts	ADC value of sensor excitation voltage
20	Sensor1AdcCnts	ADC value of 3 wire sensor output voltage
22	Sensor2AdcCnts	ADC value of 2 wire sensor output voltage

Computing the full scale ADC voltage from RefCnts:

AdcFullScaleV = 1.225*2*RefCnts

Computing 3-wire Sensor Output Voltage:

SensorOutputV = 1.225*SenorAdcCnts/RefCnts

The example below shows the conversion of the Maxbotix 3 wire sensor followed by the 2 wire temperature measurement.

For each data a name is assigned. The XFieldname associates this name with the parsed data and also converted data. So in the example below E1ExcitationV has been assigned both the parsed (raw) data from byteoffset 18, 19 and the converted value of ="1.225*2*x/y". pIn any further data conversions in the XML file the parsed value of E1ExcitationV will be used; for data storage, data display the converted value will be used.

```
<!-- Data -->
            <XField name="E1ReferenceADC" byteoffset="16" length="2" type="uint16"/>
            <XField name="E1ExcitationV" byteoffset="18" length="2" type="uint16">
                <XConversion function="1.225*2*x/y" returntype="float">
                    <XConvParam variablename="x" fieldname="E1ExcitationV"
type="float"/>
                    <XConvParam variablename="y" fieldname="E1ReferenceADC"
type="float"/>
                </XConversion>
            </XField>
        <!-- maxbotix conversion - report volts -->
            <XField name="RangeV" byteoffset="20" length="2" type="uint16">
                <XConversion function="1.225*x/z" returntype="float">
                    <XConvParam variablename="x" fieldname="RangeV" type="float"/>
                    <XConvParam variablename="z" fieldname="E1ReferenceADC"
type="float"/>
                </XConversion>
            </XField>
            <XField name="Temperature" byteoffset="22" length="2" type="uint16">
                <XConversion function="((1/(.001307050 + 0.000214381 * log( (10000 *x/
(1024-x))   ) + 0.000000093 * (log( (10000 *x/ (1024-x)) )^3))) - 273.15)"
returntype="float">
                    <XConvParam variablename="x" fieldname="Temperature"
type="float"/>
                    </XConversion>
            </XField>


        </XFields>
        <XFilter>
```

For the third block the only value that should be changed is the fieldvalue in the boardId section. The value must correspond to the “Sensor Id” value in the SIS structure of the sensor assembly. No two XML files can contain the same values.

```
       <!-- LOGIC:  SocketID==XSensorĒKo AND BoardID (SensorId) AND PacketID==0 -->
            <XCondAnd>
                <XCond name="IsEqual">
                    <XFilterParam name="fieldname" value="socketId"/>
                    <XFilterParam name="fieldvalue" value="0x34"/>
                </XCond>
                <XCond name="IsEqual">
```

```
                    <XFilterParam name="fieldname" value="boardId"/>
                    <XFilterParam name="fieldvalue" value="119"/>
                </XCond>
                <XCond name="IsEqual">
                    <XFilterParam name="fieldname" value="packetId"/>
                    <XFilterParam name="fieldvalue" value="0x0"/>
                </XCond>

            </XCondAnd>
        </XFilter>
```

The last block determines how data is stored in the eG2100 database, presented in ēKoView, and logged to CSV files. There are three sections:

- Generic Print Datasink: Determines the variables and order for screen display if running Xserve in terminal mode. Normally this doesn't need to be present.
- Generic File Datasink: XServe can log data directly to CSV files when running. If logging to files in enabled three CSV (Comma Separated Variable) files can be logged to the local flash memory in the gateway. The three files are:
    - Sensorname_raw.csv - contains all data from each data packet. Not usually used.
    - Sensorname_Parsed – contains the all the parsed data as determined by the order of the XML variable parsing.
    - Sensorname_Converted – contains the converted data as determined by the order of the XML variable parsing.
- Sensor Log Datasink: Defines that database table for the sensor and how it's displayed in ēKoView. For the following lines in the XML file:
    - sensorid must be the same as the board id
    - tablename defines the database table; must be unique
    - sensorname determines the name that will appear for the sensor in ēKoView
    - node_Id should not be changed. This associates the data with the node number that sent it. This should have 'display_order = 1"
    - The next lines determine how which data is displayed in ēKoView and parameters associated with it.
        - fieldname is a the name of the data defined in the XML parsing above.
        - displayName is the name that will appear for the sensor measurement in ēKo.
        - displayorder should just be incremented for each additional parameter in the list
        - unitName defines the engineering units for the measurement. ēKoView can plot multiple sensor measurements on the same axis if they have the same unitName.
        - unitShortname defines a truncated unit of measure, similar to the unitName.

- sensorType allows ēKoView to associate specific calculations (such as dew point) with the right sensor.
- sensorMinValue and sensorMaxValue configure the maximum and minimum sensor values for plots or bar charts.

```
     <XDataSinks>
          <XDataSink name="Generic Print Datasink">
               <XDSParam name="printstring" value="eX9119 MaxBotix[%s:%s]:\n
Parent:%s PortID:%s \n Temperature:%sC RangeV:%s V ExcitV:%s RefADC:%s"/>
               <XDSParam name="printfields"
value="boardId,packetId,ParentID,nodeId,Temperature,RangeV,E1ExcitationV,
E1ReferenceADC"/>
           </XDataSink>
           <XDataSink name="Generic File Datasink">
               <XDSParam name="rawfilename" value="eX9119_ET119_MAXBOTIX_Raw.csv"/>
               <XDSParam name="parsedfilename"
value="eX9119_ET119_MAXBOTIX_Parsed.csv"/>
               <XDSParam name="convertedfilename"
value="eX9119_ET119_MAXBOTIX_Converted.csv"/>
               <XDSParam name="delim" value=","/>
               <XDSParam name="header" value="yes"/>
               <XDSParam name="timestamp" value="%m-%d-%Y %H:%M:%S"/>
               <XDSParam name="backup" value="yes"/>
           </XDataSink>
           <XDataSink name="Sensor Log Datasink">
               <XDSParam name="sensorid"  value="119"/>
               <XDSParam name="tablename"  value="eX9119_sensor_results"/>
               <XDSParam name="sensorname"  value="eX9119 MaxBotix"/>
               <XDSParam name="columninfo"  value="fieldName = nodeId, displayName =
Node Id, displayOrder = 1"/>
               <XDSParam name="columninfo"  value="fieldName = RangeV,displayName =
RangeV, displayOrder = 2,unitName = Volts, unitShortName = V, sensorType = Voltage,
sensorMinValue = 0, sensorMaxValue = 6"/>
               <XDSParam name="columninfo"  value="fieldName = E1ExcitationV,
displayName = Excitation,displayOrder = 3,unitName = Volts, unitShortName = V,
sensorType = Voltage, sensorMinValue = 0, sensorMaxValue = 6"/>
               <XDSParam name="columninfo"  value="fieldName =
Temperature,displayName = Temperature, displayOrder = 4,unitName = Celsius,
unitShortName = C, sensorType = Temperature, sensorMinValue = -50, sensorMaxValue =
125"/>
           </XDataSink>
       </XDataSinks>
   </XFieldExtractor>


</XServeConfig>
```

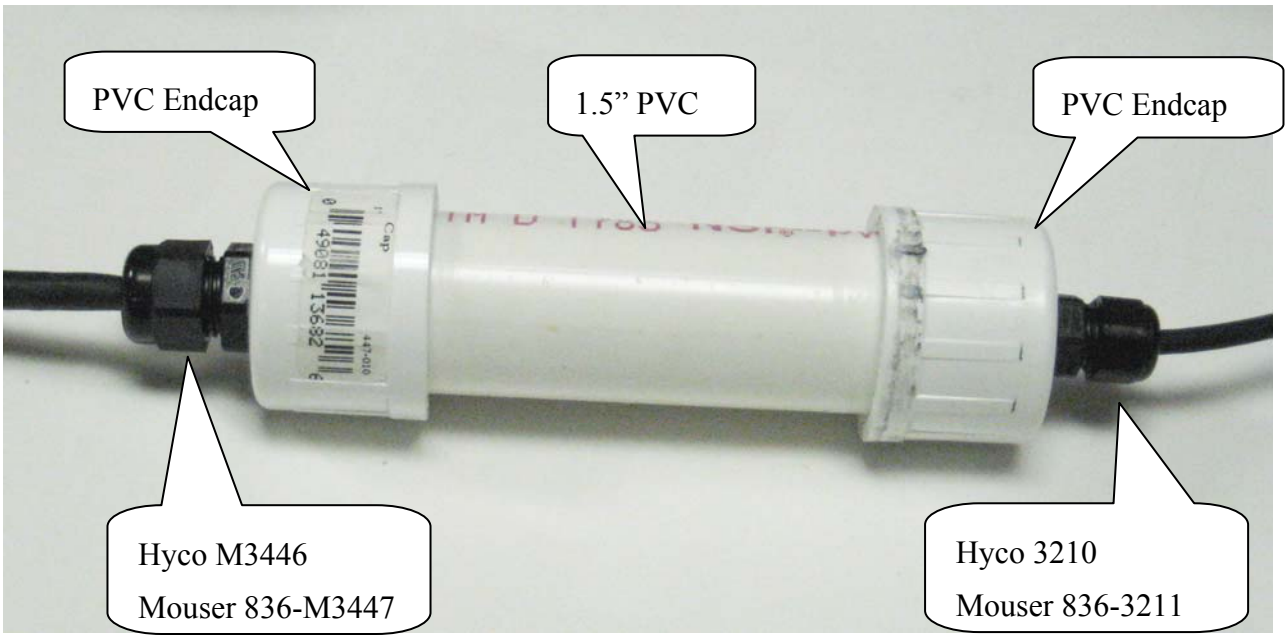
8 eS9000, eS9100 & eS9200 Cable

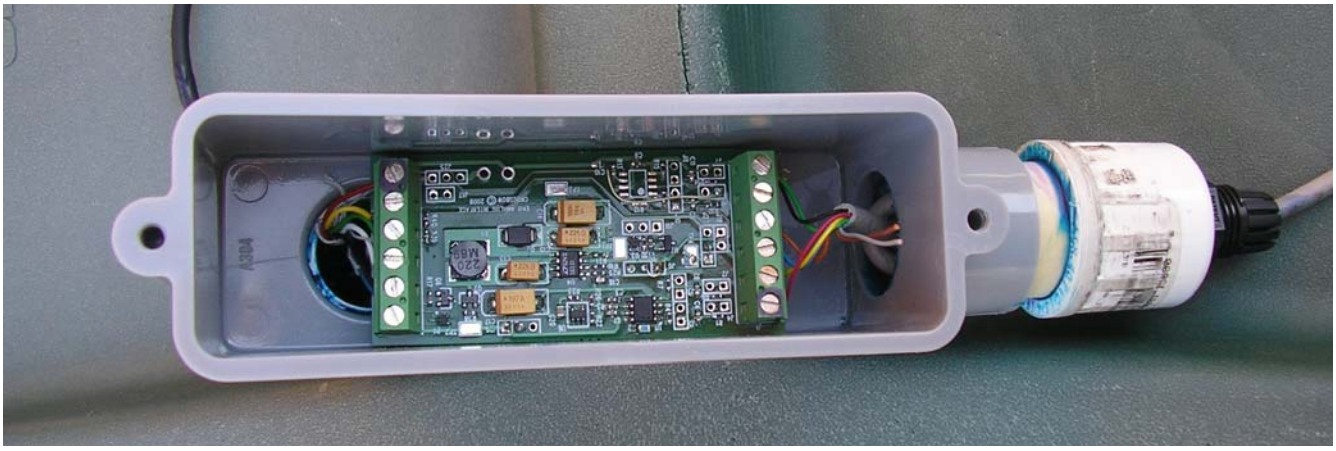
Crossbow uses the following cable to connect the eS9000/eS9100/eS9200 and Switchraft connector. Any 6 conductor cable will work; the cable below is specified for outdoor usage.

Conductors	6
AWG	26
Type	Stranded (7 strand) tinned copper
Insulation	PVC UL1061
Color Code	Blk, Brn, Red, Org, Yel, Grn
Rating	600V/105C
Jacket Rating	125C
Jacket Material	Thermo Plastic Rubber UV Tolerant
Jacket Color	Black
Cable OD	0.180 +/- 0.10 inch
Compliance	ROHS

9 eS9100 & eS9200 Enclosures

The eS9100 and eS9200 are supplied as bare PCBs but need to be deployed in weather proof packaging. The production units are sized (width) to fit inside of an inexpensive 1.5” diameter PVC pipe or electrical box. Standard PVC end caps can be attached to both ends with the correct weatherproof Hyco connectors for cable feedthrough. Hyco connectors can be ordered from Mouser (www.mouser.com). The picture below shows an example of how to do this:





10 eS9100 Jumpers and Resistor Options

10.1 Sensor Signal Conditioning

	Ch 1 Jumpers		Ch2 Jumpers		Notes
Single ended Input	Short	Open	Short	Open	
No Attenuation	J2,J4	J5	J7,J9	J10,J12	
Attenuation (0.5)	J2,J5	J4	J7,J10	J9,J12	Remove surface mount resistors and install through hole resistors for other attenuations.
Differential Input					
No Attenuation		J4,J2, J3,J5	J9	J7,J9,J8,J 10,J12	Standard Config for Ch1
Attenuation (0.5)	J3,J5	J2,J4	J8,J10	J7,J9,J12	Remove surface mount resistors and install through hole resistors for other attenuations.
4-20ma Input	J2,J4,J5		J7,J9,J10	J12	Install R43 or R46 through hole resistors for correct current to voltage full scale.
Amplifier Reference Offset					
No Offset	J6(2->1)		J11(2->1)		Standard Config for Ch1
1.2V Offset	J6 (2->3)		J11(2->3)		
Output					
Amplifier output	J28(1->2)		J29(1->2)		Standard Config for Ch1
Direct output, no amplifier	J28(2->3)		J29(2->3)		

Amplifier Gain	**Resistors ohm(R2,R12)**		**Rx = 100K(G-1) (Use 0.1%, 25 ppm)**
2	100K	Digikey # P100KYCT-ND	Remove surface mount R2 Install through hole R41
2.96	51K	Digikey # P50KYCT-ND	Standard Config for Ch1
10	11 K	Digikey # P11KYCT-ND	
101	1K	Digikey # P1KYCT-ND	
501	200	Digikey# P200YCT-ND	

10.2 Sensor Power Options

	Short	Open	Notes
VCC_BOOST from VCC_EKO	J25(2->1)	J24	eKo Node powers booster Standard Config
VCC_BOOST from External Voltage	J25(2->3)	J24	External supply powers booster
Enable Booster	J17		Standard Config
Disable Booster		J17	
VCC_LDO from VCC_EKO	J26(2->1)		Source for LDO output Standard Config
VCC_LDO from VCC_BOOST	J26(2->3)		13.V max
VCC_SENSOR from VCC_EKO	J18	J27	
VCC_SENSOR from VCC_BOOST	J27(2->3)	J18	Standard Config
VCC_SENSOR from VCC_LDO	J27(1->3)	J18	
Booster Voltage (V)	R21 (Ohm)		R21=R18/(VCC_BOOST/1.23 – 1)
10.23	143K		Standard Config
7.38	200K		Remove surface mount R21, Install through hole R51
12	113K		“
LDO Voltage (V)	R23 (Ohm)		
2.5	174K		Standard Config
4.0	383K		Remove surface mount R23 and install through hole R52
5.0	523K		“

11 eS9200 Jumpers and Resistor Options

11.1 Sensor Input Options

RS232 Inputs/Outputs:

	Rx J1(5)	Tx J1(4)
Enable RS232 Converter Input/Outputs	J4 (7->8)	J2(7->8)
Enable logic level Input/Outputs	J4(9->10)	J2(7->10)

Analog-Digital Sensor Inputs:

J1 pin	uP Analog Channel	Jumper
2	ADC0	J6(1->2)
3	ADC1	J3(1->2)
4	ADC2	J6(1->2)
5	ADC3	J6(1->2)

SPI Bus:

J1 pin	uP Signal	Jumper
2	SCK	J6(7->8)
3	MOSI	J3(7->8)
4	MISO	J2(7->8)

I2C Bus :

J1 pin	uP Signal	Jumper
2	SCL	J6(5->6)
3	SDA	J3(5->6)

Timers:

J1 pin	uP Analog Channel	Jumper
4	TIMER2	J3(3->4)
3	TIMER3	J2(3->4)

11.2 Sensor Power Options:

	Short	Open	Notes
VCC_BOOST from VCC_EKO	J25(2->1)	J24	eKo Node powers booster Standard Config
VCC_BOOST from External Voltage	J25(2->3)	J24	External supply powers booster
VCC_LDO from VCC_EKO	J26(2->1)		Source for LDO output Standard Config
VCC_LDO from VCC_BOOST	J26(2->3)		13.V max
VCC_SENSOR from VCC_EKO	J18	J27	
VCC_SENSOR from VCC_BOOST	J27(2->3)	J18	Standard Config

VCC_SENSOR from VCC_LDO	J27(1->3)	J18	

Crossbow Technology, Inc.
4145 N. First Street
San Jose, CA 95134
Phone: 408.965.3300
Fax: 408.324.4840
Email: info@xbow.com
Website: www.xbow.com