Eurotherm®

by Schneider Electric

Eurotherm PAC

Operations Viewer, Shell Application Internals Manual

Issue 5

October 2017
HA028882

Table of Contents

Key Shell Application Features

What is the ‘Shell Application’ and what does it provide beyond a bare Wonderware InTouch application?

The shell application has been developed to assist users in developing a set of screens to be used for operating their plant. The aim was to provide as many features as possible that are common to the requirements of all users and thereby save significant engineering effort on every job that includes InTouch view screens. Since the initial product release efforts have been made to improve and add to the shell application by including elements of development made on customer applications. This does make the shell application large and, therefore, difficult to modify. This document provides reference information on how the standard features are implemented.

User Interface Framework

The application display area is split into three sections:

- **Alarm Banner** - the top section is primarily for alarm annunciation but also has some interactive features that allow alarm pages and help screens to be called up.
- **Display Area** - the centre section is the main display area and would normally contain the 'home' display or overview by default.
- **Button Bar** - the bottom section is a collection of buttons and menus that allows the user to call up both standard displays and custom displays.

Standard Displays

There are a number of standard displays that provide:

- **Alarm Summary** - a list of current alarms by alarm group.
- **Alarm History** - alarms and events as logged to a SQL database.
- **Help Display** - brief help on the 'User Interface Framework'.
- **History Display** - real time and historic trends.
- **Point Pages** - predefined display with tag parameter list and other tag related data.
- **Faceplates** - predefined displays for operator interaction.

There are many more windows that provide support for the standard displays and graphical wizards.

Graphical Wizards

Graphical wizards allow relatively complex graphical objects to be placed onto displays and configured through simple dialog boxes. They are constructed using Microsoft Visual C++ and a Wonderware toolkit. Two wizard libraries are installed with the shell application.

InHouse Wizards - most of these wizards are used in predefined displays and do not need to be used by a user although there are some faceplates which can be placed onto custom displays.

Control Module Wizards - these wizards are designed to complement the more complex LIN blocks known as 'Control Modules'. As well as valves and motors there is also the 'LIN Data Text' wizard that is used to display data and also allow secured user writes.

Server Redundancy

Computers within the system can be defined as servers or clients. The servers will provide data and alarms for display on the clients. In almost all cases the server is also a client i.e. the InTouch graphics is running on the system.

If two servers are configured as a redundant pair then the standard graphics will automatically monitor the health of the servers and choose which server to use as the source of data and alarms. There is no need for the servers to communicate because they always provide data.

Security

Levels of access and levels of confirmation can be configured for users using Security Manager. If security is enabled then a user logging into the View station will have their access controlled and logged. The system does not just control access to tags (and fields) but also to displays, recipes and to the operating system. An interface is provided to allow a user to change their password. There is automatic logout after a configurable period of inactivity.

Language Support

The shell application contains a script to translate displayed or logged text on the fly. This means that the language may be switched at runtime.

The Alarm Banner

How the alarm banner presents the alarm information to the user.

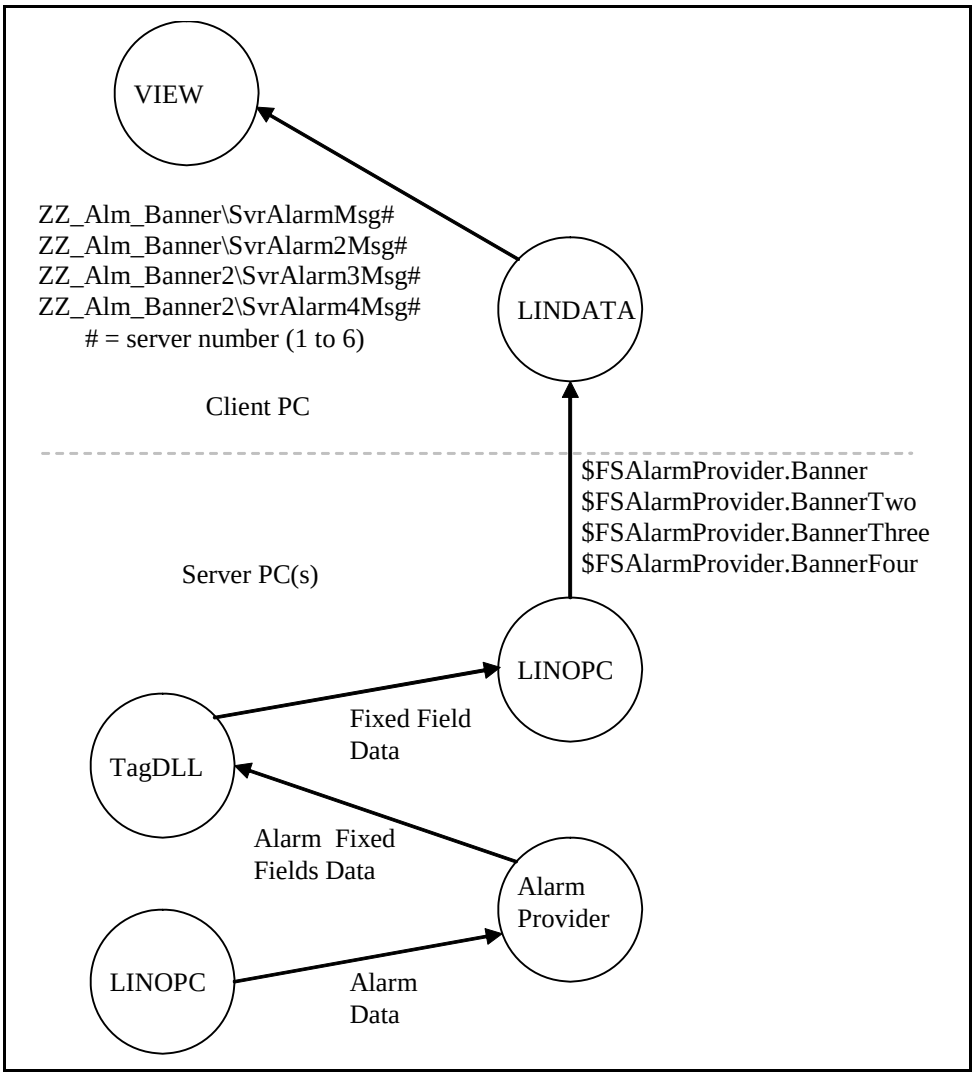
The alarm banner picture is called **ZZ_AlarmBanner** but there is a lot more to it than just the picture. The data that is shown is derived from data that is sourced from one or more servers and that data has to be transferred and interpreted before it is displayed. There are three parts of the alarm banner that would not necessarily reveal how they are implemented when inspected from WindowMaker:

- One is the alarm group that shows a summary of alarms across the system in priority order in the form of alarm group names surrounded by coloured boxes.
- A closely related item is the ‘two line’ alarm summary that shows the highest priority alarms from the system.
- Then there is the server alarm feature that will indicate if any of the servers have a problem with providing data and alarms.

Alarm Group Boxes

There are eight alarm group boxes across the bottom of the **ZZ_AlarmBanner** window. If more than eight groups contain current alarms then the user can display all of the alarm groups by displaying the **ZZ_Expanded Alarm Buttons** window (there is a button on the alarm banner for this that looks like a drop-down).

Each of the alarm group boxes is a wizard. If the wizard is opened and the dynamic features are examined it can be seen that there are some ‘script extension functions’ which return data about the alarm groups. The reason for the script extensions is that they allow for much more efficient processing of the data as the collation of all the alarm group data from the system would overload the InTouch scripting engine. These script extension functions are written in Microsoft Visual C++ and the functions conform to an interface standard as defined by Wonderware. The following figure shows the flow and transformation of data from the servers to the graphic.



There are a number of steps that bring about the setting up for the alarm banner to work, as follows:

1. Operations Server Configurator is used to configure the systems in the computer network and the access names to be used to get data from the servers. Access names for the data provider ‘LINDATA’ define which servers provide alarm information from LINOPC. Access names that provide data from ‘VIEW’ define which servers provide InTouch alarms.

2. When View starts it runs the script *ZZEaseStartup* which does a lot of initialisation (look for the comment "This sets up access names etc…"), including setting up the pre-built access names and all the tags in the tag dictionary to allow alarm data to be collected. The script extension function *EaseServerAccessName* is used to return information from the script extension DLL (originally from the project database) and to give a count of servers that need to be set up. The function *EaseServerProvidesAlarms* is used to determine if there will be alarm data from the server.

Note: The script referred to here is also important for server health monitoring and the server redundancy logic.

3. Once the reading of the data has been set up it then has to be monitored and processed. This is done in the 'While Showing' script of the window **ZZ_AlarmBanner**. The data from the alarm providers that is now in string form in the tag dictionary is fed into the script extension DLL using the function *EaseUpdateBanner*. Once all the banner information has been fed in then the function *EaseBannerChanged* is called which re-evaluates the alarm banner information for the whole system and indicates if it has changed. If it has changed then the tag in the tag dictionary *ZZ_ABSeqNo* is incremented as a flag to the display system. The script also refreshes the query on the 'two-line' alarm summary if it is necessary (this is usually when redundancy changeovers take place).

4. When the tag *ZZ_ABSeqNo* changes, the buttons on the alarm banner are triggered to refresh because they reference that tag. The alarm buttons use the script function *EaseAlarmButtonPrio* and *EaseAlarmButtonName* to display the alarm button data with the correct text and colour (appropriate to the priority).

Alarm Summary

The alarm summary window in the display is often referred to as the 'two-line' alarm summary. At the default resolution (1024 by 768) it will show two alarms, although it may be scrolled to view others. The object is a standard InTouch distributed alarm object (often referred to as a DAO by Wonderware). The colours are configured to match the standard alarm colours used in the wizards. The only way in which the dynamics of the alarm summary object are affected is by changing the query that is used to update it. The alarm summary will need to query for LINDATA alarms from all the servers (excluding the standby servers in server pairs) and also query for VIEW alarms on the servers that have been configured with VIEW access names.

The data that drives the queries at runtime is the state of all the servers that have been configured and the access name data that has been configured using Operations Server Configurator. The server state is monitored by the script QuickFunction called *ZZServerCheck* (see next section for more on server monitoring). The script functions *EaseServerFailed*, *EaseServerOK* and *EaseServerHealth* are used to feed data on the state of the servers into the script extension DLL.

Once the server data has been updated then the script in the 'On Show' and the 'While Showing' areas of the **ZZ_AlarmBanner** window calls *EaseAlarmQRefreshed* to determine if the alarm summary needs to be re-queried. This script function actually modifies the query for the named alarm summary by directly updating the INI file in the application folder (named *almgrp.ini*). The re-querying of the alarm summary does not happen frequently and is most likely when a redundant server changeover takes place.

Server Alarms and Redundancy

At the top of the alarm banner window are six green boxes containing the "#" symbol (when looked at from WindowMaker). Each box shows the state of a server at runtime, with server 1 being at the right hand side and server 6 at the left. If a server is completely healthy or no server is configured then nothing will be shown.

Each server box shows red if the LINOPC watchdog has failed, orange if the alarm provider watchdog has failed and green if the alarm provider health is less than 100%. The size of the green bar indicates the server percentage health.

Which servers are to be monitored is determined by the access names that have been configured (using Operations Server Configurator) as LINDATA access names. All the servers that are referenced by LINDATA access names without standby servers become servers that will be monitored. The maximum number that the application can show is six although it is normal to have either one or two, and systems with two would normally be dual redundant. This means that where there is a dual redundant pair an access name is required for each of the individual servers and one for the dual redundant pair. The reason for the single route access name being required is that the watchdog data must be read from the server being monitored, and this requires an access name that never gets switched to look anywhere else.

On startup the script extension DLL reads the access name information from the project database and constructs a list of servers to be monitored. This data is then returned into the scripting environment in the script *ZZEaseStartup* using the script extension function *EaseServerAccessName* that will return the names of the access names for all the servers to be monitored. This allows the script to set up a number of tag dictionary tags for monitoring *$LINOPC.OPCWatchdog*, *$FSAlarmProvider.APHealth* and *$FSAlarmProvider.APWatchdog*. These three items for each server are used to determine the state of each server. Before looking at the monitoring script there is a slight complication to the start-up, in the form of variables in the PersistentData table in the project database.

The PersistentData table in the project database is configured using the Operations Server Configurator and the ‘System Variables’ button. The variables that can be set are shown in the following table:

Variable Name	Usage
BackOffSrvrChkInit	Number of seconds after starting View before checking the state of the server(s). To allow for a busy system on start-up.
BackOffSrvrCheckInitOK	If 0 then show server(s) unhealthy before checking, if 1 then show the server(s) as healthy before the first check cycle.
BackOffSrvrChk	If there is a redundancy changeover, how many seconds to wait before checking server health. On occasions there is sufficient loading at changeover to cause an apparent server failure. Setting this can avoid flip-flop changeovers.
ServerCheckRate	How often, in seconds, to check the state of the server(s). By default equal to 10, means that it may take up to 20 seconds to detect server failure.

After the initialisation the script function that checks the state of the server(s), *ZZServerCheck*, is called from the Application 'While Running' script. It is called every second and the script then determines if it is time to check the server watchdogs. The script reads all watchdog and health variables and then passes this information into the script extension DLL using the functions *EaseServerFailed*, *EaseServerOK*, and *EaseServerHealth*. If there has been a redundancy changeover due to the failure or health status change of a server then the calling of the function *EaseChangedAccessName* is used to reconfigure the redundant access names.

The data that is generated by the monitoring of the servers is used to drive the graphics at the top of the alarm banner.

Computer Alarm Views

A feature exists which allows the view of alarms from a View station to be limited to a part of the system. In order to use this feature alarm groups need to be placed into 'Alarm Group Sets' that represent the sections of the system (use Plant Model tool for this). The sets may then be associated with a computer using the Operations Server Configurator. If a computer has no sets associated with it then it will show all alarms from the system. If one or more sets are associated with a client as a 'Computer Alarm View' then only alarms from those sets will be visible.

When sets are associated with a computer the alarm summaries and the alarm banner will be restricted accordingly. Alarms will still be visible on individual faceplates and point pages, no matter where the tag is in the alarm groups. Most of the logic for the restricted view is in the script extension DLL but there are some script functions that may be used to view and control the alarm views. These script functions are not currently used in the shell application but could be used in a custom application. They all start *EaseAlmGrp* and are documented in the *EASEEXT.HLP* file. An example custom application would be to enable the viewing of more alarms by either pressing a button on the screen or on the automatic detection of another system failure.

Using InTouch Alarms

Particularly when using third-party I/O it may be desirable to use InTouch alarms to get alarms into the system. It is very unusual for there to be a separate alarm provider and there is no support for one even if there was. If the user configures InTouch tags to generate alarms then the alarms can be presented through the alarm banner and alarm summaries.

InTouch alarms require minimal configuration. The InTouch alarm group names must be configured in the plant model; i.e. they must be the same names as those configured for Operations Server. There must also be an access name configured for VIEW. This access name is used to indicate the source of VIEW (or InTouch) alarms and data. If the access name is a dual redundant access name then the InTouch alarms will only be sourced from one of the servers at a time. Single route access names will cause the alarms to be always displayed from that server.

Note: The InTouch tags, because the application is used on all stations, will exist on all the stations but the data will not be live and reliable unless they have been configured to use remote I/O server.

The desirable configuration is to have a single server machine generating the data and alarms and then use the access name configured for InTouch data to access the tags. This means that the access name for the I/O server should be a remote access name so that all machines gather the InTouch data. This does mean that the InTouch tags on all the stations will have the I/O data but it will not be used other than for historic trending. The display alarms and data will always come from the server node(s).

The Button Bar

How the button bar features such as display navigation, tool tips and list boxes are implemented.

The button bar at the bottom of the screen is the window **ZZ_ButtonBar**. It is the main user interface for controlling access to displays.

Display Navigation

Once a user has configured their screens, they can use the display navigation editor to link the displays together, starting from the Overview. The Overview display may be different on different computers but is usually the same on all computers in a single system. The Overview will always be shown on start-up.

The main buttons on the button bar all have script that calls the QuickFunction *ZZFunctionButton* with the name of the button. All the main button functions can be found in this single QuickFunction. A lot of the functions, including display navigation, call the QuickFunction *ZZDBlkLoad* (which is short for 'Display Block Load'). The first argument to this QuickFunction is the type of navigation to be carried out or the display block name. The following table shows the possible values:

Value	Display Shown
Overview	The overview mimic (*EaseDBlkOverviewMimic*)
Processcell	The process cell mimic for the current display
Alarm	The alarm mimic for the current display
Trend	The trend mimic for the current display
Left	The navigate left mimic for the current display
Right	The navigate right mimic for the current display
Up	The navigate up mimic for the current display
Down	The navigate down mimic for the current display
Back	The previous display
Mimic	Display the current mimic (display block)

Other script functions used by *ZZDBlkLoad* are *EaseDBlkMimic*, *EaseDBlkMimicName*, *EaseDBlkName* and *EaseDBlkExists*. The *ZZDBlkLoad* function may be used from user script with a display block name and the second parameter set to 1 (this indicates normal operation).

ZZFunctionButton first calls *ZZCanNavigate* to determine if display navigation is currently allowed. This relates to a security feature that allows the View station to be locked when a user logs out.

ZZFunctionButton next calls *ZZCustomFunctionButton*, which may be used to customise the result of pressing any of the standard buttons. If the custom function returns a non-zero result then the standard navigation will be prevented. Other than the standard navigation buttons *ZZFunctionButton* also deals with the standard fixed displays (trend, alarm summary, alarm history, point, operator and trend groups, login and all the menus) that can be produced from the button bar.

List Boxes

List boxes, or menus, can be produced from many of the buttons on the button bar. It is worth mentioning here because the method by which it is populated and then an option is selected is not obvious. In all cases a variable is set to indicate the type of list required and then the window **ZZ_List Box** is shown.

In the ‘On Show’ script of **ZZ_List Box** the type of list box required is tested and the list box is then populated with the options. A user wanting to implement their own list box just needs to add another type to this list and append the script to populate the list box.

When the user selects an item in the list box the selected entry is written to the tag in the tag dictionary named *ZZ_Item_Sel*. This then triggers the Data Change Script for *ZZ_Item_Sel* where the list box type is again tested and the action of the selection is carried out.

This sequence of events always starts with script that is something like:

```
Hide ("ZZ_List Box");
EaseSetString("ListBoxType", "MimicSel");
CALL ZZShowAtRel ("ZZ_List Box",0, -145);
```

This example is for the mimic selection list box.

Tool Tips

As the mouse is passed over the button bar in View tool tips appear in order to show the user what will happen when the mouse is clicked. The starting point of this is the ‘On Show’ script for the **ZZ_ButtonBar** window.

The Wonderware script function *WWContext* is called. It specifies a region of a window that will cause a value in the tag dictionary to be set to a string value when the mouse is over that region. In this case the tag is *ZZ_DBlk\Description*.

When the Data Change Script for *ZZ_DBlk\Description* is triggered the script then works out which region the mouse is over and sets the tool tip text up and then shows the **ZZ_ToolTip** window in the appropriate position. Tool tips could be added to this from any position on the screen so long as they do not clash with other tool tip areas.

Security

Users wanting security set it up using Security Manager but how does it work within the shell application and how can new security features be implemented?

Once a user has configured security and enabled it then there are features within the shell application to support user logins and protection of the appropriate data, with logging of any attempted breaches in security. There are also security features within Security Manager, which have been created for customising the security of the shell application itself.

Application Security Features

The following table lists the security features that can be found in Security Manager (in a user group on a zone containing 'Operations Server PCs'):

Security feature	Usage
Sign	Can user sign for changes? Applies to any change that requires a signature.
Authorise	Can user authorise changes? Applies to any change that requires authorisation.
Print	Is the user allowed to print the screen? (See Key Script - "Ctrl+p")
TagEdit	Is the user allowed to run *TagEdit*? (See ActiveX Event Script "TagEditTag")
Operator Point Display	Does the user see operator point pages instead of the standard point pages? (See "On Show" script for window ZZ_Pointpage).
Export historical Trend	Is the user allowed to export historical trend data? (See disable state on button of ZZ_HistTrendExport window).
Faceplate Configurator	Can the user save faceplates? (See Action script on save button on window ZZ_Faceplate_Config).
Inactivity timeout	After how many minutes is the user logged out if no keyboard/mouse activity? (See Condition Script "$InactivityWarning == 1").
Display access level	Defines access to display block mimics (implemented in script extension DLL and ZZDBlkLoad QuickFunction).
Synchronise Files	Can the user manually synchronise files? (See the Action script on "Synch File" button on ZZ_ABOUT display).
Override Server Redundancy	Can the user force fail servers? (See the Action script on "Fail Comms to this Server" on ZZ_ABOUT display).
Task Switch	Can the user task switch from view? (See Action script on "Task Switch" button on ZZ_login display, and ZZLogin QuickFunction).

Security feature	Usage
Operator Group Global	Can the user save other user's operator group displays? (See Data Change script ZZ_Item_Sel).
Debug	Can the user switch on script trace? (See the buttons on the ZZ_EaseErrors window).
Operator Groups	Can the users save their own operator groups? (See Data Change script ZZ_Item_Sel).
Trend Global	Can the user save other user's operator trend displays? (See Data Change script ZZ_Item_Sel).
AlarmHistMaxItems	Can the user increase the maximum entries on the alarm history display? (See the bottom right hand button on ZZ_AlarmHistoryFilter window).
Trends	Can the user save their own operator trend displays? (See Data Change script ZZ_Item_Sel).
Recipe Download	Can the user download recipes? (See the QuickFunction ZZ_RecipeLoad).
Global Alarm Acknowledge	Can the user acknowledge multiple alarms? (See the "Ack Display" button on the window ZZ_AlarmPage).
Faceplate Modify	Can the user modify faceplates? (See Key Script "Ctrl+Shift+f").
Change Language	Can the user change the display language? (See the "Language" button on the ZZ_ABOUT window).
Offline data writes	*TagEdit* security feature.
IO data writes	*TagEdit* security feature.
System User writes	What level of logging of writes for system users only? LINOPC security feature. Allows the inhibiting of logging of writes which is needed where OPC clients are making continuous writes.
Custom1-Custom5	Five spare features for custom use. These are all Yes or No access.
Custom6-Custom10	Five spare features for custom use. These are all of Confirmation Level type so that required signatures may be requested.

Recipe Security

In the Security Manager, a list of recipes may be defined, each with an associated user name that is the name of the user to carry out the recipe writes. For a logged in user to be able to download a recipe they must have the ‘Recipe Download’ privilege and the recipe must be in the list of recipes that the user's security group has access to. The user carrying out the writes would be expected to be a system user.

The recipe name can be pattern matched, which means that wildcard characters may be entered. The format of the recipe name is <recipe file>/<unit>/<recipe> (e.g. ‘MyRecipes.csv/Boiler1/VeryHot’). When the user attempts to download a recipe the first entry that matches in the list of recipes will allow the download and the system user for that entry will be used to carry out the writes. The simplest entry for recipes that will allow the user to download any recipe is a single asterisk (*).

How We Download Recipes

Because of problems with the InTouch *RecipeDownload* function when using access names to remote I/O rather than the tag dictionary, a method for downloading recipes has been implemented. This is the QuickFunction *ZZRecipeLoad*, which then calls *ZZSystemRecipeLoad*. Using this, we have good control over the user name used for carrying out the writes. There are major problems, however, if there is a requirement to upload recipes because we have not implemented the equivalent of *RecipeSave* and, presumably, *RecipeSave* would have the same problems as *RecipeLoad*.

What if I need to upload my recipe?

If there is a requirement to upload a recipe then using the standard *RecipeSave* is the only easy way to achieve it. This means that the recipe must only use tags in the tag dictionary and then *RecipeLoad* and *RecipeSave* functions may be used. All the tags in the tag dictionary should use the same access name and this access name should NOT be one that is entered in Operations Server Configurator. The reason for this is that the access names that the system is aware of get reconfigured when a user logs in or out. The recipe writes should not come from the logged in user but from a fixed user name. In order to inform the underlying security which user is carrying out the writes the access name topic should be set to
<Node>@<AccessName>@<UserName>
(e.g. ‘ThisPC@RCPDATA@ESDataSrv’). If security is disabled then the access name setup is unimportant.

Tag Security

In Security Manager the level of access a user has to tag security areas can be defined for each security zone. A user will be allowed to write to items on tags in the security areas if their access level is greater than or equal to the restriction placed on the item. In order to view and/or change the access level on a field use the Tag Profile Configurator.

There are three QuickFunctions that are used to write to LIN tag data. They are *ZZSecureTextWrite*, *ZZSystemWriteTagValue* and *ZZWriteTagValue*.

ZZSecureTextWrite is used by many of the standard user interface (e.g. point display) to put up an appropriate styled data entry window based on the type of the data being written to. Once the user enters a value the Data Change script *ZZ_Write_String* performs the security checks, gets any user confirmation and then either carries out the write or shows an error message.

ZZWriteTagValue effectively calls the *ZZ_Write_String* script because the value to be written is passed into the script function as a string. This function can be used on a custom button for a user to write to a LIN field.

ZZSystemWriteTagValue is a version of *ZZWriteTagValue* that is given the name of the user to do the write and just does it. This is for behind the scene script that might be automatically raising an alarm by writing to a digital in an instrument, for instance.

Station Lockdown

When no user is logged in or the currently logged in user does not have the privilege to task switch then the user should not be able to get from View to the operating system. This is handled by some script at the bottom of the Application ‘While Running’ script. Two new script extension functions have been introduced called *SecureView* and *UnsecureView* that are used to increase security beyond the standard InTouch features. When View is secured it will be brought to the front so that if the user has switched to another program and then the automatic logout takes place, the focus will be forced back to View.

Another security feature exists in the ‘Management Data’ for a security zone called *Lockout Level*. This can have one of the values ‘None’, ‘No Navigation’, ‘No Alarms’ and ‘No Displays’. This represents the extent to which View is locked out when nobody is logged in. The level of lockout is progressively increased so that each option includes the previous options.

- ‘No Navigation’ prevents any change of display,
- ‘No Alarms’ prevents any navigation and hides the alarm information in the alarm banner, and
- ‘No Displays’ is the same except the main display is blanked.

For all lockout levels other than ‘None’ and ‘No Display’, the overview display is restored. The script behind this is in the *ZZLogon* QuickFunction and at the bottom of the **ZZ_AlarmBanner** window ‘While Showing’ script. The script extension DLL also applies logic to hide the alarm buttons when the lockout level is ‘No Alarms’ or ‘No Displays’.

Language Support

How to configure and extend the application multi-language support.

Language support has been implemented in the shell application in a relatively simple fashion because there is no easy support of special fonts such as Japanese or Chinese. All the displayed and logged text in the shell application can be replaced with alternative language representations.

There is a single script extension function that is used to convert text called *Translate*. A string is passed into the function and the function returns the translated string. The input string can surround text that should not be translated with %n where n is a 'parameter' number. This is so those context-specific object names such as tags and names are not translated but do appear in the resultant string in the correct position. For example:

```
Translate( "The user %1Joe Barrett%1 has just written %243.2%2 to
%3TIC0101.SL%3" );
```

When the Translate function is called it converts the string into a 'parameterless' form, which in this example would be:

```
"The user %1 has just written %2 to %3"
```

This string is then looked up in the INI file as defined by the currently selected language. The entry in the INI file would look like this:

```
The user %1 has just written %2 to %3=%2 has just been written to
%3 by %1
```

The looked up string is then reconstituted with the parameters to give the final output string:

```
43.2 has just been written to TIC0101.SL by Joe Barrett
```

In this example the language has not been translated but transformed. Some users may want to use translation in this way if they would prefer different wording and do not want directly edit the shell application.

The language file to be used is set automatically on start-up based on the regional settings on the computer. Each language has a three letter code that is used to prefix the file name *LANG.INI*. The name of the file being looked for is logged to the Wonderware logger on start-up. If the file is not found then the three letter prefix is reduced to the first two letters (which is the same for all variants of a language) and this file is looked for, and again logged to the Wonderware logger. If this is not found then there will be no translation.

A template file is provided to make it easier to create the language file that is called *ENLANG.INI*. Most of the text that needs to be translated is in this template file. This should be copied to the appropriate language file name (the list of language prefixes is in the *EASEEXT.HLP* file in the InTouch folder under the topic *SetLanguageID*). The translations may then be entered.

If a menu of languages is required for the application then the file *EPALANG.INI* needs to be edited. This lists the available languages. The **ZZ_ABOUT** window has a button on it for allowing the user to change the language. The script function *SetLanguageID* is used to switch the language on the fly. Pressing the ‘Home’ key on the keyboard will cause all the windows to refresh their text (see the Key Script ‘Home’).

Graphical Wizards

How to use the standard Operations Server Wizards on InTouch pictures.

There are two standard wizard libraries supplied with the system. One is called **Eurotherm Control Modules** and the other is **Eurotherm InHouse**. The Control Modules library contains wizards that are most likely to be used on a customer system and the majority of them apply to specific control modules. The In-house library mostly contains wizards that are used in the standard graphics but it does contain two faceplates that may be used on custom pictures.

The user may use any of the wizards on pictures but may also need to deploy updated wizards from the in-house library as part of a software upgrade to the InTouch application.

Note: These wizards cannot be deployed into a bare InTouch application and are designed to be deployed into a Operations Server application that already has supporting tags and script functions that the wizards make use of.

To be able to use the wizard libraries they must first be 'Installed' into WindowMaker. This can be done from the menu **Special** > **Configure** > **Wizard/ActiveX Installation...** Then select the required library from the list of uninstalled wizards at the bottom of the dialog. Install the library by clicking on the **Install** button.

Control Modules Library

The control modules library contains the following wizards which are designed for use with specific control modules. They all have an icon that represents the shape of the item they apply to and indicate the current state and mode of the control module. They are designed so that they can be connected into a pipe on a mimic and some have alternate formats (e.g. vertical and horizontal) so that they can be arranged sensibly on the picture. The library appears as **Eurotherm CtrlMods** in the WindowMaker Wizard Selection dialog.

Wizard Name(s)	Control Module
Valve	Vlv1In or Vlv2In
Motor, Stirrer, Heater	Mtr3In
DMS	DgManSt
AMS	AnManSt
3WayValve	Vlv3Way
ControlValve	PID or PID_CONN or PID_LINK

The **PushButton** wizard is designed to help the user implement security checks against an operator action. The wizard allows a number of different security check types which allow security to be applied even when there does not appear to be a standard feature to allow it. The wizard will create a button with action scripts and expressions attached to it. If the user wishes to put the same checks on a different graphical object then they should create the button using this wizard and then copy the script onto the appropriate object. The following table lists the type of security checks that the wizard provides.

Security Option	Details
No Security Checks	No security checks applied.
Confirmation Only	The operator will be asked if they are sure that they want to carry out the action.
Display Access Level	The operator's display access level (as configured in Security Manager) is checked against the entered value and then the subsequent level of confirmation will be requested.
Application Feature	The entered Application Feature is checked to see if the operator has access and what level of confirmation is required. There are a number of Application Features in Security Manager (see Chapter 4, "Security.").
Tag Security	The operator is allowed access if they have access to the Tag.Field as entered onto the wizard. The confirmation level is also determined from the security on the tag and field.

The **LIN Data Text** wizard displays a single value from a LIN instrument. It will indicate loss of communications (with @@@) and bad data (with ????). The format, justification and colour (foreground and background) of the data can be defined. A visibility expression may be entered and an option to link the item to a point page display is available. The point page option causes the current point page to be updated when the item is clicked so that if the current point page button (or F4) is clicked then the point page for the item is shown. The item can also be made writeable so that the operator may write to the item by clicking on it (this may also change the current point page at the same time).

Earlier versions of the Control Module Wizards produced pictures of differing sizes. When upgrading these earlier Wizard versions they may lose their proportionality. This can be corrected, however, by editing the *EuroWiz.ini* file. Simply add the following to the *EuroWiz.ini* file.

```
[Control Modules]
FixedSize=1
```

InHouse Wizards

The InHouse Wizards library mostly contains wizards with special functions that are used on the standard displays. There are two faceplate wizards called *Mimic Fascia* and *Small Fascia* that are designed for use on customer pictures. They reside in the same library because they share code with the *Fascia* wizard (strictly for use only where it has already been applied). The two faceplate wizards allow the user to select the tag and point type to be used although if default type is selected the wizard will derive the type from the project database. At runtime they provide buttons for mode changes and value changes (on outputs and setpoints) as well as the ability to call up the point page.

Some of the standard wizards will respond to any alarm colour scheme changes on the system. There is an INI file within the application folder named *EuroWiz.ini* that may be configured to change alarm colours and alarm colour band thresholds. If *EuroWiz.ini* is modified in any way then the wizards will need to be re-deployed (double-click and OK each one) to pick up the changes. This applies to the all the fascia wizards and the *Alarm Button* wizard. The standard displays which are affected are **ZZ_Fascia***, **ZZ_PointPage**, **ZZ_AlarmBanner** and **ZZ_Expanded Alarm Buttons**. When re-deploying wizards it is best to select (double-click) the wizard while holding down the shift key because this prevents it from being accidentally moved.

System Variables

How to make use of the System Variables in Operations Server Configurator to customise the application.

If you run Operations Server Configurator you will see a button on the main page labelled ‘System Variables...’. If this is selected a table of variables is shown. All of these variables allow the InTouch application to be set up and/or customised. The table below lists the variables and how they should be used. Each variable has a type, most of them being ‘ViewConfig’. The type specifies which package the variable is used in, where ‘ViewConfig’ means the InTouch application. In the case of the InTouch application the same variable may be entered with the type as the name of the computer instead of ‘ViewConfig’ to give a computer specific value to a variable rather than a global.

Variable	Usage
AlmHistDateFormat	Date Format for Alarm History ActiveX - not normally used (see LHAlarmObjects.hlp for details, in index as "DateFormat Property").
AlmHistPrefixOnTag	Prefix on tags logged by the alarm provider, not normally used.
AlmHistTypeIsLIN	Use LIN alarm name in the alarm type column when logging or the equivalent InTouch type. Normally ‘**Yes**’. Can be ‘**No**’.
AlmHistValStrType	The contents of the value string for alarm messages. Can be ‘**AlmThenVal**’ (default), ‘**ValThenAlm**’ or ‘**ValueOnly**’.
BackOffSrvrCheckInitOK	*When the shell application starts up does it indicate everything healthy before it knows the state of the servers? Default 0 (=No).*
BackOffSvrCheck	*How many seconds to wait before checking the servers again after a server failure. Can prevent failovers caused by failovers. Default 0.*
BackOffSvrCheckInit	*How many seconds to wait before checking server status on startup. Can prevent false failure detections and allow a server to detect itself as healthy on startup. Default 0.*
EaseHealthDeadband	*Percentage by which one server must become healthier than the other to allow a changeover. Does not apply when one is at zero. Prevents flip-flopping.*
ServerCheckRate	*The frequency in seconds at which servers are checked. Default 10 seconds.*
bAlarmGroupsAsOpGroups	PCView. Operator Group names are restricted to the Alarm Group names. Default 0 (=No).
bTrendAlarmGroups	PCView. Causes the Trend list to display alarm group names so that you load/save trends by alarm group name. Default 0 (=No).
bTrendLink	PCView. Use Canary Trends instead of InTouch/InSQL.
bOpGroupHidesPrevious	*Does the operator group obscure (1) the current picture or overlay it (0) ?*
bPreventOpGroupConfig	*Do not allow any operator group configuration from View (1 = Yes, 0 = No)*

Variable	Usage
DisableFasciaConfirms	Prevent the default confirmation dialog when using buttons on the standard faceplates. Applies when security is disabled. Default 0 (=No).
EnableQuickNavigation	*Enables the 'Quick Navigation' display for mimics. Default 1 (=Yes).*
QuickNavigationXCoord	*The X coordinate for the initial position of the quick navigation window top left in 1024 scale screen. Default 874.*
QuickNavigationYCoord	*The Y coordinate for the initial position of the quick navigation window top left in 768 scale screen. Default 85.*
InactivityTimeoutWarning	Time of inactivity in seconds after which data entry dialogs are closed. Default 60 seconds. Automatic logout times configured in Security Manager.
OverviewFallbackTime	Time of inactivity in seconds before automatically reverting to the overview. Default 0 (=disabled). Superseded by 'Lockout Level' in Security Manager when security is enabled.
InitialChartLen	*The length in minutes of the historic trend display when initially displayed. Default 10 minutes.*
MaxChartLen	*The maximum length in days of the historic trend chart. Prevents potential crashes when excessive lengths are used but may be increased. Default 1 day.*
OnLineTagBrowserHierarchy	An initial point in the browse hierarchy (a string) for the on-line tag browser. Not normally used.
PointPageListMaxItems	*The maximum number of tags to list in the point page (F4) button menu. Default 10.*
Review_Installed	Is the 'Tamperproof Alarm History' display enabled. Default 0 (=No).
Review_Path	The path to the Review database, including the database name. The full path to and name of the Review database used by the 'Tamperproof Alarm History' display.
TrendInSQLSvrDSN	*The DSN of the primary trend InSQL server. Only used for InSQL trend displays. Default '' (not setup).*
TrendInSQLSvrBackupDSN	*The DSN of the backup trend InSQL server. Only used for InSQL trend displays. Default '' (not setup).*
TrendInSQLSvrUser	*The user name to login to the InSQL server to get trend data. Default wwUser.*
TrendInSQLSvrPswd	*The password to login to the InSQL server to get trend data. Default wwUser.*
ZZMHPLogErrors	Indicates if errors that occur when logging messages should be logged as messages themselves (a diagnostic aid). Default 0 (=No).

User Variables

How to make use of the User Variables in Operations Server Configurator to customise the application.

If Operations Server Configurator is run, a button on the main page labelled ‘User Variables...’ is shown. If this is clicked, a table of variables is shown. All of these variables allow the system to be customised and only differ from ‘System Variables...’ in that they do not directly apply to the InTouch application, but are used by other programs. The table below lists the variables and how they should be used. Each variable has a type, where ‘Alarms’ is for the alarm provider, ‘InTouchTrend’ is for the build of InTouch trends and ‘ProjectBuildFlag’ will appear on the ‘Build’ tab of the project properties dialog.

Variable	Usage
CommsPriority	Alarm provider can generate communications alarms when it cannot establish communication with a block/instrument. This can be set to a value between 0 and 15 inclusive. Default 15.
OneAlarmPerBlock	Default 0. Set to 1 and alarm provider will only show the highest priority alarm. Currently shows all alarms on a block at the same time.
DeadbandAbsoluteValue	*Tag builder will use this to set up the deadband on trend tags. Default 1.0. The smaller of the two deadband calculations is used.*
DeadbandFractionOfRange	*Tag builder will use this to set up the deadband on trend tags. Default 0.005 (= 0.5%). The smaller of the two deadband calculations is used.*
DontBuildBlankUICTags	If there is no tag in the comment on the channel ‘Type’ field in a clone file then don’t automatically build a tag in the project database. Default = 1 (Yes).

QuickFunction Reference

How to make use of the Quick Functions in the InTouch application and which ones not to use.

The shell InTouch application has a large list of QuickFunctions, all starting with ‘ZZ’ to push them to the end of the list, which are used by the standard displays. Many of these functions are useful for implementing custom features. Others, however, should never be used and are there to be called by other QuickFunctions. The following table lists the functions that can be used and if a function is not listed in this table then do not attempt to use it. This is a fairly brief reference but most of the functions have useful comments in them about what they do and how to use them.

QuickFunction Name	Usage
ZZApplicationCustomStartup	Normally empty. This function is called at startup and on refresh of the application and is to allow customisation without changing the standard startup script *ZZEaseStartup*. Normally reserved for initialisation of variables, do not expect any I/O data to be available.
ZZCanLaunchProgram	Tests to see if a user has the privilege and the state of the application allows for the launching of another program. Only applies to programs which will put up a user interface on top of the Viewer. Only parameter is the program name which will be used in displayed messages in the event of access being denied.
ZZCanNavigate	The ‘Lockout Level’ security feature is implemented by this function. For custom buttons on pictures that should be inhibited then call this function in the Action script. IF 1 == ZZCanNavigate() THEN
ZZCustomFunctionButton	Normally empty. This function allows custom script to be associated with buttons on the standard button bar. Setting the return value to 1 inhibits the standard script on the button.
ZZCustomLogin	Normally empty. Can be used to take custom action when a user logs in.
ZZDialogStringEntry	Ask the operator to enter a string. Parameters are default value, a prompt, a description and the X and Y top left coordinates. Returns 1 if a string has been entered and the value of the string is *EaseGetString("DialogStringEntry")*.
ZZDisplayAlarmSummary	Display an alarm summary. Only parameter is the group name.
ZZDisplayCustomAlarmSummary	Allows the alarm group buttons to be customised. Takes the button number (1 to 96) as an argument. Could be used for an alternative to the alarm summary display. Returns 1 to prevent the standard display. To get the group name use *EaseAlarmButtonName(ButtonNumber)*.
ZZDisplayMimic	Standard function for calling up pictures using a display block. Use this to call up pictures from custom buttons. Parameter is the display block name. Using this function will maintain all the up, down, previous and next references and implement any display access security.
ZZLINBlockHelp	Would not normally be used for customisation but could be used to implement a custom text-based pop-up help system.

QuickFunction Name	Usage
ZZLogEvent	Use to log an event to the alarm history and, possibly, tamperproof audit trail. Is used by the standard application to log errors and actions but could be used to log custom messages (see function for parameter details).
ZZLogMessageBox	As *ZZLogEvent* but also show a message box.
ZZMHP	See chapter on Message Handling Package.
ZZOpenPDFFile	Launches Acrobat Reader and opens the specified file. Can also jump to a named destination in the file. Has no in-built security so remember to put a security test in if required.
ZZPushButton	This script allows the user to perform either a single write, a recipe download or run a script - whilst a given condition is active.
ZZPushButtonCustomScript	This function is called from *ZZPushButton* and is used to run customised script from a pushbutton dialog.
ZZPushbuttonMultiWrite	Allows multiple writes to be associated with a single user action. *ZZPushButtonSetupTag* should be called once for each value to be written before calling this function.
ZZPushButtonSetupTag	Call this to setup a tag/value pair that will be written by *ZZPushbuttonMultiWrite*.
ZZRampField	This script is used to display a ramp input window for a numeric field. A ramp input is not the default entry window for any field types so it has to be forced if it is required.
ZZRecipeLoad	Used to download an InTouch recipe. Works with remote reference I/O whereas the standard *RecipeLoad* will not.
ZZRecipeSave	Puts security around the standard *RecipeSave*. Will not work for recipes with remote reference I/O.
ZZRestart	Reset the application to initial state. Re-displays all the windows. This is normally associated with the pressing of the Home key.
ZZSecureTextWrite	This function is used to provide a dialog entry for writing a value to a LIN Tag and allows configurability for the tag description and discrete button text. The word Secure is there to imply that standard security will be applied to the write request.
ZZShouldILogThis	Not just for logging. On a multi node system using NAD, scripts written on the configuration station will run on ALL clients. If background scripts are used to perform actions like recipe downloads, unless otherwise catered for, all clients will perform the download. This script determines if this node is the first available server to deal with such an event.
ZZShowAt	Used to show a window at coordinates (centre) but scaled according to the screen resolution (relative to 1024 * 768). Only required where multiple resolutions are used in the system.
ZZShowAtRel	As for *ZZShowAt* but relative to the current mouse position.
ZZShowTopLeftAt	Used to show a window at coordinates (top left) but scaled according to the screen resolution (relative to 1024 * 768). Only required where multiple resolutions are used in the system.
ZZShowTopLeftAtRel	As for *ZZShowTopLeftAt* but relative to the current mouse position.
ZZShowDialogWindow	See chapter on Message Handling Package.
ZZShowMessageBox	Display a message box with optional timeout.

QuickFunction Name	Usage
ZZShowPointPage	Show the point page for the specified tag.
ZZSynchroniseFile	This function is used to synchronise modified files in the local application with the one in the master application directory and the other computers in the system. To be called after the file has changed/been saved on disk.
ZZSystemRecipeLoad	Download a recipe using a specified user. To perform a recipe download as part of the application as opposed to a user requested recipe download.
ZZSystemWriteTagValue	This script should only be used to perform secure writes that are not linked to the operator, e.g. background scripts not initiated by an operator.
ZZTagType	This script is used to parse a tag of the format AccessName:Tag.Field.SubField or any combination. Potentially useful in general script functions. Used by the standard script functions.
ZZUserAccessConfirm	Used to check if a user has access to an ‘Access Right’. Will get the necessary signing/authorisation if required. e.g. IF 1 == ZZUserAccessConfirm("Custom1", "Open Front Door", "Are you sure ?", "") THEN
ZZWriteEaseValue	Allows the user to set a global variable via a dialog.
ZZWriteTagValue	To perform an input-less write to an I/O point from script using the currently logged in user.
ZZXScale	Scales an X coordinate from 1024 resolution to that on the current monitor. Only required where multiple resolutions are used in the system.
ZZYScale	Scales a Y coordinate from 768 resolution to that on the current monitor. Only required where multiple resolutions are used in the system.

Tag Dictionary Reference

What are the tags in the tag dictionary used for and which ones can be used for customisation?

The shell InTouch application has some predefined tags, all starting with '*ZZ*', that are required for the standard displays to operate. They would not normally be used by custom script but some can be used for reading and writing temporary data and others may well be useful for monitoring purposes. The table below shows the tags that could be useful in customising an application. There are many other tags beginning with '*ZZ*' which are all important to the standard application and should not be used in custom script.

Tag Name	Usage
ZZ_Node	Read only. Message containing the name of the local computer.
ZZ_Shutdown	Read only. If this is set to 1 then View will shut down.
ZZ_Write_Mode	This variable determines what the main interface for user entry is going to be. This affects the way dialogs appear. Possible values are 'Mouse' (default), 'Keyboard' and 'TouchScreen'. This can be changed permanently to match the input mode on the system or changed in *ZZApplicationCustomStartup* if it varies from screen to screen.
ZZDiscreteTag	Read/write. Memory discrete for temporary storage of data where functions require a tag rather than a value.
ZZIntegerTag	Read/write. Memory integer for temporary storage of data where functions require a tag rather than a value.
ZZMessageTag	Read/write. Memory message for temporary storage of data where functions require a tag rather than a value.
ZZRealTag	Read/write. Memory real for temporary storage of data where functions require a tag rather than a value.

Customising Point Pages and Faceplates

Point pages and faceplates can be customised to show alternative names for fields, different fields and different layouts.

The standard point display and faceplates support a level of customisation. Some aspects of customisation can now be controlled through the Tag profile Configurator that can be launched by double-clicking the ‘Tag profiles’ icon in the root of the project. The profiles allow, amongst other things, fields to have different displayed names and different enumeration. The displayed names will appear on point pages and faceplates instead of the actual field name. Part of a profile is the **Point Type** that will normally be set to ‘default’.

The point type is used to load the point page configuration file, as follows:

1. Does the file of name <point type>.ini exist in the PtPgCnf folder below the application folder?

Note: If it is ‘default’ then skip next step and go to step 3.

2. If yes, then use that file when loading the point page.

3. If no, then use the <block type>.ini file (which should always be there).

As a rule, the point page INI files that are provided as standard should not be edited. If they are then the changes may need to be re-made after a software upgrade. The INI files will appear something like as follows (extracts from *PID.ini*):

```
[Alarms]
Count=6
[FieldNames]
Field1=Mode
Field2=Alarms
Field3=FallBack
Field4=
.
Field28=SelMode
.
.
[SelMode]
Count=8
SubField1=SelHold
SubField2=SelTrack
SubField3=SelRem
SubField4=EnaRem
SubField5=SelAuto
SubField6=SelMan
SubField7=
SubField8=SelFMan
.
.
[TrendPoints]
TrendPoint1=PV
TrendPoint2=OP
TrendPoint3=SP
```

The [Alarms] section gives the alarm count that would not normally be changed. The list of [FieldNames] defines which field is shown at which position on the point page, where the odd numbers form the first column and the even numbers form the seconds column. Field2 is always ‘Alarms’ and changing it will have no effect. For each field that has sub-fields there is a section that contains the names of the sub-fields and the section has the name of the field (see SelMode in the example from *PID.ini*). The final section of the INI file is [TrendPoints] which allows fields to be allocated to pens on the point display, the first four of which will be displayed while other pens can be allocated at run-time.

If the appearance of a faceplate is to be modified then there is a simple tool that can be accessed from View while running the standard application (and, therefore, one derived from it). Pressing ‘Ctrl’ and ‘Shift’ and ‘f’ keys at the same time can access it. The name of a faceplate can be entered at the top of the display and then an icon can be clicked to load the faceplate. The parameters, which affect the appearance of the display, can then be modified and there is a Refresh button that allows the changes to be tested. The standard faceplates are not normally modified so it is recommended that the name of the new faceplate is entered before clicking on the Save icon. This faceplate can then be associated with a tag from the **TagEdit** display by modifying the name on the ‘FacePlate’ field on the ‘SCADA’ tab. If a standard faceplate is changed/corrected then there is a risk of losing the change on an upgrade. Because all the faceplates are held in the InTouch recipe file *ZZ_Faceplate.csv* in the application folder it will be necessary to copy the contents into the new *ZZ_Faceplate.csv* file when the application is upgraded.

Fixed Fields Reference

What data can be read from the system using fixed fields and what items they apply to.

Fixed fields provide a means of addressing data that is related to instrument data but not directly available from the instrument. The data may be derived from instrument data or may come from another part of the system, very commonly from the project database. Tags are created in the project database and there is much tag related data that can be read and sometimes written using fixed fields. The fixed fields may be used as part of an item reference string from within InTouch or any SuiteLink/DDE/OPC client that is connected to the LIN system (meaning the LINData SuiteLink driver or LINOPC). There are two top-level classes of fixed field, one being 'System fixed fields' that have a fixed syntax, the other being 'Fixed fields' that are applied to tags or blocks.

Note: If there is no project database then only one of the fixed fields will be available, namely '$LINOPC.OPCWatchdog'.

The systems fixed fields are shown in the following table. Some are unlikely to be used in general but are used by the system internals. Writeable fields are shown in bold text, like **this**.

System Fixed Field	Usage
$AlarmGroupNNN.TotActAl	The total active alarms on alarm group NNN (= group name or alarm group ID).
$AlarmGroupNNN.TotUnackAl	The total unacknowledged alarms on alarm group NNN (= group name or alarm group ID).
$AlarmGroupNNN.TotAl	The total (active or unacknowledged) alarms on group NNN (= group name or alarm group ID).
$AlarmGroupNNN.GrpDescription	The description of alarm group NNN (= alarm group ID).
$AlarmGroupNNN.GrpIn	Indicates if any active alarms on group NNN (= group name or alarm group ID). 1 = Yes, 0 = No.
$AlarmGroupNNN.GrpUnAck	Indicates if any unacknowledged alarms on group NNN (= group name or alarm group ID). 1 = Yes, 0 = No.
$AlarmGroupNNN.GrpPriority	The effective priority of group NNN (= group name or alarm group ID) = 16 * GrpUnAck + highest current alarm priority
$AlarmGroupNNN.GrpName	The name of group NNN (= alarm group ID).
$AlarmGroupNNN.TotNewAlarms	A counter of new alarms which increases every time there is a new alarm in the group NNN (where NNN is the alarm group ID or the group name). Once it reaches 32767 it will return to 1.
$EASEConfiguration.Security	*Is security enabled? 1 = Yes, 0 = No.*
$EASEConfiguration.AlarmGroupSeq	*Alarm group sequence numbers from project DB. Changes when alarm groups are reconfigured.*
$EASEConfiguration.TagDataTableSeq	*Tag data sequence numbers from project DB. Changes when tag data is reconfigured.*
$EASEConfiguration.TotalAlarmGroups	*Count of configured alarm groups.*
$EASEConfiguration.SecuritySeq	*Security sequence numbers from project DB.*

System Fixed Field	Usage
$EASEConfiguration.AlarmProvSeq	*Sequence number changes each time data changes that requires an alarm provider rebuild.*
$EASEConfiguration.TagTableAddrSeq	*Sequence number changes when any tag data that affects the addressing of the tag is changed.*
$EASEConfiguration.DisplayBlockSeq	*Sequence number changes when Display Blocks configuration is modified.*
$EASEConfiguration.SQLServer	*The path to the SQL Server for the system.*
$EDBUF_<Name>.UpdateCnt	Exception data buffer <Name>, count of updates. <Name> can be EASEALARM for the alarm provider exception data buffer or LIS_EDBUF for the LINData exception buffer.
$EDBUF_<Name>.UpdateErrCnt	Count of update errors into the exception data buffer.
$EDBUF_<Name>.UpdBufSize	Size (bytes) of the exception data buffer.
$EDBUF_<Name>.UpdBufLow Water	Minimum free space in the exception data buffer since startup.
$FSAlarmProvider.APVersion	*Version string from alarm provider.*
$FSAlarmProvider.Banner, BannerTwo, BannerThree, BannerFour	*Strings containing codes used to generate the alarm banner display.*
$FSAlarmProvider.APWatchdog	*Health counter for the alarm provider. Should change every second.*
$FSAlarmProvider.APErrors	*Count of update errors into the alarm provider.*
$FSAlarmProvider.APHealth	*Percentage of alarm data that is being communicated with. 100 = healthy.*
$FSAlarmProvider.APNewAlarms	*A counter of total alarms which increases every time there is a new alarm. Once it reaches 32767 it will return to 1.*
$FSAlarmProvider.APActAl	*A counter of the total number of active alarms.*
$FSAlarmProvider.APUnAckAl	*A counter of the total number of unacknowledged alarms.*
$FSAlarmProvider.APAl	*A counter of the total number of all alarms, active or unacknowledged.*
$FSDataServer.Version	Version string from LINData.
$GroupData.AddCnt	*Count of AddItem calls into the OPC group.*
$GroupData.AddErrCnt	*Count of errors from AddItem calls into the OPC group.*
$GroupData.AddErr	*Last error returned from AddItem in the OPC group.*
$GroupData.WrtCnt	*Count of writes to items in the OPC group.*
$GroupData.WrtErrCnt	*Count of errors on writes to items in the OPC group.*
$GroupData.WrtErr	*Last error returned from WriteItem in the OPC group.*
$GroupData.UpdErrCnt	*Count of update errors on items in the OPC group.*
$GroupData.UpdErr	*Last error returned by an item update in the OPC group.*
$GroupData.UpdCnt	*Count of updates on items in the OPC group.*
$GroupData.AddErrFlag	*A bitmask used with AddErrStr array. Setting to 0 clears all error strings. Setting individual bits clears the specified entries in the AddErrStr array.*
$GroupData.AddErrStr[0->9]	*Array of 10 strings (to be addressed separately) containing the last ten AddItem error strings.*

System Fixed Field	Usage
$GroupData.WrtErrFlag	*A bitmask used with WrtErrStr array. Setting to 0 clears all error strings. Setting individual bits clears the specified entries in the WrtErrStr array.*
$GroupData.WrtErrStr[0->9]	*Array of 10 strings (to be addressed separately) containing the last ten WriteItem error strings.*
$GroupData.ResetStats	*Set to 1 to reset all the error statistics on the group.*
$GroupData.ItemsInGroup	*Count of items currently in the group (including this one!).*
$GroupData.ItemsNotResolved	*Count of items that have been added but not resolved (not communicating).*
$License.LicState	Integer indicating state of the license. See table in section ‘LicState and LicStateString’, below.
$License.LicStateString	License state - see table in section ‘LicState and LicStateString’, below.
$License.ViewBlocks	Number of blocks that Operations Viewer client(s) have cached.
$License.ViewBlocksMax	Maximum number of blocks that Operations Viewer client(s) can cache.
$License.ViewState	View blocks state: Integer, 0 = Less than max - 5 blocks cached, 1 = more than max - 5 blocks cached, 2 = max blocks cached.
$License.OPCBlocks	Number of blocks that all LINOPC OPC clients have cached.
$License.OPCBlocksMax	Maximum number of blocks that all LINOPC OPC clients can cache.
$License.OPCState	OPC blocks state: Integer, 0 = Less than max - 5 blocks cached, 1 = more than max - 5 blocks cached, 2 = max blocks cached.
$LINOPC.OPCUpdCnt	*Count of updates to all items in LINOPC.*
$LINOPC.OPCUpdErr	*Error code from last failed update.*
$LINOPC.OPCUpdErrCnt	*Count of update errors on all items in LINOPC.*
$LINOPC.OPCAddCnt	*Count of AddItem requests in LINOPC.*
$LINOPC.OPCAddErr	*Error code from last failed AddItem.*
$LINOPC.OPCAddErrCnt	*Count of AddItem errors in LINOPC.*
$LINOPC.OPCAddErrFlag	*A bitmask used with AddErrStr array. Setting to 0 clears all error strings. Setting individual bits clears the specified entries in the AddErrStr array.*
$LINOPC.OPCAddErrStr[0->9]	Array of 10 strings (to be addressed separately) containing the last then AddItem error strings.
$LINOPC.OPCWrtCnt	*Count of writes to all items in LINOPC.*
$LINOPC.OPCWrtErr	*Error code from last failed write.*
$LINOPC.OPCWrtErrCnt	*Count of write errors on all items in LINOPC.*
$LINOPC.OPCWrtErrFlag	*A bitmask used with WrtErrStr array. Setting to 0 clears all error strings. Setting individual bits clears the specified entries in the WrtErrStr array.*
$LINOPC.OPCWrtErrStr[0->9]	*Array of 10 strings (to be addressed separately) containing the last ten WriteItem error strings.*
$LINOPC.OPCResetStats	*Set to 1 to reset all the error statistics on the group.*

System Fixed Field	Usage
$LINOPC.OPCItemsInGroup	*Count of items currently in LINOPC (including this one).*
$LINOPC.OPCItemsNotResolved	*Count of items that have been added to LINOPC but not resolved (not communicating).*
$LINOPC.OPCClients	*Count of OPC clients.*
$LINOPC.OPCGroups	*Count of OPC groups.*
$LINOPC.LINSRV32Version	*Version string for linsrv32.dll.*
$LINOPC.LINOPCVersion	*Version string for LINOPC.exe.*
$LINOPC.OPCWatchdog	*Health counter for the alarm provider. Should change every second (if the data rate is sufficient).*
$LINOPC.Message	*When a string is written to this (usually a special format) it will be logged to the alarm history by the alarm provider.*
$PrinterN.Available	Used with Network Printer Status application. Is Printer N available/communicating? 1 = Yes, 0 = No.
$PrinterN.OK	Is PrinterN OK? 1 = Yes, 0 = No.
$PrinterN.Warning	Is there a possible problem with PrinterN? 1 = Yes, 0 = No.
$PrinterN.Error	Is there an error on PrinterN? 1 = Yes, 0 = No.
$PrinterN.Status	Status text of PrinterN.
$PrinterN.Name	The name of PrinterN.
$TagDLL.TagDLLVersion	*Version string for tag dll.*

LicState and LicStateString

LicState Value	LicStateStr	Description
0	Not Initialised	LicenseInit with package has not been called (LINOPC)
1	OK	License initialised - valid
2	Temporary	License initialised - temporary
3	ACK Required	License initialised - temporary - Acknowledge required
4	Expired - No Acknowledgement	License initialised - temporary - expired no acknowledgement
5	Temporary Date Expired	License initialised - temporary - date expired
6	Invalid	Invalid license found

The other fixed fields all relate to tags in the project database. The type of item to which fixed fields apply is variable. In the following table the variable parts are enclosed in angle brackets which define the type of item and syntax required.

Fixed Field	Usage
<Item>.AccessName	The access name to use to read an item where <Item> is a tag for a field or sub-field.
<Item>.LINFieldType	Integer representing the LIN field type of an item where <Item> is a tag for a field or sub-field.
<Item>.IsWritable	Is an item writeable? 1 = Yes, 0 = No. Where <Item> is a tag for a field or sub-field.
<BlockTag>.<Alarm>.AlmIn	Is the tag's alarm in alarm? 1 = Yes, 0 = No. Alarm can be alarm name or Alarms[N] where N is alarm number 1 to maximum alarms or 0 (for current alarm).
<BlockTag>.<Alarm>.UnAck	Is the tag's alarm unacknowledged? 1 = Yes, 0 = No.
<BlockTag>.<Alarm>.AlPriority	Priority (0 to 15) of the tag's alarm.
<BlockTag>.<Alarm>.FieldName	The displayed name for the tag's alarm.
<BlockTag>.CurrAlNo	The current alarm number in a LIN block.
<BlockTag>.CurrAl	The current alarm byte in a LIN block.
<BlockTag>.HistAlNo	The historic alarm number in a LIN block.
<BlockTag>.HistAl	The historic alarm byte in a LIN block.
<BlockTag>.BlockName	The name of the block (not necessarily the same as the tag).
<BlockTag>.Template	The block's template type
<BlockTag>.CommAl	The communication alarm byte in a LIN block.
<BlockTag>.CurrentRate	The current poll rate (in milliseconds) for the block.
<BlockTag>.NodeName	The name of the LIN node that the block is in.
<BlockTag>.NetworkName	The name of the network that the block's node is on.
<BlockTag>.MasterDBF	The source DBF (or layer) that contains the block.
<BlockTag>.DefaultDBF	The output DBF (the one to be run in the instrument) that contains the block.
<BlockTag>.TaskNo	The task number for the block.
<BlockTag>.<Field>.Enums or <FieldTag>.Enums	The field enumerations in the form <Count>,<Enum1>,<Enum2>,...
<BlockTag>.<Field>.FieldName or <BlockTag>.<Field>.<SubField>. FieldName or <FieldTag>.<FieldName>	The displayed name for a field may be different to the field name.
<BlockTag>.HiRange or <BlockTag>.<Field>.HiRange or <FieldTag>.HiRange	The high range of the item may be from the LIN block, a tag profile or the tag range.
<BlockTag>.LoRange or <BlockTag>.<Field>.LoRange or <FieldTag>.LoRange	The low range of the item may be from the LIN block, a tag profile or the tag range.
<BlockTag>.<Field>.% or <FieldTag>.%	The value of the field expressed as a percentage of the range. The range of the item may be from the LIN block, a tag profile or the tag range.

Fixed Field	Usage
<Tag>.<EventPriority>	The priority associated with any non-alarm events for the alarm history.
<Tag>.<AlMask>	Are alarms masked for this tag? 1 = Yes, 0 = No.
<Tag>.TagName	Allows the tagname to be edited.
<Tag>.BriefDesc	The brief description for the tag. Appears at the bottom of faceplates.
<Tag>.FullDesc	The full description for the tag. Used in logged messages.
<Tag>.Units	The units for the tag. Appears on faceplates.
<Tag>.UserStr1,UserStr2, UserStr3	User-definable strings associated with a tag.
<Tag>.TagDataRate	The normal rate for reading displayed data (milliseconds).
<Tag>.AlarmRate	The rate for reading alarm data (milliseconds).
<Tag>.HomeDisp	The name of the display block that defines the mimic designated as the home display for a tag (can be displayed from the point page).
<Tag>.IconDisp	The name of the display block which defines the mimic designated as the icon display for a tag (not currently used, i.e. spare).
<Tag>.FasDisp	The name of the faceplate display, used to override the default from the template name.
<Tag>.PtDisp	The name of the point display. Point displays can be defined in the tag profile configurator. Only pre-defined point types may be entered here.
<Tag>.PhyAddr	The fully qualified physical address (FQA) for the tagged item (block or field).
<Tag>.SecurityArea	The security area the tag is assigned to. Must already exist in the list of security areas.
<Tag>.HighDesc	The high descriptor for digital tags. Appears on faceplates and may be used as enumeration of the primary variable TRUE state.
<Tag>.LowDesc	The high descriptor for digital tags. Appears on faceplates and may be used as enumeration of the primary variable FALSE state.
<Tag>.LoopNumber	The loop number for the tag.
<Tag>.PandID	The P & I diagram reference for the tag.
<Tag>.Deadband	The deadband for the tag.
<Tag>.Trended	Is the tag InTouch trended? 1 = Yes, 0 = No.
<Tag>.InSQL	Is the tag InSQL trended? 1 = Yes, 0 = No.
<Tag>.LinkTrendRanges	1 = Yes, 0 = No.
<Tag>.TagType	Integer representing the ‘component type’. 9 = block, 10 = field, 11 = subfield.
<Tag>.PlantUnit	The name of the plant unit the tag is assigned to.
<Tag>.PlantUnitName	The name of the plant unit the tag is assigned to. Same as ‘PlantUnit’.
<Tag>.PlantUnitDesc	The description of the plant unit the tag is assigned to.

Fixed Field	Usage
<Tag>.ProcessCell	The name of the process cell the tag is assigned to.
<Tag>.ProcessCellName	The name of the process cell the tag is assigned to. Same as ‘ProcessCell’
<Tag>.ProcessCellDesc	The description of the process cell the tag is assigned to.
<Tag>.AlarmGroups	A comma separated list of alarm groups the tag is assigned to.
<Tag>.GenericType	The generic type of the tag. One of AIN, AnalogField, ANY, AOUT, CM, DigitalField, DIN, DOUT, GeneralField.
<Tag>.MachTags	A comma separated list of machine tags (from the LIN database) associated with the same item.
<Tag>.TagAddrSeq	A sequence number that changes every time the tag physical address is affected.
<Tag>.TagDataSeq	A sequence number that changes every time tag related data is changed.
<FieldTag>.BlockTag	The tag for the block which contains this field or sub-field.
<FieldTag>.FTagFld	The field name that is tagged.
<FieldTag>.FTagSubFld	The sub-field name that is tagged, if any.
<FieldTag>.FTagWrite	Is the tagged field writeable? 1 = Yes, 0 = No.
<FieldTag>.FTagType	The default rendered type for this field or sub-field. 1 = boolean, 7 = integer, 8 = float, 10 = string.

Scan for local contents

Eurotherm Ltd
Faraday Close
Durrington
Worthing
West Sussex
BN13 3PL
Phone: +44 (0) 1903 268500
www.eurotherm.co.uk

HA028882 Issue 5 (CN35935)