Sixnet RAM6K series SDK

Revision 2

July 9, 2012

Sixnet Inc. ● 331 Ushers Road ● Ballston Lake, NY 12019 ● USA
T +1 518 877 5173 ● F +1 518 877 8346 ● www.sixnet.com

Revision History

Revision 2 - July 9, 2012

- Emphasized RAM6K features
- jbmupdate is now snupdate
- Application reflash persisitance has been disabled by default
- Added instructions for managing SDK application with the GAU

Revision 1 - April 5, 2012

- Added instructions for using VirtualBox SDK

Sixnet Inc. ● 331 Ushers Road ● Ballston Lake, NY 12019 ● USA
T +1 518 877 5173 ● F +1 518 877 8346 ● www.sixnet.com

Contents

1 Overview

This technote describes how to set up a toolchain to cross-compile code which interfaces with the RAM6K series hardware and runs natively on that platform.

1.1 Supported Platforms

1.1.1 Build Host

Only Linux build hosts are supported. Any major modern Linux distribution which supports rpm files, either natively or through a translation program, should work. At time of writing, this includes:

- Fedora 14
- Ubuntu 10.04
- OpenSuse 11.3

1.1.2 Target Hardware

The software produced using this toolchain and libraries is supported on Sixnet SN series hardware.

2 SDK Installation

The toolchain and bootstrap environment have all been packaged and preconfigured in a virtual machine. See the VirtualBox SDK Environment section for more information.

2.1 ELDK Toolchain Installation

2.1.1 Overview

Software development for the SN series products utilizes the DENX Embedded Linux Development Kit (ELDK), version 4.2, for ARM. Full documentation for this toolchain can be accessed at:

```
http://www.denx.de/wiki/DULG/ELDK
```

In order to not swamp DENX's bandwidth, however, Sixnet maintains a mirror of the toolchains we use at our public FTP server:

```
ftp://ftp.sixnet.com/eldk/arm-2008-11-24.iso
```

2.1.2 Requirements

1. A relatively modern Linux distribution
2. The `rpm` command installed on your distribution. To check, execute `which rpm` in a terminal. If you get a reply like `/usr/bin/rpm`, then you have it installed. If not, please consult your distribution's documentation on how to install it.
3. At least 2.5 GB of free space.

2.1.3 Installation Instructions

Note: By default, the Makefile in the bootstrap build environment will expect the toolchain to be installed into `/opt/sixnet/eldk4.2`. If you install to this location, you won't have to change anything in the example. If you do not, you will have to change the `TOOLCHAIN_PATH` export in Makefile.

1. Download the iso from the following URL:

   ```
   ftp://ftp.sixnet.com/eldk/arm-2008-11-24.iso
   ```

2. Either burn the .iso to a CD with your favorite application or loopback mount the .iso as follows:

   ```
   sudo mkdir /media/iso
   sudo mount -o loop arm-2008-11-24.iso /media/iso/
   ```

   Then run the installer, substituting values for the placeholders (in < >'s) as appropriate:

   ```
   mkdir <target directory>
   /media/iso/install -d <target directory>
   ```

   Once installed, unmount the .iso and remove the directory we created:

   ```
   umount /media/iso
   rmdir /media/iso
   ```

2.2 Supplemental Library Installation

2.2.1 Overview

The libraries in the supplemental tarball are supplied by Sixnet to ease third-party development of applications which run on the SN series hardware. They are supplied as a tarball which is designed to be an "overlay" over the toolchain.

2.2.2 Installation

Untar the library tarball over the installed target directory. Note that `<target directory>` here should be the same as where you installed the ELDK.

```
tar -zxf ram6ksdk.<version>.tgz -C <target directory>
```

2.3 Bootstrap Build Environment

2.3.1 Overview

The bootstrap build environment is provided as a convenience to users who wish to get up and running quickly. It contains a sandbox folder with scripts for streamlining compiling and packaging a program, and a folder of reference examples/documentation.

2.3.2 Installation

Untar the tarball to some convenient spot in which you would like to develop code. For example:

```
tar -zxf ram6ksdkbootstrap.tar.gz
```

Will untar it to the current directory.

2.3.3 Usage

1. Place all of your source code in the `sandbox/source` folder. By default, all `.c` files will be included in the compile.
2. Configure the build environment for your program and create accessory files by executing the following from the sandbox directory in terminal.

```
./configure.sh
```

   This script will create a Makefile in the sandbox directory, as well as sdk.init , sdk.conf, and install.sh in the accessory folder.

3. Compile your program and create a .zip package file with:

```
make
make install
```

- The configure script copies default files for your program and modifies `install.sh` if you specify whether your program should start when the device boots and/or persist through reflashing. To bypass this script, copy the default files needed for package installation like so:

```
cp defaults/Makefile .
cp defaults/sdk.init accessory/
cp defaults/sdk.conf accessory/
cp defaults/install.sh accessory/
```

- `sdk.conf` is a configuration file that you can edit through the GAU Sub-systems menu in Expert Mode. The configure script creates a default in the accessory folder. You can rewrite this file to be used by your program. On the device, this file will be located here:

```
/etc/jbm/sdk.conf
```

  sdk.conf is format independent and it is up to the SDK user to implement a compatible reader for any selected content.

  See "Managing the Example Program" for more information on using this file with the GAU

Note: See `iodbTemplate.c` in the reference directory for example usage of the `daemon()` function to add an application debug mode. Daemonizing will redirect all standard output and errors to `/dev/null` as is the default behavior based on the sdk.init script. In the example, this is overridden by the `-d` command-line option.

2.4 VirtualBox SDK Environment

As an alternative to setting up your own environment, the toolchain, supplemental SDK libraries and bootstrap have been installed and preconfigured in a virtual machine.

2.4.1 VirtualBox Installation and Setup

This has been tested with VirtualBox version 4.1.10.

1. Download the platform package from the following URL:

   ```
   https://www.virtualbox.org/wiki/Downloads
   ```

   While not required, it is recommended to also download the VirtualBox Extension Pack. This will allow for sharing folders between guest and host.

2. Download the latest version of the virtual machine from here:

   ```
   http://susestudio.com/u/sixnet
   ```

   extract using your favorite archive program or `tar -zxf <filename>.tar.gz` from a linux terminal

3. Install the VirtualBox platform package using default settings. Once installed, the extension pack can be applied through virtualbox itself:

   (a) Select Preferences from the File menu in VirtualBox

   (b) Choose Extensions from the left sidebar

   (c) Click the icon on the right to Add Package, browse to the extension file you downloaded and open it.

4. Create a new virtual machine in VirtualBox as follows:

   (a) Select "New" from the main window and hit next.

   (b) Give a descriptive name to your virtual machine. The operating system is Linux and the Version is openSUSE (NOT the 64 bit option). Click Next

   (c) Leave 512 as the amount of memory. Click Next

   (d) For Virtual Hard Disk, select "Use existing hard disk," then click the icon and browse to the .vmdx file extracted from the tar.gz archive and open it. Click next, then create.

   You can now boot into the virtual machine using the Start arrow in the VirtualBox Manager window.

2.4.2 Using the Virtual Machine

- The bootstrap described above has been extracted to the desktop of the VM.
- To import source code to the VM, you can use a usb drive, download it from a local source using firefox, or use shared folders if you have installed guest additions.
- Packages can be installed directly to a device through firefox.
- For administrating the VM, root's password is gateway

2.4.3 VirtualBox Guest Additions

To enable file sharing between the host and guest systems, this must be installed on the guest system.

1. Boot into the VM and log into a terminal as root with command `su` password `gateway`
2. Select "Install Guest Additions" from the "Devices" menu on the VM's window. This should mount a drive containing installation scripts in `/media`

   If the drive isn't mounted automatically, execute the following:

   ```
   mkdir /media/temp
   mount /dev/scd0 /media/temp
   ```

3. `cd` to the mounted drive and run the following script:

   ```
   ./VBoxLinuxAdditions.run
   ```

4. Add user sixnet to the vboxsf group with the following command:

   ```
   sed -i s/vboxsf\:\!\:1000\:/vboxsf\:\!\:1000\:sixnet/ /etc/group
   ```

   This will append sixnet to the vboxsf group in the /etc/group file.

5. Reboot the virtual machine.
6. To share a folder with the virtual machine, select "Shared Folders" from the "Devices" menu of the virtual machine, then click the icon on the left to add a folder. Make sure "Auto-mount" and "Make permanent" are checked. The folder will show up in /media on the guest. You may have to reboot the VM.

3 SDK Library Function Calls

3.1 IODB

3.1.1 Overview

These library functions are used within applications to exchange I/O with a RAM6K station's I/O registers. Upcoming is a list of the function prototypes. Each function topic includes a C example.

Refer to "Ram6kHwSdk.pdf" for more information on using these library calls.

Refer to the supplied sample program, iodbEx.c, which uses these library function calls.

3.1.2 Modules

Here is a list of all modules:

3.1.3 I/O Updating Functions

These functions are used to read and write station I/O registers.

3.1.3.1 IODBRead

Use this function to:

Read one or more consecutive values from the I/O database based on type number and address.

Arguments:

`TypeNum`	in	Register type
`Addr`	in	Starting address
`NumRegs`	in	Number of registers to read
`pbuff`	out	Buffer to read data into
`StaName`		Not used. Pass NULL.

Return Values:

`ENOERROR`	on success
`EOUTOFRANGE`	if Addr+NumRegs exceeds values vailable for TypeNum
`ENOSTATION`	if Addr or TypeNum is invalid for station
`EINVALIDTYPE`	if TypeNum is invalid
`EDBERROR`	if database cannot be opened.

Prototype:

```
IODBerr IODBRead (
USHORT      TypeNum,
USHORT      Addr,
USHORT      NumRegs,
void       *pbuff,
const char *StaName
);
```

Examples:

iodbEx.c.

3.1.3.2 IODBReadTag

Use this function to:

Read one or more consecutive values from the I/O database based on register info defined by a tag.

Arguments:

`TagName`	in	Specifies the Sixtag name
`pbuf`	out	Buffer to read the data into
`BufSize`	in	Size of return buffer
`DataType`	out	Data type associated with sixtag
`NumRegs`	out	Number of registers associated with tag.

Return Values:

`ENOERROR`	on success
`ETAGNOTFOUND`	if TagName isn't found
`EDUPLICATETAG`	is TagName has multiple definitions
`ENOSTATION`	if TagName has invalid address information for station
`EOUTOFMEMORY`	if TagName is too long

Prototype:

```
IODBerr IODBReadTag (
const char *TagName,
void       *pbuf,
USHORT     *BufSize,
USHORT     *DataType,
USHORT     *NumRegs
);
```

Examples:

iodbEx.c.

3.1.3.3 IODBSetTag

Use this function to:

Create a tag in the tags database.

This tag is associated with the register type, starting address and number of registers.

Arguments:

`TagName`	in	Tag to be associted with register
`TypeNum`	in	Register type
`Addr`	in	Starting address
`NumRegs`	in	Number of registers of this type.

Return Values:

`ENOERROR`	on success.
`EOUTOFMEMORY`	if TagName is too long or NumReg <= 0
`EOUTOFRANGE`	if Addr+NumRegs exceeds values vailable for TypeNum
`EINVALIDTYPE`	if TypeNum is invalid

Prototype:

```
IODBerr IODBSetTag (
const char *TagName,
USHORT      TypeNum,
USHORT      Addr,
USHORT      NumRegs
);
```

Examples:

iodbEx.c.

3.1.3.4 IODBWrite

Use this function to:

Write one or more consecutive I/O values into the I/O database based on type number and address.

Arguments:

`TypeNum`	in	Register type
`Addr`	in	Starting address
`NumRegs`	in	Number of registers to write
`pbuff`	in	Buffer containing data to write
`StaName`	in	Not used. Pass NULL.

Return Values:

`ENOERROR`	on success
`EOUTOFRANGE`	if Addr+NumRegs exceeds values available for TypeNum
`ENOSTATION`	if Addr or TypeNum is invalid for station
`EINVALIDTYPE`	if TypeNum is invalid
`EDBERROR`	if cannot open database for write.

Prototype:

```
IODBerr IODBWrite (
USHORT      TypeNum,
USHORT      Addr,
USHORT      NumRegs,
void       *pbuff,
const char *StaName
);
```

Examples:

iodbEx.c.

3.1.3.5 IODBWriteMask

Use this function to:

Write selected I/O values into I/O database.

The bit mask is used to indicate whether an I/O register is to be written to or not. A 0 indicates no, a 1 indicates yes.

Arguments:

`TypeNum`	in	Register type
`Addr`	in	Starting address
`NumRegs`	in	Number of registers to write
`pbuff`	in	Buffer containing data to write
`pMask`	in	Controls which values to write
`StaName`	in	Not used. Pass NULL.

`pMask` specifies the write mask (a 1 indicates the point will be updated, and a 0 indicates the point will remain unchanged). The bit order of the mask is as follows: the least-significant bit of the first byte of `pMask` corresponds to the register at address `Addr`. The bit positions continue in ascending order to the most-significant bit and then continues on to the least-significant-bit of the next byte.

Return Values:

`ENOERROR`	on success
`EOUTOFRANGE`	if Addr+NumRegs exceeds values vailable for TypeNum
`ENOSTATION`	if Addr or TypeNum is invalid for station
`EINVALIDTYPE`	if TypeNum is invalid
`EDBERROR`	if cannot open database for write.

Prototype:

```
IODBerr IODBWriteMask (
uint16_t    TypeNum,
uint16_t    Addr,
uint16_t    NumRegs,
void       *pbuff,
uint8_t    *pMask,
const char *StaName
);
```

3.1.3.6 IODBWriteTag

Use this function to:

Write one or more consecutive values to the I/O database based on the Sixtag.

Arguments:

`TagName`	in	Sixtag name
`pbuf`	in	Buffer containing data to write

Return Values:

`ENOERROR`	on success
`ETAGNOTFOUND`	if TagName isn't found
`EDUPLICATETAG`	is TagName has multiple definitions
`ENOSTATION`	if TagName has invalid address information for station
`EOUTOFMEMORY`	if TagName is too long
`-1`	if TagName or pbuf is NULL.

Prototype:

```
IODBerr IODBWriteTag (
const char *TagName,
void       *pbuf
);
```

Examples:

iodbEx.c.

3.1.4 Advanced Driver Functions

These functions get information about the station and its configuration.

3.1.4.1 IODBGetFile

Use this function to:

Retrieve information about the current project file associated with a RAM.

Arguments:

`PathName`	out	Full path name of project file
`ConfigName`	out	IODB configuration name
`ConfigDate`	out	Time of last modification

Prototype:

```
void IODBGetFile (
char   *PathName,
char   *ConfigName,
time_t *ConfigDate
);
```

3.1.4.2 IODBGetNextTag

Use this function to:

Query the IODB for information about the next valid I/O Tag Name (Sixtag) in the database.

Successive calls to this function will return a list of all valid Sixtags.

Arguments:

`ListPos`	in/out	Current position in tags db, -1 to start from beginning.
`matc h`	in	Only return tags that contain this match string. Pass NULL to get all tags.
`TagName`	out	Tag name found
`StaName`	Not used. Pass NULL.	
`TypeNum`	out	Register type for TagName
`DataType`	out	Discrete, analog, long or float
`TypeSize`	out	Size of data type
`Addr`	out	Starting address

TagName, TypeNum, DataType, TypeSize, and Addr may be NULL if that information is not needed.

Return Values:

`ENOERROR`	on success.
`ETAGNOTFOUND`	at the end of the database (no more tags)

Prototype:

```
IODBerr IODBGetNextTag (
short  *ListPos,
char   *match,
char   *TagName,
char   *StaName,
USHORT *TypeNum,
USHORT *DataType,
USHORT *TypeSize,
USHORT *Addr
);
```

3.1.4.3 IODBGetNextType

Use this function to:

Query the I/O database for information about the next valid I/O type in the configuration.

Successive calls to this function will return a list of all valid I/O types.

Gets I/O type information identified by the next valid type following ListPos. Use this function by setting ListPos to -1 the first time you call it. For successive calls, pass in the ListPos that is returned from the previous call.

Arguments:

`ListPos`	in/out	Current position going through list of types. -1 to start from beginning.
`TypeNum`	out	Register type
`DataType`	out	Discrete, analog, float, long
`TypeSize`	out	Size in bytes for data, except discetes are 0
`NumRegs`	out	Number of allocated register for this type
`TypeName`	out	Description of type
`StaName`		Not used. Pass NULL.

Return Values:

`ENOERROR`	on success
`EINVALIDTYPE`	if ListPos is invalid.

Prototype:

```
IODBerr IODBGetNextType (
short       *ListPos,
USHORT      *TypeNum,
USHORT      *DataType,
USHORT      *TypeSize,
USHORT      *NumRegs,
char        *TypeName,
const char *StaName
);
```

3.1.4.4 IODBGetTag

Use this function to:

Query the I/O database for register information associated with the supplied Sixtag (I/O tag).

Arguments:

`TagName`	in	Sixtag name or address string (e.g.,"X1","AY23")
`StaName`		Not used. Pass NULL.
`TypeNum`	out	Register type
`DataType`	out	Discrete, analog, long or float
`TypeSize`	out	Bytes in an element of this type
`NumRegs`	out	Number of register associated with tag
`Addr`	out	Starting address

TypeNum, DataType, TypeSize, NumRegs, and Addr may be NULL if that information is not needed.

Return Values:

`ENOERROR`	on success
`ETAGNOTFOUND`	if TagName isn't found
`EOUTOFMEMORY`	if TagName is too long
`-1`	if TagName is NULL.

On success, if DataType is not NULL, it will point to one of the following values (defined in iodb.h):

```
ANALOGtype
DISCRETEtype
BYTEtype
LONGtype
FLOATtype
DOUBLEtype
USERtype
```

and TypeSize (if not NULL) will point to one of the following values (based on DataType):

`2`	(Analog)
`0`	(Discrete)
`1`	(Byte)
`4`	(Long)
`4`	(Float)
`8`	(Double)

1-220 (User-Type)

Prototype:

```
IODBerr IODBGetTag (
const char *TagName,
char       *StaName,
USHORT     *TypeNum,
USHORT     *DataType,
USHORT     *TypeSize,
USHORT     *NumRegs,
USHORT     *Addr
);
```

Examples:

iodbEx.c.

3.1.4.5 IODBGetType

Use this function to:

Query I/O database about a particular I/O type.

Arguments:

`TypeNum`	in	Register type
`DataType`	out	Discrete, analog, long or float
`TypeSize`	out	Bytes in registers of this type
`NumRegs`	out	Number of registers allocated of this type
`TypeName`	out	String indicating type of I/O

DataType, TypeSize, NumRegs, and TypeName may be NULL if that information is not needed.

Return Values:

`ENOERROR`	on success
`EINVALIDTYPE`	if TypeNum is invalid
`ETAGNOTFOUND`	if TagName isn't found
`EOUTOFMEMORY`	if TagName is too long
`EDBERROR`	if cannot open the database

On success, if DataType is not NULL, it will point to one of the following values (defined in iodb.h):

```
ANALOGtype
DISCRETEtype
BYTEtype
LONGtype
FLOATtype
DOUBLEtype
USERtype
```

and TypeSize (if not NULL) will point to one of the following values (based on DataType):

`2`	(Analog)
`0`	(Discrete)
`1`	(Byte)
`4`	(Long)
`4`	(Float)
`8`	(Double)
`1-220`	(User-Type)

Prototype:

```
IODBerr IODBGetType (
USHORT  TypeNum,
USHORT *DataType,
USHORT *TypeSize,
USHORT *NumRegs,
char   *TypeName
);
```

3.1.4.6 IODBGetTypeRange

Use this function to:

Query the I/O database about the configuration range (first and last available) of an I/O type.

Arguments:

`TypeNum`	in	Register type
`StaName`		Not used. Pass NULL.
`MinAddr`	out	First register in type. Always 0.
`MaxAddr`	out	Number of registers of this type.

Return Values:

`ENOERROR`	on success
`EINVALIDTYPE`	if TypeNum is invalid
`EDBERROR`	if cannot open the database.

Prototype:

```
IODBerr IODBGetTypeRange (
USHORT      TypeNum,
const char *StaName,
USHORT     *MinAddr,
USHORT     *MaxAddr
);
```

3.1.4.7 IODBVersion

Use this function to:

Retrieve the current version of the DLL running for this library.

Return Values:

Library version. The most significant byte is the major version. The least signficant byte is the minor version.

Prototype:

```
USHORT IODBVersion (void);
```

3.1.5 I/O Status Functions

These functions get information about registers and tags.

3.1.5.1 IODBGetDescription

Use this function to:

Get the user defined description (comment text) of an I/O register in a RAM station.

The description is defined using the I/O Toolkit.

Arguments:

`StaName`		Not used. Pass NULL.
`TypeNum`	in	Register type
`Addr`	in	Register address
`Description`	out	User-defined description. May be 40 chars + NUL.

Return Values:

`ENOERROR`	on success.
`ETAGNOTFOUND`	if tag isn't found.
`EINVALIDTYPE`	if TypeNum is invalid.
`EDBERROR`	if cannot open database.
`EFILEACCESS`	if cannot read tags.

Prototype:

```
IODBerr IODBGetDescription (
const char *StaName,
USHORT      TypeNum,
USHORT      Addr,
char       *Description
);
```

3.1.5.2 IODBGetFormat

Use this function to:

Get the integer format of an analog input register in a RAM station.

Defined in the I/O toolkit and stored in the associated project file.

Arguments:

`StaName`		Not used. Pass NULL.
`TypeNum`	in	Register type (must be analog type)
`Addr`	in	Register address
`Format`	out	NOFORMAT, SFORMAT, UFORMAT, LFORMAT

Return Values:

`ENOERROR`	on success.
`ENOTANALOGTYPE`	if TypeNum is isn't for an analog type
`EDBERROR`	if cannot open database.

Prototype:

```
IODBerr IODBGetFormat (
const char *StaName,
USHORT      TypeNum,
USHORT      Addr,
BYTE       *Format
);
```

3.1.5.3 IODBMinMax

Use this function to:

Get the engineering units and the raw and scaled min and max of an I/O register based on I/O type, and register address.

Arguments:

StaName		Not used. Pass NULL.
TypeNum	in	Register type
Addr	in	Register address
RawMin	out	Minimum for raw readings
RawMax	out	Maximum for raw readings
ScaleMin	out	Minimum for scaled values
ScaleMax	out	Maximum for scaled values
Units	out	Units string

RawMin, Rawmax, ScaleMin, ScaleMax and Units may be NULL if that information is not needed.

Return Values:

ENOERROR	on success
EINVALIDTYPE	if TypeNum is invalid
EDBERROR	if cannot open database
EFILEACCESS	if tags cannot be read
EREGISTERNOTFOUND	if register isn't found

Prototype:

```
IODBerr IODBMinMax (
const char *StaName,
USHORT      TypeNum,
USHORT      Addr,
long       *RawMin,
long       *RawMax,
double     *ScaleMin,
double     *ScaleMax,
char       *Units
);
```

3.1.5.4 IODBMinMaxTag

Use this function to:

Get the engineering units and the raw and scaled min and max of an I/O register based on station name, I/O type, and register address.

Arguments:

`StaName`		Not used. Pass NULL.
`TagName`	in	Tag name to look up
`RawMin`	out	Minimum for raw readings
`RawMax`	out	Maximum for raw readings
`ScaleMin`	out	Minimum for scaled values
`ScaleMax`	out	Maximum for scaled values
`Units`	out	Units string

RawMin, Rawmax, ScaleMin, ScaleMax and Units may be NULL if that information is not needed.

Return Values:

`ENOERROR`	on success.
`EOUTOFMEMORY`	if TagName is too long
`EFILEACCESS`	if Tag file cannot be read
`ETAGNOTFOUND`	if TagName doesn't exist

Prototype:

```
IODBerr IODBMinMaxTag (
const char *TagName,
long       *RawMin,
long       *RawMax,
double     *ScaleMin,
double     *ScaleMax,
char       *Units
);
```

3.1.5.5 IODBScale

Use this function to:

Get the engineering units and conversion factor for an I/O register.

Arguments:

`StaName`		Not used. Pass NULL.
`TypeNum`	in	I/O type number
`Addr`	in	Register address
`Span`	out	Span correction factor
`Offset`	out	Offset correction factor
`Units`	out	Units string

Return Values:

`ENOERROR`	on success
`EDBERROR`	if the database cannot be opened
`ENOTANALOGTYPE`	if TypeNum is not an analog type
`EREGISTERNOTFOUND`	if the register is not found
`EFILENOTFOUND`	if the project file is not found
`EFILEACCESS`	if the project file cannot be accessed
`ENOSIXNETFILE`	if the file does not support SIXNET configuration

Prototype:

```
IODBerr IODBScale (
const char *StaName,
USHORT      TypeNum,
USHORT      Addr,
double     *Span,
double     *Offset,
char       *Units
);
```

3.1.5.6 IODBScaleTag

Use this function to:

Get the engineering units and conversion factor for an I/O register.

Arguments:

`StaName`		Not used. Pass NULL.
`TagName`	in	Tag name to look up
`Span`	out	Span correction factor
`Offset`	out	Offset correction factor
`Units`	out	Units string

Return Values:

`ENOERROR`	on success
`EOUTOFMEMORY`	if TagName is too long
`-1`	if the database cannot be opened
`ENOTANALOGTYPE`	if TagName does not refer to an analog type
`EDUPLICATETAG`	if TagName is not unique
`ETAGNOTFOUND`	if TagName is not found
`EFILENOTFOUND`	if the project file is not found
`EFILEACCESS`	if the project file cannot be accessed
`ENOSIXNETFILE`	if the file does not support SIXNET configuration

Prototype:

```
IODBerr IODBScaleTag (
const char *TagName,
double     *Span,
double     *Offset,
char       *Units
);
```

3.1.6 File List

Here is a list of all documented files with brief descriptions:

3.1.6.1 iodb.h File Reference

Declaration of SIXNET IODB functions.

```
#include "utypes.h"

#include <time.h>
```

Defines

- #define **IODB_VERSION** 0x0100
- #define **EDBERROR** -1
- #define **ENOERROR** 0
- #define **EOUTOFRANGE** 1
- #define **EINVALIDTYPE** 2
- #define **ENOSTATION** 3
- #define **EFILEACCESS** 4
- #define **EFILENOTFOUND** 5
- #define **EIODBNOTFOUND** 6
- #define **EOUTOFMEMORY** 7
- #define **ETASKERROR** 8
- #define **ENONSIXNETFILE** 9
- #define **ETAGNOTFOUND** 10
- #define **EDUPLICATETAG** 11
- #define **EBUFFERSIZE** 12
- #define **ENOTANALOGTYPE** 13
- #define **EREGISTERNOTFOUND** 14
- #define **EEXCEEDEDMAXTAGS** 15
- #define **NOFORMAT** 1
- #define **SFORMAT** 2
- #define **UFORMAT** 3
- #define **LFORMAT** 4
- #define **ULFORMAT** 5
- #define **RESERVEDtype** 0
- #define **ANALOGtype** 1
- #define **DISCRETEtype** 2
- #define **BYTEtype** 3
- #define **LONGtype** 4
- #define **FLOATtype** 5
- #define **DOUBLEtype** 6

- #define **USERtype** 7
- #define **READSCAN** 0
- #define **WRITESCAN** 1
- #define **WRITEREAD** 2
- #define **EXCEPTION** 3
- #define **ASSIGNED** 4
- #define **SIXTRAK** 0
- #define **IOMUX** 1
- #define **VERSAMUX** 2
- #define **S60IBMN** 3
- #define **LOCALCOMP** 4
- #define **IODB** 5
- #define **MISCTYPE** 6
- #define **VERSATRAK** 7
- #define **REMOTETRAK** 8
- #define **ETHERTRAK** 9
- #define **SYSPLAN_PROG** 0
- #define **SPF_PROG** 1
- #define **SCR_PROG** 2
- #define **SCRADV_PROG** 3
- #define **SCR_1131** 4
- #define **SCR_1131_ADV** 5
- #define **NO_LICENSE** -1
- #define **TEMPORARY_LICENSE** 0
- #define **DEMO_LICENSE** 1
- #define **SINGLE_LICENSE** 2
- #define **SITE_LICENSE** 3
- #define **OEM_LICENSE** 4
- #define **BYTE** unsigned char
- #define **USHORT** unsigned short
- #define **IODBerr** unsigned short

Functions

- IODBerr **IODBCleanup** (void)
- IODBerr IODBSetTag (const char *, USHORT, USHORT, USHORT)

  Create a tag in the tags database.

- IODBerr IODBRead (USHORT, USHORT, USHORT, void *, const char *)

Read one or more consecutive values from the I/O database based on type number and address.

- IODBerr IODBReadTag (const char *, void *, USHORT, USHORT *, USHORT *)

  Read one or more consecutive values from the I/O database based on register info defined by a tag.

- IODBerr IODBWrite (USHORT, USHORT, USHORT, void *, const char *)

  Write one or more consecutive I/O values into the I/O database based on type number and address.

- IODBerr **IODBWriteMask** (USHORT, USHORT, USHORT, void *, BYTE *, const char *)
- IODBerr IODBWriteTag (const char *, void *)

  Write one or more consecutive values to the I/O database based on the Sixtag.

- IODBerr IODBGetNextTag (short *, char *, char *, char *, USHORT *, USHORT *, USHORT *, USHORT *)

  Query the IODB for information about the next valid I/O Tag Name (Sixtag) in the database.

- IODBerr IODBGetDescription (const char *, USHORT, USHORT, char *)

  Get the user defined description (comment text) of an I/O register in a RAM station.

- IODBerr IODBGetFormat (const char *, USHORT, USHORT, BYTE *)

  Get the integer format of an analog input register in a RAM station.

- IODBerr IODBMinMax (const char *, USHORT, USHORT, long *, long *, double *, double *, char *)

  Get the engineering units and the raw and scaled min and max of an I/O register based on I/O type, and register address.

- IODBerr IODBMinMaxTag (const char *, long *, long *, double *, double *, char *)

  Get the engineering units and the raw and scaled min and max of an I/O register based on station name, I/O type, and register address.

- IODBerr IODBScale (const char *, USHORT, USHORT, double *, double *, char *)

  Get the engineering units and conversion factor for an I/O register.

- IODBerr IODBScaleTag (const char *, double *, double *, char *)

  Get the engineering units and conversion factor for an I/O register.

- void IODBGetFile (char *, char *, time_t *)

  Retrieve information about the current project file associated with a RAM.

- IODBerr IODBGetNextType (short *, USHORT *, USHORT *, USHORT *, USHORT *, char *, const char *)

Query the I/O database for information about the next valid I/O type in the configuration.

- IODBerr IODBGetTag (const char *, char *, USHORT *, USHORT *, USHORT *, USHORT *, USHORT *)

  Query the I/O database for register information associated with the supplied Sixtag (I/O tag).

- IODBerr IODBGetType (USHORT, USHORT *, USHORT *, USHORT *, char *)

  Query I/O database about a particular I/O type.

- IODBerr IODBGetTypeRange (USHORT, const char *, USHORT *, USHORT *)

  Query the I/O database about the configuration range (first and last available) of an I/O type.

- USHORT IODBVersion (void)

  Retrieve the current version of the DLL running for this library.

3.1.6.2 Detailed Description

Declaration of SIXNET IODB functions. Copyright 2009 SIXNET, LLC. All rights reserved.

3.1.7 Examples

Here is a list of all examples:

- iodbEx.c

3.1.7.1 iodbEx.c

This code is written to give an example or two on how to access the SIXNET I/O database from external OEM applications. NOTE: Discretes are bitpacked. 8 bits per byte. Floats are treated as floats (4 bytes), not doubles. Longs are treated as longs (4 bytes).

```
/*
 * Example implementation of iodb functions
 *
 * Copyright 2012 SIXNET, LLC.  All rights reserved.
 */
/* -*- mode:c;  c-basic-offset: 4; tab-width: 8; -*- vi: set sw=4 ts
   =8:*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <iodb.h>

/* Types */
#define INT16   short
#define INT32   long
#define UINT16  unsigned short
#define UINT32  unsigned long
#define ULONG   unsigned long
#define BYTE    unsigned char
#define BOOL    int

/* IODB Types */
#define ANALOG_IN       0
#define ANALOG_OUT      1
#define DISCRETE_IN     10
#define DISCRETE_OUT    11
#define LONG_IN         20
#define LONG_OUT        21
#define FLOAT_IN        22
#define FLOAT_OUT       23

int main()
{
    int i;
    UINT16 addr;          /* Starting Register */
    UINT16 regs;          /* Number of Register */
    UINT16 datatype;      /* Type of I/O */
```

```
    UINT16 typesize;
    UINT16 typenum;
    INT16 ecode;
    INT16 a_value;
    INT16 analog_values[8];
    INT16 discrete_values;
    char tagname[32];

    /*
     * Write 1 analog value to analog output register 0.
     * Then read the value back in...
     */
    addr = 0;
    regs = 1;
    a_value = 37;
    printf("\nWriting 1 analog, Reading 1 back...\n");
    printf("a_value: %d\n", a_value);
    ecode = IODBWrite(ANALOG_OUT, addr, regs, (void *)&a_value, NULL);
    a_value = 0;
    ecode = IODBRead(ANALOG_OUT, addr, regs, (void *)&a_value, NULL);
    printf("a_value (returned): %d\n", a_value);

    /*
     * Write 3 analog values to 3 consecutive registers,
     * starting at analog output register 1.
     * Then read the values back from register 0.
     */
    addr = 1;
    regs = 3;
    analog_values[0] = 11;
    analog_values[1] = 22;
    analog_values[2] = 33;
    printf("\nWriting 3 analogs, Reading 4 back...\n");
    for (i = 0; i < regs; i++) {
        printf("analog_values[%d]: %d\n", i, analog_values[i]);
    }
    ecode = IODBWrite(ANALOG_OUT, addr, regs, (void *)&analog_values,
       NULL);

    memset((void *)&analog_values, '0', sizeof(INT16) * 8);
    addr = 0;
    regs = 4;
    ecode = IODBRead(ANALOG_OUT, addr, regs, (void *)&analog_values,
       NULL);
```

```
    for (i = 0; i < regs; i++) {
        printf("analog_values[%d]: %d\n", i, analog_values[i]);
    }

    /*
     * Write 1 discrete value to discrete output register 0.
     * Then read the value back in...
     */
    addr = 0;
    regs = 1;
    discrete_values = 1;  /* Turning discrete ON */
    printf("\nWriting 1 discrete, Reading 1 back...\n");
    printf("discrete_values: %d\n", discrete_values);
    ecode = IODBWrite(DISCRETE_OUT, addr, regs, (void *)&
       discrete_values, NULL);
    discrete_values = 0;
    ecode = IODBRead(DISCRETE_OUT, addr, regs, (void *)&discrete_values
       , NULL);
    printf("discrete_values (returned): %d\n", discrete_values);

    /*
     * Write 4 discrete value to discrete output register 4.
     * Then read 8 discretes back, starting at register 0...
     *    D0 = ON  (Previous Write)
     *    D4 = ON
     *    D6 = ON
     *    If nothing else is ON, should return 81.
     */
    addr = 4;
    regs = 4;
    discrete_values = 5;  /* D0 & D2 are on, D1 & D3 are off */
    printf("\nWriting 4 discretes, Reading 8 back...\n");
    printf("discrete_values: %d\n", discrete_values);
    ecode = IODBWrite(DISCRETE_OUT, addr, regs, (void *)&
       discrete_values, NULL);
    discrete_values = 0;
    addr = 0;
    regs = 8;
    ecode = IODBRead(DISCRETE_OUT, addr, regs, (void *)&discrete_values
       , NULL);
    printf("discrete_values (returned): %d\n", discrete_values);

    /*
     * Set a tag to correspond to an discrete output register
```

```
     */
    strcpy(tagname, "do_tag");
    addr = 2;  /* Start at register 2 */
    regs = 1;  /* Only one register */
    printf("\nAssigning tagname (%s) to discrete output register 2...\n
       ",
           tagname);
    ecode = IODBSetTag(tagname, DISCRETE_OUT, addr, regs);

    /*
     * Write one discrete output value to tagname.
     * Then read back the same discrete output value, via tagname.
     */
    discrete_values = 1;  /* Turning discrete ON */
    printf("\nWriting/Reading 1 discrete output values to/from %s...\n
       ",
           tagname);
    printf("discrete_values: %d\n", discrete_values);
    ecode = IODBWriteTag(tagname, (void *)&discrete_values);
    discrete_values = 0;
    ecode = IODBReadTag(tagname, (void *)&discrete_values,
                        sizeof(discrete_values), NULL, NULL);
    printf("discrete_values associated with %s (returned): %d\n",
           tagname, discrete_values);

    /*
     * Set a tag to correspond to a block of analog output registers
     */
    strcpy(tagname, "ao_tag");
    addr = 4;  /* Start at register 4 */
    regs = 4;  /* 4 consecutive registers */
    printf("\nAssigning tagname (%s) to analog output registers: 4, 5,
       6, 7\n",
           tagname);
    ecode = IODBSetTag(tagname, ANALOG_OUT, addr, regs);

    /*
     * Write 4 analogs output values to tagname
     * Then read back the same 4 analog output values, via tagname.
     */
    analog_values[0] = 9;
    analog_values[1] = 8;
    analog_values[2] = 7;
    analog_values[3] = 6;
```

```
    printf("\nWriting/Reading 4 analog output values to/from %s...\n",
           tagname);
    printf("Datatype & the numbers of registers will also be returned.\
       n");
    for (i = 0; i < 4; i++) {
        printf("analog_values[%d]: %d\n", i, analog_values[i]);
    }
    ecode = IODBWriteTag(tagname, (void *)&analog_values);
    memset((void *)&analog_values, '0', sizeof(INT16) * 8);
    ecode = IODBReadTag(tagname, (void *)&analog_values,
                        sizeof(analog_values), &datatype, &regs);
    printf("%d registers, of type (%d), are associated with %s...\n",
           regs, datatype, tagname);
    for (i = 0; i < regs; i++) {
        printf("analog_values[%d]: %d\n", i, analog_values[i]);
    }

    /*
     * Get information related to tagname
     * Any unwanted values can be NULL'ed out
     */
    printf("\nGetting information associated with %s...\n", tagname);
    ecode = IODBGetTag(tagname, NULL, &typenum, &datatype, &typesize,
                       &regs, &addr);
    printf("Retrieved the following information:\n");
    printf("\ttypenum: %d\n", typenum);
    printf("\tdatatype: %d\n", datatype);
    printf("\ttypesize: %d\n", typesize);
    printf("\tregs: %d\n", regs);
    printf("\taddr: %d\n\n", addr);

    exit (0);
}
```

4 Application Development Lifecycle

4.1 Building the Example Program

4.1.1 Overview

The example program is available as native C source code in the bootstrap environment, along with the necessary Makefile(s) and directory structure required to build a package for installation on your RAM6K series product.

To build the example program (iodbEx), simply navigate to the sandbox directory in the root of your bootstrap environment installation, and:

```
cp ../reference/iodbEx.c source
./configure.sh
```

Just hit enter at both prompts for defaults, then:

```
make
make install
```

`sdk.zip` will be created in the packages folder.

4.2 Installing the Example Program

4.2.1 Overview

Now that you have successfully compiled the example program (make) and built a package for installation on your RAM6K series product (make install), you need to install the package.

RAM6K series package files are simple .zip archives, containing full directory structures that will corespond to the installation location ono your RAM6K series product. Further, there is a special (optional) file that can control more advanced behavior during package installation.

4.2.2 Controlling Package Installations

Let's start by examining the contents of the example package archive:

```
Archive:  iodbEx.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2012-04-04 18:23   etc/
```

```
         0  2012-04-04 18:23   etc/jbm/
        27  2012-04-04 18:23   etc/jbm/sdk.conf
        27  2012-04-04 18:23   etc/jbm/sdk.conf.orig
         0  2012-04-04 18:23   etc/rc.d/
         0  2012-04-04 18:23   etc/rc.d/init.d/
       964  2012-04-04 18:23   etc/rc.d/init.d/sdk
      1159  2012-04-04 18:23   install.sh
         0  2012-04-04 18:23   usr/
         0  2012-04-04 18:23   usr/local/
         0  2012-04-04 18:23   usr/local/bin/
     11096  2012-04-04 18:23   usr/local/bin/sdk
---------                     -------
     13273                     12 files
```

Notice the file in the package manifest named install.sh. This bash script will be executed, if present, before any package installation is executed. The sample install.sh file included in the bootstrap environment is a straightforward example that simply extracts the contents of the archive to the RAM6K series product and logs a message to syslog.

Without this special file, the example program would simply be extracted from the archive by the package installer, with no additional logic.

Two lines are key in allowing advanced functionality for the program.

The following line enables the init script for the program meaning it will start whenever the device boots. Uncomment to enable this functionality.

```
#chkconfig sdk on
```

This line copies the package file into a special directory causing it to be reinstalled next time the device is flashed. Uncomment to enable.

```
#cp -f $FILE_NAME /storage/sdk/install/
```

WARNING: Configuring an untested package to install every time, or on bootup, could render your device inoperable. The SDK is a powerful tool. Please test extensively before applying permanent changes.

Also, see additional document snupdate.pdf for more information on creating package .zip files.

4.2.2.1 install.sh

```
#!/bin/sh

# Check that /etc/version exists. This pretty much only true on JBM/SN
   devices
if [ ! -e "/etc/version" ]; then
    echo "Unit does not appear to be a JBM/Sixnet unit, exiting."
    exit 1
fi

# The only argument we are passed is the file name of the package
FILE_NAME=$1

# Make sure we can find that file before we continue
if [ ! -e "$FILE_NAME" ]; then
    echo "Cannot find package file $FILE_NAME"
    exit 2
fi

# Extract the rest of the contents of the package
# -d             The destination directory for the unpacked contents
# -o             Overwrite existing files without prompting (use this!)
# $FILE_NAME     The file name of the package to unzip
# -x install.sh  Exclude install.sh when extracting since that has
   already been unpacked
unzip -d / -o $FILE_NAME -x install.sh

# Uncomment to turn on autostart of sdk process at boot time
#chkconfig sdk on

# Uncomment the following line to allow the program to persist across
   reflashing
#cp -f $FILE_NAME /storage/sdk/install/

# Now that the contents are extracted, we can do any other action the
   package needs to perform
logger -t $FILE_NAME "Initial installation..."


exit 0
```

4.3 Managing the Example Program

4.3.1 Installing the Example Program

There are three ways to install the example program package:

- **Sixview Manager** - Create a job and add the package file to it
- **RAM6K Series Product On-Board Web Interface** - Use the Admin->Package Installation page
- **Command Line Interface** - Upload to /tmp via TFTP, FTP, SCP or ZModem and install via:

```
[root@SNgateway-v3_09 tmp]# snupdate iodbEx.zip
snupdate v1.15
** Package contents appear valid.
** Running included installation script.
Archive:  /tmp/iodbEx.zip
  inflating: /usr/local/bin/iodbEx
  Install success
```

4.3.2 Example Application Persistence

In order to allow your application(s) to persist across firmware upgrades, they must be stored in a special filesystem and format on the RAM6K series product. The simplest way to do this is to move your tested installation package to the /storage/sdk/install directory on your RAM6K series product. This can be accomplished with:

```
cp -f /tmp/iodbEx.zip /storage/sdk/install/
```

This has been added to the install.sh script, and is disabled by default.

4.3.3 GAU Interaction with SDK Application

The SDK subsystem can be managed through the GAU in expert mode. This is available by going to Advanced > Expert Mode > Configure Sub-systems. Selecting "SDK" from the drop-down menu will display an editable configuration file and buttons for starting and stopping the application.

If the sdk application uses the configuration file, it can be modified here. A copy of the original file is created by the bootstrap when packaging the program, and can be restored by clicking the "Default" button.

See "Usage" of the Bootstrap Build Environment section for more information on creating the configuration file.

Index