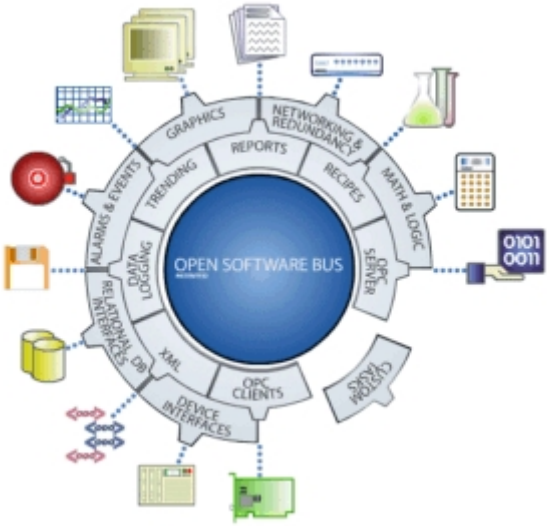
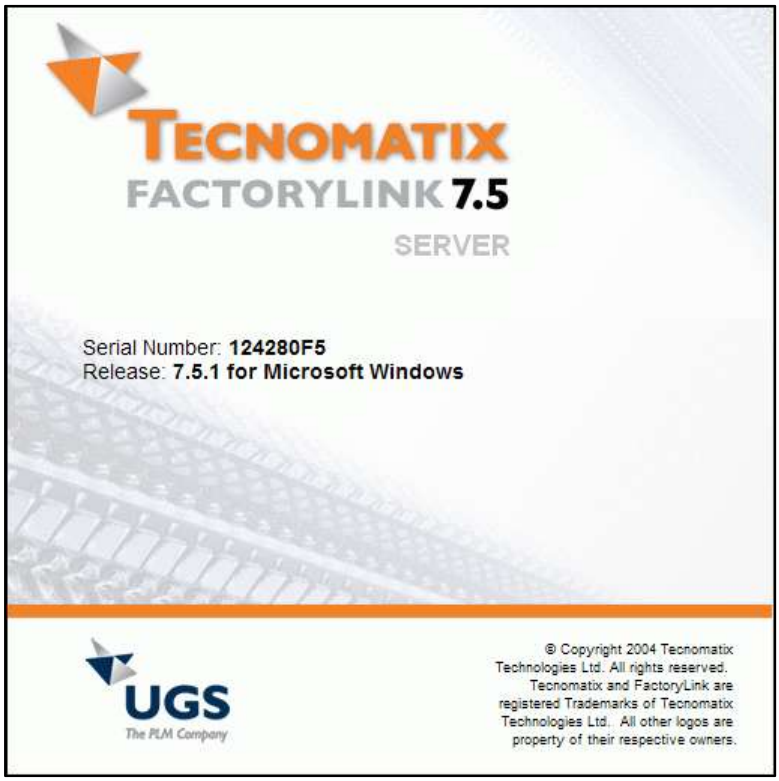


FactoryLink IM-BAS Protocol Driver

for

FactoryLink 7.5.x



IM-BAS

Version 2.0
Printed: Thursday, August 25, 2005

IM-BAS Protocol Driver

The IM-BAS protocol driver is a FactoryLink task, which supports the IM-BAS protocol to communicate with BAS Control subsystems. The driver is part of a FactoryLink system, in IM-BAS terminology this system is called Installation Management (IM) and communicates with one or more BAS Control subsystems. BAS Control subsystems are a collective noun for all systems under technical supervision by and/or reporting errors to Installation Management.

.

RLD Automation
Van Sonsbeeckstraat 11
5344 JB Oss
The Netherlands

tel. +31(0) 412 655 990
fax +31(0) 412 655 991
e-mail : rldauto@tiscali.nl

Version

Date	Version	State and changes
13-05-2002	0.1	Alpha release
	0.5	Beta release
4-07-2002	1.0	First release IM-BAS driver
18-03-2003	1.5	Release of this version, update task-status behavior and improved shutdown recovery.
10-11-2004	1.5	Conversion of task for FactoryLink release FI7.2.2.
03-08-2005	2.0	Conversion of task for FactoryLink release FL7.5.x, and adaption for site ‘Schiphol-Centraal’
	2.1	Backporting of driver to FL7.2.2

Contents

1 Introduction

1.1 IM-BAS Protocol driver

The IM-BAS protocol driver is a FactoryLink task, which supports the IM-BAS protocol to communicate with BAS Control subsystems. The driver is part of a FactoryLink system, in IM_BAS terminology this system is called Installation Management (IM) and communicates with one or more BAS Control subsystems. BAS Control subsystems are a collective noun for all systems under technical supervision by and/or reporting errors to Installation Management.

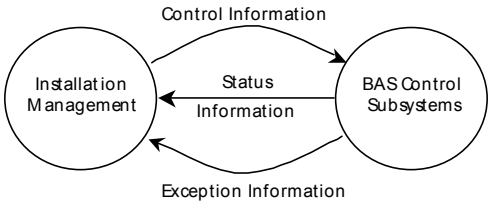


Figure 1.1 Communication between FL7 (IM) and BAS control system

This manual is intended for use by a technician who is familiar with the FactoryLink software. Covered in this manual are the principles of operation, installation and configuration of the software. Included in the shipment is a demo-application, which can be used as an example of how to implement the features in your FactoryLink application.

1.2 Contents of shipment

Please check the package you received with the checklist below. Should there be an item missing contact RLD Automation to correct the problem. There is a limit of 90 days after shipment to report problems!

This package includes the following:

- A IM-BAS driver build contained on a CD-ROM or diskette(s) labelled "Wizard Information Systems IM-BAS Driver"
- This manual.

You should also have:

- A IMX based translator e.g. the IO-Translator.
- The correct hardware and software for the TCP/IP stack on your platform.
- A BAS Control subsystem capable of network communication via TCP/IP.
- An Ethernet-network with at least both the BAS Control subsystem and the FactoryLink system connected to it.

1.3 References

[GLOS] Glossary, Baggage Handling Systems Schiphol, document code T31-391R, version 6.0, dated 11.06.2001

[DATADICT] Data Dictionary, Baggage Handling Systems Schiphol, document code T31-446S, version 2.0, dated 11.06.2001

[BAS_NET] IP-address scheme of the new BAS network, document code T31-597R, version 2.0, dated 09.06.2000

[BAS_SYS] System Communication Identifications and Definitions, Baggage Handling Systems Schiphol, document code T31-713.S, version 1.0, dated 11.06.2001

[BAS_IRS] Project BESBAS Interface Requirement Specification (IRS), Installation Management – BAS Control subsystems, document code T31-691S, version 3.0, dated 11.06.2001

[FL7_CONFIG] FactoryLink 7.x.y Task Configuration Reference Guide

2 Driver principle

2.1 The IO-translator RAP Driver principle

RAPD stands for Rapid Application Protocol Driver. The RAPD principle was adopted so that protocol drivers can be easily and rapidly configured for a FactoryLink application. RAPD is based on the Intertask Mail Exchange standard or IMX, which defines how a protocol driver task communicates with an I/O Translator task. The RAPD system consists of a protocol driver that communicates with external devices (RTUS, PLC's, etc.) and a translator that controls data storage (going to and coming from a protocol driver) in the FactoryLink real-time database. All data collected by the protocol driver is referenced as contiguous blocks or ranges within the device. This enables communication between the driver and a device to be very efficient. All data is referenced between the driver and the translator in terms of datasets. Datasets, described in the next section, define memory regions or locations of data within a device.
The protocol driver and the I/O Translator communicate with one another via FactoryLink mailbox tags, according to the IMX standard. Every task (translator and protocol driver) has its own mailbox, so for full communication between a translator and a protocol driver a mailbox database element for every task has to be defined. The IMX standard is especially designed for the following situation. To use one translator and several protocol drivers. For example the translator together with the ISO-TP4 driver and the Modbus Plus protocol driver. Aside from storage duties, the translator provides data conversions (i.e. analog, IEEE conversions, etc.) for I/O data to/from a protocol driver.

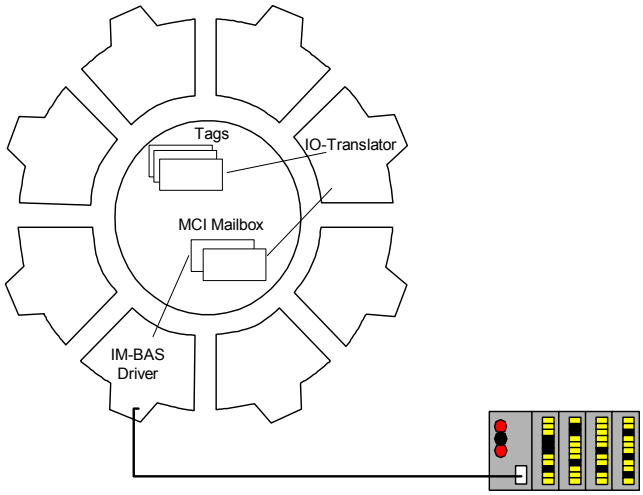


Figure 2.1 The RAPD principle.

2.1.1 FactoryLink domain selection

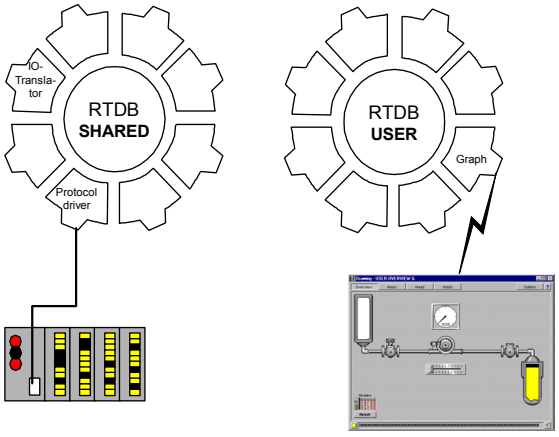


Figure 2.2 Domain selection for the driver

The standard domain for the IM-BAS driver is the SHARED domain. The protocol driver communicates with a dedicated piece of hardware, there fore only one task should be able to access the hardware. If only one

program accesses the hardware, the task should be located in the shared domain and therefore started by the shared runtime-manager.

Important: The protocol driver and the translator must be in the same domain (either SHARED or USER).

2.2 The IM-BAS Protocol

Installation Management is a system for technical supervision of the subsystems and equipment of the luggage transport facilities within BAS Control 2003. The scope of supervision includes communication networks (e.g. process and real-time networks) and supporting systems (e.g. LPC server).
Installation Management is a collective noun for all systems under technical supervision by and/or reporting errors to Installation Management. Due to the nature of 'Installation Management' (e.g. PC's or PLC's) this interface requirement specification is split into separate sets of requirements for IT-systems and PLC-like systems. The IM-BAS protocol driver is a Installation Management systems which only communicates with Installation Management of the IT-type, it is not capable to communicate with subsystems of the PLC-type.
With respect to retrieving the information from the Installation Management, Installation Management will take the initiative.
The IT-subsystems are connected through an Ethernet network compliant with IEEE 802.3. The interface uses TCP/IP as the network/transport layer protocol. The connection on application level will be established with the default port number 7366, but the port number is configurable.

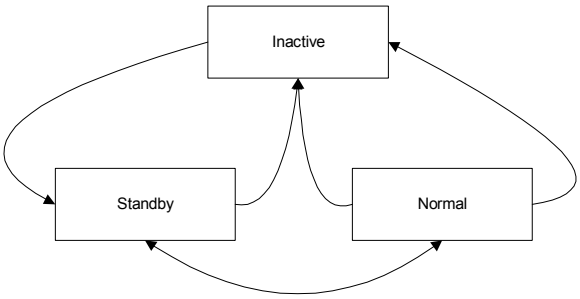


Figure 2.3 States of the driver task.

The IM-BAS driver is based on redundant server technology, and incorporates a standby mode for a redundant system in hot-standby. A hot-standby redundant system is basically two (identical) FactoryLink applications running on two different systems. The driver will be active on both systems, the mode of the driver will be standby for the application running on the hot-standby server. In standby mode the driver accepts connection request on TCP/IP level from subsystems, but rejects application level connections. Switching to and from standby mode is done with a FactoryLink tag, so you have full control over the mode of the driver in your application. In case your redundant FactoryLink system consists of two applications both running that is the hot-standby system is used for normal operation together with the main system, both drivers will be in normal mode. In this situation it is possible that the subsystems are 'distributed' on both systems, to force the subsystems to use a preferred system a digital tag can be used to disconnect all subsystem, that is disconnect them on application and on TCP/IP level. During the time the value of the tag is 'ON', incoming connection request on TCP/IP level are rejected, this will force the subsystems to use the 'other' server. With this tag you can force all subsystems to use the same server! This option is used when the two redundant systems are both up and running, but can not communicate with each other, both servers will now assume they are the 'master' system and that the 'other' system is down. This means both drivers are in normal mode, however the subsystems may be distributed on both systems, now the 'disconnect' tag can be used to let all the subsystems connect to the same server. If all the subsystems are connected to the same server, this server will have full functionality, it can communicate with all subsystems. The other server will have no subsystems connected! This done by setting the 'disconnect' tag to 'ON' (and after a short time to 'OFF') on the server which should have no subsystem(s) connected.
Note: Using redundancy requires the availability of IM-systems which are capable fo communicating with two (the two FactoryLink servers) systems, having different IP-addresses. The IM-system will only exchange data with one IM-system, but will switch to the inactive connection in case the comunication with the current system fails. Note that both the driver and the IM-system should redundancy 'aware'!

Three different types of data exchange are defined for the communication between FactoryLink and Installation Management. For the different type of data exchange, different functionality for the communication between the driver and the subsystem is needed.

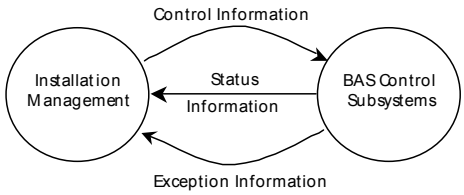


Figure 2.4 Communication between driver and BAS control subsystem.

Control information: These are commands and parameters send to the BAS control subsystem, for the driver this will be 'writes'. Fetching the same parameters from the subsystem is a 'read' for the driver.

Status information: These are system status and statistics reported by the subsystem, on request of the driver. The driver performs a 'read' operation to retrieve the values.

Exception information: The subsystem send exception messages to the driver whenever there is an exception monitored by the installation management. For the driver these kind of messages are 'unsolicited receives'.

2.3 Devices

Every BAS Control Subsystem connected to the driver via the network is called a device, this term is defined in the IMX standard as a logical external device (e.g. a PLC). A BAS Control Subsystem is called a device in the FactoryLink application and has a logical name (only used in the application configuration). In the network every device (or BAS Control Subsystem) has its own unique identifier, this identifier is an alphanumeric string of two characters.

The TCP/IP protocol is a connection-oriented protocol, to exchange data between the driver and a device a connection must be present. A connection is a bi-directional point-to-point communication link through the network. Each connection requires an IP-address and a port number on both sides of the connection. The IP-addresses are resolved from names by using DNS. On one side, the server side, the port number is pre-defined; here the driver acts as the server part of the communication link.
Before data exchange can take place, a connection must be established on application level. To establish this connection the subsystem sends life-signs to the driver, included in these life-sign messages are an unique subsystem identifier. With this identifier the driver is able to uniquely identify the subsystem on the network, even two or more subsystems using the same IP-address can be identified in this way. The driver responds to the life-signs by sending a reply to the subsystem. This reply is used by the subsystem to accept a connection at application level. The connection is maintained by sending life-signs at regular intervals, if a life-sign (or response) is not received within a certain time limit, the connection is assumed to be lost.

2.4 Communications functions

Data exchange between a FactoryLink application with a IM-BAS driver and Installation Management is performed by using the *read*- and *write*-services of the IM-BAS protocol, see for detailed information [BAS-IRS]. The protocol driver task uses the *read*-service to read (or fetch) data from the subsystem, and it uses the *write*-service to write data to the subsystem. Both these actions can be initiated by the driver task, on the other hand the subsystem may use the *write*-service for sending data to the FactoryLink station, this is called an unsolicited receive from the perspective of the protocol driver.

The IM-BAS protocol supports different types of read and write communication functions, the ones supported are described in the next paragraphs. In the table below are all the possible functions listed together with the information if they are supported or not.

Read/Write Service	Supported
Block read	Yes
Block write	Yes
Exception write	Yes
Encoded write	**No**
Unsolicited Receive	Yes

2.5 Datasets

Data exchange is, according to the definitions stated in the IMX standard, is based upon datasets. A dataset is a contiguous area, such as a register or memory location, of data in the external device. All datasets are defined as digital tags. The protocol driver defines all specifics, such as data type about datasets, starting address, or length, while the IOX task references the dataset by tag name for read/write operations. When the IOX task initiates a trigger on a dataset, the IMX system notifies the protocol driver that an event has occurred on that dataset. The event notification, in turn, causes a protocol specific message to be generated and sent to an external device. The scenario works in reverse when the protocol driver receives unsolicited data from a device.

Dataset	Maps to external device
Referenced by dataset control tag	Memory location, defined by area type, start address, and length
Tag 1	Address: n
Tag 2	Address: n + 1
Tag 3	Address: n + 2
Tag 4	Address: n + 3
Tag x	Address: n + m

A dataset can be read or written to with one command or trigger. With the exception of actions on the complete dataset, most protocol drivers are capable of addressing specific elements or a group of elements in the dataset. How these actions are performed depends on the size of a basic element in the dataset, which can be bit, byte, word, or long, depending on the data area in the device or the communication protocol.

2.5.1 Block read and block write

The read and block write commands are performed on a contiguous block of data in the external device specified by datasets. The FactoryLink station initiates both commands. The developer configures digital triggers for the IOX task to start the read/write commands. Data received from a device as a result of a read request is stored in FactoryLink tags. For a write request, data is taken out of a FactoryLink tag and sent to the device.

2.5.2 Unsolicited receive

In case of an unsolicited receive the external device sends a dataset to the FactoryLink workstation, without a specific request from that workstation. The device sends a dataset to the FactoryLink workstation for an unsolicited receive, though the workstation does not originate the request to do so. The I/O translator task places the data received from the device into FactoryLink tags.
The protocol driver will check if the received data is configured in one of the datasets. If this is the case, the data will be sent with the dataset information to the I/O translator, which converts and updates the data. If no dataset is specified regarding to the received data, it will be discarded.

The unsolicited receive function is a very powerful functionality for fast communication because no request has to be placed by the protocol driver. This way less overhead is involved in communication. Data that alters unpredictably such as alarms are very suitable for the unsolicited receive function. Some extra programming effort has to be done in the external device because the device is in charge of sending the data.

2.5.3 Exception write

The FactoryLink workstation initiates an exception write and causes a write of data to the external device. Exception writes are most often used when a tag value has changed within the real-time database.

The protocol driver receives a request for an exception write from the I/O translator task. The first action the driver takes is to check if the write can be accomplished with one write operation or if the data must be read first, then written, which results in more than one operation. In situations where the exception data element size is smaller than the size of the device data type element, the data area, or register, must be read first. Then the data to be written out must be masked or OR'ed into the value just read.

Finally, the new value is written back to the device.

Caution: If this is not completed, data will be unintentionally overwritten in the device.

An example of an exception write is if a bit in a device has to be set and the boundary of the data area is a 16 bit word, which implies word addressing, the specific data word has to be read first because the smallest element in this device area is a word. The bit has to be patched into the word and the word is then written back to the device. In this illustration it is possible the device may change the word value during the read-before-write operation. In this case, once the driver writes the new word value, elements of the last word change are lost. To avoid the latency problem associated with read-before-write operations, an encoded write is an alternative option. An encoded write can perform a write operation in one request instead of the two or more (depending on the protocol) requests generated by a read-before-write operation.

2.5.4 Encoded write

The encoded write function is similar to the exception write in that both functions write single data elements. The difference is the exception-write directly accesses the desired data area in the external device and that data area is changed. On the other hand, the encoded write composes an encoded write command. The encoded write command can be thought of as a protocol message to the device referencing a certain memory location. A device program, such as a ladder logic algorithm in a device, running within the device detects the memory location has been written to. This causes the device program to make the change to a single data element. The advantage of this method is only one write command is generated from the protocol driver and the device internal program does the actual operation on the data element. This means no reading occurs before writing. Another advantage is the device program controls all encoded writes because they are all written to one location in the device.

Note: The encoded write is not supported for the IM-BAS protocol

2.5.5 Datasets for IM-BAS protocol

The datasets that can be present in a device, which supports the IM-BAS protocol, are summarized in the table below. The message types described in the table define the different kind of possible datasets, for a detailed description see [BAS-IRS]. The boundaries are directly related to the addressing: one address relates to the size of the boundary, e.g. word boundary has word-addresses, every address is an value of two bytes. For the function column the following characters are used: R – for read functionality.
W – for write functionality.
U – unsolicited receive functionality.

Message type, Data type	Function	Start	Length	Boundary	Conversion I/O translator
Command, Bool {IMBC01}	W	0	Subsystem dependent	Byte	Byte
Parameter, Bool {IMBC02, IMBC03}	RW	0	Subsystem dependent	Byte	Byte
Parameter, Integer {IMBC02, IMBC03}	RW	0	Subsystem dependent	Short	Word
Parameter, Long {IMBC02, IMBC03}	RW	0	Subsystem dependent	Long	Long
Parameter, Real {IMBC02, IMBC03}	RW	0	Subsystem dependent	Long	DBL
Parameter, String {IMBC02, IMBC03}	RW	0	String, length = 20 bytes.	Long	MSG, length = 20
Status, Bool {IMBC11}	R	0	Subsystem dependent	Byte	Byte
StatInfo, Bool {IMBC12}	R	0	Subsystem dependent	Byte	Byte
StatInfo, Integer {IMBC12}	R	0	Subsystem dependent	Short	Word
StatInfo, Long {IMBC12}	R	0	Subsystem dependent	Long	Long
StatInfo, Real {IMBC12}	R	0	Subsystem dependent	Long	DBL
StatInfo, String {IMBC12}	R	0	Subsystem dependent	Long	MSG, length = 20
StatInfo-Del, Bool {IMBC12}	R	0	Subsystem dependent	Byte	Byte
StatInfo-Del, Integer {IMBC12}	R	0	Subsystem dependent	Short	Word
StatInfo-Del, Long {IMBC12}	R	0	Subsystem dependent	Long	Long
StatInfo-Del, Real {IMBC12}	R	0	Subsystem dependent	Long	DBL
StatInfo-Del, String {IMBC12}	R	0	Subsystem dependent	Long	MSG, length = 20

Message type, Data type	Function	Start	Length	Boundary	Conversion I/O translator
Exception {BCIM21}	U	0	4 bytes.	Byte Address 0 = Time stamp, 14 bytes. Address 14 = Equipment code, 16 bytes. Address 30 = Exception level, 1 byte. Address 31 = Description, 50 bytes	Depends on the variable, there are four different variables in the dataset.

The id between {} in the column 'Message type' refers to the command id's used in the document [BAS-IRS].

2.6 Multi-threading of the driver

The IM-BAS driver is a high performance RAP driver, for speed considerations the driver is build as a multi-threaded task. On start-up there is initially one thread active, this thread is the 'main' thread, first it will load the configuration defined with the 'Configuration Explorer'. In the configuration are all the device definitions found, next the protocol driver starts a thread for every device or subsystem. One thread for every device is needed to perform the 'read' and 'write' commands, in combination with IMX this thread will wait for any read or write command to occur. If there is a command it is immediately executed and the response is send to the I/O translator.
A second thread for every device is needed to receive and respond to exception messages, 'unsolicited receives'. The unsolicited receives are the exception message and the life signs.

The total number of threads for the device will be: **2 * number of devices + 2**. The first of the two 'extra' threads is the main thread, responsible for dispatching the read and write commands from the I/O translator to every read/write thread of a device. The second 'extra' thread is needed to set out a listen for connection requests from subsystems, if a connection request is received and accepted, a new thread is started. This thread will be used for the unsolicited receives, only after receiving the first life-sign the subsystem id is known and device information (stored in global memory) is associated with the thread. After the life sign the thread for read/write commands is also associated with the device information.
The logical order of starting and creating threads is the following:

1. Task is started, one thread present. Configuration data is loaded, among it are the device definitions.
2. For every device a thread for the read/write commands is started (IMX).
3. A listen thread is started for accepting connections.
4. After accepting a connection, a thread is started for any 'unsolicited receive' functionality, and the thread is associated with the device information after receiving a life-sign. Also the read/write thread is associated with this device information.
5. On error, or stopping of the task, life signs are not answered and next TCP/IP connections are closed.
6. On changes from standby to normal or vice verse, the answering to life signs starts or vice versa is stopped. TCP/IP connections are not closed when going from normal to standby state.
7. On request (from the digital 'disconnect' tag) all connections are closed o application level and on TCP/IP level. A disconnect on TCP/IP level will cause the receive thread to exit. New incoming TCP/IP connection requests are handled by the listen thread, see 3.

3 Installation

3.1 System requirements

The IM-BAS driver will install and run on any IBM PC or PC-compatible computer equipped with:

- A processor equivalent to an Intel 80486 or later. Wizard Information Systems recommends at least an Intel Pentium-class or compatible processor.
- A CD-ROM.
- At least 5 MB of free hard disk space.
- Microsoft Windows NT 4.0, with at least service pack 4 installed.
- USDATA FactoryLink 7.0, with at least service pack 1 September installed.

3.2 IM-BAS Driver installation

The main objective of the IM-BAS driver installation is copying a group of files from the installation media to your computer system. In this group all the files related to the FactoryLink environment are copied to your FactoryLink system directory (normally set by the 'FLINK' environment variable), including both the runtime and configuration options.
To start the installation procedure: locate the 'setup.exe' file on the installation media, and execute the setup utility.

Figure 3.1. Setup the application (large icon view).

After starting the setup, some questions have to be answered, the most important one is where your FactoryLink system directory is located. The setup assumes that the environment variable %FLINK% is correct, but gives you the chance to change the directory. Most of the steps in the installation process are presented in the following figures.

Figure 3.2. The welcome wizard panel.

The first screen displayed is the 'Welcome' screen, click 'Next >' to continue. The next wizard panel displays the IM-BAS driver end-user license agreement. Read this agreement carefully. If you install the IM-BAS driver you agree to abide by the terms of the license.
If you do not agree to the license terms, click 'I do not accept the agreement' folowed by 'Cancel'. Setup will quit immediately. Otherwise, click 'I accept the agreement' followed by 'Next >' to continue. The next screen will list the minimum requirements for your system needed by the IM-BAS driver. It also explains how to choose the program location in the next screen, press 'Next >' when you are ready.

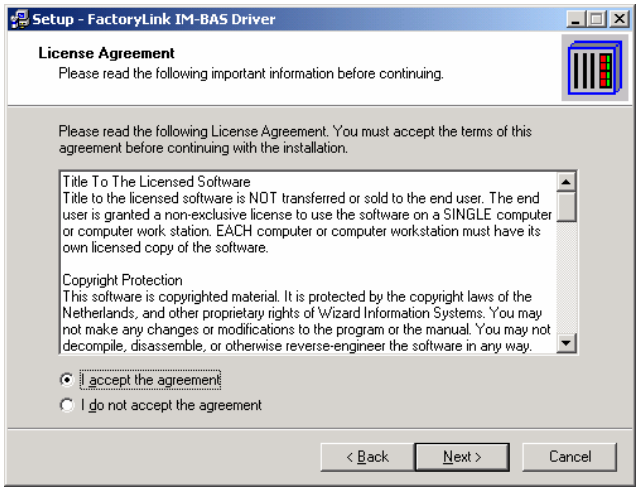


Figure 3.3. The software license agreement.

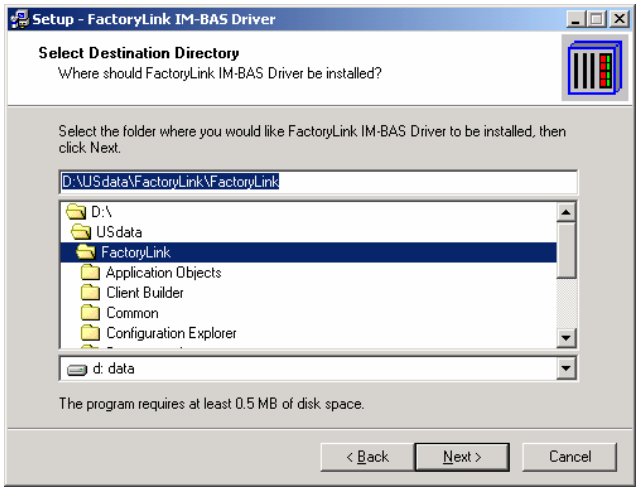


Figure 3.4. Select destination directory

Use the browse window to specify a directory for the FactoryLink system files, in case the one selected by the set-up wizard is not the one where you want to install this FactoryLink driver.

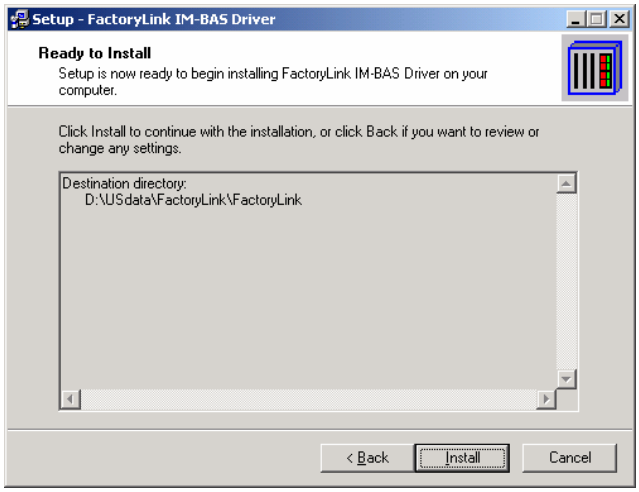


Figure 3.5. Ready to install the application.

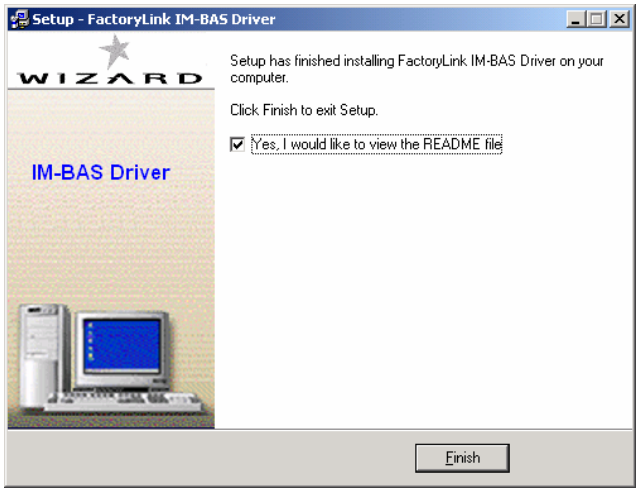


Figure 3.6. Installation is completed.

You need to confirm your settings and requirements, before the actual installation process is started. After completion of the installation process, the last page of the wizard is shown, and pressing the 'Finish' button closes the setup window.

You can view the 'readme' file with the latest information about the driver, which did not make it to the documentation, by placing a vink in the check box.

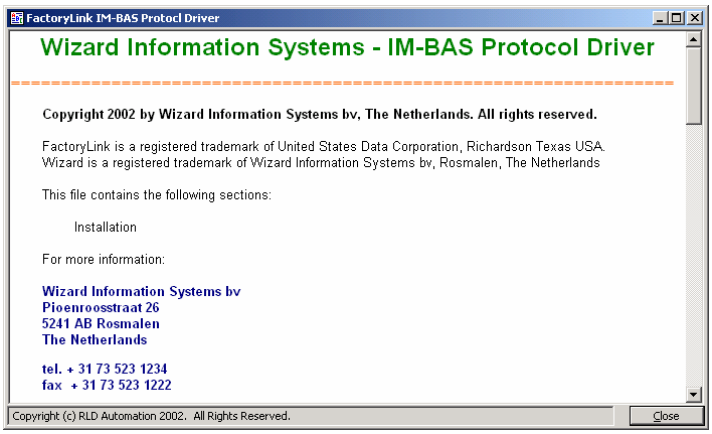
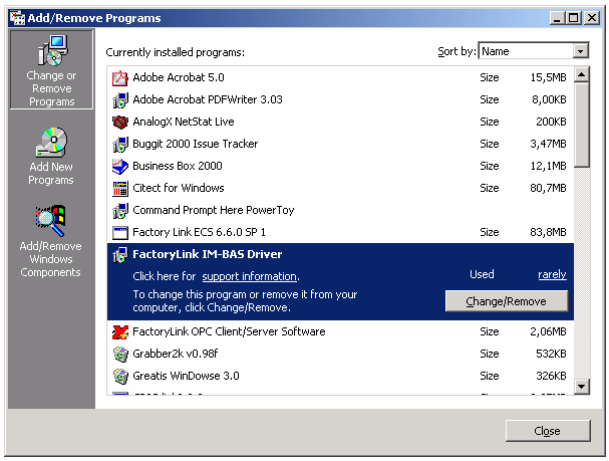


Figure 3.7. The 'readme' file will look like this.

3.3 IM-BAS driver de-installation

De-installation of the IM-BAS driver is done through the Control Panel applet ‘Add/Remove Programs’. The IM-BAS driver has an entry in the list of applications, which can be de-installed: ‘FactoryLink IM-BAS driver’.



Information window for the IM-BAS driver.

Figure 3.8. Control Panel applet for removing the IM-BAS driver.

Clicking on the button Change/Remove allows you to remove the IM-BAS driver from the system. Before the deleting of files starts, you are asked to confirm the removal of the application.

4 IM-BAS driver configuration

4.1 FactoryLink task configuration

The configuration of the IM-BAS driver consists of three main parts, two of these parts need some configuration, the third part has already a default configuration. This default configuration will in most cases suit the needs of the application. The configuration main parts are: Run time manager configuration, INI-file configuration of the IM-BAS driver and driver table configuration.
The first (Run time manager configuration) and the last (driver table configuration) configuration, are normally done with the configuration explorer, and involves changing or adding settings in configuration tables. Second part of the configuration is the task's INI-file, a default file is already supplied during installation, changes in this file influences your system, not only the application. The file is located in your FactoryLink system directory (%FLBIN%), and not part of an application.

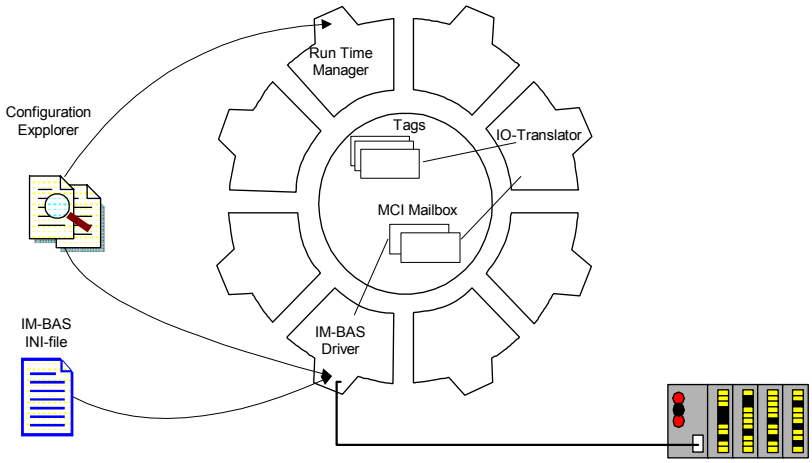
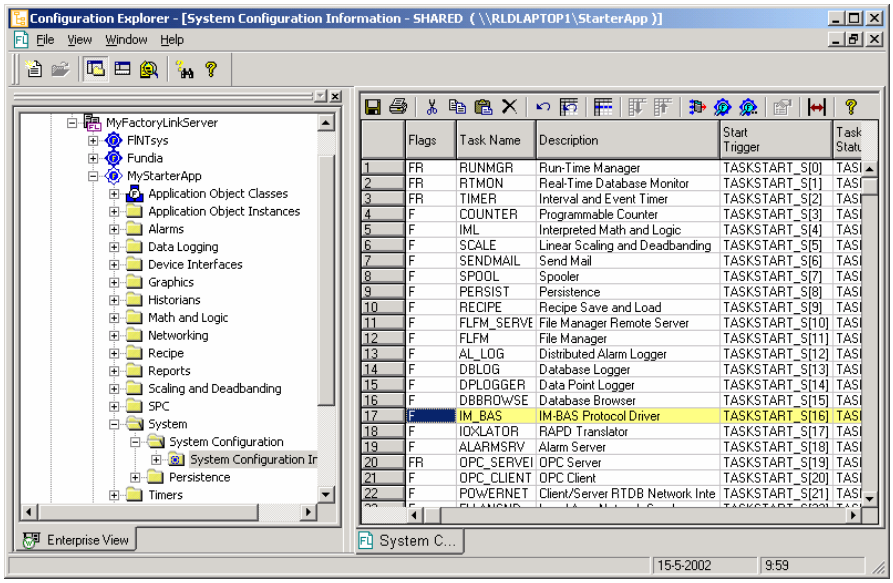


Figure 4.1 IM-BAS driver configuration.

4.2 Run time manager configuration

In order to activate the IM-BAS driver as a FactoryLink task it needs some configuration in the 'system configuration' panel of the FactoryLink Configuration Explorer.



	Flags	Task Name	Description	Start Trigger	Task Statu
1	FR	RUNMGR	Run-Time Manager	TASKSTART_S[0]	TASI
2	FR	RTMON	Real-Time Database Monitor	TASKSTART_S[1]	TASI
3	FR	TIMER	Interval and Event Timer	TASKSTART_S[2]	TASI
4	F	COUNTER	Programmable Counter	TASKSTART_S[3]	TASI
5	F	IML	Interpreted Math and Logic	TASKSTART_S[4]	TASI
6	F	SCALE	Linear Scaling and Deadbanding	TASKSTART_S[5]	TASI
7	F	SENDMAIL	Send Mail	TASKSTART_S[6]	TASI
8	F	SPOOL	Spooler	TASKSTART_S[7]	TASI
9	F	PERSIST	Persistence	TASKSTART_S[8]	TASI
10	F	RECIPE	Recipe Save and Load	TASKSTART_S[9]	TASI
11	F	FLFM_SERVE	File Manager Remote Server	TASKSTART_S[10]	TASI
12	F	FLFM	File Manager	TASKSTART_S[11]	TASI
13	F	AL_LOG	Distributed Alarm Logger	TASKSTART_S[12]	TASI
14	F	DBLOG	Database Logger	TASKSTART_S[13]	TASI
15	F	DPLOGGER	Data Point Logger	TASKSTART_S[14]	TASI
16	F	DBBROWSE	Database Browser	TASKSTART_S[15]	TASI
17	F	IM_BAS	IM-BAS Protocol Driver	TASKSTART_S[16]	TASI
18	F	IOXLATOR	RAPD Translator	TASKSTART_S[17]	TASI
19	F	ALARMSRV	Alarm Server	TASKSTART_S[18]	TASI
20	FR	OPC_SERVEI	OPC Server	TASKSTART_S[19]	TASI
21	F	OPC_CLIENT	OPC Client	TASKSTART_S[20]	TASI
22	F	POWERNET	Client/Server RTDB Network Inte	TASKSTART_S[21]	TASI

Figure 4.2 FactoryLink task configuration.

The required settings for the IM-BAS driver are its task name and executable name, the last one is not visible in the figure above, but can be reached by panning through the configuration row. The following settings apply:

Task Name = IM_BAS
Executable File = bin/im_bas

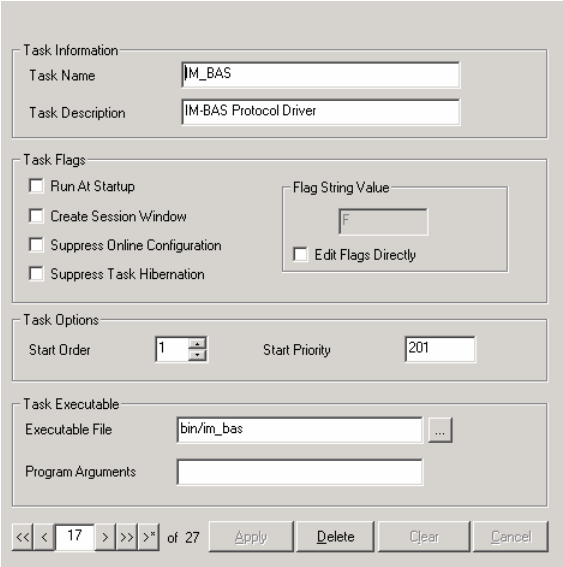


Figure 4.3 IM-BAS task definition.

4.2.1 Program arguments

-LSn Local Station id with number *n*, needed if the protocol driver and IO-translator reside on different nodes (or FactoryLink workstations). This must be an unique (station) number for the network. Two IMX tasks (the IOX task and a protocol driver) can run on different computers communicating with one another. Because the two computers can differ in data direction (different platforms), IMX supplies information specifying where the dataset came from and what kind of data orientation it has.
The station identification is a unique number of the host station on the LAN. The user must assign this number and keep it unique over the LAN. The station identification can be passed to the driver task as a program argument.
If tasks are to support communications over a LAN, some information must be exchanged between the tasks before they can begin communications. The tasks are probably running on separate nodes in two different FactoryLink applications, which means all tag ids are most likely not consistent. IMX uses the dataset control tag for identification of the dataset. But, if a problem prevails, the dataset control tags cannot be exchanged as they exist because the remote application does not recognize the dataset control tag.

4.3 INI-file

The IM-BAS driver uses an INI-file, located in the FactoryLink 'bin' directory (%FLBIN%), and is named im_bas.ini. The location makes the settings in this file system wide, and not application settings. Note that this file is not saved in an application save file! It should be reviewed if the target system differs from the development system.

INI-file: im_bas.ini

```
* Debug section
* The level 1..4 is defined with DebugLevel.
* Seperate logging to file is requested with Log2File
* Seperate logging to the event viewer is requested with Log2EventViewer

[Debug]
DebugLevel=2
Log2Screen=1
Log2DebugViewer=1
Log2File=1
Log2EventViewer=1

* Task section
* A task can be forced to use polling mode with Polling,
* this is the opposite of event based.
* The polltime is a value in millseconds the task will
* sleep before a new search for command is done
* A protocol driver normally uses two task id's to exchange data
* with the FL kernel, if desired this setting can be overruled
* with the key Driver2Ids set to zero, the task switchs (automatically)
* to forced polling mode.

[Task]
Polling=0
PollTime=5
Driver2Ids=0

* Translator section
* This section is only used by protool drivers, it defines
* the absolute maximum number of messages there can be in
* a mailbox read by a translator.

[Translator]
MaxMessage=1000

* Protocol section
* Oly used by drivers, and is protocol dependent, see the
* manual of the driver.

[Protocol]
Source=IM
Destination=BC

* Terminator is specified as hex number
Terminator=10
Version=02
LifeSeq=00

* Exception message have a fixed length in addresses and bytes
ExceptionByteLength0=134
ExceptionByteLength1=16
ExceptionByteLength2=1
ExceptionByteLength3=50

* Exception messages, time conversion: local or UTC
ExceptionUseLocalTime=1

* Trimming of strings, remove leading and/or trailing spaces
StringTrimLeft=0
StringTrimRight=1
StringTrimBothSides=0

* Non existent numerical elements in TM have a default value
NonExistentValue=-1

* Driver can keep or close TCP/IP connection after a protocol error
```

```
DisconnectProtocolErrors=1

* Lifesign loss: use timeouts and/or disconnect from TM
UseLifesignTimeout=1
DisconnectLifesignLoss=0
```

An INI-files has sections, and in every section one or more key's, every key is associated with a setting. Section names are enclosed in brackets []. A key and it's value are separated by the '=' character.

4.3.1 Section: [Debug]

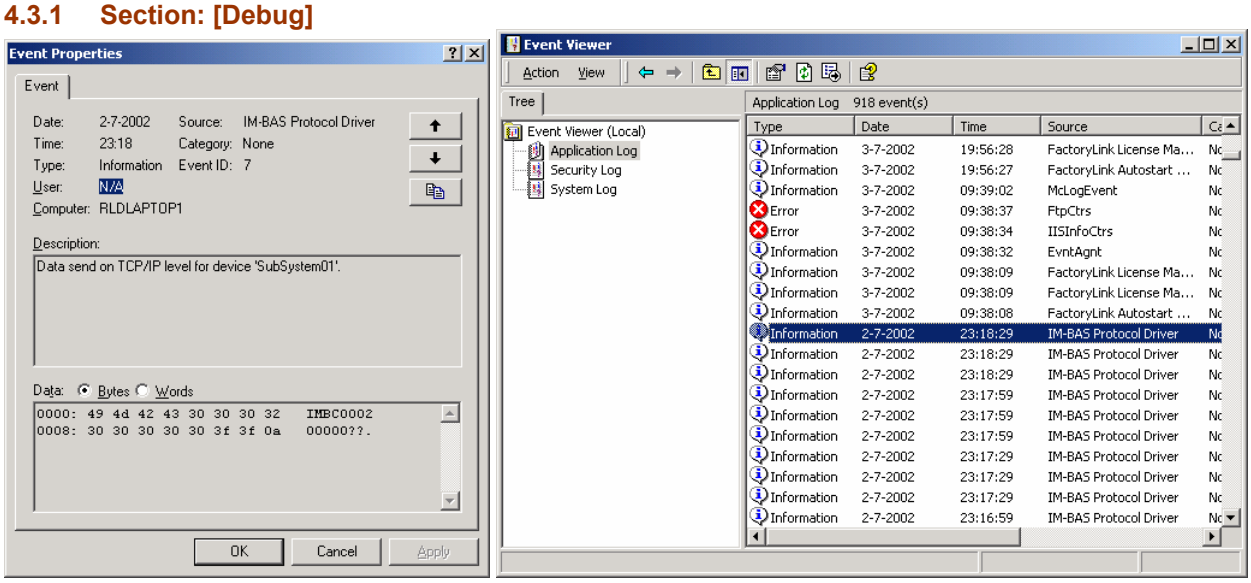


Figure 4.4 Logging to the Microsoft Event Viewer.

This section defines if and how many debug information is generated. The key DebugLevel defines if and how detailed the debug information will be, a level of zero disables debugging info. The maximum level is 4, and this level includes the data send and received from devices. The lowest level is level 1, for this level only the run-time manager messages are used as debug information.
Normally the debug information is displayed in a debug window of the Configuration Explorer, or to a command window if FactoryLink is started from the command prompt; for debugging information in the Configuration Explorer or the command window the key 'Log2Screen' must have any other value then zero. By setting the key Log2File to a value not equal to zero the debug information is logged to a file in the application directory: {FLAPP}\{FLNAME}\{FLDOMAIN}\log\im_bas.log. For default environment settings this name will be: {FLAPP}]flapp1\shared\log\im_bas.log, where {FLAPP1} is your application directory. Every time the driver is started the log file is cleared, during run-time you have no control over the size of the log-file! Additional to logging to a file it is possible to log to the Microsoft event viewer, all you have to do is set the key Log2EventViewer to a value not equal to zero. Logging to the log-file and the event viewer can be active together, or you can choose only one (or none) of the logging methods.

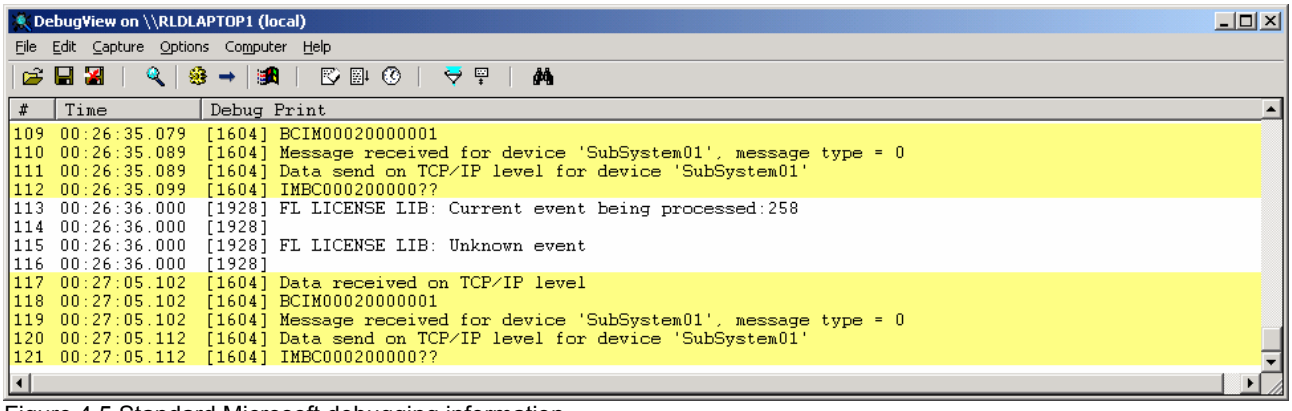


Figure 4.5 Standard Microsoft debugging information.

Microsoft supports debugging of applications with a 'DebugViewer', an application that is capable of showing, filtering and saving debug information. In your shipment is included a debug viewer application. In order for this application to show the debug information, the driver should generate the Microsoft compatible debug information, to do so turn the option 'Log2DebugViewer'. The debug viewer included supports remote debugging, use any system in your network to view, filter or save the debug information form the driver, the only thing you have to do is connect to the server where the driver is running.

4.3.2 Section: [Task]

In this section is defined how the driver task interacts with the FactoryLink real-time database, and how multi-threading is configured for optimal speed, or real-time database access. A normal task uses one name, the task name (see previous paragraph) to get access to the real-time database.

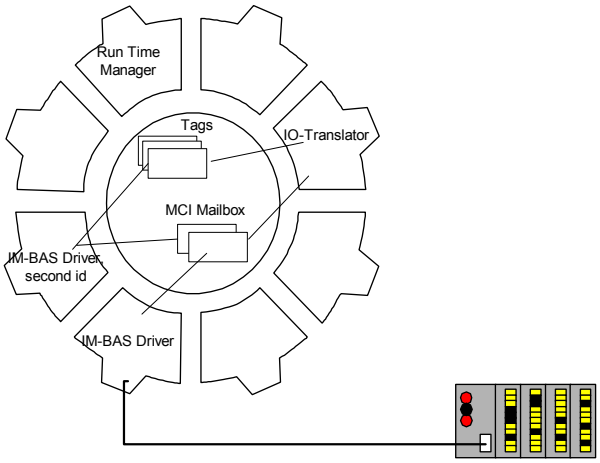


Figure 4.6 Second task-id for driver.

The run-time manager task regulates the access to the real-time database by supplying so called task-id's to every task that registers to the FactoryLink kernel.
For optimal speed a task can use two task-id's instead of one, one task id will be used to receive commands from the I/O translator mailbox('s) and the second id will be used to reply to the I/O translator. Using two id's allows a task to simultaneously read and write from/to the real-database, with one id these actions are carried one after another. Using two id's is set with the key 'Driver2Ids' to a value not equal to zero.
For the second task-id there must be an empty entry in the system configuration table, not all 30 task entries should be filled in. During start-up the driver will automatically acquire two task-id's.
A normal task, with one task-id, can operate in a polled or event mode, default the event mode is used. However if desired, the polled mode is activated with the key 'Polling' set to a non-zero value. The poll-time is set with the key 'PollTime' to a time in milliseconds; this is the time between successive polls of the real-time database for commands or instructions.

4.3.3 Section: [Translator]

In this section there is currently only one key defined 'MaxMessages', this number limits the maximum number of messages in any mailbox read by the translator and filled by the driver to the given value. The limit is normally only reached if the translator task is not active, messages are not removed from the mailbox. The driver will stop filling the mailbox at the specified maximum, this way the number prevents the system from running out of resources caused by an unlimited filling of mailboxes.

4.3.4 Section: [Protocol]

In this section several settings used by the protocol driver to implement the IM-BAS protocol are defined. The settings include a number of fixed settings, which are not likely to change often. You are strongly advised not to change the default settings, doing so can cause the driver to report errors or stop the communication at all.
There are however settings to influence data handling by the driver: ExceptionUseLocalTime, StringTrimLeft, StringTrimRight, StringTrimBothSides and NonExistentValue.

ExceptionUseLocalTime – Defines how timestamps of exception messages are converted. A 'true' value (any value not equal to zero) forces the driver to convert the time in an exception from UTC to local time (including daylight saving if this option is activated on the system)

StringTrimLeft, StringTrimRight, StringTrimBothSides – Define the trimming of spaces from string variables. String data is always retrieved from the subsystem with a fixed length, the subsystem adds space to fill the string to the fixed length. With these settings you can remove spaces from all (received) string data.

NonExistentValue – In case the driver does a block read, there ca be non existent data in this block. The subsystem will not give any values for these addresses to the driver, the will now be 'empty' places in the dataset, they can be filled with a predefined value. This key defines the default value for nonexistent data, note that string data is always initialized to an empty string.

DisconnectProtocolErrors – In case the driver recognizes a protocol error, the driver can disconnect the TCP/IP connection with the subsystem, or only discard the message and continue using the connection (on TCP/IP and application level). If the value of the key is not equal to zero, the TCP/IP connection is reset on every protocol error. Protocol errors include invalid message id's, invalid protocol version, invalid terminator, invalid source id, invalid destination id and invalid subsystem id.

UseLifesignTimeout – Setting this key to zero configures the driver not to react on life sign timeouts. The life sign timeout is defined as a part of the device (see next chapter), and normally the driver detects and reacts on missing life signs in the specified time period. If the use of the life sign timeout is enabled (any value not equal to zero), the behavior is determined with the next key: DisconnectLifesignLoss.

DisconnectLifesignLoss – Reaction of the driver on detection of a life sign timeout: key of zero means don't do anything, only an error is reported. If the key value is non-zero the TCP/IP connection with the subsystem that failed to send life signs is reset.

IdLifeSign, IdCommand, IdParameter, IdSetParameter, IdStatus, IdStatistical and IdException – These are the 2-byte message identifiers used in the IM_BAS protocol. Do not change these settings unless you are familiar with the protocol used by the subsystems connected to the driver, see reference [BAS_SYS].

DebugInfoForReals – The value of this setting is true (value not equal to zero) or false (value is zero), if true a debug message is given for every floating point. Floating point values are received as a string: xxxxxxx.yyyyyyyy. Due to conversion errors the value presented in an application can be different from the value received as an ASCII string. To help you to identify the conversion error, the driver can report the result of the first conversion from string to float.

ThreadDelaySec – The value of this setting is used as a delay in seconds. After connection establishment with a subsystem, a thread for reading, writing and receiving data is started. Before the thread starts waiting for data to receive or commands from the IO-translator, it waits the specified number of seconds. If you have a redundant system, this delay can be used on the 'slave' system (not on the 'master' system) to give the 'master' time to react earlier on life sign requests. If both server are up and running and accept connections, the master will first accept a connection on application level, the subsystem in return will disconnect from the system which responds slower.

ErrorOnDataLimit – Boolean value for generating errors if the driver enforces a data limit value, data limit errors are only reported to the IO-translator if the value is not equal to zero. Otherwise write actions will complete successful, even if the driver limited values written to a TM to it's minimum or maximum.

FirstLifeSign – After accepting a TCP/IP connection the driver waits for the first lifesign of the connected TM, this parameter defines the maximum time in seconds to wait for this first lifesign. If no lifesign is received within this period the TCP/IP connection is closed and reset. The TM should then request for a new connection on TCP/Ip level

4.3.5 Section: [RuntimeManagerError]

In this section error settings for the presentation of the IM_BAS driver in the runtime manager mimic are defined. If the driver encounters an error, an error string and code is send to the runtime manager, the runtime manager displays these results in tags. Normally there is a mimic in the application were these tag values are shown. The task status represents the actual state of a task, this can be: inactive, running,

running with error, starting or stopping (decimal values: 0, 1, 2, 3, 4). In his section the filtering of error for presentation by the runtime manager is set.
Error%d - %d is the error number to enable or disable for setting the error state of the driver in the runtime manager. If error number 329 should not set the task status to ‘running with error’ the entry ‘Error329=0’ must be present in this section of the ini-file. Note that if the task shows an error, the analog task-status is 2, subsequent errors which are disabled will not be shown on the task message line.

4.4 Configuration tables

The configuration tables for the IM-BAS driver can be accessed with the FactoryLink Configuration explorer. For general information about entering data in FactoryLink configuration tables refer to the FactoryLink Manual. In the Configuration Explorer select ‘Device Interfaces’. A table entry appears: ‘IM-BAS Protocol Driver. This entry gives access to the configuration tables, two tables are present, ‘IM-BAS Driver Definition’ and ‘IM-BAS Device Definition’. The first table to configure is common options for the driver, the second configures all the subsystems that will be present in the network, and defines the datasets for every device.

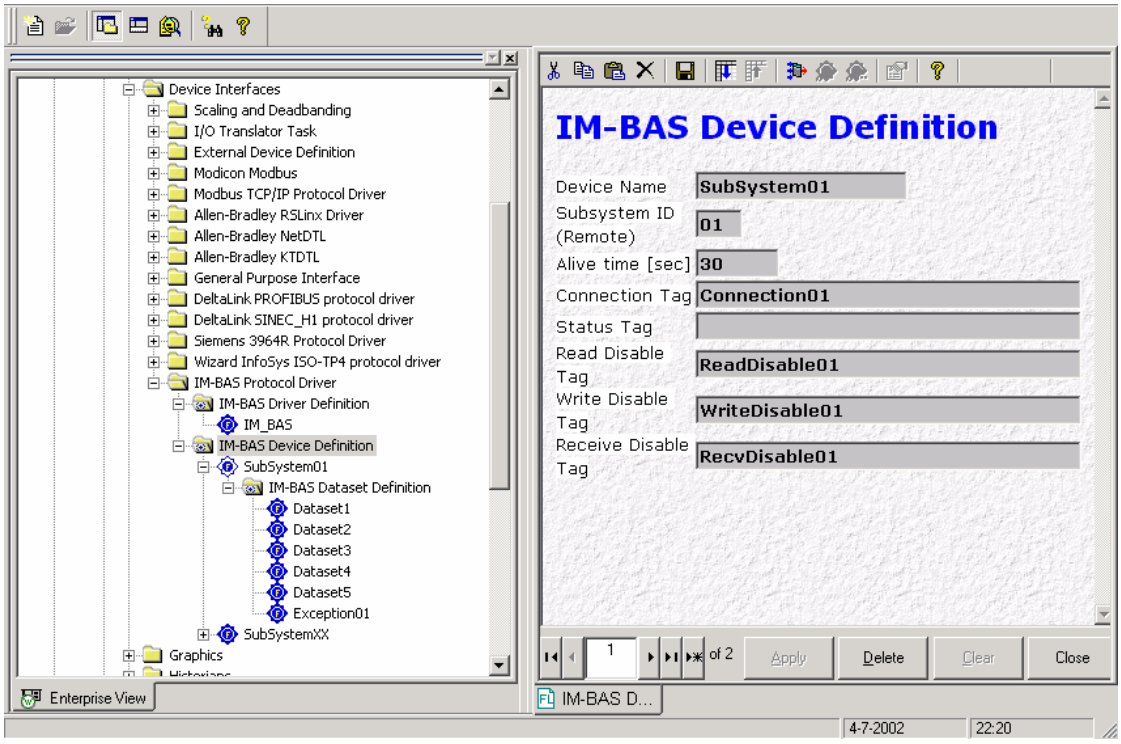


Figure 4.7. FactoryLink Configuration explorer.

4.4.1 IM-BAS Driver Definition

The ‘IM-BAS Driver Definition’ panel is used to define global settings for the driver. For these global settings only one record or row is needed, if you define more then one row, the driver will ignore the second and following rows. The first row however is required, an empty table will generate a task error and cause the driver to stop. The fields ‘Protocol Driver Mailbox Tag’ and ‘Subsystem ID’ are required and do not have a default value. Every definition in the configuration panel completely defines a mail message; don’t forget to define at least one recipient for the given list of recipients.

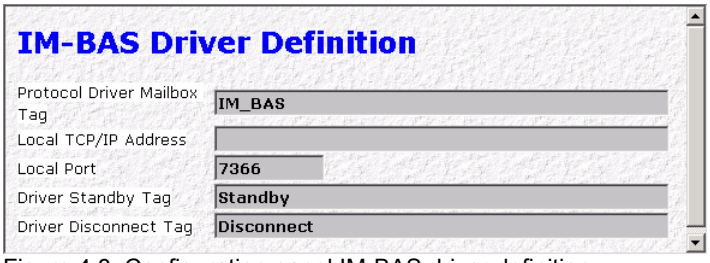


Figure 4.8. Configuration panel IM-BAS driver definition.

Protocol Driver Mailbox Tag

The I/O-translator will send read and write requests to IM-BAS driver using IMX, only one mailbox is used to send requests to the protocol driver. For the driver this the ‘receive’ mailbox, every translator will wirte

requests for the driver in this mailbox. To make sure IMX is working properly the mailbox tagname entered should only be used for sending requests to the IM-BAS driver!

entry Required
entry type Tag name, tag type: MAILBOX

Local TCP/IP Address

This entry is optional, only if there is more then one Ethernet card present in your computer system the field may require an address. If this field is left blank, the driver will locate every Ethernet card in your system, and start listening to connect requests from BAS Control subsystems.
If a DNS name or (dotted) IP address is filled in only connection request from subsystems using the Ethernet card with that address or name are accepted. All other cards are ignored, connection requests arriving on these cards will also be ignored.

entry Optional
entry type String of up to 80 characters

Local Port

The connection between the driver and a device is defined with the IP-addresses of both systems and the port number on both sides. The IM-BAS protocol driver acts as a server, and will listen to connection requests from subsystems, therefore only the local port number is defined. The driver will listen for connection requests on this port for every Ip-address (see the previous field).

entry Required
entry type Numerical value

Driver Standby Tag

In a redundant FactoryLink system are two applications active, one of them can be in (hot) standby mode. For the IM-BAS protocol is defined that a susbsystem only communicates with one server, so the driver in the standby system will NOT communicate with susbsystems.

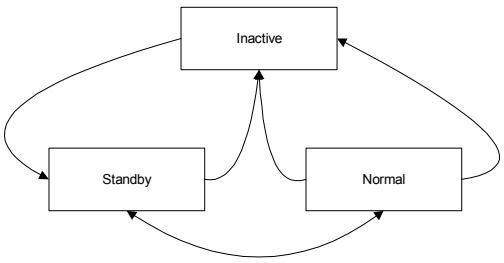


Figure 4.9 States of the driver task.

Driver Standby Tag	Value Standby Tag	Mode
NOT defined	-----	Normal
Digital tag defined	Off	Normal
Digital tag defined	On	Standby

To direct the driver in in standby or normal mode, you need to define a digital tag in this field, the value of the tag forces the driver to switch to one of the two modes. A value of zero ‘OFF’ instructs the driver to switch (or maintain) the normal mode, the switches to standby whenever the tag value goes from ‘OFF’ to ‘ON’.

entry Optional
entry type Tag name, tag type: DIGITAL

Driver Disconnect Tag

In a redundant FactoryLink system, with both servers running and not one in hot standby, both drivers will be in running in normal mode. As a result of maintenance or errors the subsystems may not all be connected to the same server. For normal operation this will be no problem, if one of the two systems fails the subsystems will try to connect with the other system, and continue to exchange data with the running server. However if both servers are running, but fail to communicate with each other, the scattering of the subsystems over both servers, leaves both servers with incomplete communication data. Set the tag specified in this field to ‘ON’ and all subsystems will try to connect to the ‘other’ server, so one server will have all the communications.

Incoming connection requests from subsystems are ignored while the tag value is 'ON', changing the tag value to 'OFF' will tell the driver to accept incoming connection requests.

entry	Optional
entry type	Tag name, tag type: DIGITAL

4.4.2 IM-BAS Device Definition

This table defines all the subsystems from which connection request are accepted. Acceptation of a connection is defined as connected on TCP/IP and application level. If there is a subsystem trying to connect, but not defined in this table, the TCP/IP connection is accepted, but life-signs are not returned to the subsystems. By accepting the connection on TCP/IP level the connection-requests coming from the subsystem will be at intervals with a minimum of the timeout for connection loss (defined in the subsystems). This will prevent heavy network loads in case a subsystem after a rejection immediately sends another connection-request.

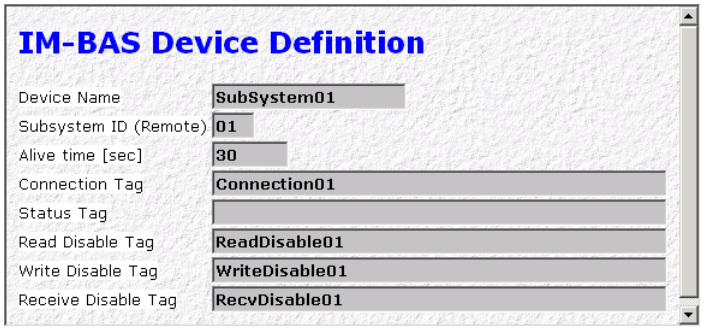


Figure 4.10 Configuration panel IM-BAS device definition.

For every subsystem or device (as it is called in context of the driver) several settings are defined in this table. Among them are a logical device name, only used in FactoryLink for referencing the subsystem with a logical user defined name. The device name is used to select the datasets for this device from table behind the current table.
The subsystem id is used to uniquely identify the subsystem on the network, just like the local system ID of the driver. Only after accepting the TCP/IP connection and reception of the first life-sign, the relation between connection and device is known, and is internally in the driver the connection linked to the device.
Life sign messages are exchanged between the driver and a subsystem for a connection on application level. An alive time in seconds is used to determine when an connection on application level is lost. The connection on application level is lost when the time between two life sign exceeds the given value, and there still is a connection on TCP/IP level.
The actual state of the connection is stored as a tag value in the real time database of FactoryLink.

Device Name

Logical name assigned by the user to represent a particular communication device. This field is used as a selection criterion for the next panel: IM-BAS Dataset Definition.

entry ✘ Required
entry type String of up to 16 characters

Subsystem ID (Remote)

The subsystem id is used to uniquely identify the subsystem on the network, just like the local system ID of the driver. Only after accepting the TCP/IP connection and reception of the first life-sign, the relation between connection and device is known, and is internally in the driver the connection linked to the device.

entry ✘ Required
entry type String of up to 2 characters

Alive Time [sec]

The alive time specifies the maximum number of seconds admitted between the reception of two life sign messages. If the time between two successive life signs exceeds the entered value, the connection (on application level) is assumed to be lost, but the driver does not close the connection.
This value is used to timeout read and/or writes, if the time between command and response is greater then the alive time, the command fails with a timeout error. If the command fails the connection is assumed to be lost and the driver will close the connection, on application level and TCP/IP level.

entry Required
entry type Numerical value, with 123 as default
default If not configured the value defaults to 20 seconds

Connection Tag

The connection status for every device can be made available throughout the application, the only thing to do is specify a tag in this field. The value of the tag will represent the connection state of the device. Please remember that a life sign is needed before the subsystem id is known of the device to which a driver has a TCP/IP connection. Due to this definition, a device has still the disconnected state after acceptation of a TCP/IP connection request, but before a life sign message is received. Before receiving the life sign, there is no relation between a device (or actually the subsystem id) and the TCP/IP connection.

entry Optional
entry type Tag name, tag type: DIGITAL, ANALOG, MESSAGE

Action	Digital tag	Analog tag	Message tag
Listening for TCP/IP connection	0	0	Listening, no connection
TCP/IP connection accepted	0	0	Listening
Life sign received	1	1	Connected
Disconnecting	0	2	Disconnecting

The string-values for the message tag are defined in the task translation file: im_bas.txt. This file is located I the directory {FLINK}/msg/en, depending on the selected language for the FactoryLink system the last sub-directory can be ‘fr’ for French texts or ‘de’ for German texts. For flexible configuration of these translation texts, there is a second directory to store this file: {FLAPP}/msg. The different location implicates that the file is part of the application and will travel from one system to another along with saves and restores of the application. The translations are loaded into memory during start-up of the driver, first from the file in the FactoryLink system directory and next from the translation file in the application directory. Entries in the second file are loaded on top of the previous ones, a duplicate entry will override an existing translation in memory.

Figure 4.11. Translation examples for 'connection' texts.

Status Tag

The IM-BAS protocol driver writes to this real-time database element, the value of the tag indicates the status of communication commands. The status element can be referenced by any task to handle communication error situations. A value of zero indicates no error (success), any other value represents an error, see the next chapter how to solve or prevent errors. A detailed list of all possible error codes also can be found in the next chapter.
If a digital tag is used to evaluate errors, only the error situation is indicated, the value of the digital is ‘ON’ to indicate an error. A digital tag only has two values (‘ON’ and ‘OFF’), and cannot have the error value.

entry Optional
entry type Tag name, tag type: DIGITAL, ANALOG

Read Disable Tag

Real-time digital database element used to enable/disable read-commands for the logical device. Read-commands are enabled in case there is no tag defined, or the status of the digital tag is ‘OFF’. Read-commands are disabled if a tag is defined and the status of the tag is ‘ON’.

entry Optional
entry type Tag name, tag type: DIGITAL

Write Disable Tag

Real-time digital database element used to enable/disable write-commands for the logical device. Write-commands are enabled in case there is no tag defined, or the status of the digital tag is ‘OFF’. Write-commands are disabled if a tag is defined and the status of the tag is ‘ON’.

entry Optional
entry type Tag name, tag type: DIGITAL

Receive Disable Tag

Real-time digital database element used to enable/disable receive-commands for the logical device. Receive-commands are enabled in case there is no tag defined, or the status of the digital tag is ‘OFF’. Receive-commands are disabled if a tag is defined and the status of the tag is ‘ON’.

entry Optional
entry type Tag name, tag type: DIGITAL

4.4.3 IM-BAS Dataset Definition

The 'IM-BAS Dataset Definition' panel is used to define one or more datasets for the selected device. Different logical device names are selected in the panel 'IM-BAS Device Definition'.

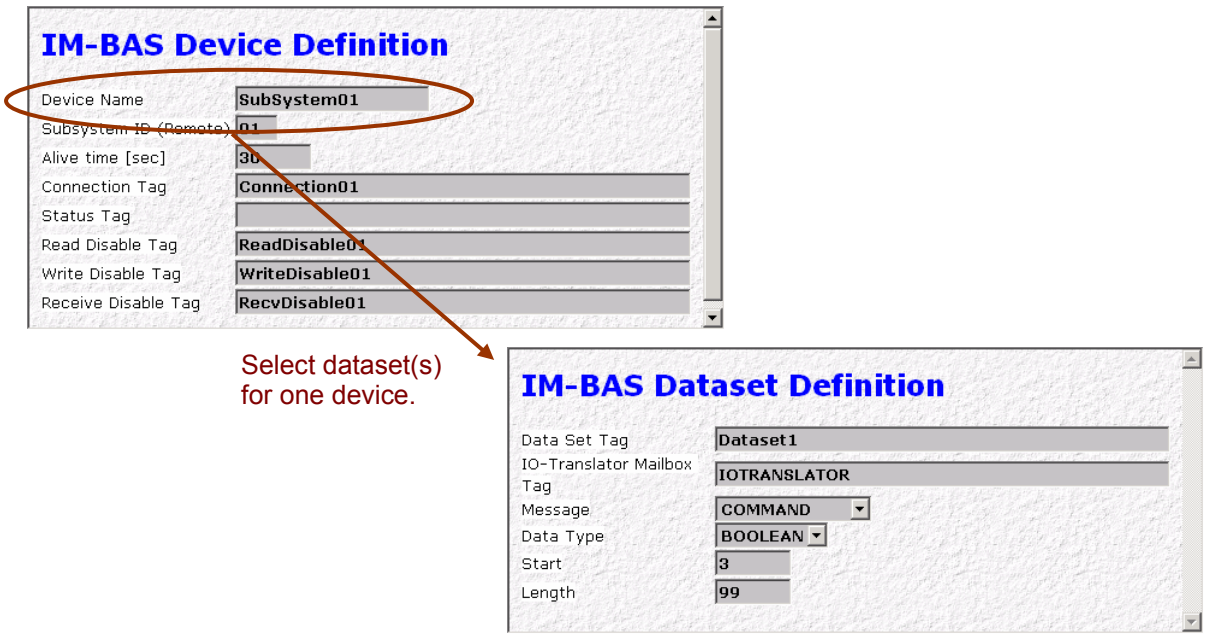


Figure 4.12 Configuration panel IM-BAS dataset definition.

A dataset is a contiguous data area in the device, and is identified with its type, start and length. The IO-Translator mailbox defines the IO-Translator to use for this data set. The maximum number of messages in this mailbox at any given time before the driver stops writing messages into the mailbox is defined in the ini-file.
The dataset tag is a digital tag used to identify the dataset, the same data set tag will be present in the configuration for the IO-Translator. The user is responsible for defining an unique tag name for every data set.
A mail message can have more then one recipient definition, two or more mail messages can share the same definition of recipient(s). The list of recipients for one mail message is defined with the name in the first field of the 'send mail definition' panel.

IO-Translator Mailbox Tag

Tag name of a translator mailbox element, to be referenced by the IM-BAS protocol driver task (**N.B.** IMX must be supported by the translator task). The I/O translator task uses this mailbox to receive data from the IM-BAS protocol driver.

entry Required
entry type Tag name, tag type: MESSAGE

Dataset Tag

Tag name representing a (unique) logical name for a dataset. The tag is used internally in the I/O translator and protocol driver for activating a dataset read from the external device. The I/O translator and the protocol driver internally reference a dataset by the tag name defined in this field. As a consequence the tag specified must be unique. The same tag should be used in configuration of the I/o translator to select the conversions for the dataset.

entry Required
entry type Tag name, tag type: DIGITAL

Message

This field defines the type of the data area in the logical device (subsystem). The datasets that can be present in a device, which supports the IM-BAS protocol, are summarized in the table below. The message types described in the table define the different kind of possible datasets, for a detailed description see [BAS-IRS]. The boundaries are directly related to the addressing: one address relates to the size of the boundary, e.g. word boundary has word-addresses, every address is an value of two bytes. For the function column the following characters are used: R – for read functionality.
W – for write functionality.
U – unsolicited receive functionality.

Message type, Data type	Func-tion	Start	Length	Boundary	Conversion I/O translator
Command, Bool {IMBC01}	W	0	Subsystem dependent	Byte	Byte
Parameter, Bool {IMBC02, IMBC03}	RW	0	Subsystem dependent	Byte	Byte
Parameter, Integer {IMBC02, IMBC03}	RW	0	Subsystem dependent	Short	Word
Parameter, Long {IMBC02, IMBC03}	RW	0	Subsystem dependent	Long	Long
Parameter, Real {IMBC02, IMBC03}	RW	0	Subsystem dependent	Long	DBL
Parameter, String {IMBC02, IMBC03}	RW	0	String, length = 20 bytes.	Long	MSG, length = 20
Status, Bool {IMBC11}	R	0	Subsystem dependent	Byte	Byte
StatInfo, Bool {IMBC12}	R	0	Subsystem dependent	Byte	Byte
StatInfo, Integer {IMBC12}	R	0	Subsystem dependent	Short	Word
StatInfo, Long {IMBC12}	R	0	Subsystem dependent	Long	Long
StatInfo, Real {IMBC12}	R	0	Subsystem dependent	Long	DBL
StatInfo, String {IMBC12}	R	0	Subsystem dependent	Long	MSG, length = 20
StatInfo-Del, Bool {IMBC12}	R	0	Subsystem dependent	Byte	Byte
StatInfo-Del, Integer {IMBC12}	R	0	Subsystem dependent	Short	Word
StatInfo-Del, Long {IMBC12}	R	0	Subsystem dependent	Long	Long
StatInfo-Del, Real {IMBC12}	R	0	Subsystem dependent	Long	DBL
StatInfo-Del, String {IMBC12}	R	0	Subsystem dependent	Long	MSG, length = 20
Exception {BCIM21}	U	0	4 bytes.	Byte Address 0 = Time stamp, 14 bytes. Address 14 = Equipment code, 16 bytes. Address 30 = Exception level, 1 byte. Address 31 = Description, 50 bytes	Depends on the variable, there are four different variables in the dataset.

The id between {} in the column 'Message type' refers to the command id's used in the document [BAS-IRS].

Note that exception write behavior depends on the type of data area. To know how an exception write is executed, one should regard the boundary of the data area and the element size involved with the exception write. For e detailed description refer to chapter 2.

entry Required
entry type String of up to 15 characters, selection is made form:
COMMAND, PARAMETER, STATUS, STATINFO, STATINFO-DEL , EXCEPTION, DEVX

The data type 'DEVX' is not an actual data area in the subsystem, it is only a data area contained in and maintained by the driver. This type of data can only be read by the IO-translator, it has values like IP-address, device name, total count of read and write commands, the time between two reads, the time

needed to read a data area from the subsystem etc. The boundary for this dataset is byte, the addressing of the data fields is according to the table below.

Internal dataset DEVX			
Address	**Name**	**Type, Length in bytes**	**Description**
0	Device	MSG, 20	Name of the device, this the of this device as entered in the 'IM-BAS Device Definition' table.
20	LocalIP	MSG, 20	Local IP address, used for communication with this subsystem, the IP-address has the form: 0.0.0.0
40	RemoteIP	MSG, 20	Remote IP address, used by the subsystem to communicate with the IM-BAS driver, the IP-address has the form 0.0.0.0
60	LocalPort	WORD, 2	Local port number used for TCP/IP communication
62	RemotePort	WORD, 2	Remote port number used for TCP/IP communication
64	Connection	WORD, 2	Connection status, value is also presented on the 'Connection Tag'
66	Count1	LONG, 4	Total number of read commands processed
70	Count2	LONG, 4	Total number fo write commands processed
74	Count3	LONG, 4	Not used, always zero
78	DataTime1	LONG, 4	Time in milli-seconds to process a read command
82	DataTime2	LONG, 4	Time in milli-seconds to process a write command
86	DataTime3	LONG, 4	Not used, always zero
90	WaitTime1	LONG, 4	Time in milli-seconds between two successive read commands
94	WaitTime2	LONG, 4	Time in milli-seconds between two successive writes commands
98	WaitTime3	LONG, 4	Not used, always zero
102	Spare	MSG, 100	Not used, reserved for feature use

The drier ignores the start and length definitions entered in the next columns, it sets these values to zero (start address) and the maximum length of this data area.

Data Type

Every message has its own is data type, the choices are: boolean, integer, long, real and string. Depending on the message type not all choices are possible, see the table in the previous filed description. The combination of message and data type uniquely defines a data area in a subsystem. The exception on this rule is the 'exception' message, this message has a fixed layout, as described in the table above. For this message You should use the entry 'ignore' in this field, the driver will not use the entry of this field in combination with the message type 'exception'.

entry ✘ Required
entry type String of up to 15 characters, selection is made form:
IGNORE, BOOLEAN, INTEGER, LONG, REAL or STRING

Start

Start address number of the dataset, it is the start address of the data area in the subsystem. This start address is the same as the field index defined in the subsystem for the particular data area. There are two situations for which the start address and the length (set in the next field) are calculated different, here in the driver and in the IO-translator. The driver task always uses the addressing of the subsystem, that is element number as start address and the length is the total number of elements. First are messages with the data type string, in the IO-translator the start address will be a start-address for boundary long. The length of strings is fixed, 20 bytes for each string. An example: string 5 will have start address 25 (= 5 * 5) but still a length of 20 bytes for the MSG conversion.
For the message type 'exception' the driver overrules the entries for 'start' and 'length', the start address is zero and the length is 14 + 16 + 1 + 50 = 81 bytes. This fixed internally, the entries you make here are always overruled!
The two differences in calculating the start address and length are a result of the boundary definition of the data area, the boundary for a data area can be found in the table above. The addressing is directly related to the boundary of the data area, the boundary defines the size (in bytes) of a single item in the data area. For integers the boundary is integer, as a result the start address and field index will match.

Entry ✘ Required

entry type Numerical value: 0 – 999

Length

Length of the dataset in elements, the length used by the IO-translator is a number of bytes, words, longs or doubles, depending on the type of the dataset, see the table for the field ‘Message’. The minimum length allowed is ‘1’.

Entry Required
entry type Numerical value

5 Error codes and messages

5.1 Error codes

Error numbers are part of the error messages described in the next paragraph, and the error number is written to the 'status-tag' of a device in case of an error. See the previous chapter how to define a 'status' tag for a device (or subsystem). In the table the error codes are grouped for location where the error occurred.

Error #	FactoryLink errors
1	Internal error, general error code
2	Out of memory, memory allocation error because of a system exhaustion or FactoryLink allocation
3	Unable to map kernel semaphores
4	Can not locate real-time database for given {FLNAME}
5	Can not locate real-time database for given {FLDOMAIN} + {FLUSER}
6	Incorrect function, reserved error code
7	Incorrect argument, bad input parameter passed into function
8	TAG structure received with an invalid offset member
9	TAG structure received with an invalid type or offset member
10	Null pointer assignment
11	Change read/wait function call failed because no changes for given tag array
12	Procedure table full, maximum number of tasks has registered for the given {FLNAME}/{FLDOMAIN}/{FLUSER}
13	Bad procedure name
14	Bad user name
15	Bad option
16	Incorrect checksum
17	No options
18	No key
19	Bad key
20	No port available
21	Port busy
22	Initializing task with same name as one currently running
23	Attempted to unlock the kernel, without holding a lock
24	Unable to attain lock to the kernel
25	Attempted to release a lock that has expired on the kernel
26	Invalid task ID passed into fl_wait()
27	Termination flag for given task ID set to ON
28	Internal error, failed to allocate mailbox queue
29	Internal error, inconsistent queue size
30	Reserve, no tag list
31	Reserved, tag list changed
32	Failure occurred awakening task sleeping on the kernel
33	No signals pending for caller
34	Task awakened by signal from another task
35	Called mailbox function with a tag not f type MAILBOX
36	No mailbox message pending for mailbox tag
37	Can not read a mailbox message targeted for another task
38	Attribute name table full
39	Invalid attribute name
40	Cannot find requested attribute
41	Attempted to start application with same {FLNAME} as one running
42	Unable to map to kernel memory area for {FLNAME}
43	Task protection bit not set for initializing task
44	Attempted to start restricted task from running on a lite system
45	Unable to get or release semaphore
46	Mailbox message space exhausted
50	Float (FLP) is not-a-number
51	Float (FLP) is positive infinity
52	Float (FLP) is negative infinity
101	FLDTP general failure indicator

Error #	FactoryLink errors
102	FLDTP invalid argument passed dtp function
103	FLDTP invalid tag passed to dtp function
104	FLDTP memory allocation failure
105	FLDTP can not find requested object
106	FLDTP illegal call to procedure at given time. Some dtp functions are not re-entrant
107	FLDTP kernel function call failed.
108	FLDTP tag of unknown type passed into function
109	FLDTP no tag changes are pending

Error #	IMX errors
150	Bad message type
151	Message with dataset control tag not found in queue
152	No messages available to query
153	Bad receive mailbox tag
154	Bad mailbox send tag
155	Bad dataset control tag
156	Message cannot be adjusted
157	Operation to big for variable
158	Unknown boundary
159	Function not supported
160	No message for this index present
161	The remote dataset Is not defined on this system
162	The received dataset was not registered
163	The message is not queued
164	Message is rejected due error in the remote IMX
165	Illegal method of addressing bits on bit boundary
166	Element cannot be written
167	Invalid buffer specified
168	Block write function impossible
169	Maximum number of messages in MBX reached
170	No memory left
171	Error registering standard dataset
172	Error writing message in pipe
173	Not supported IMX message
174	Error creating pipe
175	Error starting threat
176	Error server connecting to pipe
177	Error child connecting to pipe
178	Error reading the pipe
179	Error number of bytes read from pipe
180	Error writing into the pipe
181	Error number of bytes written into the pipe
182	Error reading dataset from the mailbox
183	No communication buffers assigned
184	Error decrementing semaphore
185	Error writing to pipe, no space left
186	Error querying no. of messages for translator
187	Max. No. of messages in translator MBX reached

Error #	TCP/IP layer errors
210	Creation of socket failed
211	Unexpected handle for socket, internal error
212	Device has no valid dataset registered for read or write command.
220	Creation of receive thread failed
230	Receive failed, no data received, most likely the remote partner did close the connection
231	Send failed, no data send, partner closed connection or internal network resource failure
232	Sending of message to TM system failed, unknown command id

Error #	IM-BAS protocol errors
300	Communication error, command rejected, driver is in standby mode
301	Request for reading dataset rejected, read functionality is disabled by user
302	Request for writing dataset rejected, write functionality is disabled by user
303	Incoming data is not send to translator, receive functionality of device is disabled by user
305	Maximum number of messages reached in mailbox for IO-translator (or ECI), number is configured in the im_bas.ini file
310	Read or write request is ignored, device is not connected on application level
311	Memory allocation error, requested amount of memory is allocated
312	Dataset adjusted to allowed minimum or maximum values for start address and or length, informational error code only
320	Source id in received data is not 'IM'
321	Destination id in received data is not 'BC'
322	Invalid version found in received data, protocol version number of received data does not matches with implemented version in driver
323	Unexpected type received as message identifier
324	The sequence number is not equal to '00' for received life sign
325	Received message has no terminator
326	Invalid functionality (read/write) requested for the given message type
327	No data in the message
328	Unknown data type found in received message
329	Life sign from subsystem not received within timeout interval
330	Unexpected message received, data is ignored
331	Unexpected data type received for a read/write request, data is ignored
332	Length or start address of block write does not match with defined start address and/or length
333	Addressing information in received data from sub-system is not processed
430	Invalid exception write, value is less then boundary. This kind of exception writes is not supported by the driver
431	Encode write is not supported
432	Time out error, remote device did not respond within alive time
440	Lower or upper limit set to one or more data values

5.2 Error messages

If an error condition is present in the application, an appropriate message is send to the FactoryLink kernel. The message can be seen in the 'run-time manger' screens that are part of the starter application. In the standard screens the error message will appear to the right of the 'IM-BAS' task. A red button marks errors, these errors are not critical the task can and will continue. Severe errors cause a FactoryLink task to stop, these errors are marked with an error description and a gray button (same as an inactive task).

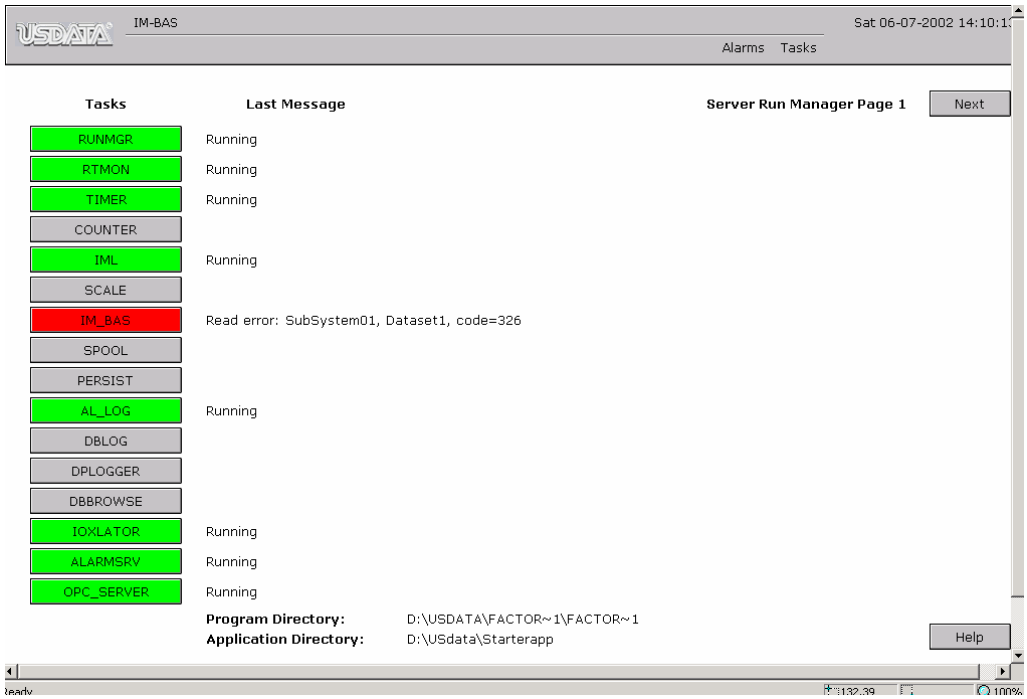


Figure 5.1. Run-time manager screen.

IM_BAS **Can't open CT file**

The application was unable to access the task specific CT-file, the file was not found in the application directory.

Possible solutions: Make sure the IM-BAS protocol driver has a (valid) configuration in the Configuration Explorer, if so please check if this is the same application as you started.
Check if there is a configuration file 'im_bas.ct present in the application, directory: {FLAPP}\shared\ct. If there is no such file present, see if FactoryLink is instructed to generate this file. In the ASCII-file '{FLINK}\ctgen\ctlist' there must be an entry: **im_bas: im_basm im_basp im_basd**. If this entry is missing, try to re-install the IM-BAS protocol driver.

IM_BAS **No triggers found in CT-file**

The SendMail task found a configuration file 'im_bas.ct' in the application directory, however there are no triggers defined for the task.

Possible solutions: Make sure the IM-BAS task has a valid configuration in the Configuration Explorer, if so please check if this is the same application as you started.
Check if there is a configuration file 'im_bas.ct present in the application, directory: {FLAPP}\shared\ct. If there is no such file present, see if FactoryLink is instructed to generate this file. In the ASCII-file '{FLINK}\ctgen\ctlist' there must an entry: **im_bas: im_basm im_basp im_basd**.
If a configuration file (CT-file) present, please generate a list file of the configuration, before contacting the support desk. The listing file is generated with a utility called "ctlist.exe", check first if the program is present in the {FLINK}\bin directory, if so open a command prompt and go to the application directory. In the application directory is heck if there is a configuration file 'im_bas.ct present in the application, directory: {FLAPP}\shared\ct. In the application directory go to the subdirectory: shared\ct, from there run the

command: “ctlist im_bas.ct”. The output can be redirected to a file, e.g. appending: “> d:\sendmail.txt” to the command line. Contact the support desk for more help.

IM_BAS **Failed to load winsock stack**

The application was unable to load the stack for Windows Socket version 2.

Possible solutions: The Windows socket is part of the operating system, for NT4 and Windows 2000. If this stack is missing, the operating system is not installed without network or the settings of the operating system are corrupted. Try reinstalling the operating system, specific the network part.

IM_BAS **Winsock stack does not support requested version**

The application was able to load the Windows socket stack, but the requested minimum version (this is 1.1) is not supported by the system.

Possible solutions: The Windows socket is installed on the system, but the version is less then the minimum requested version 1.1. Update your system with a version of Windows sockets, normally this should at least be 2.0.

IM_BAS **No memory**

The application was unable to allocate the amount of memory needed.

Possible solutions: Increase the amount of memory, either real or virtual memory.

IM_BAS **FactoryLink error %d**

An internal error occurred, during interaction with the FactoryLink kernel, in the message there is an error code. An explanation of the code is given in the previous table.

Possible solutions: The solution or prevention the error depends on the error code given in th eactual message. Please call support if you need further assistance.

IM_BAS **Task was unable to get second taks-id from kernel**

The driver was unable to get a second task-id from the kernel. The use of the second task id is requested in the ini-file of the task.

Possible solutions: If a task is unable to get a second task id, th emost likely course is the lack of a free task slot. It is a second task id, which means the definition of this task id is not found in the ‘system configuration’ tables of FactoryLink, but there must be an empty task slot present. An empty task slot means that less then 31 tasks are defined in the system configuration table.

IM_BAS **Listen thread reported WinSock error: %d**

An error on the network layer occurred in the thread listening for incoming connection requests of subsys-tems.

Possible solutions: The error code reported in message can be found in the Microsoft documentation ‘Winsock’, or in an explanation of general error codes. The error code will give detailed information about the error, and will indicate possible solutions.

IM_BAS **Driver mode standby, command for %s cancelled**

Driver is standby mode and the IO-translator initiated a read or write command. In standby mode read and write commands are not processed, the driver responds with an error report to the IO-translator. The IO-

translator will not update tags associated with a read command that reports this error, the last values read without an error will still be the tag values after completion of the command.

Possible solutions: To avoid this error message, don't generate read or write commands while the driver is in standby mode, or put the driver back into normal mode.

IM_BAS **Device %s not connected on application level**

Device or subsystem has no connection on application level with the TM-system and the IO-translator initiated a read or write command. Without a connection on application level read and write commands are not processed, the driver responds with an error report to the IO-translator. The IO-translator will not update tags associated with a read command that reports this error, the last values read without an error will still be the tag values after completion of the command.

Possible solutions: To avoid this error message, don't generate read or write commands while the driver is not connected.

IM_BAS **Write disabled for device %s**

Write functionality for the reported device is disabled and the IO-translator initiated a write command. In a disabled state a write command is not processed, the driver responds with an error report to the IO-translator.

Possible solutions: To avoid this error message, don't generate write commands while this type of commands is disabled for the device, or enable the write functionality for this device.

IM_BAS **Read disabled for device %s**

Read functionality for the reported device is disabled and the IO-translator initiated a read command. In a disabled state a read command is not processed, the driver responds with an error report to the IO-translator. The IO-translator will not update tags associated with a read command that reports this error, the last values read without an error will still be the tag values after completion of the command.

Possible solutions: To avoid this error message, don't generate read commands while this type of commands is disabled for the device, or enable the read functionality for this device.

IM_BAS **Read error: %s, %s, code=%d**

A read command for the specified device and dataset completes with the reported error. The same error code is reported by the IO-translator.

Possible solutions: A possible solution for this error depends on the cause of the error; the cause is determined by examining the error number. For a description of the error, see the table in the beginning of this chapter.

IM_BAS **Write error: %s, %s, code=%d**

A write command for the specified device and dataset completes with the reported error. The same error code is reported by the IO-translator.

Possible solutions: A possible solution for this error depends on the cause of the error; the cause is determined by examining the error number. For a description of the error, see the table in the beginning of this chapter.

IM_BAS **IMX error:%d, sending data for %s**

The driver failed to send a reply to the IO-translator, the error code and dataset name are found in the error message

Possible solutions: A possible solution for this error depends on the cause of the error; the cause is determined by examining the error number. For a description of the error, see the table in the beginning of this chapter.

IM_BAS **IMX error:%d, sending error for %s**

The driver failed to send an error-reply to the IO-translator, the error code and dataset name are found in the error message

Possible solutions: A possible solution for this error depends on the cause of the error; the cause is determined by examining the error number. For a description of the error, see the table in the beginning of this chapter.

IM_BAS **IMX error:%d, initialization failed**

The driver failed to start, initialization of the IMX library failed, the cause is given by the error code in the error message.

Possible solutions: A possible solution for this error depends on the cause of the error; the cause is determined by examining the error number. For a description of the error, see the table in the beginning of this chapter.

IM_BAS **IMX dataset registration failed %s**

The driver failed to start, the specified dataset failed to register with the IMX library.

Possible solutions: There can be errors in your application, e.g. the dataset control tag I the ct-file is not a digital tag (or non-existent). Rebuilding all the configuration files might help, from the command line run 'ctgen –r'. I

IM_BAS **IMX dataset not unique: '%s'**

The reported dataset control tag is not unique in the configuration of the driver. The dataset control tag may only be used for one (and one only) dataset.

Possible solutions: Use every dataset control tag only once for all the datasets. If the configuration meets this requirement, try to rebuild the configuration files with the command 'ctgen –r'. Run this utility from the command line and set the environment variable FLAPP to your application directory prior to running the utility.

IM_BAS **IMX semaphore creation failed**

During initialization of the IMX library the creation of a semaphore failed, this is a fatal error.

Possible solutions: Contact the helpdesk.

IM_BAS **IMX failed to start thread**

During initialization of the IMX library the creation of a thread failed, this is a fatal error.

Possible solutions: Contact the helpdesk.

6 Exception alarms

Subsystems will send exception messages to the FactoryLink system on their own initiative, these are so called unsolicited receives for the IM-BAS driver. In most cases these messages are interpreted as alarms for the subsystem, which send the message. This appendix describes a configuration for the Alarm logger task, the IO-translator, Interpreted Math and Logic and the IM-BAS driver to accept incoming exception messages and generate an alarm. The alarms are made visible with the standard alarm OCX-control in the Client Builder application. Every exception message has a severity level indicator, this is a byte-value with the values '1', '2', '3' or '4' (the decimal equivalents are 49 until 52). To be able to process the alarms of different in case the severity level differs, there are four alarm groups defined. For every alarm group the acknowledgement requirement, coloring and logging of the alarm text's can be different.
Alarms will be generated in the following way:

1. The driver receives an exception message and sends this message to the IO-translator task using a mailbox tag (IMX)
2. The IO-translator evaluates the mailbox message and updates tags in the appropriate dataset with the newly received values. After updating the tags a completion tag is triggered, this tag is used to trigger an IML procedure.
3. The IML task is triggered whenever an exception message is received and the data is stored in tags. The IML procedure uses the severity in the exception message to trigger an alarm in the corresponding alarm group. As there are four severities, there are four alarm groups defined for the alarm logger. In every group one alarm for every equipment code is defined, the equipment code is used as the 'area' name for an alarm.
4. The alarm logger task is triggered by the raised alarm condition and generates an alarm, in the alarm text the exception description is a parameter. The timestamp of the exception message is used to supply the alarm logger with an alarm-timestamp for the specific alarm. The condition used to trigger the alarm is the 'toggle' condition (TGL), this condition matches most closely the alarm-behavior of an exception message: The alarm is only active on the moment of reception, that means the alarm comes and goes to normal. This is exactly what the 'toggle' condition is, the alarm is only active when the tag value changes. So the alarm logger detects a change, generates an alarm, and the alarm goes to normal (after reading the changed value the value no longer changes…).

The configuration presented uses a minimum of entries in the configuration to generate alarms for exception messages received. By doing so a new subsystem is easily configured with the option of alarm generation after receiving an exception message. The next steps will describe the principles for this configuration.

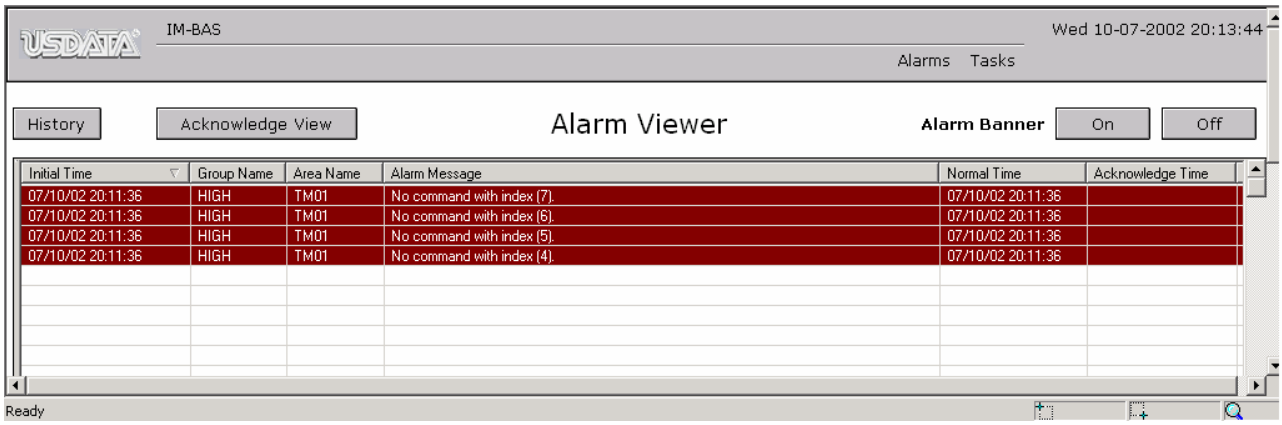


Figure 6.1. Alarm screen with exception message.

6.1 IM-BAS driver

This part of the configuration defines the datasets used to receive the exception messages. For every subsystem a dataset to receive exceptions should be defined. If the driver receives an exception message, the subsystem id in the message header is used to identify the subsystem and find this subsystem in its configuration. If the subsystem is defined with the Configuration Explorer, and a dataset with type ‘exception’ is defined for this subsystem the exception data is send to the IO-translator, using IMX.

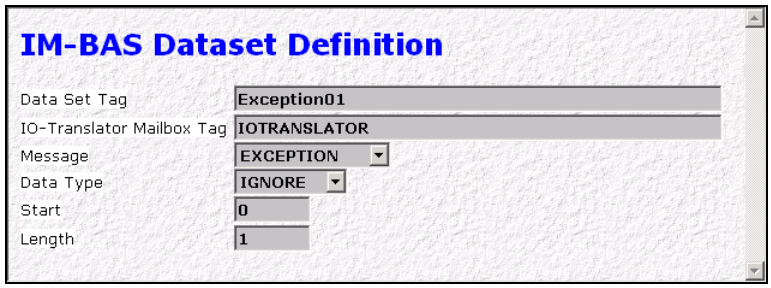


Figure 6.2. Dataset definiton fro exception message.

In the figure above the definition of a dataset used to receive exception messages is given. For every subsystem, or device (the name used for a subsystem in the driver configuration) one (and only one) dataset should be defined to make the reception of exception messages possible. The difference between two dataset definitions will be the name of the dataset control tag, and of course the device for which the dataset is defined, see the figure below.

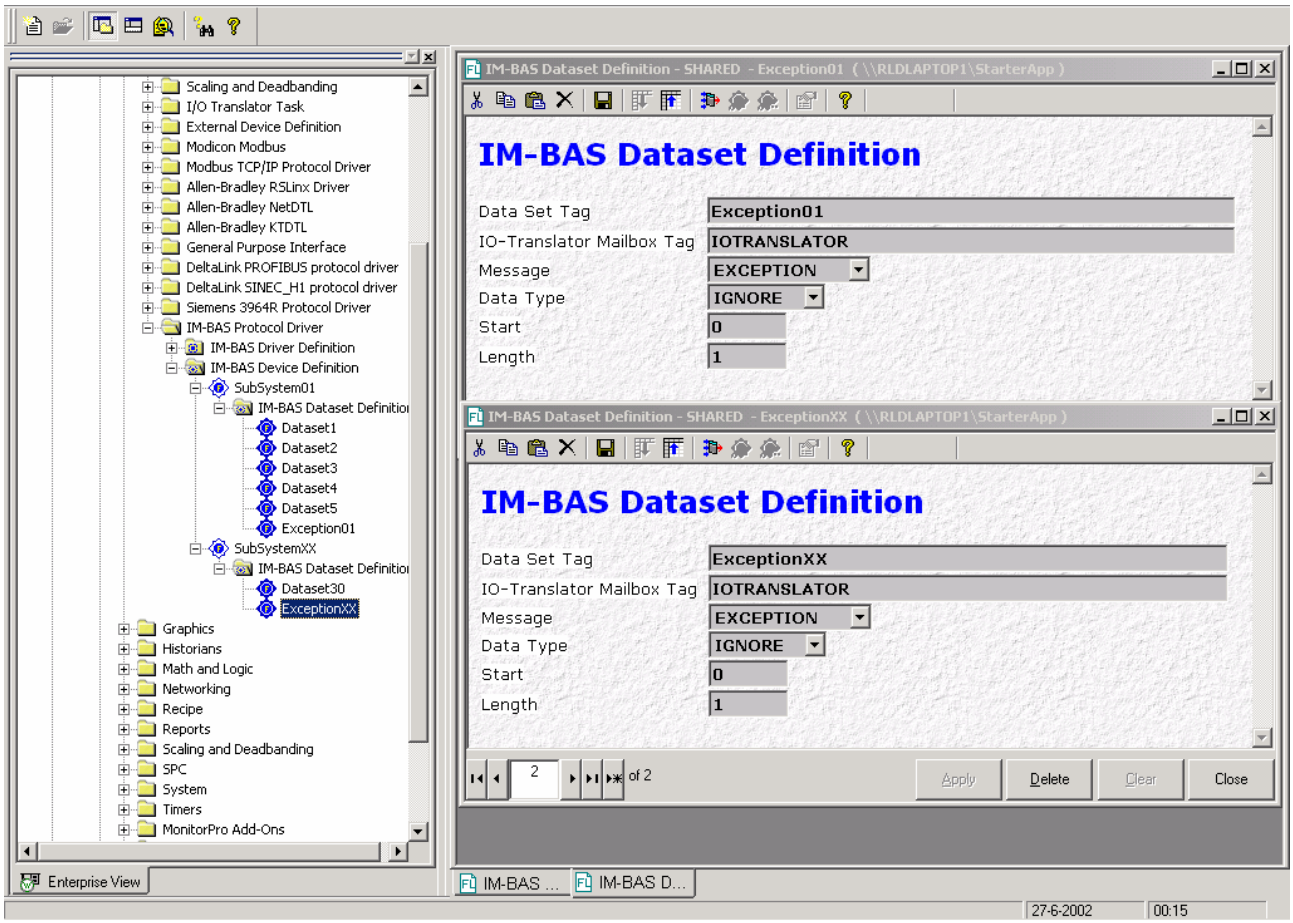


Figure 6.3. Dataset definitons.

The data of the exception message is send to the IO-translator task using IMX, the dataset control tag is used to identify the data send to the IO-translator task. The same dataset control tag needs to be defined for the IO-translator task; this allows the task to associate FactoryLink tags with the dataset (and of course the data itself).

6.2 IO-translator

For the IO-translator every dataset for exception messages are defined, see the figures below for an example. Every dataset for an exception message has the same layout, and even the same tags for the data in the message are used. The use of the same tags makes it very easy to configure more then one dataset (note that every dataset represents a different subsystem or device) for exception messages. Just define a dataset configuration and use the same name for the field: Data Set Name, automatically the already defined data-tags are associated with this dataset.

	Data Tag	Dimension	Address	Bit or Length	Conv
1	IO_TimeStamp	1	0	14	MSG
2	IO_Equipment	1	14	16	MSG
3	IO_Level	1	30		BYTE
4	IO_Description	1	31	50	MSG
*					

Figure 6.4. Dataset tags for exception message.

Instead of the completion tag the synchronization is used to generate a trigger after receiving data and updating the data tags in the dataset. Both tags are forced to 'ON' when data is received and the data-tags are updated. The synchronization tag adds another functionality to the completion trigger, it allows for synchronization of received data. Setting the digital tag to 'ON' instructs the IO-translator task to hold incoming data for the dataset, newly arrived data is not used to update the data-tags in the dataset. The data is send by the driver to the IO-translator using a mailbox tag, the data will stay in the mailbox tag as long as the value of the synchronization tag is 'ON'.
The **same** synchronization tag is used for every dataset defined for exception messages. After receiving one exception message and updating the data-tags, this sync-tag is automatically set to 'ON' (by the IO-translator task itself). As a result all other incoming exception messages (received from the driver) are buffered in the mailbox tag used to communicate with the IM-BAS driver.

I/O Translator Dataset Definition

Data Set Name	EXCEPTION
Data Set Control Tag	Exception01
Read Trigger	
Write Trigger	
RD	N
EW	N
BW	N
UR	Y
Abs	Y
Status	StatusExc01
Read Completion	
Write Completion	
Unsolicited Rcv Completion	
Read Disable	
Write Disable	
Unsolicited Rcv Disable	
Read Sync.	
Write Sync.	
Unsolicited Rcv Sync.	SyncException

I/O Translator Dataset Definition

Data Set Name	EXCEPTION
Data Set Control Tag	ExceptionXX
Read Trigger	
Write Trigger	
RD	N
EW	N
BW	N
UR	Y
Abs	Y
Status	StatusExcXX
Read Completion	
Write Completion	
Unsolicited Rcv Completion	
Read Disable	
Write Disable	
Unsolicited Rcv Disable	
Read Sync.	
Write Sync.	
Unsolicited Rcv Sync.	SyncException

Figure 6.5. Dataset control for exception alarms.

Now is guaranteed that the alarm logger is able to generate an alarm and read the data-tags associated with the exception message, while these tags are consistent. A newly arrived exception message will not overwrite the data-tags! After triggering of the alarm, the alarm logger will fire an event, which causes a reset

of the synchronization tag, the value is set to ‘OFF’. And this allows the IO-translator to update the data-tags with a newly arrived exception message etc.
The use of the synchronization tags as described above makes it possible to define only one set of data-tags for all the ‘exception’ datasets (and of course for every device or subsystem). Adding a new device or subsystem requires for the alarm handling only the definition of a dataset, like the one in the figure above. Note that the data set control tag must be unique for every dataset (and most likely this applies also to the status tag).

Before the actual alarm can be generated by the alarm logger task the IML task will evaluate the data and generate a trigger to activate the alarm.

6.3 Interpreted Math and Logic

The IML task does some pre- and post-processing of the alarm data. With this processing the configuration of the alarm logger task is minimized to four alarm groups and one alarm in every group.
The pre-processing of the IML task evaluates the severity level of the exception; this can be high, medium, low or info (corresponding decimal values are 49, 50, 51 and 52, equal to the ASCII values ‘1’, ‘2’, ‘3’ en ‘4’). This severity level and equipment name is used to trigger one of the alarms defined for exception messages in the alarm logger.
The post-processing of the IML task detects whenever the alarm logger has evaluated the alarm condition, and then resets the synchronization tag to ‘OFF’. Allowing the IO-translator task to update the data-tags with new exceptions.
Only two procedures for the IML task are needed for the exception/alarm handling, one procedure for pre-processing and one procedure for post-processing.
The procedure for pre-processing is triggered every time the synchronization tag is forced to ’ON’ by the IO-translator task. The severity level is directly used to trigger one alarm in one of the four groups (the alarm triggered has match between equipment name and area definition), the alarm logger takes care of the alarm functionality.
The post-processing procedure is triggered with every status change of an alarm, and it merely rests the synchronization tag for the IO-translator.

	Trigger Tag	Procedure	Mode	Description
1		ExcAlarm	INTERPRETED	File declaration: alrams generated for exception messages
2	SyncException	ExcTrigger	INTERPRETED	Trigger exception alarms
3	ExcAlarmActive[0]	ExcResetSy	INTERPRETED	Reset exception message synchronisatoin
4	ExcAlarmActive[1]	ExcResetSy	INTERPRETED	Reset exception message synchronisatoin
5	ExcAlarmActive[2]	ExcResetSy	INTERPRETED	Reset exception message synchronisatoin
6	ExcAlarmActive[3]	ExcResetSy	INTERPRETED	Reset exception message synchronisatoin
*				

	Tag Name
1	IO_Level
2	IO_Description
3	IO_Equipment
4	IO_TimeStamp
5	ExcAlarmTrigger[0]
6	ExcAlarmStatus[0]
7	SyncException
8	Equipment[0]
9	Description[0]
10	TimeStamp[0]
*	

Figure 6.6. IML Configuration for exception handling.

IML file: ExcAlarm.prg

```
#Procedure used for file declaration
PROC ExcAlarm
BEGIN
END

DECLARE SHORT _ExvAlarmInit
DECLARE STRING _ExcEquipmentNames[ 100]
CONST _MaxEquipmentIndex           100

#=========================================================================
#Initialization procedure, this procedure links alrm index and area names
#of alarms. The area names of the alarms should be equal to the equipment
#names used in exception messages. In this demonstratio is assumed every
#subsystem incorporates only one equipment name.
#=========================================================================
```

```
PROC ExcInit
BEGIN

  _ExcEquipmentNames[   0] = "TM01"
  _ExcEquipmentNames[   1] = "TM02"
  _ExcEquipmentNames[   2] = "TM03"
  _ExcEquipmentNames[   3] = "TM04"
  _ExcEquipmentNames[   4] = "TM05"
  _ExcEquipmentNames[   5] = "TM06"
  _ExcEquipmentNames[   6] = "TM07"
  _ExcEquipmentNames[   7] = "TM08"
  _ExcEquipmentNames[   8] = "TM09"
  _ExcEquipmentNames[  10] = "TM10"

END #ExcInit


#=============================================================================
#Procedure to determine exception level in received data
#and trigger the alarm with the group name corresponding
#with the found level.
#There are four levels and four alarm groups, defining four alarm
#groups alllows you to define different (alarm) behaviour for
#every exception level.
#
#Trigger: Synchronisation tag of dataset for which exception messages
#are received.
#=============================================================================
PROC ExcTrigger
BEGIN
  DECLARE SHORT _level
  DECLARE SHORT _start, _end, _i
  DECLARE STRING _equipment
  DECLARE SHORT _ExcAlarmSync

  #save exception level, to reduce tag usage in IML
  _level = IO_Level - 49
  _ExcAlarmSync = 0

  #check for initialization
  IF (_ExvAlarmInit = 0) THEN
    CALL ExcInit
  ENDIF

  #Synchronisation tag is forced to ON by IO-Translator, trigger alarm
  IF (_level >= 0) AND (_level <= 3) THEN
    _start = _level * _MaxEquipmentIndex
    _i = _start
    _end = _start + _MaxEquipmentIndex
    _equipment = upper(IO_Equipment)

    WHILE (_i < _end)

      #search for corresponding area name
      IF (_ExcEquipmentNames[ _i mod _MaxEquipmentIndex] = _equipment) THEN

        #copy tag values
        Description[ _i] = IO_Description
        TimeStamp[ _i] = IO_TimeStamp

        #trigger alarm
        _ExcAlarmSync = 1
        ExcAlarmTrigger[ _i] == 1
        #stop the search
        _i = _end
      ENDIF

      _i = _i + 1
    WEND
  ENDIF

  IF (_ExcAlarmSync = 0) THEN
    SyncException = 0
  ENDIF
END #ExcTrigger
```

```
#=============================================================================
#Procedure to clear synchronisation tag used in IO-translator, the tag is
#set to OFF whenever the alarmstatus changes
#
#Trigger: The procedure is triggered with the tag ExcAlarmStatus, the same
#tag is used for every exception alarm. This is possible because on one
#point in time, only one alarm is triggered (becoming active). For resetting
#the synchronisation tag, we only want to know if the alarm is processed
#by the alarm logger, and the status tag just provides this information as
#a trigger (even if the same tag is used for more then one alarm).
#=============================================================================
PROC ExcResetSync
BEGIN

  #clear the exception synchronisation tag for the IO-translator
  SyncException = 0
END #ExcResetSync
```

6.4 Alarm Logger

As already stated four alarm groups are defined for exception messages, one group for each severity of an exception. Every group should be defined according to the requirements for the alarm handling (that is acknowledgement, coloring of the alarm text, logging to a database etc.).

	Group Name	Group Text	Group Composite Status Tag	Group Number Active Tag	Ack	Aud	Alarm Stat Print Dev	Log
1	HIGH	HIGH			YES	NO		YES
2	MEDIUM	MEDIUM			YES	NO		YES
3	LOW	LOW			YES	NO		YES
4	INFO	INFO			NO	NO		YES
*								

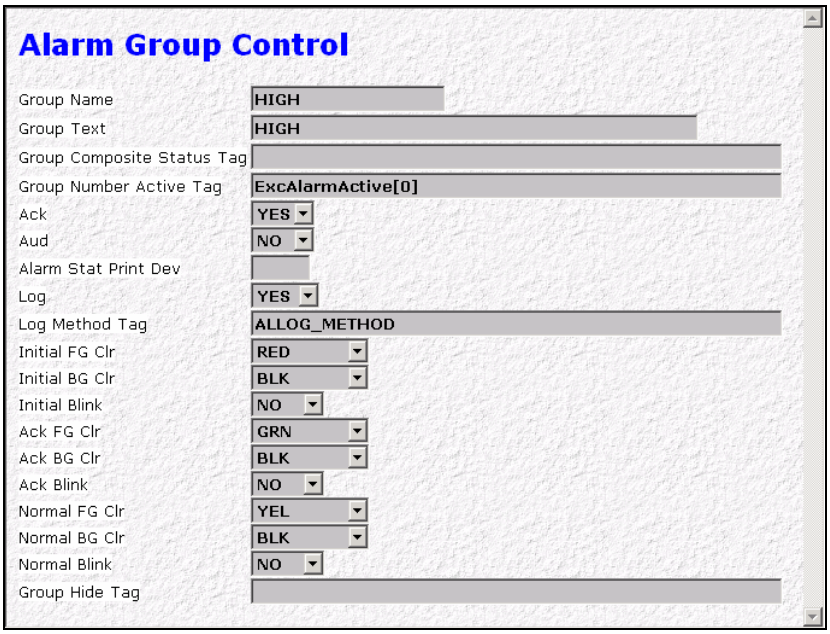


Figure 6.7. Alarm group definition fro exception alarms.

In every group one alarm for every equipment name is defined, this gives a total number of four * (number of equipments) alarms to configure for the alarm handling of the exception messages. The alarm itself is triggered by forcing the tag ExcAlarmTrigger[severity] to 'ON', this is done by the pre-processing part of IML.

	Unique Alarm ID	Alarm Tag Name	Cond.	*Limit	*Deadband	Message Text
1		ExcAlarmTrigger[0]	TGL			$VA1$
2		ExcAlarmTrigger[1]	TGL			$VA1$
3		ExcAlarmTrigger[2]	TGL			$VA1$
4		ExcAlarmTrigger[3]	TGL			$VA1$
5		ExcAlarmTrigger[4]	TGL			$VA1$
6		ExcAlarmTrigger[5]	TGL			$VA1$
7		ExcAlarmTrigger[6]	TGL			$VA1$
8		ExcAlarmTrigger[7]	TGL			$VA1$
9		ExcAlarmTrigger[8]	TGL			$VA1$
10		ExcAlarmTrigger[9]	TGL			$VA1$
*						

Figure 6.8. Several alarms are needed for one severity, the alarms differ in area name.

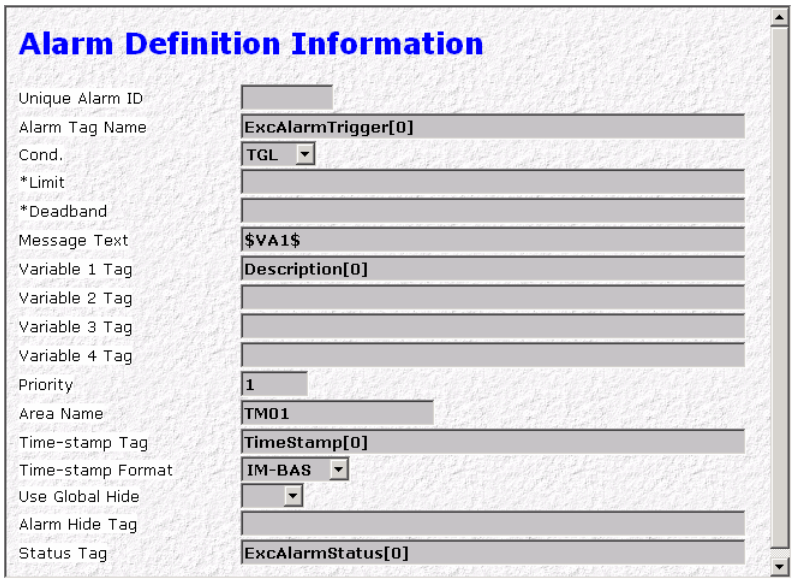


Figure 6.9. Individual alarm definition.

The alarm text consists out of the content of a message tag, the description of the exception (received from the subsystem). The standard variable tag for an alarm is used to build the complete alarm message. The maximum length of the description is 50 characters, this exceeds the limit of 44 characters for variable tags. This means that the text element is truncated before its value is stored in the alarm database, and before it is used to build the complete alarm text. For this behavior the 'alarm message' is defined as something like: $VA1$.
If you need to save the full length of the alarm variables in the alarm message, they should be used prior to truncating. To do so a format sp ecifier of the form %s should be used to incorporate the variable tags in the alarm message. But note that the %s also means that the alarm text is updated whenever a variable tag changes, so using %s means alarms will have the same text. To avoid this, use more then one alarm for every equipment, and trigger the alarm which is not (yet) active, every alarm should have separate variable tags (to prevent the logger to update other alarms with the newly received description).

All the alarm groups have a group count tag, these tags are used to trigger the post-processing of the IML task for the alarm handling.

In the received exception message is a timestamp present, a message tag is updated this timestamp by the IO-translator. The timestamp is a string of 14 characters: yyyymmddhhmmss, year, month, day, hour, minutes and seconds. Default the system clock is used to timestamp the alarms, however every alarm has the option to use a tag for time stamping. To do so a time stamp tag is defined for all four alarms (the same tag for every alarm), together with the tag a format for the timestamp is defined. In this configuration is used the timestamp layout 'IM-BAS'. This timestamp layout is normally not present, you have to change some files in the system directory of FactoryLink. Below are the required additions in these plain text files presented as bold and blue text.

File: %FLINK%\key\de\al_tsfmt.key
File: %FLINK%\key\en\al_tsfmt.key
File: %FLINK%\key\fr\al_tsfmt.key

```
*
* Copyright 1984-1995 United States Data Corporation. All Rights Reserved
*
* AL_STS.KEY
* DISTALOG key file
*
*
NULL       | -1 | NULL
USA        | 0 | USA
```

```
EUROPE     | 1 | EUROPE
USA_HS     | 2 | USA_HS
EUR_HS     | 3 | EUR_HS
USA_L      | 4 | USA_L
EUR_L      | 5 | EUR_L
USA_HSL    | 6 | USA_HSL
EUR_HSL    | 7 | EUR_HSL
IM-BAS     | 8 | IM-BAS
```

Add this line

File: %FLINK%\msg\de\al_fmt.txt
File: %FLINK%\msgy\en\al_fmt.txt
File: %FLINK%\msg\fr\al_fmt.txt

```
*****************************************************************************
*  Copyright 1984-2000 United States Data Corporation. All Rights Reserved.
*****************************************************************************
*                                - NOTICE -
*
*  The information contained herein is confidential information of United
*  States Data Corporation, a Delaware corporation, and is protected by
*  United States copyright and trade secret law and international treaties.
*  This information is not to be disclosed, used or copied by or transferred
*  to any individual, corporation, company or other person without the
*  express written permission of United States Data Corporation.
*****************************************************************************
*
*         File:  AL_FMT.MLT (used to generate AL_FMT.TXT)
*      Purpose: Define formats used for viewing and logging alarms.
*
*   Change Log:
*   DATE   WHO  SPR     DESCRIPTION
* 04/05/99 knc   SPR8678 Added tokens for Event alarm support
* 02/15/99 knc   SPR8229 Updated documentation for tokens
* 07/25/97 knc   SPR5180 Added formats for ASCII logfiles
*****************************************************************************



* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*                           SECTION 1: ALARM FORMATS
*
* PURPOSE:
* The formats in this section are used to control the appearance of alarm
* messages as they appear in various locations (Viewer display, print device,
* database, etc.).
*
* DESCRIPTION:
* The formats defined in this section may contain a mixture of regular text
* and tokens. The tokens will be replaced when the message is formatted. A
* brief description of the tokens and their syntax rules are given below.
* For more complete information, please see the FactoryLink documentation.
*
* RECOGNIZED TOKENS:
* The following tokens are recognized by the Distributed Alarm Task.
*
*   Token : Replaced With...
*   ----- : ----------------
*     AID : Unique Alarm ID Number
*     TAG : Alarm TAG Name
*     STS : Alarm Status (INI|ACK|NRM)
*     DAT : Initial Date        --See (*1) below
*     TIM : Initial Time        --See (*3) below
*     ADT : Acknowledge Date           --See (*1) below
*     ATM : Acknowledge Time           --See (*2) below
*     OPR : Acknowledge Operator
*     NDT : Normal Date                --See (*1) below
*     NTM : Normal Time                --See (*2) below
*     GMS : Group Message
*     MSG : Alarm Message
*     VA1 : Variable 1
*     VA2 : Variable 2
*     VA3 : Variable 3
*     VA4 : Variable 4
```

```
*     GRP : Group Name
*     ARE : Area Name
*     PRI : Alarm Priority
*     LOG : Logbook Entry Indicator
*           Exclamation mark '!' indicates a logbook entry, blank otherwise
*     ACK : Acknowledgement Indicator
*           Asterix '*' indicates alarm is unacknowledged, blank otherwise
*     LAN : LAN station ID alarm origin
*
* The following tokens are available in FL6.5 and later:
*     SEQ : Alarm Sequence ID
*     DUR : Alarm Duration (NTM - TIM) --See (*2) below
*
* NOTES:
*     (*1) The output of the tokens DAT|ADT|NDT are controlled by the DATE
*          keyword defined later in this file.
*
*     (*2) The output of the tokens ATM|NTM|DUR are controlled by the TIME
*          keyword defined later in this file.
*
*     (*3) The output of the TIM token is controlled by the HSTIME
*          keyword defined later in this file.
*
* SYNTAX RULES:
*    * Each token must be surrounded by dollar-signs ($).
*      Example: $MSG$
*
*    * A length specifier may be added to force replacement text to be
*      a specific number of characters. If the text is longer, it is
*      truncated. If the text is shorter, it will be padded with blanks.
*      Example: $MSG40$ (truncates or pads data to exactly 40 characters)
*
*    * There must be at least one space between two tokens
*      Example: $DAT$ $TIM$ will work
*               $DAT$$TIM$  will NOT work
*
* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


* ****************************************************************************
* ALARM VIEWER FORMATS
*    Purpose: These formats are used to display alarms on the Viewer.
* FLCM Panel: "Alarm View Control" panel, "Line Format" column
*
* View Format 1
VIEW_1    $DAT$ $TIM$ $LOG$ $ACK$ $GMS$ $MSG$
*
* View Format 2
VIEW_2    $TIM$ $TAG16$ $GMS$ $MSG$
*
* View Format 3
VIEW_3    $DAT$ $TIM$  $LOG$  $AID5$ $STS8$ $GMS$ $MSG$
*
* View Format 4
VIEW_4    $LOG$ $OPR8$ $TIM$   $TAG16$ $GMS$ $MSG$
*
* View Format 5
VIEW_5    $LAN$ $TIM$ $LOG$ $STS4$ $GMS$ $MSG$
*
* View Format 6
VIEW_6    $TIM$ $LOG$ $GMS$ $MSG$
*
* View Format 7
VIEW_7    $TIM$ #$LAN$ $OPR8$ $TIM$   $TAG16$ $GMS$ $MSG$
*
* View Format 8
VIEW_8    $LOG$ $OPR8$ $TIM$   $TAG16$ $GMS$ $MSG$
*
* View Format 9
VIEW_9    #$LAN3$-$AID4$ at $TIM$ : $GMS$ $MSG$
*
* Old (Pre-FL4.4) View Format Defintions
* VIEW_1    $DAT$ $TIM$ $LOG$ $TAG16$ $ACK$ $GMS$ $MSG$
* VIEW_5    $TIM$ : $STS8$ : $GMS$ $TAG16$
* VIEW_6    $TIM$ : $OPR8$ : $AID4$ $GMS$ $MSG$
```

```
* ************************************************************************
* ALARM PRINT DEVICE FORMATS
*    Purpose: These formats are used to print alarms on the print device
* FLCM Panel: "Alarm Group Control" panel, "Active Stat Print Dev" column
*
* Initial Alarm Occurrance
PRTINI    $DAT$ $TIM12$ $TAG16$ $GMS$ $MSG$
*
* Alarm Acknowledgement
PRTACK    $ADT$ $ATM12$ $TAG16$ Acknowledged by: $OPR$
*
* Alarm Return-To-Normal
PRTNRM    $NDT$ $NTM12$ $TAG16$ Normal Status
*
* Alarm "Unknown State" Format
PRTUNK    $DAT$ $TIM12$ $STS$ $AID$ UNKNOWN STATUS  $GMS$ $MSG$ $OPR$
*
* Event Occurance
PRTEVT    $DAT$ $TIM12$ EVENT $TAG16$ $GMS$ $MSG$


* ************************************************************************
* ALARM PRINT ACTIVE FORMATS
*    Purpose: These formats are used to print active alarms on the print device
*             when the "Print Active" tag is triggered
* FLCM Panel: "General Alarm Setup Control" panel, "Print Active Alarms Tag" col
*             "General Alarm Setup Control" panel, "Active List Print Device"
*
* Active Alarm Format
PRTACT    $DAT$ $TIM12$ $TAG16$ $STS$ $GMS$ $MSG$ $OPR$
*
* Logbook Entry Format (Part 1: Operator Name)
PRTOPR    "\tLogbook Message from operator %s\n"
*
* Logbook Entry Format (Part 2: Logbook Text)
PRTLOG    "\t%s\n"


* ************************************************************************
* ALARM ASCII LOGFILE FORMATS
*    Purpose: These formats are used to log alarms to an ASCII text file
*             when the "Log Method" is set to "FILE".
*             They are similar to the PRTINI, PRTACK, and PRTNRM formats
* FLCM Panel: "Alarm Group Control" panel, "Log" column
*             "Alarm Group Control" panel, "Log Method Tag" column
*
* Initial Alarm Occurrance
ALFINI    $DAT$ $TIM12$ $LOG$ $ACK$ $AID$ ACTIVE $GMS$ $MSG$
*
* Alarm Acknowledgement
ALFACK    $ADT$ $ATM12$ $LOG$ $ACK$ $AID$ Acknowledged by: $OPR$
*
* Alarm Return-To-Normal
ALFNRM    $NDT$ $NTM12$ $LOG$ $ACK$ $AID$ Normal Status
*
* Alarm "Unknown State" Format
ALFUNK    $DAT$ $TIM12$ $LOG$ $STS$ $AID$ UNKNOWN STATUS  $GMS$ $MSG$ $OPR$
*
* Alarm "Event" Occurance
ALFEVT    $DAT$ $TIM12$ $AID$ EVENT: $GMS$ $MSG$

* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*                      SECTION 2: DATE AND TIME FORMATS
*
* PURPOSE:
* The formats in this section are used to control the appearance of date
* and time stamps placed on alarms. The tokens in this section should NOT
* be directly entered into any alarm format definitions in Section 1.
*
* DESCRIPTION:
* The formats defined in this section may contain a mixture of regular text and
* tokens. The tokens will be replaced when the message is formatted. A brief
* description of the tokens and their syntax rules are given below.
* For more complete information, please see the FactoryLink documentation.
*
```

```
* RECOGNIZED TOKENS:
* The following tokens are recognized by the Distributed Alarm Task.
*
*    year: four digit year          example: 1999
*     yr : two digit year           example: 99
*     mo : two digit month          (01 .. 12)
*     mon: three letter month       (jan .. dec)
*     dy : two digit day            (01 .. 31)
*     hr : two digit hour           (00 .. 23)        24-hour clock
*     ah : two digit hour           (01 .. 12)        12-hour clock
*     ap : AM/PM indicator          (AM for hr < 12, PM for hr >= 12 )
*     mi : two digit minutes        (00 .. 59)
*     sc : two digit seconds        (00 .. 59)
*    mse : three digit milliseconds (000 .. 999)
*
* NOTES:
* These tokens are CASE sensitive: YR != yr
*
* SYNTAX RULES:
* No spaces are required between tokens.
*
* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


* ****************************************************************************
* TIMESTAMP FORMATS
*    Purpose: These formats are used to format the time-related tokens
*             listed in Section 1.
* FLCM Panel: "Alarm Definition Information" panel, "Time Stamp Format" column
*
* Standard USA format
USA             yrmodyhrmisc
*
* Standard European format
EUROPE   dymoyrhrmisc
*
*
* "High Speed" USA format (with milliseconds)
USA_HS   yrmodyhrmiscmse
*
* "High Speed" European format (with milliseconds)
EUR_HS   dymoyrhrmiscmse
*
*
* "Long" USA format (with separators)
USA_L    yr:mo:dy:hr:mi:sc
*
* "Long" European format (with separators)
EUR_L    dy:mo:yr:hr:mi:sc
*
*
* "High Speed, Long" USA format (with milliseconds and separators)
USA_HSL  yr:mo:dy:hr:mi:sc.mse
*
* "High Speed, Long" European format (with milliseconds and separators)
EUR_HSL  dy:mo:yr:hr:mi:sc.mse

* IM-BAS driver
IM-BAS  yearmodyhrmisc

* ****************************************************************************
* VIEW DATE FORMATS
*    Purpose: These formats are used to format the time-related tokens
*             listed in Section 1.
*       These tokens will not be recognized if they are used directly
*             in an alarm view format string.
*
* DATE FORMAT
* (controls format of DAT|ADT|NDT tokens defined in Section 1)
DATE            mo/dy/yr
*
* ACK/NORMAL TIME FORMAT
* (controls format of ATM|NTM|DUR tokens defined in Section 1)
TIME            hr:mi:sc
*
```

Add these lines

```
* INITIAL TIME FORMAT
* (controls format of TIM token defined in Section 1)
HSTIME   hr:mi:sc
*
* INITIAL TIME FORMAT (alternate defintion -- includes milliseconds)
* (controls format of TIM token defined in Section 1)
* To use this alternate format, comment out HSTIME above
* and uncomment this version
**EN HSTIME      hr:mi:sc.mse
**FR HSTIME      hr:mi:sc.mse
**DE HSTIME      hr:mi:sc.mse


* ************************************************************************
* DATABASE DATE FORMATS
*    Purpose: These formats are used to format the date and time when
*             logging alarms to the ALARMS database.
*       These tokens will not be recognized if they are used directly
*             in an alarm view format string.
*
* ACK/NORMAL DATABASE TIME FORMAT
* (controls format of Ack and Return-to-Normal Alarm Times in ALARM database)
LOGTIME  yearmodyhrmisc
*
* INITIAL DATABASE TIME FORMAT
* (controls format of Initial Alarm Time in ALARM database)
LOGHSTIMEyearmodyhrmisc.mse


* ************************************************************************
* MISCELLANEOUS DATE FORMATS
*    Purpose: These are miscellaneous formats used by the Alarm Logger Task
*
* NAME OF ARCHIVE ALARM FILE
* Used to create archive filename for the ALARMLOG.TXT file
* (Archive file will be named al{ARCDATE}.txt)
ARCDATE  yrmody
*
* USDATA DEBUG TIME FORMAT
* (controls format of Initial Alarm Time in task debug logs)
DATETIME mo/dy/year hr:mi:sc


* ************************************************************************
* APPLICATION-SPECIFIC DIGITAL TAG TEXT
*    Purpose: These formats are used with the "%3B" format specifier to allow
*             application-specific text for digital tag values. If the tag
*             value is zero, the DIGOFF string is printed; otherwise, the
*             DIGON string is printed.
* FLCM Panel: "Alarm Definition Information" panel, "Message Text" column
*
DIGON    On
*
DIGOFF   Off


* ************************************************************************
* EVENT KEYWORD
* This text is logged into the "Acknowledging Operator" field ('OPR') of
* the alarms database. It is used to easily extract EVENT records from the
* database.
EVENT    EVENT



* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*                  SECTION 3: STANDARD TRANSLATION TOKENS
*              Tokens in this section are not likely to change
* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


* ************************************************************************
* TAG DATA FORMAT
*    Purpose: These formats are used to format data from tags listed in the
*             "Variable 1 Tag" .. "Variable 4 Tag" columns.
* FLCM Panel: "Alarm Definition Information" panel, "Variable N Tag" columns
```

```
*
* Digital Tag
DIGFMT    "%1B"
*
* Analog Tag
ANAFMT    "%d"
*
* Float Tag
FLTFMT    "%.3f"
*
* Long Analog Tag
LANAFMT   "%ld"
*
* Message Tag
MSGFMT    "%s"


* ****************************************************************************
* ALARM STATUS TEXT
*     Purpose: These are standard FL translation tokens for the Alarm State.
*
* Alarm Status is NORMAL (tag value does not meet alarm condition)
NORMAL          Normal
*
* Alarm Status is ACTIVE (tag value meets alarm condition)
ACTIVE          Active
*
* Alarm Status is ACKNOWLEDGED (operator has acknowledged alarm)
ACK             Ack


* ****************************************************************************
* MONTH NAMES (THREE-LETTER ABBREVIATIONS)
*     Purpose: These are standard FL translation tokens for month names.
*
JAN             Jan
FEB             Feb
MAR             Mar
APR             Apr
MAY             May
JUN             Jun
JUL             Jul
AUG             Aug
SEP             Sep
OCT             Oct
NOV             Nov
DEC             Dec
```

7 License Agreement

The use of this program is subject to the following terms and conditions.

Title To The Licensed Software
Title to the licensed software is NOT transferred or sold to the end user. The end user is granted a non-exclusive license to use the software on a SINGLE computer or computer workstation. EACH computer or computer workstation must have its own licensed copy of the software.

Copyright Protection
This software is copyrighted material. The copyright laws of the Netherlands, and other proprietary rights of Wizard Information Systems protect it. You may not make any changes or modifications to the program or the manual. You may not decompile, disassemble, or otherwise reverse-engineer the software in any way.

Limited Warranty
Wizard Information Systems does not warrant that the licensed software will meet your requirements or that the operation of the software will be uninterrupted or error free. The warranty does not cover any media or documentation that has been subjected to damage or abuse by you or others. The software warranty does not cover any copy of the licensed software that has been altered or changed in any way. In other words there is no warranty either implied or expressed.

ANY IMPLIED WARRANTIES INCLUDING ANY WARRANTIES OF MERCHANTABILITY OR FITNESS OR A PARTICULAR PURPOSE ARE LIMITED TO THE TERM OF THE EXPRESS WARRANTIES. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you.

Other Warranties
The warranties set forth above are in lieu of any and all other express or implied warranties, whether oral, written, or implied, and the remedies set forth above are the sole and exclusive remedies.

Limitation Of Liability
Wizard Information Systems is neither responsible nor liable in anyway for any problems or damage caused by the licensed software that may result from using the licensed software. This includes, but is not limited to, computer hardware, computer software, operating systems, and any computer or computing accessories. End user agrees to hold Wizard Information Systems harmless for any problems arising from the use of the software.
Wizard Information Systems SHALL NOT IN ANY CASE BE LIABLE FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT OR OTHER SIMILAR DAMAGES ARISING FROM ANY BREACH OF THESE WARRANTIES EVEN IF Wizard Information Systems OR ITS AGENTS OR DISTRIBUTORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

In no case shall Wizard Information Systems liability exceed the license fees paid for the right to use the licensed software.

The above constitutes the license agreement for this program. It supersedes any and all previous license agreements.

LICENSE