GE VERNOVA

PROFICY® SOFTWARE & SERVICES

# PROFICY iFIX HMI/SCADA

## Simulation 2 Driver

GE VERNOVA

**Proprietary Notice**
The information contained in this publication is believed to be accurate and reliable. However, GE Vernova assumes no responsibilities for any errors, omissions or inaccuracies. Information contained in the publication is subject to change without notice.



**Trademark Notices**
"GE VERNOVA" is a registered trademark of GE Vernova. The terms "GE" and the GE Monogram are trademarks of the General Electric Company, and are used with permission.

Microsoft® is a registered trademark of Microsoft Corporation, in the United States and/or other countries.

All other trademarks are the property of their respective owners.

We want to hear from you. If you have any comments, questions, or suggestions about our documentation, send them to the following email address:
doc@ge.com

# Table of Contents

# Simulation 2 Driver

The Simulation 2 (SM2) driver provides a matrix of addresses that lets you test your process database to learn how its block and chains respond to different conditions. You can also use the SM2 driver with a C application to extract data from legacy systems and store it in a process database.

The SM2 driver has no driver configuration program so no configuration of the driver is required. Instead, you specify the SM2 address you want to use in a database block's I/O Address field and then place the block on scan. To learn more about using SM2 addresses, refer to Accessing SM2 Registers.

The SM2 driver supports the following HMI software products:

- FIX for Windows NT version 6.15 or greater.
- iFIX version 2.5 or greater.

# SM2 Driver Features

The SM2 driver is similar to the SIM driver supplied with your FIX and iFIX software. Both drivers:

- Provide a matrix of addresses that database blocks can read from and write to.
- Support analog and digital database blocks.
- Support text blocks.

However, the SM2 driver differs from the SIM driver in several important ways:

| The SM2 driver... | The SIM driver... |
|---|---|
| Provides three independent sets of registers. Analog blocks automatically access the analog registers, digital blocks automatically use the digital registers, and Text blocks automatically access the text registers. | Provides one set of registers shared by both analog, digital, and text blocks. |
| Changing a register in one set does not change the same register in the other set. For example, if you change the value of the analog register 1000, the value of the digital register 1000 is unchanged. | Changing an analog register in the SIM driver modifies the register for analog, digital, and text reads. For example, if you change the value of the analog register 1000, you also modify the value of the same digital register. |
| Provides 20,000 analog, 20,000 16-bit digital registers, and 20,000 text registers. | Provides 2000 analog and digital registers, a total of 32,000 bits. |
| Stores analog values in 64-bit floating point registers, numbered 0 to 19999. Incoming values are not scaled. | Stores analog values in 16-bit integer registers, numbered 0 to 2000. Incoming 32-bit values are scaled to 16-bit values (0 - 65535). |
| Digital values are stored in 16-bit integer registers, numbered 0 to 19999. | Digital values are stored in 16-bit integer registers, numbered 0 to 2000. |
| Text values are stored in 8-bit registers numbered 0 to 19999. Each register holds one text character for a total of 20,000 bytes of text. | Text values are stored in the same area as analog and digital values, numbered 0 to 2000. |
| Provides a register to simulate communication errors. | Cannot simulate communication errors. However, the SIM driver does provide registers RA through RK and RX through RZ to generate random numbers. For more information, refer to the Using Signal Generation Registers in the SIM Driver section Building a SCADA System electronic book. |
| Supplies a C API that allows applications to access SM2 analog, digital, and text values. | Does not support a C API for accessing SIM values. |
| Supports exception-based processing. | Does not support exception-based processing. |
| Supports latched data for Analog Input, Analog Alarm, Digital Input, Digital Alarm and Text blocks when a simulated com- | Does not support latched data. |

| | |
|---|---|
| munication error is enabled. | |
| Can read and write the individual alarm status of each SM2 register. | Cannot read and write the individual alarm status of any SIM register. |
| Does not provide alarm counters. | Provides alarm counters that show the general alarm state of a SCADA server. For more information, refer to the Using Alarm Counters chapter of the Implementing Alarms and Messages electronic book. |

Obviously, you can use the SIM driver for many of the same tasks as the SM2 driver. However, you may prefer to use the SM2 driver when one or more of the following conditions occur:

- You have more test data than the SIM driver can hold.
- You want to determine how the database responds to 32-bit values.
- You need to access the driver from a C program.

# Accessing SM2 Registers

The SM2 driver matrix consists of three independent sets of registers: one for analog values, one for digital values, and one for text values. Analog database blocks read from and write to analog registers only. Once a block writes a value, other analog blocks can read the value from the register written to. Digital database blocks work the same way, reading and writing from the digital registers. FIX (or iFIX) clears all SM2 values when FIX or iFIX starts.

The SM2 driver does not use the Hardware Options or Signal Conditioning fields.

### To use the SM2 register:

1. Enter SM2 in the primary block's Device field.
2. Complete the I/O Address field with the following syntax:

   For Analog values: *register*

   For Digital values: *register:bit*

   For Text values: *register*

**SM2 Address Examples**

| Analog Examples | Digital Examples | Text Examples |
|---|---|---|
| 1000 | 5000:10 | 2000 |
| 16435 | 23:15 | 10000 |

# Generating Bad Data

The SM2 driver provides an S register to simulate a communication error. Using this register, all analog and digital reads return an error as if communication to the process hardware has been lost.

To use this feature, set the S register to 1.

**NOTE:** The SM2 driver latches data when a simulated communication error is enabled.

# Using the SM2 C API

**NOTE:** You must have the iFIX Integration (EDA) Toolkit installed to use the C API.

You can access SM2 analog, digital, and text values through the C API that the driver supplies. The file SM2API.H describes the API and the functions reside in the file SM2API.LIB. You can link this library file to your C application to access the API's functions. You can find both files in your Base path. By default, this path is C:\Program Files (x86)\Proficy\iFIX, C:\IFIX, or C:\Dynamics depending where you installed iFIX.

## Example

Suppose you are using the SM2 driver to store data from a legacy system. Using the C API and a number of preconfigured analog blocks, you can extract your data from the legacy system and store it in your process database.

**C API Functions**

| Syntax | Values read, written, and returned |
|---|---|
| UINT16 GetAnalog(UINT16 index, FLOAT *data); | GetAnalog reads an analog value (32-bit float) to the register indicated by 'index'.<br>FE_OK is returned if the operation succeeds.<br>FE_IO_ADDR is returned if the register index is out of range.<br>FE_RANGE is returned if the analog value exceeds the range of a 32-bit float.<br>**NOTE:** GetAnalog and GetDouble access the same table in the SM2. |
| UINT16 SetAnalog(UINT16 index, FLOAT data); | SetAnalog writes an analog value (32-bit float) to the register indicated by 'index' and causes an exception for the specified register even if the data has not changed.<br>FE_OK is returned if the operation succeeds.<br>FE_IO_ADDR is returned if the register index is out of range.<br>**NOTE:** SetAnalog and SetDouble access the same table in the SM2. |
| UINT16 GetDouble (UINT16 index, DOUBLE *data); | GetDouble reads an analog value (64-bit float) to the register indicated by 'index.'<br>FE_OK is returned if the operation succeeds.<br>FE_IO_ADDR is returned if the register index is out of range.<br>**NOTE:** GetAnalog and GetDouble access the same table in the SM2. |
| UINT16 SetDouble (UINT16 index, DOUBLE data); | SetDouble writes an analog value (64-bit float) to the register indicated by 'index' and causes an exception for the specified register even if the data has not changed.<br>FE_OK is returned if the operation succeeds.<br>FE_IO_ADDR is returned if the register index is out of range.<br>**NOTE:** SetAnalog and SetDouble access the same table in the SM2. |
| UINT16 GetDigital(UINT16 index, UINT16 | GetDigital reads 16 digital values (all 16 bits in one of the 20,000 digital registers) to the register indicated by 'index'.<br>FE_OK is returned if the operation succeeds. |

| | |
|---|---|
| *data); | FE_IO_ADDR is returned if the register index is out of range.<br><br>**NOTE:** The API can only read and write the entire 16 bit digital register at one time. If you want to change 1 bit, you can read the register, modify the desired bit and write the register. However, when you modify a single bit, ensure that only one thread in one application is accessing a digital register at one time. |
| UINT16 SetDigital(UINT16 index, UINT16 data); | SetDigital writes 16 digital values (all 16 bits in one of the 20,000 digital registers) to the register indicated by 'index' and causes an exception for all 16 bits of the specified register even if the data has not changed.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range.<br><br>**NOTE:** The API can only read and write the entire 16 bit digital register at one time. If you want to change 1 bit, you can read the register, modify the desired bit and write the register. However, when you modify a single bit, ensure that only one thread in one application is accessing a digital register at one time. |
| UINT16 SetDigitalEx(UINT index, UINT16 data, UINT16 mask) | SetDigitalEx writes 16 digital values (all 16 bits in one of the 20,000 digital registers) to the register indicated by 'index' and causes an exception for specific bits selected from a mask. An exception is triggered for the bits set in the mask even if the data has not changed.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range.<br><br>**NOTE:** The API can only read and write the entire 16 bit digital register at one time. If you want to change 1 bit, you can read the register, modify the desired bit and write the register. However, when you modify a single bit, ensure that only one thread in one application is accessing a digital register at one time. |
| UINT16 GetText (UINT16 index, char *data, int size) | GetText reads the text specified by `data' from text registers starting at the register indicated by `index'. The number of characters to read is indicated by `size'. GetText does not automatically add a null terminator to the text being read. If you require null-terminated strings, make sure your program adds a null terminator after reading text.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range. |
| UINT16 SetText (UINT16 index, char *data, int size) | SetText writes the text specified by `data' to text registers starting at the register indicated by `index'. The number of characters to write is indicated by `size'. Exception-based processing is not supported for text values. SetText does not automatically add a null terminator to the text being written. If you require null-terminated strings, make sure your program adds a null terminator prior to writing the text.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range. |
| UINT16 GetCommError (UINT16 *data); | GetCommError reads the communication error flag to the S register. The flag is a 1 bit integer value.<br><br>FE_OK is returned always. |
| UINT16 | SetCommError writes the communication error flag to the S register. The flag is a 1- |

| | |
|---|---|
| SetCommError (UINT16 data); | bit integer value. You should only pass 0 or 1 to the SetCommError function. Using any other value can have unpredictable results.<br><br>FE_OK is returned always. |
| UINT16 GetAnalogAlarm (UINT16 index, INT16 *alm); | GetAnalogAlarm reads an alarm status from an analog register indicated by 'index'. To learn more about available alarm statuses, refer to Understanding Alarm Statuses.<br><br>**NOTE:** As of iFIX 4.5, only the alarm statuses IA_OK and IA_COMM are supported for use through the SM2 driver.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range. |
| UINT16 SetAnalogAlarm (UINT16 index, INT16 alm); | SetAnalogAlarm writes an alarm status to an analog register indicated by 'index' and causes an exception for the specified register even if the data or alarm has not changed. To learn more about available alarm statuses, refer to Understanding Alarm Statuses.<br><br>**NOTE:** As of iFIX 4.5, only the alarm statuses IA_OK and IA_COMM are supported for use through the SM2 driver.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range. |
| UINT16 GetDigitalAlarm (UINT16 index, INT16 *alm); | GetDigitalAlarm reads an alarm status from a digital register indicated by 'index'. To learn more about available alarm statuses, refer to Understanding Alarm Statuses.<br><br>**NOTE:** As of iFIX 4.5, only the alarm statuses IA_OK and IA_COMM are supported for use through the SM2 driver.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range. |
| UINT16 SetDigitalAlarm (UINT16 index, INT16 alm); | SetDigitalAlarm writes an alarm status to a digital register indicated by 'index' and causes an exception even if the data or alarm has not changed. The same alarm is associated with all 16 bits in the digital register. To learn more about available alarm statuses, refer to Understanding Alarm Statuses.<br><br>**NOTE:** As of iFIX 4.5, only the alarm statuses IA_OK and IA_COMM are supported for use through the SM2 driver.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range. |
| UINT16 GetTextAlarm(UINT16 index INT16 alm) | GetTextAlarm reads an alarm status from a text register indicated by `index'. To learn more about available alarm statuses, refer to Understanding Alarm Statuses.<br><br>**NOTE:** As of iFIX 4.5, only the alarm statuses IA_OK and IA_COMM are supported for use through the SM2 driver.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range. |
| UINT16 SetText- | SetTextAlarm writes an alarm status to a text register indicated by 'index'. When a |

| | |
|---|---|
| tAlarm(UINT16 index, INT16 alm) | block reads data from the SM2 driver, the alarm status of the first byte is returned. The status of additional bytes is ignored. To learn more about available alarm statuses, refer to Understanding Alarm Statuses.<br><br>**NOTE:** As of iFIX 4.5, only the alarm statuses IA_OK and IA_COMM are supported for use through the SM2 driver.<br><br>FE_OK is returned if the operation succeeds.<br><br>FE_IO_ADDR is returned if the register index is out of range. |

# Understanding Alarm Statuses

iFIX and FIX can process alarm status information from I/O drivers. This information complements the alarms generated by iFIX and FIX database blocks. When an alarm is returned from a driver, iFIX and FIX compares the driver alarm against the block alarm. The alarm with the higher severity is used as the block alarm and the other alarm is ignored.

**NOTE:** As of iFIX 4.5, only the alarm statuses IA_OK and IA_COMM are supported for use through the SM2 driver.

iFIX defines the following alarms with the following severity:

| Severity | Alarm Status | Description |
|---|---|---|
| 16 (highest) | IA_COMM | Communication error ("BAD" value). |
| 16 (highest) | IA_IOF | General I/O failure. |
| 16 (highest) | IA_OCD | Open circuit. |
| 16 (highest) | IA_URNG | Under range (clamped at 0). |
| 16 (highest) | IA_ORNG | Over range (clamped at MAX). |
| 16 (highest) | IA_RANG | Out of range (value unknown). |
| 16 (highest) | IA_DEVICE | Device failure. |
| 16 (highest) | IA_STATION | Station failure. |
| 16 (highest) | IA_ACCESS | Access denied (privilege). |
| 16 (highest) | IA_NODATA | On poll, but no data yet. |
| 16 (highest) | IA_NOXDATA | Exception item, but no data yet. |
| 16 (highest) | IA_MANL | Special code for MANL/MAINT (for inputs). |
| 8 | IA_FLT | Floating point error. |
| 8 | IA_ERROR | General block error. |
| 8 | IA_ANY | Any block alarm. |
| 8 | IA_NEW | New block alarm. |
| 7 | IA_HIHI | The block is in the HIHI alarm state (High High). |
| 7 | IA_LOLO | The block is in the LOLO alarm state (Low Low). |
| 7 | IA_COS | Change of state. |
| 7 | IA_CFN | Change From Normal (Digital block only). |
| 7 | IA_TIME | Time-out alarm. |
| 7 | IA_SQL_LOG | Not connected to database. |
| 6 | IA_HI | The block is in the HI alarm state (High). |
| 6 | IA_LO | The block is in the LO alarm state (Low). |
| 6 | IA_RATE | Value exceeds rate of change setting since last scan period. |
| 6 | IA_SQL_CMD | SQL command not found or invalid. |
| 5 | IA_DEV | Deviation from the set point. |
| 5 | IA_DATA_MATCH | SQL command does not match data list. |
| 4 | IA_FIELD_READ | Error reading tag values. |

| | | |
|---|---|---|
| 4 | IA_FIELD_WRITE | Error writing tag values. |
| 1 | IA_DSAB | Alarms disabled. |
| 0 (lowest) | IA_OK | The block is in normal state. |

FIX defines the following alarms with the following severity:

| Severity | Alarm Status | Description |
|---|---|---|
| 16 (highest) | IA_COMM | Communication error ("BAD" value). |
| 7 | IA_HIHI | The block is in the HIHI alarm state (High High). |
| 7 | IA_LOLO | The block is in the LOLO alarm state (Low Low). |
| 7 | IA_CFN | Change From Normal (Digital block only). |
| 6 | IA_HI | The block is in the HI alarm state (High). |
| 6 | IA_LO | The block is in the LO alarm state (Low). |
| 6 | IA_RATE | Value exceeds rate of change setting since last scan period. |
| 5 | IA_DEV | Deviation from the set point. |
| 0 (lowest) | IA_OK | The block is in normal state. |

Using the preceding tables, you can see that if a driver returns a HIHI alarm to a block that is in HI alarm, iFIX or FIX changes the alarm state to HIHI because the driver alarm is more severe. However, if the alarms are of equal severity, iFIX or FIX does not change the alarm state of the block. For example, if the block is in HI alarm and the driver returns a LO alarm, the block's alarm state does not change because both alarms have equal severity. Once an operator acknowledges the HI alarm, iFIX or FIX changes the block's alarm state.

**NOTE:** If you set a communication error to the S register with the SetCommError function, then all SM2 registers show a COMM alarm status. When examining the alarm status of text, only the status of the first character (byte) is read. You can control the alarm status functions of the SM2 driver using its C API only. Refer more information about this API, refer to the Using the SM2 C API section.

# Index