



DATAGUIDE

Manual

CONTENTS

Contents	2
Introduction.....	6
Functions	6
Installation	7
Configuration Files	7
Updates	7
Installation Order.....	8
Licensing.....	9
<i>Product Key</i>	10
<i>Licensed To</i>	10
<i>License Number</i>	10
<i>Licensing Restrictions</i>	11
Graphical User Interface	12
Interface Overview.....	13
<i>Menu Bar</i>	13
File	13
View	14
Help	14
Licensing	14
About	14
<i>Section Selection</i>	14
<i>Section Caption</i>	15
<i>Section Toolbar</i>	15
<i>Section Details</i>	15
<i>Status Bar</i>	15
Section Configuration	16
Overview	16
Configuration.....	17
Toolbar	17

<i>Variable Settings</i>	18
Triggers.....	19
Toolbar.....	20
<i>Interface Trigger</i>	20
<i>Startup Trigger</i>	21
<i>Shutdown Trigger</i>	21
<i>Timed Trigger</i>	21
Actions.....	23
Toolbar.....	23
Log.....	23
E-Mail.....	26
Script.....	28
Interfaces.....	29
Scripting.....	30
Capability.....	30
Variables.....	31
<i>Internal</i>	31
<i>Signal Variables</i>	31
<i>Configuration Variables</i>	32
Variable Formatting.....	32
Padding.....	32
FormatString.....	32
Recorders.....	34
Configuration Locking.....	35
Remove Configuration Password.....	36
Corrupted Files.....	37
Application Log Files.....	38
Activity Log.....	38
History Log.....	38
Appendix A: Script Commands.....	40
Remarks.....	40

Commands	40
<i>FileOpen</i>	40
<i>FileAppend</i>	40
<i>FileWrite</i>	40
<i>FileWriteLine</i>	41
<i>FileClose</i>	41
<i>Call</i>	41
<i>MessageLevel</i>	41
<i>LogLevel</i>	41
<i>Let</i>	42
Mail Functions	42
MailTo	42
MailFrom	42
MailSubject	42
MailMessage	42
MailAttach	43
MailSend	43
MailAttachDelete	43
<i>CleanupFolders</i>	43
Appendix B: ibaLogic Interface.....	44
DLL OUTPUT Interfaces.....	46
Error Numbers.....	46
Output DLLs.....	47
DLL_OutDINT.....	48
Inputs	48
Outputs	48
DLL_OutBOOL.....	49
Inputs	49
Outputs	49
DLL_OutREAL	50
Inputs	50
Outputs	50
DLL_OutSTRING	51
Inputs	51
Outputs	51
DLL INPUT Interfaces	52
DLL_InREAL.....	53
Inputs	53
Outputs	53
Sample Layouts	54
DataGUIDE_Sample_100.lyt.....	54

Appendix C: iba PDA Interface	55
PDA Interface Toolbar	55
Configuring PDA.....	56
Appendix D: TCP/IP Interface	61
Servers.....	61
Messages.....	63
Receive Messages.....	63
Send Messages	63
Protocols.....	63
Header Structure	64
Basic Header.....	65
Standard Header.....	65
Extended Header	65
Reserved Messages	66
Heartbeat Message	66
Disconnect Message.....	66
Empty Message	67
Message Data	67
Fixed Length Message Data (Binary).....	67
CSV Messages (String).....	68
Defining a Client Protocol	68
Protocol/Port Number	68
Messages.....	69
Message Type	70
CSV (Comma Separated Variable)	70
Fixed Length Binary Messages	71
Message Structure Definition	71
CSV Messages	72
Fixed Length Messages	72
FAQ.....	73
General	73
ibaLogic.....	73
PDA	74
TCP/IP Protocol.....	74

INTRODUCTION

This is the manual for *DataGUIDE*, a software component of the RTI Server (Real-Time Information Server), describing the operation function and configuration.

DataGUIDE is a program which works in conjunction with several sources of raw data, and converts it to usable information through a configurable interface.

Functions

DataGUIDE provides the following data interfaces:

- ibaLogic
- iba PDA V6 (Version 6.8 or later; via the PDA Plugin Interface)

The functionality based on those interfaces are as follows:

- Create triggers based on user configured levels

Additionally, *DataGUIDE* has the following capabilities:

- Log file generation
- E-mail generation
- Database updating

INSTALLATION

DataGUIDE is installed through a standard executable. It will install all required files for its own functionality. However, it will *not* install any third-party applications (e.g. ibaLogic or SQL Server).

Installation requires a product key (the product key is supplied with the media).

Additionally, to utilize the full functionality of *DataGUIDE* a license key is required. To obtain a license, please contact your reseller with the following information:

- Product Key (16 characters)
- Name
- Company

Configuration Files

If *DataGUIDE* is reinstalled, the option to overwrite the sample configuration files will be offered. By default, the configuration files will be retained (any configuration changes made to those files will not be lost).

If you choose to reinstall, and overwrite the sample configuration files, you will lose and configuration settings you have made to those files.

Updates

Updates, if any, can be installed without uninstalling first. Updates should only be obtained through the reseller or Steelcoder directly.

Installation Order

Some support applications, for a complete installation, must be installed in a specific order. The DataGUIDE installation detects the presence of these components.

The following (optional) components must be installed before installing DataGUIDE:

- SQL Server
- ibaLogic
- iba PDA

Note: The above components may have their own prerequisites which must be followed.

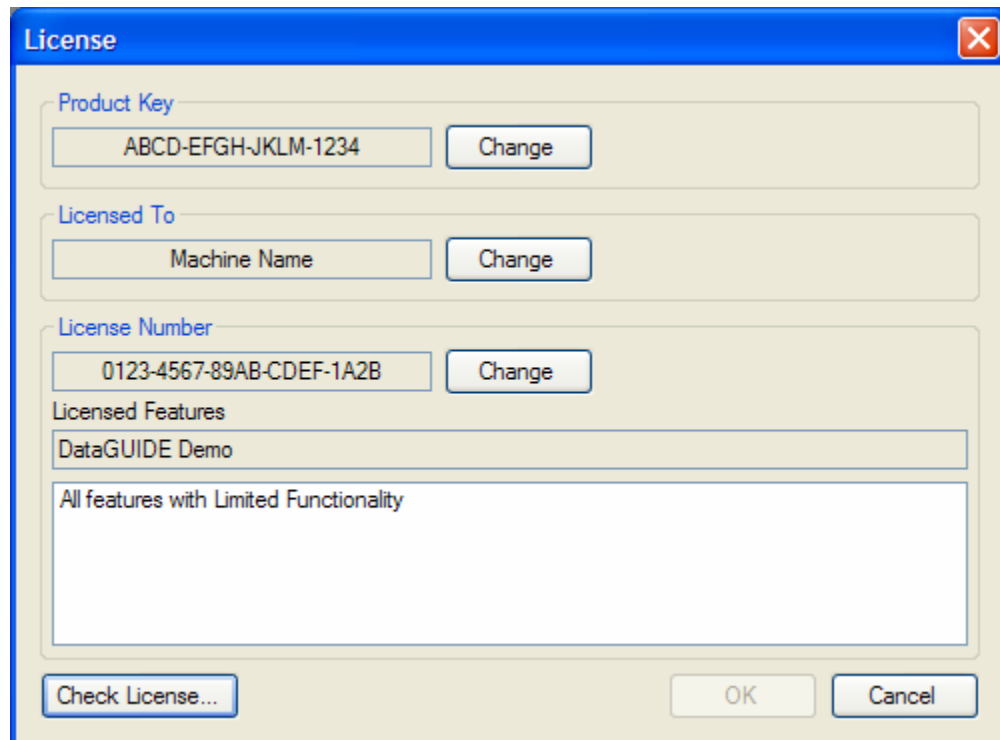
LICENSING

The features that are enabled are determined by a valid license number. The license number is a **20 digit** hexadecimal number (composed of the digits 0 to 9 and the letters A to F), For example:

ABCD-0123-EF45-6789-0FEA

A separate **License** is required for each installation: each license is tied to a specific **Product Key** and **Licensed To** user name. The License is supplied with the Licensed To user name in the **Confirmation Documentation**.

The license number can be entered from within the application on the *Help* menu by selecting the *License...* menu item:



The screenshot shows a dialog box titled "License" with a close button (X) in the top right corner. The dialog contains three sections, each with a text input field and a "Change" button:

- Product Key:** The input field contains "ABCD-EFGH-JKLM-1234".
- Licensed To:** The input field contains "Machine Name".
- License Number:** The input field contains "0123-4567-89AB-CDEF-1A2B".

Below these sections is a "Licensed Features" section with a text area containing "DataGUIDE Demo" and "All features with Limited Functionality". At the bottom of the dialog are three buttons: "Check License...", "OK", and "Cancel".

Figure 1: Licensing Screen

The *Product Key*, *Licensed To* or *Licensed* number can be changed by selecting the Change button next to each item. The license can be checked (validated) by selecting the *Check License...* button.

The features enabled by the (valid) license will be shown below the license number.

Product Key

The product key is supplied with the distribution media and is printed on an orange/yellow sticker:

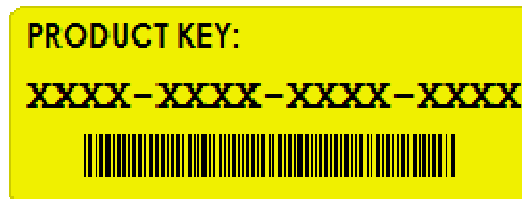


Figure 2: Product Key Sticker

This is a **16 digit** number with the letters A to Z and digits 0 to 9.

Licensed To

The Licensed To (or User Name) is the name supplied when obtaining a valid license number. It must be entered **exactly** as supplied by the Confirmation Documentation containing the license number (however, it is not case sensitive).

This can be the machine name that the installation applies to (recommended), the computer name, company name or an individual's name.

License Number

The license number is supplied in the Confirmation Documentation, along with a confirmation of the *Licensed To* name. This is a **20 digit** number.

Licensing Restrictions

Without a valid product key, Licensee and license number the application will run in demo mode. The following table outlines the restrictions based on the licensed level.

	Demo	Life	Standard	Advanced
Triggers				
Interface	4	32	64	256
Timed	1	2	4	Unlimited
Actions				
E-Mail	1	1	4	4
Log	1	1	4	4
Script	1	1	4	Unlimited
Interfaces †				
ibaLogic	4 Modules (1x each Data Type)	4 Modules (1x each Data Type)	Unlimited	Unlimited
Reflective Memory	32 Signals	64 Signals	Unlimited	Unlimited
TCP/IP	32 Signals (1 Message)	64 Signals (2 Messages)	Unlimited	Unlimited
Iba PDA	32 Signals	32 Signals	Unlimited	Unlimited

† The interfaces are licensed separately for each level. The demo version contains limited interface functionality.

GRAPHICAL USER INTERFACE

The DataGUIDE application has a visual interface accessible from the icon which resides in the system tray:




Figure 3: System Tray Icon

Double-clicking this icon or right-clicking and selecting *Show* will show the DataGUIDE Interface.

Note: if *Show* is not available, then the DataGUIDE interface is currently visible, but may be hidden behind another window. A bar will be shown on the task bar, or double-click the icon to bring it to the front.

To hide the interface to the system tray:

- Right-click on the system tray icon and select *Minimize to Tray*
- From the *View* menu, select *Minimize to Tray*
- Use  to hide *DataGUIDE* to the system tray (Note – this will *not* close the application).
- From the *System Menu*, select *Minimize to Tray*:

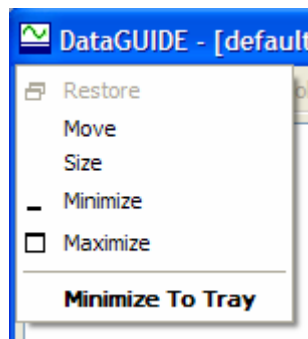


Figure 4: System Menu

To exit DataGUIDE:

- Right-click on the system tray icon and select *Exit*

- From the File menu, select *Exit*

Note: with DataGUIDE Not running, none of the functionality offered by DataGUIDE will be performed!

Interface Overview

Each property, mechanism, script, etc. is accessible in one of the following sections. Each section is selected by clicking an item on the *treeview* at the right side of the interface. The interface is divided up as shown in Figure 5 .

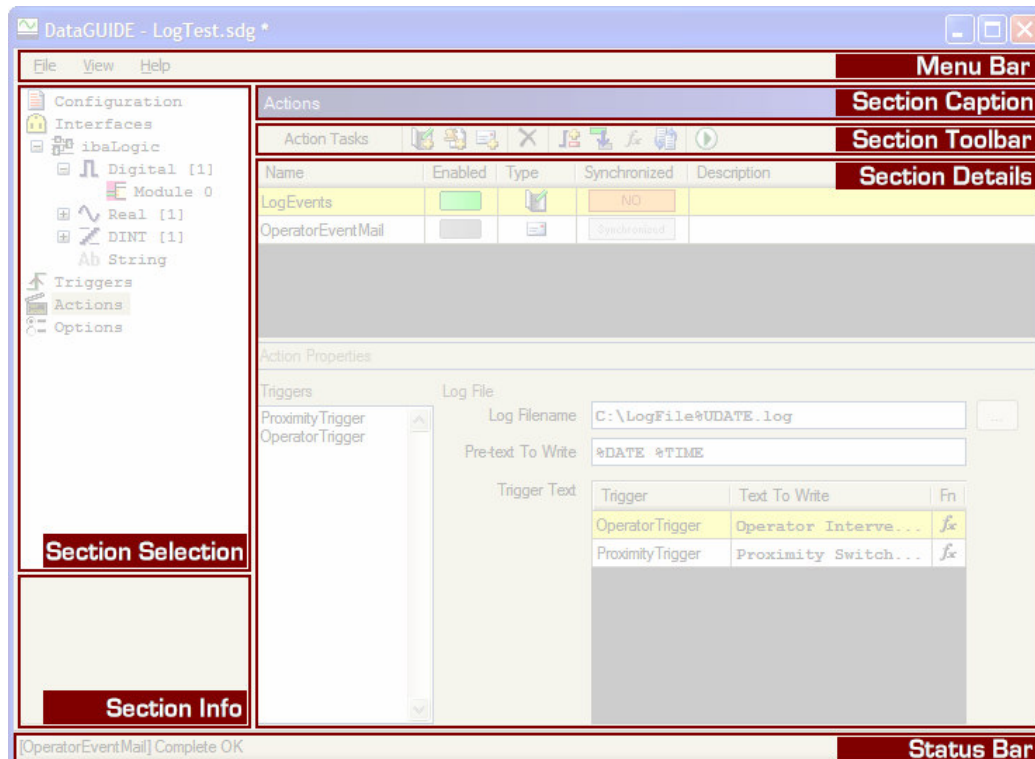


Figure 5: Interface Overview

Menu Bar

The menu bar contains common functions, such as save and loading of configurations and accessing help.

File

Contains the ability to Save, and load configurations. Additionally, the application can be closed, or exited, through this menu item only.

View

The ability to refresh the current section properties is available here, and is most useful when viewing the values of signals in the interfaces.

The application can also be minimized to the system tray by selecting the 'minimize to tray' menu item.

Help

Licensing

Shows the *Licensing* Window (see section on Licensing).

About

This shows a standard *About* Window which indicates the application version and revision number.

Section Selection

The configuration of DataGUIDE is performed through sections. Each section is accessible through this treeview.

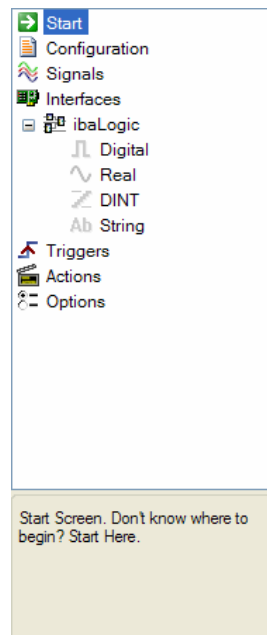


Figure 6: Section Selection Treeview

Note that some sections described in this manual may not be available, depending on the license obtained for the application.

Additional sections may be made available (with an addendum or additional documentation) depending on the license obtained for the application.

Section Caption

This area displays the title of the currently selected section.

Section Toolbar

Each section has its own toolbar, displayed below the section caption. Some of these tools are available by right clicking on the appropriate section configuration areas. Each section toolbar is described in detail in this manual, in the relevant section.

Section Details

This panel shows the details of the currently selected section. Each section is described in detail later in this manual.

Status Bar

The status bar gives an indication, or overview, of any errors or events which take place. An error will be indicated by a red circle with a cross through it.

SECTION CONFIGURATION

Overview

The application configuration will be referred to as a **Configuration** and is saved as a single text file with the extension **.SDG**

The following describes the functionality of each section.

Note: the screenshots may differ slightly from the actual application due to updates and also depend upon the license supplied with the application.

Configuration

The configuration Section allows Constants and Variables to be defined for the whole configuration. These Constants and Variables are available through emails, scripting, log file generation.

For Variable Usage, refer to the *Variables* section in this document.

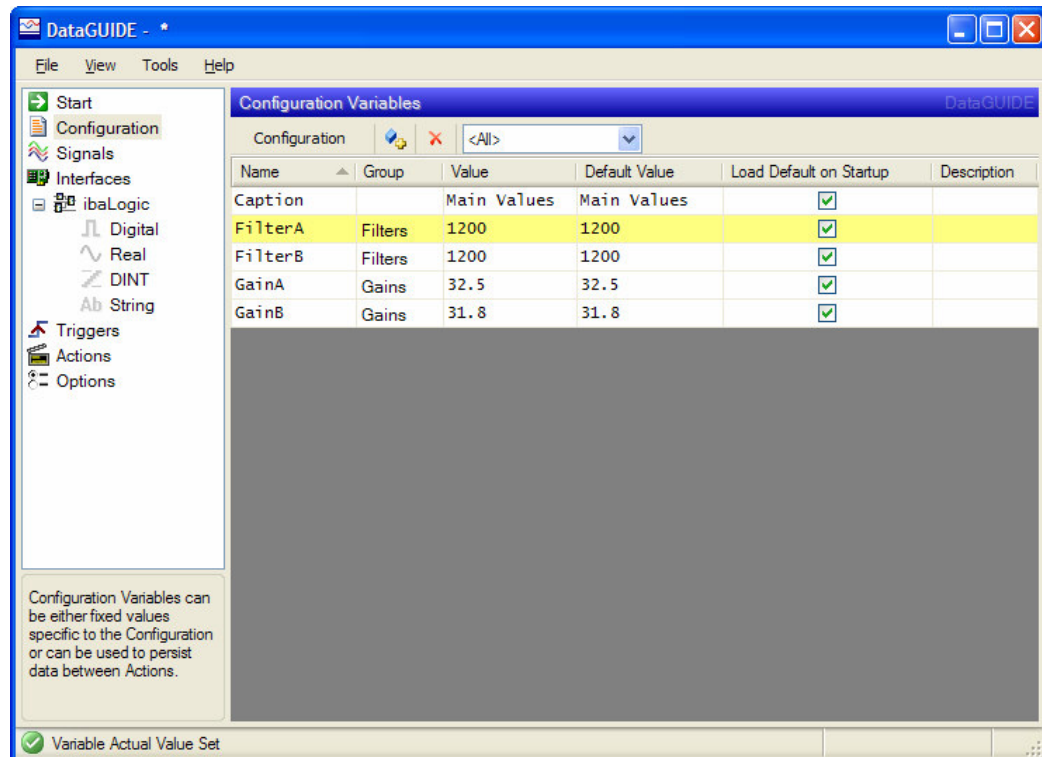


Figure 7: Configuration

Toolbar

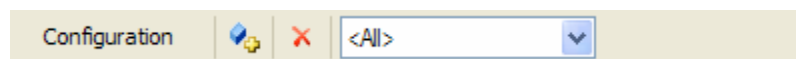


Figure 8: Configuration Variables Toolbar

Configuration Variables can be added and removed using the Toolbar:

-  Add New Variable
-  Delete the Selected Variable

The drop-down box allows the variables from a specific group to be viewed. This is useful when a large number of variables are configured.

Variable Settings

Each configuration value has the following parameters:

Name – a unique name for this Variable

Group – variables can be grouped for convenience

Value – this is the current value. The Value can be changed dynamically through scripting, for example

Default Value – this is the default value for the variable

Load Default on Startup – when *DataGUIDE* is started, the variable Value will be set to the Default Value if this checkbox is checked

Description – convenient descriptive text for the variable. This value is also available to Scripting, Log Files and emails

*Note that the current Value will persist between shutdown and startup of DataGUIDE. This data is stored in a separate file with the same name as the configuration file, with the appended file extension **.variables***

Triggers

For *DataGUIDE* to perform any actions, a Trigger must take place. This section allows selection and configuration of the triggers. There are four (4) kinds of triggers, selectable by right clicking on the grid or from the toolbar.

Note: Even after a trigger has been defined, it does not do anything until it has been assigned to an Action. See the action configuration section on assigning triggers to actions.

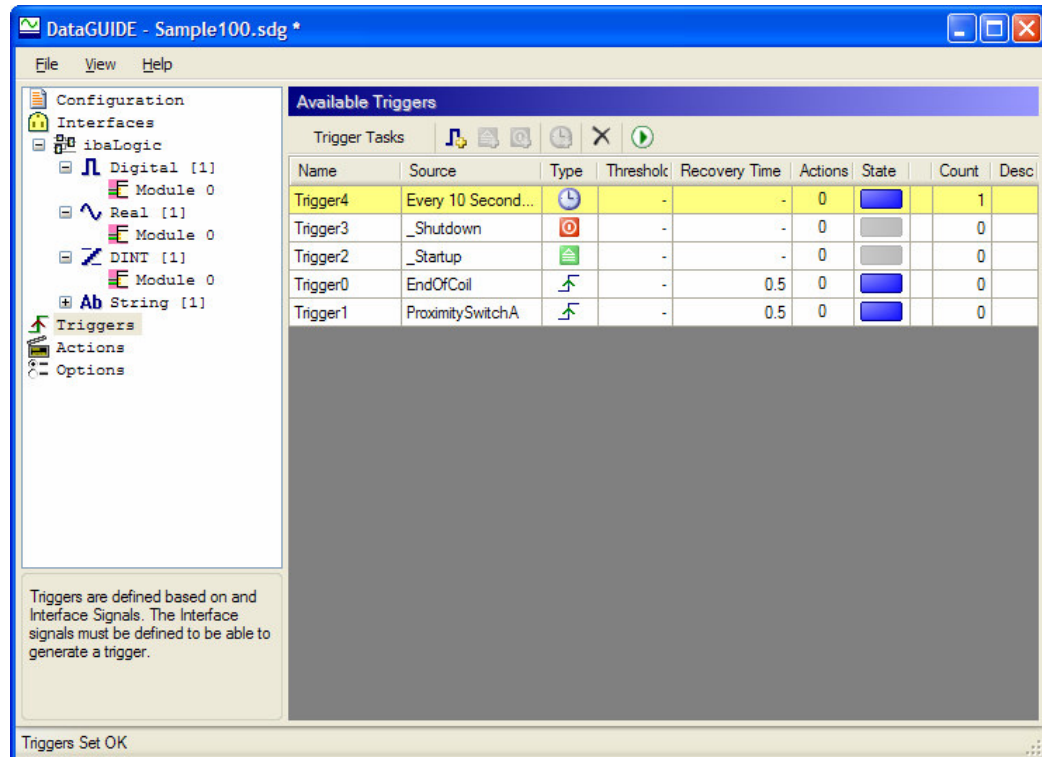


Figure 9: Triggers

For testing purposes, triggers can be run manually by forcing a trigger transition:

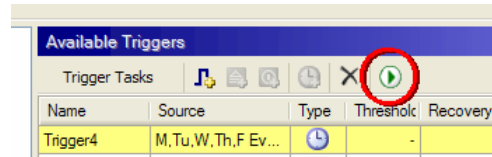











Figure 10: Force Trigger Transition Tool Button

Toolbar



Figure 11: Triggers Toolbar

Triggers can be added and removed and certain trigger properties modified from the toolbar:

-  Add or remove Signals to be used as Triggers
-  Add a Startup Trigger
-  Add a Shutdown Trigger
-  Add a Timed Trigger
-  Delete the Selected Trigger
-  Modify the Trigger Style (rising Edge/Falling edge) for the selected trigger
-  Reset the Selected Trigger Count to Zero
-  Add and Remove Actions associated with the selected trigger
-  Activate (Force) the selected trigger

Interface Trigger

Once a signal has been defined, it can be used as a Trigger. The available signals will be shown and selectable by clicking the *Add/Remove Triggers* button or right-clicking the triggers grid and selecting *Add/Remove Triggers*.

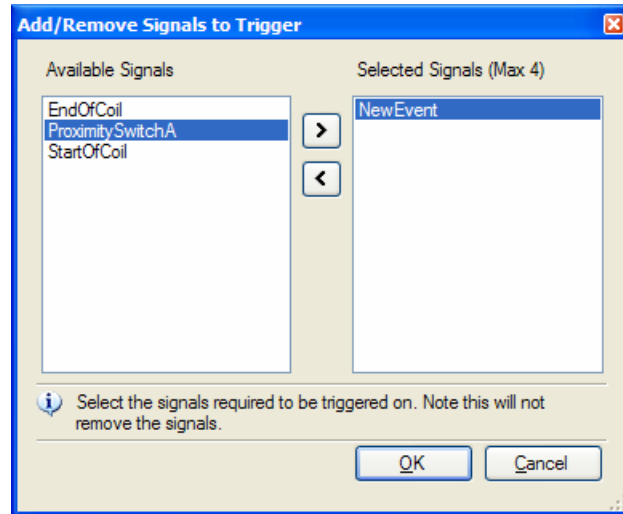


Figure 12: Add/Remove Triggers

Note: The number of triggers that can be added is based on the License.

Startup Trigger

A Startup Trigger can be added by selecting the *Add Startup Trigger* button on the toolbar. Only one Startup Trigger can be added.

This trigger will fire when *DataGUIDE* starts up or when the Configuration (or a new configuration) is loaded.

Shutdown Trigger

A Shutdown trigger can be added by selecting the *Add Shutdown Trigger* button on the toolbar. Only one Shutdown Trigger can be added.

This trigger will fire when *DataGUIDE* shuts down. Loading another Configuration will also fire the shutdown trigger (if configured) for the configuration being terminated.

Timed Trigger

Timed triggers can be added that occur at pre-defined intervals each day or at specific times each day.

Set Time-based trigger

Interval

Periodic

Every Minute

at 10 Seconds past the Minute

Between 07 : 00

and 18 : 00

Daily

at 00 : 00

and 06 : 00

and 12 : 00

and 18 : 00

Specify Days

Every Day

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

M,Tu,W,Th,F Every 10 Seconds past the Minute Between 07:00 and 18:00

Define a Periodic or a Daily trigger to be performed every day or only on specific days

OK Cancel

Figure 13: Timed Trigger Definition

Triggers can be:

Periodic – the trigger occurs either every *Hour* or every *Minute* between the specified times.

Daily – up to 4 times in a single day may be defined for the timed trigger to occur.

Specific Days – specific days of the week may also be set to restrict the days on which the trigger occurs.

Actions

There are three (3) actions that can be made:

- Log
- Script
- E-Mail

Actions can be executed on any trigger event. New actions can be created from the toolbar, or by right-clicking the Actions grid.









Additionally, each action can be executed manually, in a similar way to firing triggers, by using the Green Arrow on the Toolbar.

Toolbar



Figure 14: Actions Toolbar

The Actions Toolbar (Labeled Action Tasks) allows the addition removal and trigger definitions for the defined actions.

-  Add Log Action
-  Add Scripted Action
-  Add E-Mail Action
-  Delete Selected Action
-  Add and Remove Triggers from selected Action
-  Disable / Enable the Selected Action
-  Future Use (Disabled)
-  Run (Execute) the selected Action

Log

The Log action is used to create and write entries to a text (log) file. Each trigger which fires the action can write a different item to the file.

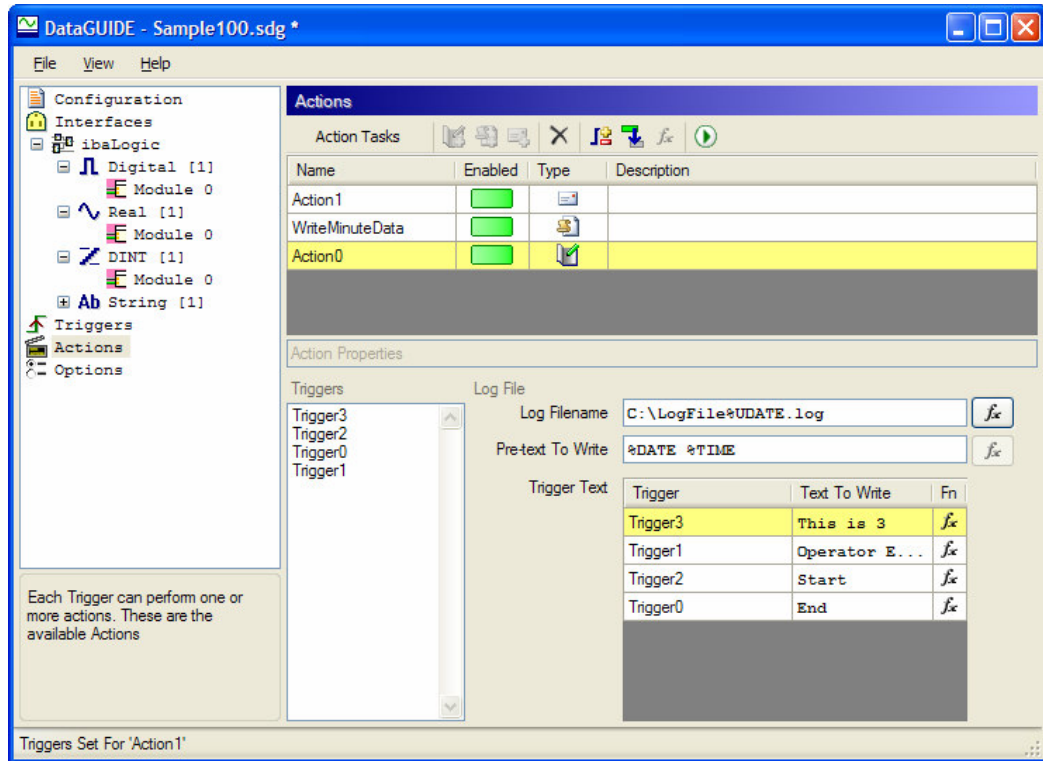


Figure 15: Log File Action

The Log Filename can be defined using any variables or signals, as well as any Internal Variables. For example, to create a Log file each day, the Internal Date variables can be used (refer to the *Variables* section later in this document) or the Name and folder can be defined using the Function Button:

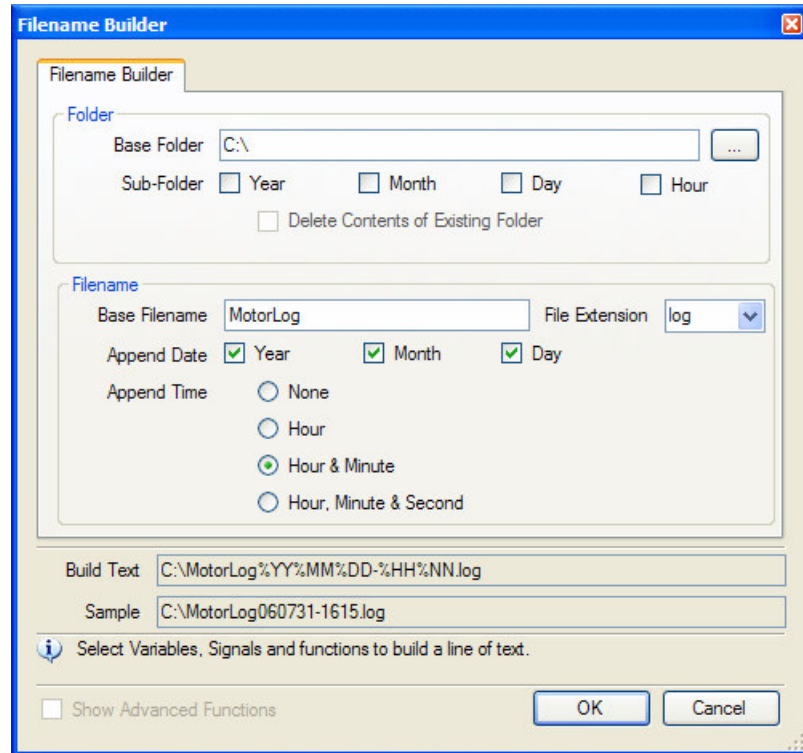


Figure 16: Filename Builder

The Filename builder dialog simplifies creating the Folder and filename for the log file.

Pre-text to write – this is the text which will be pre-pended to the line of text created by the trigger. Typically, this will be a date/timestamp, but can be any variable (refer to the *Variables* section in this document).

Each trigger can write different text. The Trigger Text Grid lists each trigger which fires this action, and the text can be edited in this grid. Once again any variable can be used in the text entry.

E-Mail

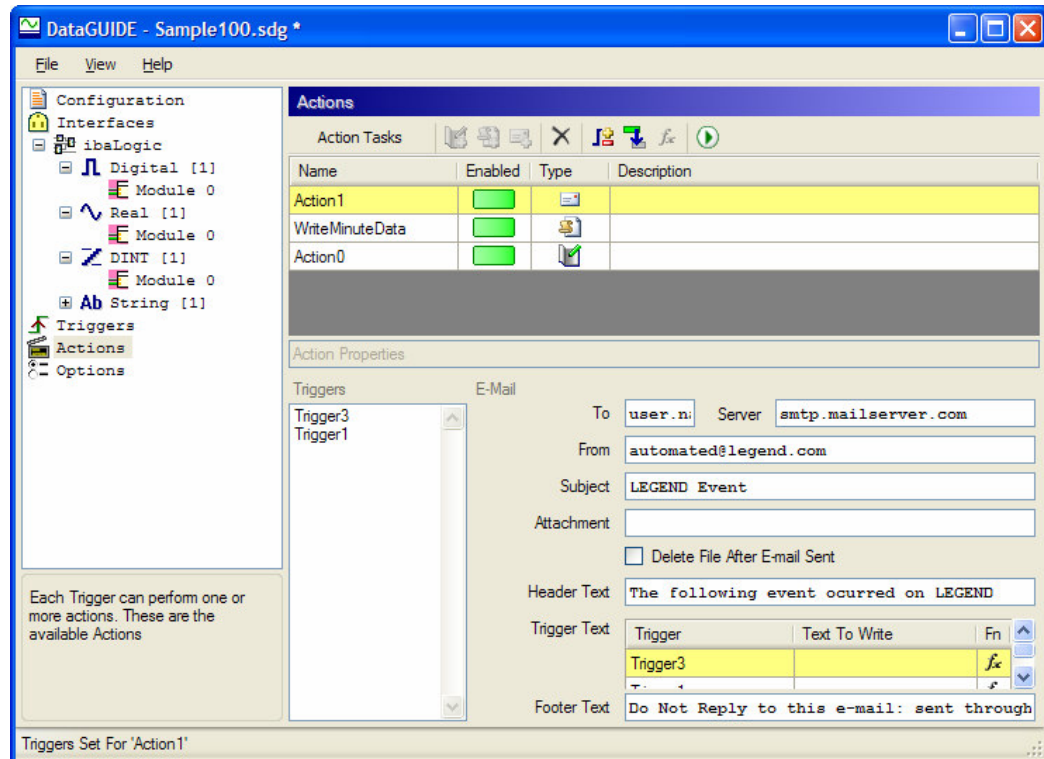


Figure 17: e-mail configuration

Triggers can also send e-mails. The following items are required to send e-mails:

SMTP Server – a SMTP capable server is required. Refer to your Information Technology department or Network Administrator to determine if an SMTP Server is available.

TO address – this is one or more valid e-mail addresses, separated by semicolons, that the e-mail is to be sent to.

From Address – this does not need to be a valid address, but must be in the form of a valid e-mail format; specifically, **name@domain**.

Subject – this is the subject for the e-mail

Attachment – a single file may be attached to the e-mail: there is an option to delete this file once the e-mail has been successfully sent.

Header Text – this text will be entered at the top of every e-mail.

Footer Text – this text will be entered at the footer of every e-mail.

Trigger Text – each trigger can write a single line of text as necessary.

Note that all these entries can contain Variables, and will be interpreted prior to the e-mail being sent. Refer to the *Variables* section in this document.

Script

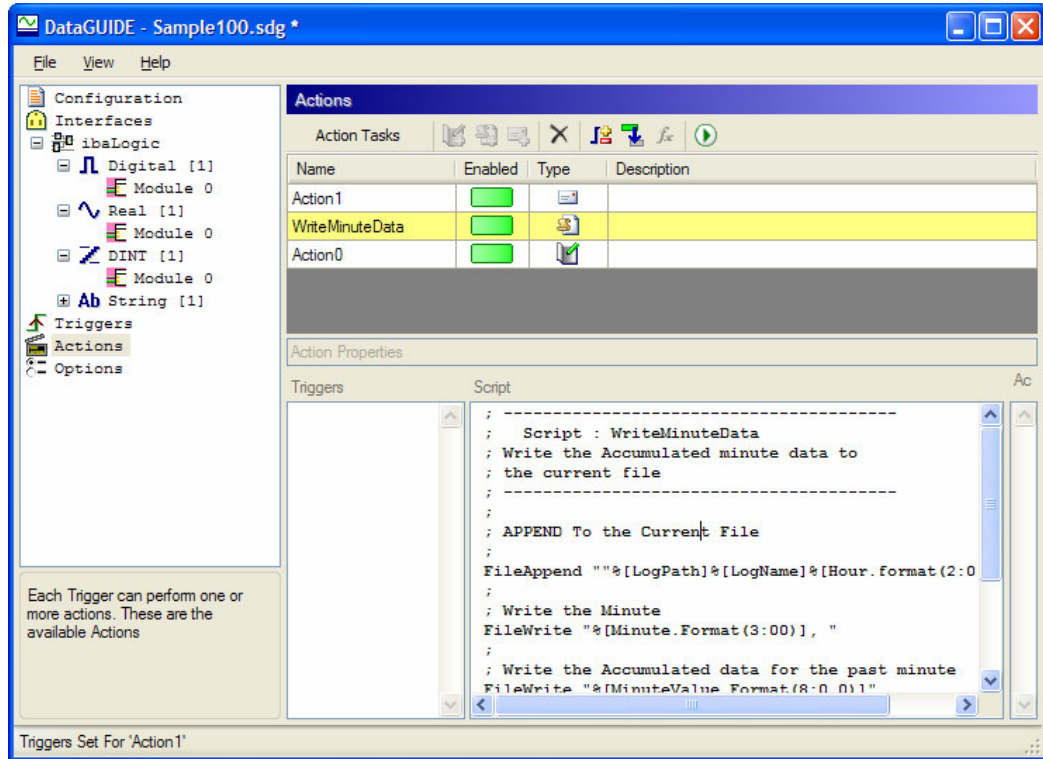


Figure 18: Script Action

The Scripting action is the most versatile, but more complex, Action. The script can send e-Mails and create log files, and is extensible to other functionality, based on the license.

Refer to the *Scripting* section in this document, which describes all the functions available in the Script Action, and the *Variables* section which describes variable usage.

INTERFACES

The INTERFACES section lists all the physical data inputs and outputs to DataGUIDE.

Each interface is described in the Appendices of this document.

SCRIPTING

Scripting is a versatile method for sequencing various actions that may not be available using the preconfigured actions or recording mechanisms.

Scripted Actions can be performed which would normally be done by an overly complex level 2 system, but are beyond the capabilities of a level 1 (PLC).

Script Commands are:

- NOT Case sensitive
- A command followed by none, one or more parameters

Capability

The appendix details the available scripting commands. In general, the following tasks can be performed by a Script:

- Open text files for writing.
- Access any defined Signal or Configuration Variable.
- Set any defined Configuration Variable.
- Create and send e-mails.
- Cleanup a Folder Structure, removing any empty folders in that root folder.

VARIABLES

Variables begin with the % symbol. If a script needs to use the % symbol, then place two (2) percent symbols together (i.e. %% will be replaced with %).

Note: Variables are not case sensitive.

Internal

Note: Internal Variables are always in UPPER CASE, preceded by a single % symbol

%DATE – Date the trigger was fired

%UPDATE – Date the trigger was fired in universal format (YYYY-MM-DD)

%TIME – Time the script was fired

%TRIGGER – the name of the trigger that fired this script (will be 'Manual' if the trigger was run manually)

%SCRIPT – the name of this script

There are several DATE variables that can be used:

%YYYY, **%YY** – formatted Year as 4 or 2 digits.

%MMMM, **%MMM**, **%MM** – Formatted Month as the full month, abbreviated Month name or as two digits.

%DDDD, **%DDD**, **%DD** – Formatted Day as a full day, abbreviated day name or as two digits.

%HH, **%NN**, **%SS** – Hour Minute and second formatting as two digits.

Signal Variables

Signal Variables, are additionally surrounded by square brackets:
e.g.

%[MyVariable]

Note: Variables are not case sensitive.

These variables are read only.

Variables have a description, if one is entered. To access the description for a variable, have the variable name, followed by a period, then 'Description'; for example:

```
%[MyVariable.Description]
```

Configuration Variables

These variables are accessed in the same way as signal variables but are readable and writeable. They will maintain their values from one function call to the next. i.e. they are *globally persistent*.

Note that these variables are also stored to the hard drive, so they will persist between executions of *DataGUIDE* (should it be shutdown for updates, for example).

*The configuration variable current values are stored in a file with the extension **.variables**.*

Variable Formatting

Variables can be formatted in a specific way, if required. This is generally used when writing to log files, or databases, for example.

To format a variable, follow the variable name with the keyword 'Format', and enclose the formatting required in parenthesis:

```
%[VariableName.Format (<Padding> : <FormatString>)]
```

Padding

An optional integer indicating the minimum width of the region to contain the formatted value. If the length of the formatted value is less than alignment, then the region is padded with spaces. If the padding value is negative, the formatted value is left justified in the region; if padding value is positive, the formatted value is right justified. If padding is not specified, the length of the region is the length of the formatted value. The comma is required if alignment is specified.

FormatString

A String of format specifiers, which follow the standard Microsoft String Format Specifiers as described in this document.

[http://msdn2.microsoft.com/en-us/library/0c899ak8\(VS.80,d=ide\).aspx](http://msdn2.microsoft.com/en-us/library/0c899ak8(VS.80,d=ide).aspx)

For example:

`% [Speed.Format (8:0.00)]`

This will right-justify the variable speed to 2 decimal places, with the width padded to a total of 8 characters (left padded).

RECORDERS

Recorders are not currently enabled for the DataGUIDE Application

DataGUIDE can store data in the following ways:

- CSV Files
- Binary Files
- Database Tables

The only limit to the number of recorders is dependent on the licensing Level.

The recorders use the defined triggers to start and stop recording.

CONFIGURATION LOCKING

Once DataGUIDE has been configured, it is unlikely that configuration changes will be needed on a day to day basis. To prevent accidental configuration changes, the configuration can be locked with a password.

NOTE: Locking with a password is NOT designed to secure the system from unauthorized changes: it is ONLY designed to prevent accidental changes.

To lock the configuration, from the menu select *Tools -> Protect Configuration...*

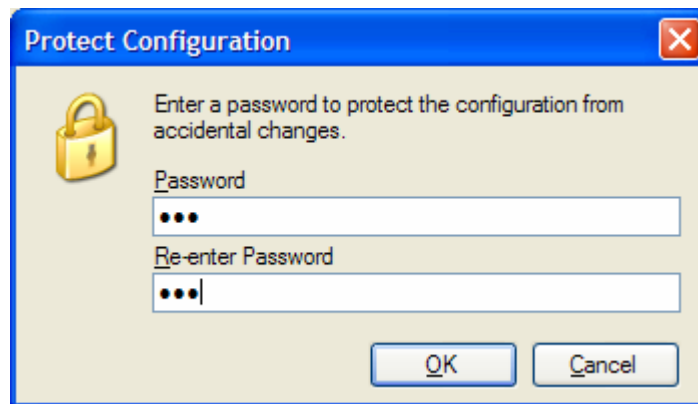


Figure 19: Protect/Lock Configuration

The OK button will be disabled unless both the Password and the Re-entered password match.

Once the configuration has been protected, no changes can be made to it. However, configurations can be saved and loaded.

The status bar will show a padlock in the bottom right, and each toolbar will have a padlock to the left.

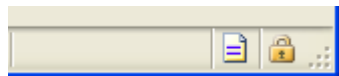


Figure 20: Locked Configuration Status

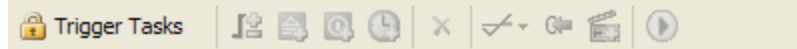


Figure 21: Toolbar Showing Locked Configuration

To unlock the configuration, from the menu select *Tools -> Unlock Configuration...*

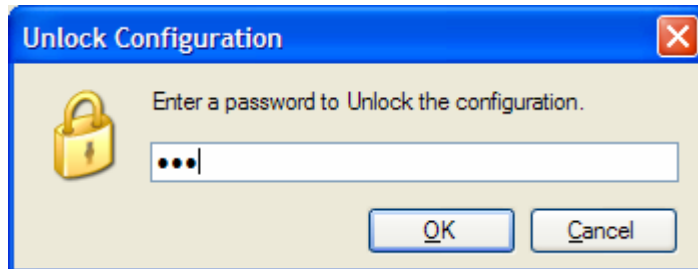


Figure 22: Unlock Configuration

The password to unlock the configuration must match the password used to lock the configuration. Should a blank password have been used, then this *Unlock Configuration* dialog will not show, and the configuration will be immediately unlocked.

Remove Configuration Password

Care must be taken when editing the configuration files. Ensure the configuration is backed up prior to making any changes. A corrupted configuration file will not be loaded by DataGUIDE.

Each configuration has its own password. To remove the configuration password the configuration file must be edited. Use a text file editor.

There is a section in the configuration file with a section heading, something like the following:

```
[Settings]
Locked=1
LockPassword=qZk+NkcGgWq6PiVxeFDCbJzQ2J0=
```

Modify the Locked entry to zero (0) so that the line reads:

```
Locked=0
```

*Do **not** modify any other lines in the file. Ensure that no spaces are added or removed from the end of the line. Doing so may corrupt the file, and it will not be readable by DataGUIDE.*

Reload the Configuration. The Configuration will now be unlocked.

Corrupted Files

Should a file become corrupted or unreadable, a backup of the file is made in the `/log` folder. This backup is made when the file is last successfully saved.

If there are configuration files of the same name, only the last successfully saved configuration file will be backed up, overwriting the previous backup file.

APPLICATION LOG FILES

Activity Log

All application Log Files are in a sub folder of the application installation folder, called **/Log**.

Any activity is recorded to a file:

current.log

This log file contains all activity for the current day. Each days log file is renamed with a date and timestamp with the following name:

dataguideYYYYMMDD.log

A Maximum of 6 days of log files will be kept. Any log files older than this will be deleted.

The current log file can be accessed from the menu File -> Log Files -> Current Log. The default editor for log files (files with the extension **.log**) will be used.

History Log

To keep track of configuration changes, a log file is kept of all changes which impact a configuration. Each time a change is made to a configuration, the change is recorded in this file:

history.log

This file is not self purging. It will continue to grow as long as changes are made to any configuration (this file is in the same folder as the activity log file: i.e. **/log**).

This file can be accessed from the menu File -> Log Files -> History Log. . The default editor for log files (files with the extension **.log**) will be used.

Additionally, an icon on the status bar can be used to access this log directly:



Figure 23: History Log File Icon

Should the log file start to get large (exceed 2 Mb in size) the icon will change to an icon with a warning symbol:



Figure 24: Large History Log File

The log file can be simply purged by deleting unwanted lines from the file using any text editor, such as *Notepad*.

APPENDIX A: SCRIPT COMMANDS

Remarks

Comments and remarks can be placed on a separate line with the first character being one of the following symbols:

/ ; \ *

Commands

Each command is placed on a separate line.

Note: Commands are NOT case sensitive.

FileOpen

```
FileOpen "D:\LogFile.txt"
```

Opens a text file ready to write data to. This will open the file, and delete the current contents of the file.

FileAppend

```
FileAppend "D:\LogFile.txt"
```

This will open a file for appending data.

FileWrite

```
FileWrite "Some Text Here",%Fred
```

Writes data to an already opened file. If a file is not already opened (with the *OpenFile* command), then this will generate an error.

Data will be written exactly as placed after the command, except for any double-quotes.

Variables and configuration parameters will be replaced by their text equivalents.

FileWriteLine

FileWriteLine *Text To Write*

Writes a line plus a new line to the currently opened file.

FileClose

FileClose

Closes the file that was opened with *FileOpen*.

Only one file can be opened at any time.

Call

Call *NextScript*

Call another Script function from within a script. Note, beware of circular references.

MessageLevel

Messagelevel = *Line*

Sets a reporting messaging level to one of the following:

- **None** – Report Nothing
- **Error** – Reports Errors Only
- **Event** – reports Errors and Events (e.g. Script Completion)
- **Line** – reports active line execution, Events and Errors
- **Full** – Reports every line execution (including Remarks and blank lines), Events and Errors.

The messaging level indicates the level at which messages are displayed to the screen only.

This command can occur multiple times in the script.

LogLevel

LogLevel = *Error*

The *LogLevel* has the same values as *MessageLevel*, but instead indicates the messages written to the *DataGUIDE* Log file.

Let

Sets a variable to another value. The variable separator is not required for the variable being set.

```
Let ConfigVar1 = %[Speed]
```

Note: This can be used to assign an Input interface variable. This value will be set for the duration of the active script.

Note: if the variable name is not found, this will create a variable with that name, which is available for use for the duration of the active script (this is a *temporary variable*).

Note: Configuration Values can be Read Only. They can be changed for the duration of the script but will retain the value they had prior to the execution of the script.

Mail Functions

The following mail sending functions are available. The mail is designed to be sent to an SMTP (*Simple Mail Transfer Protocol*) server, or a server which accepts SMTP protocol messages.

MailTo

```
MailTo "JoeMann@joesserver.com"
```

Adds a 'MailTo' address for the message to be sent. Multiple addresses can be added by scripting multiple "MailTo" commands.

MailFrom

```
MailFrom "no-reply@machineserver.com"
```

A single valid e-mail address indicating who the mail is being sent from.

MailSubject

```
MailSubject "Shift Results for January 12, 2006"
```

A subject line for the mail.

MailMessage

```
MailMessage "Summary:"  
MailMessage "-----"
```

Each line is appended to the body of the message for the e-mail to be sent.

MailAttach

```
MailAttach "C:\reports\prodreport06-01-12.csv"
```

Allows a file to be attached to the e-mail. Although there is generally no limit to the number or size of attachments, it is more efficient to send smaller e-mails.

MailSend

```
MailSend "mail.machineserver.com"
```

Performs an actual send of an e-mail via the specified server.

MailAttachDelete

```
MailAttachDelete True
```

If a mail message is sent successfully, this indicates if the attachment(s) if any are deleted after being successfully being sent.

CleanupFolders

```
CleanupFolders D:\Data
```

This will delete any empty folders that exist in this folder. If all folders can be removed, then it will delete this folder.

It is not possible to delete a root folder.

APPENDIX B: IBALOGIC INTERFACE

A valid license is required to connect to the ibaLogic interface.

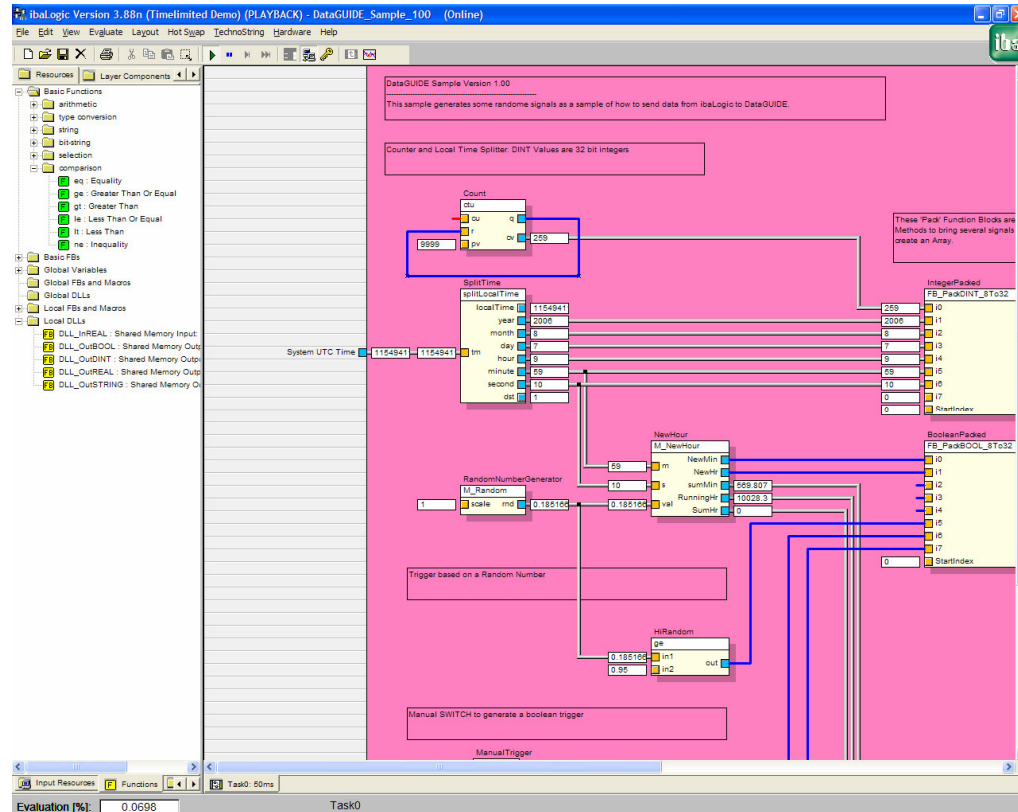


Figure 25: ibaLogic Sample

ibaLogic is a **Real Time Data Acquisition** product produced by iba (www.iba-ag.com) and has the ability for custom DLLs to be made using C++ (For example, Microsoft Visual Studio or Microsoft Visual C++ 2005 Express).

The interface to DataGUIDE is through custom DLLs. The DLLs can be found in the *Resources Treeview* under the *Functions* Tab (they can be viewed in the left hand treeview by selecting View -> Resources -> Function Blocks). At the bottom of the list are the Local DLLs.

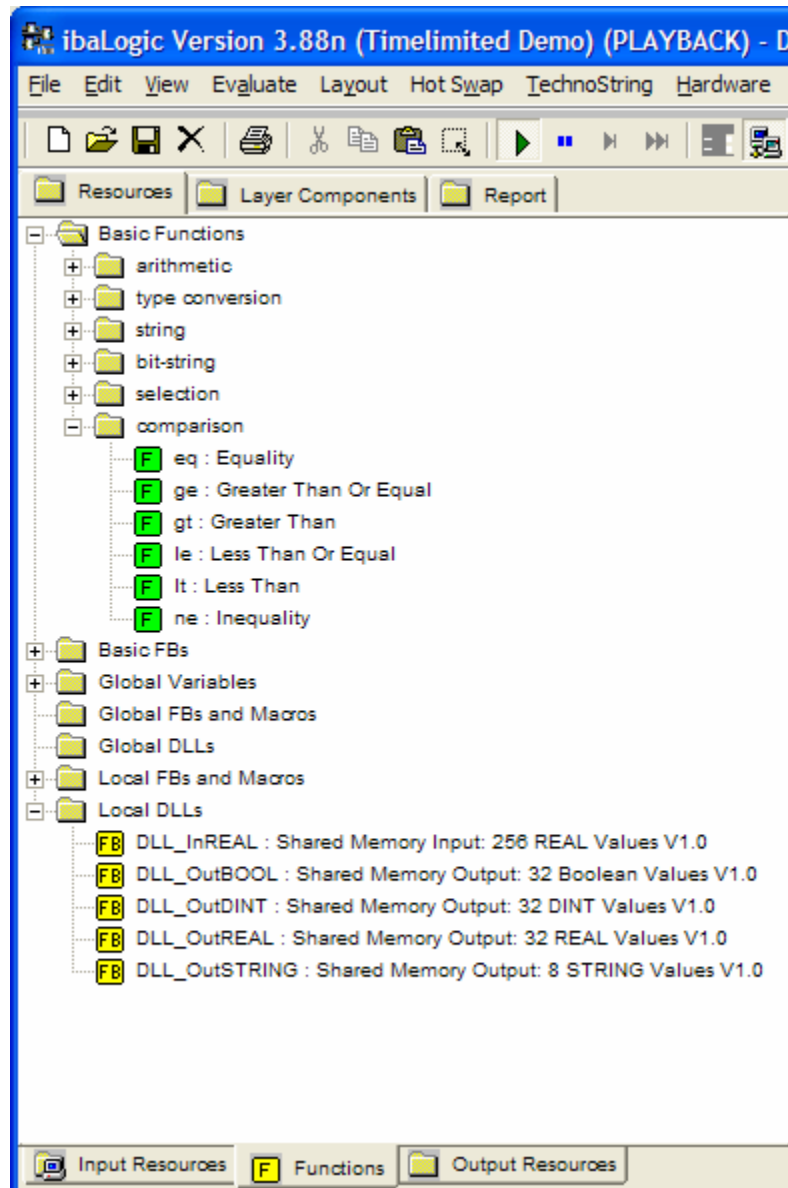


Figure 26: ibaLogic Function Tree

These DLLs can be dragged and dropped into the page like any normal function block.

The ibaLogic DLLs must be placed in the correct folder for them to be recognized by ibaLogic:

{ibaLogic Install Folder}\ configuration\DLLs

Typically, ibaLogic is installed in a root folder, since it is running on a dedicated ibaLogic machine. The folder would be:

C:\ibaLogic\configuration\DLLs

The layouts, or schematics, are in the schematics folder:

C:\ibaLogic\configuration\schematics

Note: Schematics do not have to be in this folder, but are placed here for convenience.

Upon installation of DataGUIDE, the DLLs and the sample layouts will be installed in the DLLs folder and Schematics folder, respectively. In this case, ibaLogic must be installed before DataGUIDE.

If DataGUIDE is installed before ibaLogic, the installation can be re-run to correctly place the required files, or may be manually copied from the **/support** folder in the DataGUIDE installation folder to the above described folders.

DLL OUTPUT Interfaces

For *DataGUIDE* to connect to ibaLogic, several ibaLogic DLLs are required. These DLLs are installed when *DataGUIDE* is installed.

Note: ibaLogic should be installed PRIOR to DataGUIDE being installed. This will ensure that the DLLs described here are made available to ibaLogic.

There are several DLLs associated with the *DataGUIDE* Interface, based on the data type:

- **DLL_OutDINT**
Data Type DINT (32 bit Integer) as an array of 32 values.
- **DLL_OutBOOLEAN**
Data Type Boolean (True/False) as an array of 32 values.
- **DLL_OutREAL**
Data Type REAL (32 bit floating point) as an array of 32 values.
- **DLL_OutSTRING**
Data Type String (32 bit floating point) as 8 input values.

Note: the ibaLogic (IEC) DINT value is equivalent to a standard signed Integer value of 32 bits.

Error Numbers

Each DLL has an Error output used for troubleshooting. Ordinarily, the error number will be zero. The following lists the error numbers, a description and a possible cause:

Error Number	Description	Possible Cause
0	No error	
1	Cannot Create Memory Map	This is an unusual error and indicates a fundamental problem with the computer
2	Memory Map Already Exists	Each of these DLLs (Function blocks) creates their own memory region. If the memory region exists this error is displayed. Check for other applications creating Shared Memory
3	Heartbeat Error	The internal heartbeat has been modified outside this block: check for other blocks of this type with the same ID number
4	Invalid ID Number	The ID Input is outside the possible range of values (0 to 255)

Output DLLs

The output DLLs have the name **DLL_OutXXXX** where **XXXX** signifies the data type.

*Note that different DLL types can have the same ID Number. For example, a **DLL_OutDINT** with an ID number of zero (0) will be identified separately from **DLL_OutBOOL** with the same ID Number.*

These DLLs communicate with DataGUIDE through Shared Memory. Refer to the *DataGUIDE Technical Reference* for the structure of the shared memory for each DLL.

DLL_OutDINT

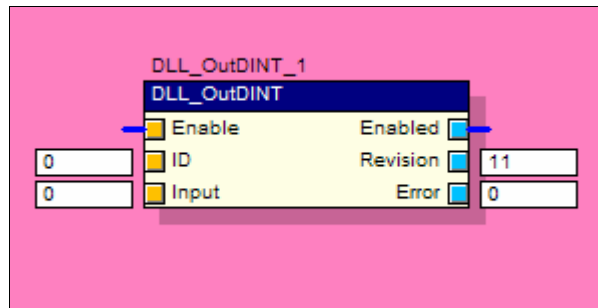


Figure 27: ibaLogic Integer Output DLL

Inputs

Name	Data Type	Description
Enable	Boolean	Enables the block for use
ID	DINT	Unique value to identify this block ranging from 0 to 255
Input	DINT[0 to 31]	Array of DINT Values to pass to DataGUIDE

Outputs

Name	Data Type	Description
Enabled	Boolean	Echoed value of the Enable Input
Revision	DINT	Compiled DLL Revision number
Error	DINT	If an internal error occurs, this will be set to a non-zero value

DLL_OutBOOL

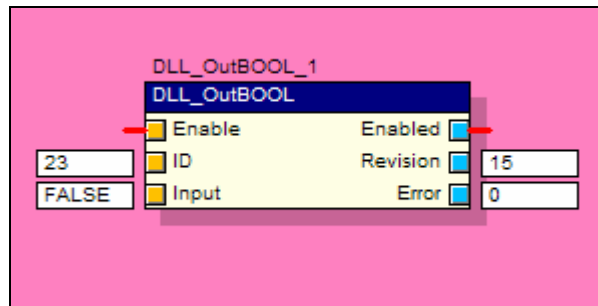


Figure 28: ibaLogic Boolean Output DLL

Inputs

Name	Data Type	Description
Enable	Boolean	Enables the block for use
ID	DINT	Unique value to identify this block ranging from 0 to 255
Input	BOOL[0 to 31]	Array of BOOL (Boolean) Values to pass to DataGUIDE

Outputs

Name	Data Type	Description
Enabled	Boolean	Echoed value of the Enable Input
Revision	DINT	Compiled DLL Revision number
Error	DINT	If an internal error occurs, this will be set to a non-zero value

DLL_OutREAL

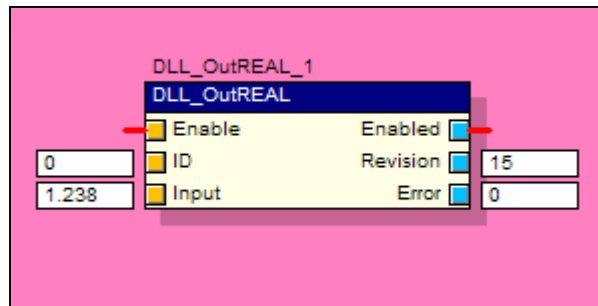


Figure 29: ibaLogic Real Output DLL

Inputs

Name	Data Type	Description
Enable	Boolean	Enables the block for use
ID	DINT	Unique value to identify this block ranging from 0 to 255
Input	REAL[0 to 31]	Array of REAL (Single, 32-bit) Values to pass to DataGUIDE

Outputs

Name	Data Type	Description
Enabled	Boolean	Echoed value of the Enable Input
Revision	DINT	Compiled DLL Revision number
Error	DINT	If an internal error occurs, this will be set to a non-zero value

DLL_OutSTRING

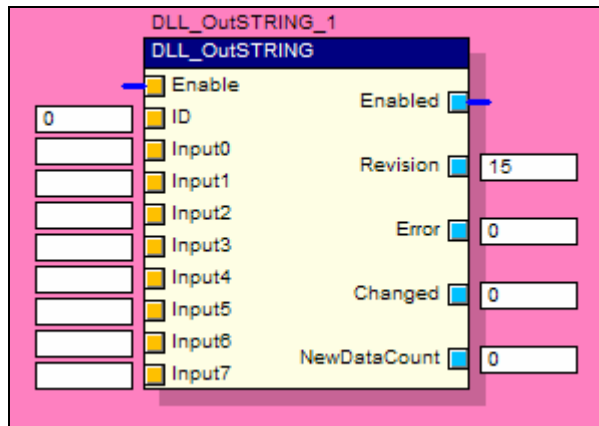


Figure 30: ibaLogic String Output DLL

Inputs

Name	Data Type	Description
Enable	Boolean	Enables the block for use
ID	DINT	Unique value to identify this block ranging from 0 to 255
Input0	STRING	A 1024 max length STRING Value
...	"	"
Input7	"	"

Outputs

Name	Data Type	Description
Enabled	Boolean	Echoed value of the Enable Input
Revision	DINT	Compiled DLL Revision number
Error	DINT	If an internal error occurs, this will be set to a non-zero value
Changed	DINT	Indicates the Input channels which changed on the current scan
NewDataCount	DINT	Incrementing counter which increments each time an input changes (maximum of 9,999)

DLL INPUT Interfaces

In addition to receiving data from ibaLogic, data can be transferred back to ibaLogic. For example, the results from a Database Query providing set points can be transferred to ibaLogic to perform real time data calculations, or may be transferred to the running machine through the many interfaces available to ibaLogic.

These DLLs are similar to the output DLLs, and are treated exactly like function blocks.

DLL_InREAL

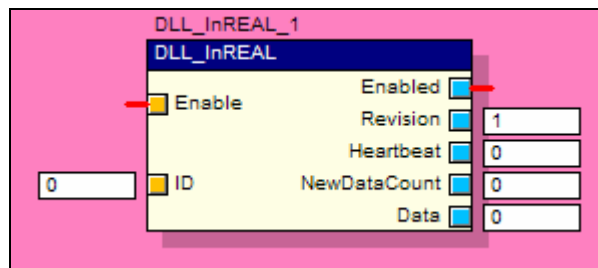


Figure 31: ibaLogic Real Input DLL

Inputs

Name	Data Type	Description
Enable	Boolean	Enables the block for use
ID	DINT	Unique value to identify this block ranging from 0 to 255

Outputs

Name	Data Type	Description
Enabled	Boolean	Echoed value of the Enable Input
Revision	DINT	Compiled DLL Revision number
Heartbeat	DINT	This value is controlled by the external Interface (DataGUIDE) and indicates a valid connection
NewDataCount	DINT	Incrementing counter which increments each time an input changes (maximum of 9,999)
Data	REAL[0 to 255]	An array of 256 REAL values

Sample Layouts

A sample test layout is also installed. This test layout demonstrates the ibaLogic DLLs and how they are connected and used.

DataGUIDE_Sample_100.lyt

This layout shows an example of the following:

- Manual Digital Trigger
- Automatic Digital Trigger
- DINT Values
- Real Values

Each Output Function Block has an ID of zero. Each data type has its own sequence of unique ID Numbers (0-7 for strings, 0-31 for other data types).

This sample layout should be loaded into ibaLogic.

APPENDIX C: IBA PDA INTERFACE

A Valid License is required to connect to the PDA Interface. Additionally PDA Version 6.8 or greater, with Plugins enabled is required.

Note: support for PDA is offered directly through iba. Please contact the iba representative in your area: www.iba-ag.com

The PDA Interface is selected from the Interfaces section.

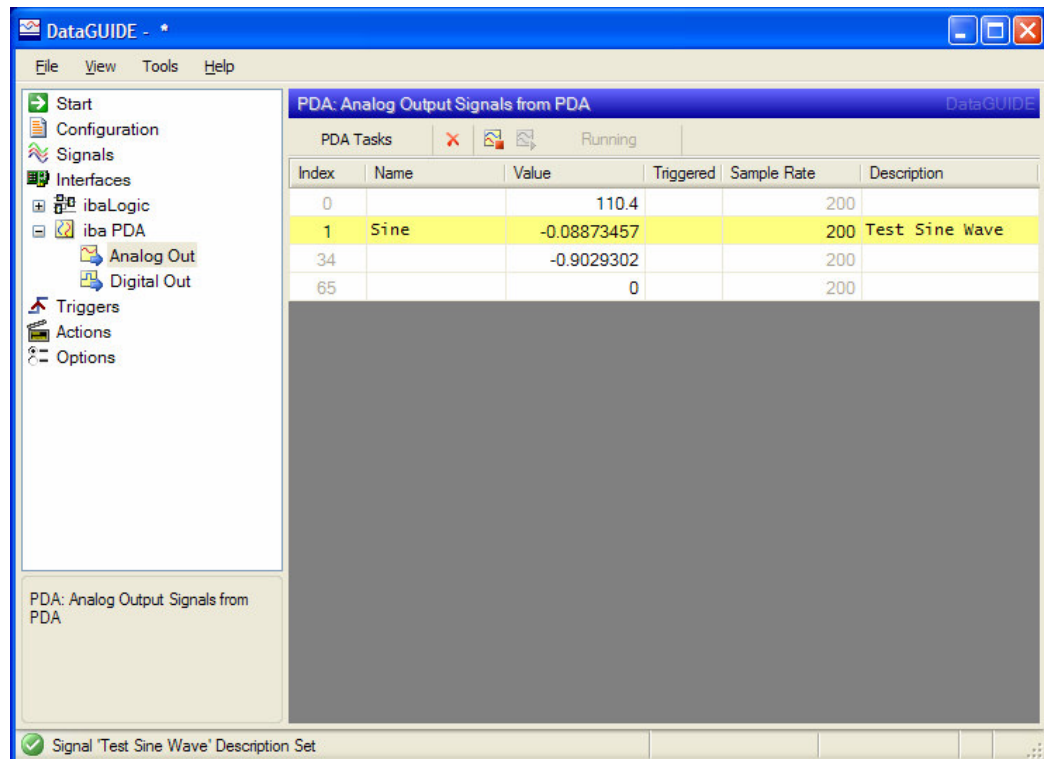





Figure 32: iba PDA Interface

PDA Interface Toolbar



Figure 33: PDA Interface Toolbar

The toolbar allows the deletion of configured signals, as well as stopping and starting the PDA Service. It also shows the current status of the PDA Data Acquisition Service:

-  Delete the selected signal – Note: deleting the signal does not remove it from PDA.
-  Stop the PDA Service
-  Start the PDA Service

Stopping the PDA Service will stop any Data Acquisition.

IMPORTANT NOTE:

For the signals to appear in the DataGUIDE, the PDA Service must be started after DataGUIDE. Once PDA is running, DataGUIDE may be shutdown and restarted at will.

Alternatively, the PDA Service can be started as the logged on user account, rather than running as a SYSTEM account.

Configuring PDA

The PDA application needs configuring to send data to DataGUIDE. Signals are configured through *Virtual Signals*.

Additionally, the Plugin DLL needs to be in the correct folder. If PDA has been installed prior to the installation of DataGUIDE, this file should already be present. The file can be copied manually using the following procedure:

- Locate the file in the DataGUIDE Install location:
`/support/ScePdaPlugin.dll`
- Copy the file to the iba PDA Installation location, typically:
`C:\Program Files\iba\ibaPDA\Server\Plugins`
- Restart the PDA Service

If the PDA Service is running, and the file already exists and is in use by PDA, then the PDA Service must be stopped prior to copying the file.

Stopping the PDA Service will stop all Data Acquisition.

Signals can be configured through the PDA Client.

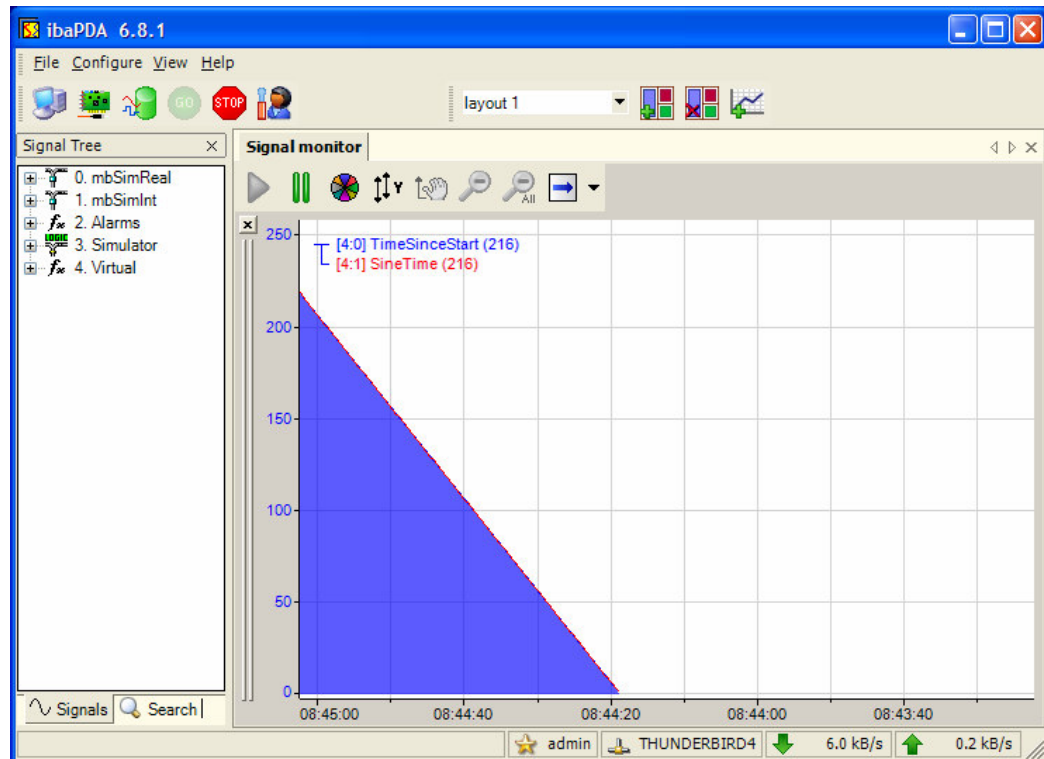


Figure 34: iba PDA Client

To begin configuring signals the PDA Service must be running. From the client, select from the Configure Menu, or select the Configure I/O Manager toolbar icon to open the I/O Manager.



Figure 35: Configure I/O Toolbar Icon

The I/O manager configures all I/O data points. The Virtual data points allow configuring of *Virtual* signals. It is in this way that signals can be sent to and from *DataGUIDE*.

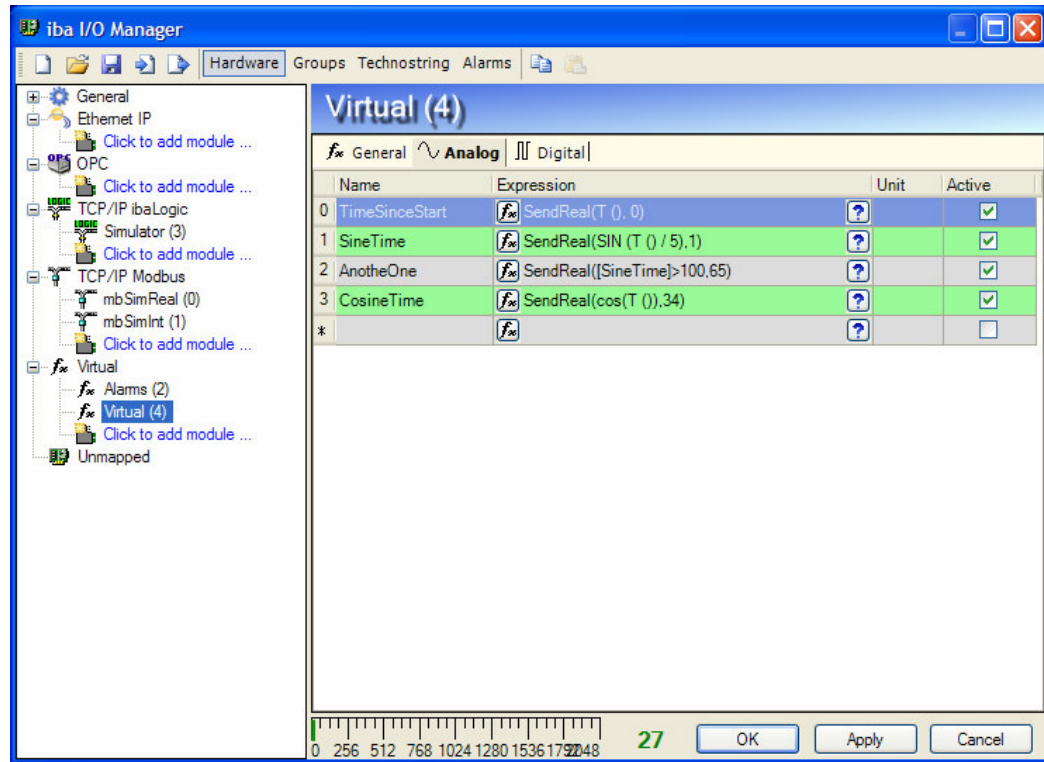


Figure 36: iba PDA I/O Manager

Either create a new virtual module, or add signals to an existing virtual module. Select the Function button (f_{sc}) to bring up the expression builder.

Under the Math Functions, there is the **Sc Send Values** Plugin. The **SendReal** Plugin will allow a single data point to be sent to *DataGUIDE*.

Up to 256 values can be sent to *DataGUIDE*.

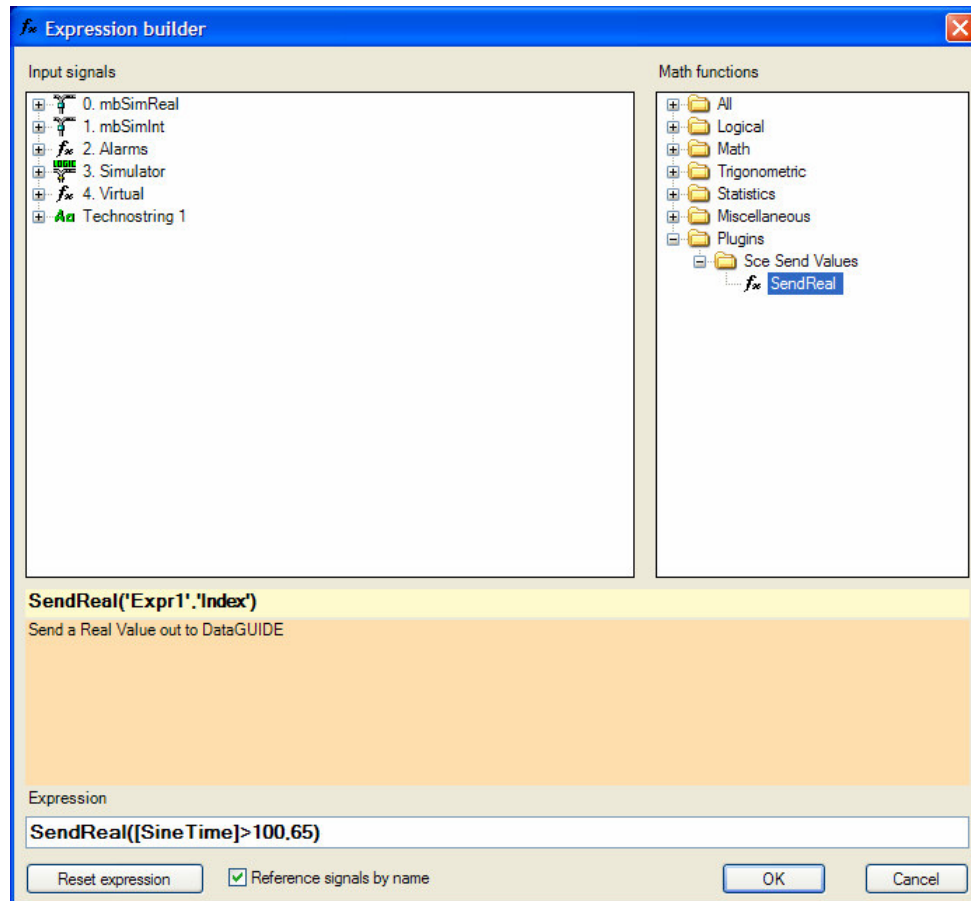


Figure 37: iba PDA Expression Builder

The `SendReal()` function takes two parameters:

- The Value to send
- The Index of the value

Each sent value must have a unique index, or else corrupted data may get sent.

Currently only Analog values can be sent. However, digital values can be used as the Value (Expr1) parameter resulting in a value of One (1) for True and Zero (0) for false.

The return value is an increasing value from 0 to 255. Should an error occur sending the value the return value will be constant, indicating the type of error.

Error Number	Description	Possible Cause
-5	Index out of range	The second parameter in the <i>SendReal()</i> function is less than zero or greater than 255
>0	Windows System Error Code	Some examples of typical error codes are as follows:
5	Access Denied	DataGUIDE is preventing communication. Stop and restart PDA from DataGUIDE .
87	Invalid Parameter	See error 5

Once signals have been configured in PDA, it will be necessary to restart the PDA service.

Signals can be accessed within DataGUIDE by the Index defined as the second parameter for the *SendReal()* function.

For support on configuring plug-ins and licensing for PDA please contact your local iba Representative www.iba-ag.com

APPENDIX D: TCP/IP INTERFACE

DataGUIDE can act as a TCP/IP server. Messages can be sent to and from DataGUIDE over a standard network using TCP/IP as a base protocol, using standard network hardware.

However, a protocol needs to be defined so that the client and DataGUIDE (server) understand one another.

Servers and messages are defined independently.

Servers

DataGUIDE can have zero, one or more servers (When the word *server* is referenced in this document, it does not refer to a separate computer, but to a Socket Server capable of accepting multiple clients on a single port number).

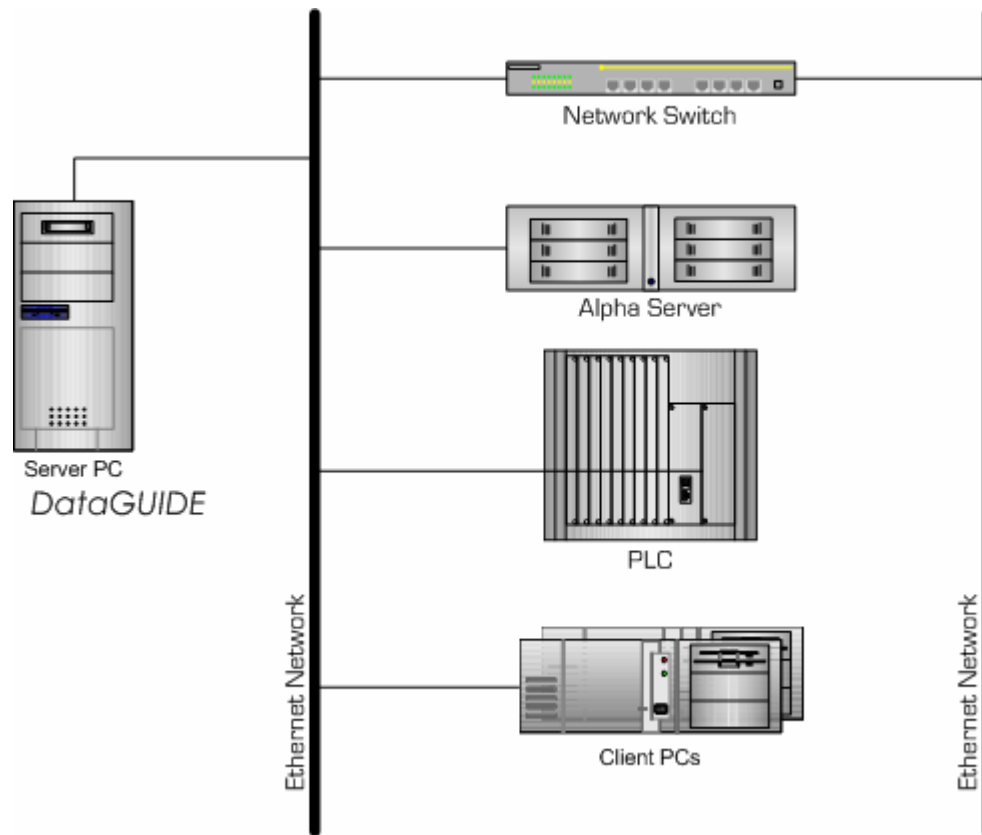


Figure 38: Example of the physical Network

Each server communicates using a specific Port Number, and uses a defined protocol. Servers must have a unique Port number. Each server can accept one or more client connections.

All messages to and from a specific server must use the same protocol to prevent garbled messages or errors.

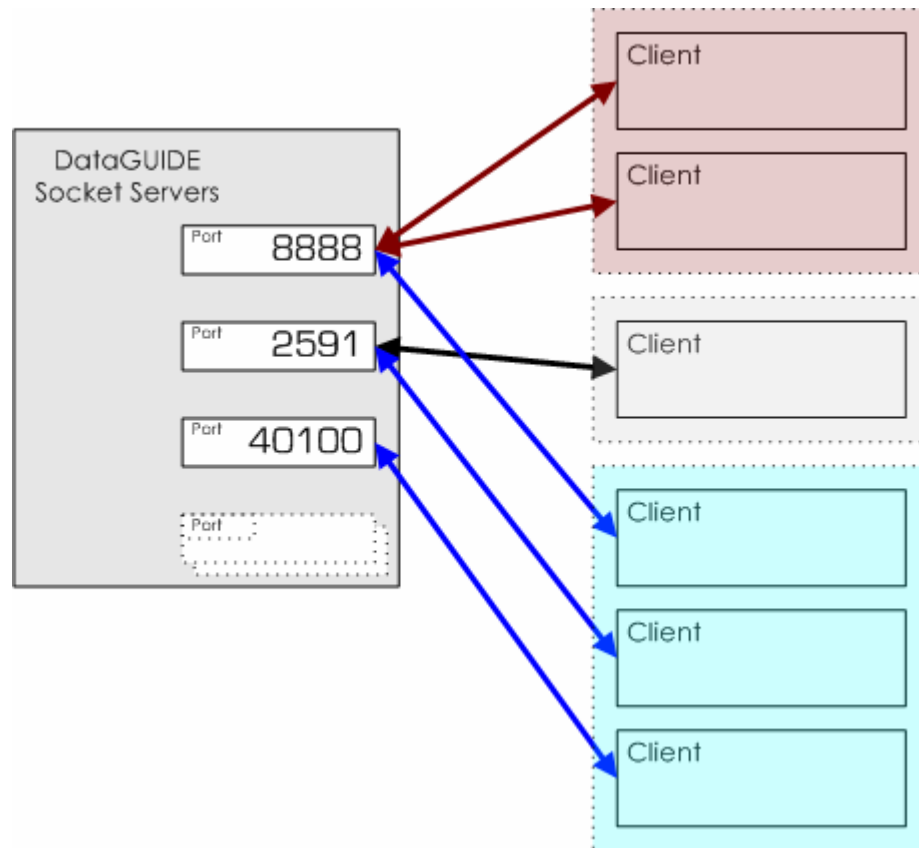


Figure 39: Multiple Servers serving Multiple Clients

Each server can be configured to send and receive specific messages independently. Messages are not tied to a specific server or client, unless configured as such.

Messages

Each defined message is assigned a unique ID number (Message ID). The message structure is independent of the protocol used, thus the same message can be sent (or received) from different servers utilizing different protocols.

Each message definition can be fixed length (bytes) or variable length (string) as a CSV String.

*A 'send' message must not have the same ID number as a 'receive' message. A Unique ID number is sent **or** received, not both.*

Receive Messages

It is possible to receive the same message on multiple servers, or from multiple clients; it is important to recognize this capability, because unpredictable data may be seen if multiple clients are sending the same message.

Send Messages

Whenever a message needs to be sent, it will be sent to all clients connected to the server that has the ability to send such a message.

Messages can be sent on a **timed** basis, or they can be **triggered** to be sent using a trigger.

Protocols

There are several built-in protocols, but they are based on a messaging structure: data is packed into uniquely identified messages for transmission. Each message has the following basic structure:

Message	Section	Description
	Header	Fixed byte length structure defined by the protocol
	Data	Zero or more bytes containing the actual data

The following, named, protocols are available, each with a different header structure:

- Basic
- Standard
- Extended

There are several other protocols which are special cases for specific applications. The messaging structure is predefined for these protocols:

- PyramidHMI
- ModbusTCP

These protocols are currently not implemented.

Header Structure

The header structure is a fixed length (bytes) structure defined by the protocol used by a specific server.

Protocol	Total Size (bytes)	Information Contained	Heartbeat Message Required
Basic	4	Message Size (2 bytes) Message ID (2 bytes)	No
Standard	6	Message Size (2 bytes) Message ID (2 bytes) Message Counter (2 bytes)	Yes
Extended	14	Message Size (2 bytes) Message ID (2 bytes) Message Counter (2 bytes) Message Timestamp (8 bytes)	Yes

Basic Header

Offset	Size (bytes)	Description
0	2	Counter indicating the TOTAL Size of the message in bytes, including header
2	2	Identity of the message (Message ID)

The basic protocol header size is **four (4) bytes**.

Standard Header

This is the **recommended** protocol to use for slow, messages (messages that get sent on a period of several seconds to minutes or hours).

Offset	Size (bytes)	Description
0	2	Counter indicating the TOTAL Size of the message in bytes, including header
2	2	Identity of the message (Message ID)
4	2	Message Counter. Each message has its own unique counter associated with it

The basic protocol header size is **six (6) bytes**. Additionally, this protocol definition also requires a heartbeat message to be sent periodically.

Extended Header

Offset	Size (bytes)	Description
0	2	Counter indicating the TOTAL Size of the message in bytes, including header
2	2	Identity of the message (Message ID)
4	2	Message Counter. Each message has its own unique counter associated with it
6	8	Timestamp for the message

The basic protocol header size is **fourteen (14) bytes**. Additionally, this protocol definition also requires a heartbeat message to be sent periodically.

The timestamp structure (8 bytes) is shown below:

Offset	Size (bytes)	Description
--------	--------------	-------------

6	1	Year (last two digits)
7	1	Month
8	1	Day
9	1	Hour
10	1	Minute
11	1	Second
12	2	Millisecond

Reserved Messages

These messages serve a special function. These ID numbers are not available for use when defining the messages.

All user defined message ID Numbers must be less than **8000h**. Message IDs above this value are reserved for special messages or future use.

Message ID zero (**0h**) is an 'empty' message. It can be sent with any data content, but will be ignored.

Heartbeat Message

Message ID: FFFFh

The Standard and Extended protocols expect a heartbeat message to be sent periodically. This is used to ensure that a connection is established.

The heartbeat message will be sent from the server approximately **once per second**. The client may simply ignore this message if so desired.

The heartbeat message consists of a **header only** (no message data).

Disconnect Message

Message ID: FE00h

Any protocols which specify a heartbeat message also use a Disconnect Message. This message is to be sent prior to the client disconnecting.

The disconnect message consists of a **header only** (no message data).

Empty Message

Message ID: 0000h

This message ID is recognized, but ignored. Do not use this for any messages that need to be interpreted. The message data may be any length, but will simply be ignored.

Message Data

Messages are defined with a unique ID number from **0001h** to **7FFFh** (1 to 32767).

The messages can be designated as fixed length (Binary) or as a CSV – comma separated variable (ANSI).

The maximum message data length that can be interpreted is **10240 bytes** (10Kb).

Messages above this length will be considered invalid messages: should the server receive such messages it will attempt to 'reset' it's receive buffer, until it receives a valid message. This may cause data loss, and is designed to prevent any kind of buffer overflow errors.

Fixed Length Message Data (Binary)

Fixed length messages are defined based on data type of the values transmitted in the message. Data types have the following sizes:

Data Type	Size (bytes)	Description
Boolean	1	Zero = False, Non-Zero = True
Integer	4	Standard 32 bit Integer
Float/ Single	4	IEEE Floating Point Representation
String	1/char	ANSI Character representation (1 byte per character)

Fixed length messages are assembled from a mixture of these data types, and are accessed by index within the message.

CSV Messages (String)

CSV Messages are string data only, and will be considered as ANSI encoded (1 byte per character), and will use the current codepage of the computer on which *DataGUIDE* is running.

If the string data is transmitted as plain ASCII representation (no character codes above 127) then the data will be interpreted correctly.

Vales will be accessed by *index* within the CSV string.

Note: Internally, DataGUIDE stores most variables as strings. This ensures that the correct data is available when the variables are used in scripts, log files, etc. and can be formatted appropriately.

Defining a Client Protocol

The following section discusses methods to help define how data is sent and received. Due to the open nature of TCP/IP communication, these are not hard rules to follow, but rather a guide. DataGUIDE has been designed around these rules, and is generally works most efficiently when these guidelines are implemented.

Protocol/Port Number

The **Standard** protocol is recommended for most clients and most usage.

The port number can be any available port, but it is recommended to keep all port numbers above the standard industry *common port numbers* (i.e. greater than 1023).

There are many port numbers which are commonly used (**1024 to 49151**), but most numbers are generally safe to use in an isolated network (not used over the internet). Even so, there are many ranges within this group which would be acceptable to use.

Port numbers above 49151 should *not* be used, and are considered dynamic, or private, port numbers.

Messages

A list of data to send and a list of data to receive should be decided upon.

This data should then be grouped by the 'period' under which it should be sent or received. Typically, data can be grouped in one of the following categories:

- **Fast periodic data** – data sent continuously at a rate faster than 1 second (1000mS).
- **Slow periodic data** – data sent continuously at a rate slower than 1 second (1000mS).
- **Event based data** – data which changes seldom, or on a trigger event, slower than a 1 second interval, with potentially many minutes or hours between events.
- **Streaming Data** – this is data which is required faster than 100mS. This is considered real-time data. **Currently DataGUIDE is not configured to accept such data at such a high rate, over TCP/IP.**

Some examples of how to categorize data are as follows:

<i>Data</i>	<i>Message type</i>	<i>Description</i>
Line Speed	Fast	Immediate indication of how fast the machine is running
Actual Tension	Fast	A Measured Value is usually transmitted periodically
Tension Setpoint	Slow/Event	Depending on the implementation, such a value may be sent as a SETUP message (Event) or as a slow value

Profile Data	Slow	Profile of a strip often isn't generated faster than every few seconds
Shape Data	Fast/Slow	The hardware used may dictate the speed at which such data is transmitted
Product ID	Event	Usually part of a setup Message
Deviation	Fast	A Measured value is usually transmitted periodically
Alarm	Event	Alarms are generally event based conditions

Message data should then be grouped by function. For example, it makes sense to group all setup data (PDI – Process Data In) into a single message, likewise for output data (PDO – Process Data Out).

Running data should be grouped into Fast and Slow messages, as appropriate.

Once data has been grouped, depending on its final destination, messages should be assigned ID numbers. It's likely that some groups (Fast, Slow, etc.) may have a great number of values to send. To better visualize such data, it may be appropriate to distribute such messages over multiple messages, each with a maximum number of indexes (typically, 32 is a reasonable size).

It may also make sense to break the groups of data into more numerous messages because it logically divides up the 'machine'.

For example, Fast data could be grouped into five (5) messages, because there are five areas of the machine (e.g. a 5 stand Hot Mill, or a 5 robot welding line).

Message Type

Once the protocol has been decided, and the signals that are required, the choice is to decide on a CSV format or fixed format message structure. The choice is often dictated by the client: machine type, or by the 'type' of data being transmitted.

CSV (Comma Separated Variable)

Advantages:

- Simple to build (in many PC languages, such as VB or C#)

- Easy to troubleshoot the actual data transmitted
- Can handle messages easily that may contain variable length data (strings).

Disadvantages:

- Messages may be longer than with fixed length messages for numeric data types
- Messages are variable length causing difficulty in troubleshooting or initially setting up the byte structure (on the client – client dependent).

Suitable For:

- Mostly string data
- High level clients – advanced string handling capability
- Variable message length not a concern

Fixed Length Binary Messages

Advantages:

- Suitable for numeric data types
- Easy to troubleshoot at the lower level – all messages should be a fixed length

Disadvantages:

- Cannot handle variable length string data – strings must be a fixed length
- Difficulty troubleshooting actual data values on client

Suitable for:

- Mostly numeric data
- Clients which has limited string capability
- Tight control of message byte length

Message Structure Definition

Messages can be defined by selecting the Interfaces->TCP/IP item in the treeview. Each message has a unique ID number, as well as a unique name. The ID Number will be transmitted in the header. Each message is defined as an INPUT message (to DataGUIDE) or an OUPUT message (From DataGUIDE).

Each message is designated a CSV message or Fixed Length.

Note: Servers can send a mixture of CSV and fixed length messages, and are not restricted to a specific message type.

CSV Messages

CSV Messages have a defined 'index count'. The values within that message are split on a comma value ','. Each item in the message will be interpreted as its string representation.

The length of this message is not shown since it is a variable length message.

{Screen Shots To Be Inserted Here}

Fixed Length Messages

Fixed Messages require that each item in the message have a data type. Additionally, strings must also have a length (in bytes) defined. Each byte is a single ANSI Character.

The length of the message, the length of each item in the message and the offset the items are shown for this message type.

Note: the length of the data portion of the message is shown; and NOT the full length of the message (including header), since this is protocol dependent.

{Screen Shots To Be Inserted Here}

FAQ

This is the frequently asked questions (FAQ), and is designed to address some common questions or issues that may arise.

General

Q1: What files are installed with *DataGUIDE*, and where are they installed?

A1: By default, DataGUIDE is installed in the folder `C:\DataGUIDE`. The following core files are installed:

`DataGUIDE.exe` – the main DataGUIDE executable

`CrashLog.exe` – helper application, should DataGUIDE crash or be caused to crash by any other applications (allows the reporting of the cause of the error)

`/Documentation/DataGUIDE - Manual.pdf` – this manual

`/Documentation/eula.rtf` – End Users License Agreement

`/Log` – no files, but all log entries will be in this folder

`/Support` – file contents varies: the files in this folder depend upon the interfaces available in the install version. See the FAQ for each interface for the files in this folder.

ibaLogic

Q1: What files are required for the ibaLogic interface to function correctly?

A1: ibaLogic requires several DLLs to be installed in the `ibaLogic/Configuration/DLLs` folder. If DataGUIDE is installed after ibaLogic has been installed and *run at least once*, then these files will be installed correctly.

If DataGUIDE is installed prior to ibaLogic being executed for the first time, then there are two methods to put the files in this folder:

1. Run DataGUIDE installation again.

2. Copy the files from the DataGUIDE installation support folder:
`/Support/ibaLogic/Configuration/DLLs` These files are:

`DLL_InReal.dll`

`DLL_OutBoolean.dll`

`DLL_OutInteger.dll`

`DLL_OutReal.dll`

`DLL_OutString.dll`

Restarting ibaLogic will now show these DLLs in the ibaLogic Resources Treeview under DLLs.

PDA

Q1: What files are required for the iba PDA interface to function correctly?

A1: iba PDA requires the Plugin DLL to be installed in the plugins folder for PDA, typically:

`c:\program files\iba\ibaPda\Server\Plugins`

If PDA is installed prior to DataGUIDE being installed then this file will be copied to this folder during installation. If PDA is installed after DataGUIDE then there are two methods for installing this plug-in file correctly:

1. Run DataGUIDE installation again.
2. Copy the files from the DataGUIDE installation support folder:
`/Support/ibaPda/Server/Plugins` These files are:

`ScePdaPlugin.dll`

The PDA server service must be stopped and then restarted for the interface function to show up as a virtual function.

Additionally, PDA must have the plug-in license for plug-in functions to be made available: please contact iba (<http://www.iba-ag.com/>) for support and availability of this functionality.

TCP/IP Protocol

Q1: Does DataGUIDE support connections to a TCP/IP Server?

A1: No, currently DataGUIDE will only accept connections from TCP/IP clients. This functionality will be incorporated into a future release of DataGUIDE.

Q2: Does it matter which protocol is used for the connection?

A2: No it doesn't matter.

Q3: is there a drawback to sending numeric values as a CSV string, or strings as fixed length messages?

A3: No. The only issue is that strings will have to be fixed length in a fixed length message. Note, however that only the maximum string size is specified: shorter strings can be sent.

Q4: if strings are sent in a fixed length message, must the string value be always the specified length?

A4: Strings may be shorter than the specified length by padding the end of the string with null characters/bytes (zeroes). Alternatively, the string can be padded with spaces.

VERSION INFORMATION

08/28/2006 Version 1.0.16

Latest DataGUIDE Version.