



BluePlant Series Utilization Manual

Rev. A 12/2013

Doc. Code: MU224600



altus

www.altus.com.br



No part of this document may be copied or reproduced in any form without the prior written consent of Altus Sistemas de Automação S.A. who reserves the right to carry out alterations without prior advice.

According to current legislation in Brazil, the Consumer Defense Code, we are giving the following information to clients who use our products, regarding personal safety and premises.

The industrial automation equipment, manufactured by Altus, is strong and reliable due to the stringent quality control it is subjected to. However, any electronic industrial control equipment (programmable controllers, numerical commands, etc.) can damage machines or processes controlled by them when there are defective components and/or when a programming or installation error occurs. This can even put human lives at risk.

The user should consider the possible consequences of the defects and should provide additional external installations for safety reasons. This concern is higher when in initial commissioning and testing.

The equipment manufactured by Altus does not directly expose the environment to hazards, since they do not issue any kind of pollutant during their use. However, concerning the disposal of equipment, it is important to point out that built-in electronics may contain materials which are harmful to nature when improperly discarded. Therefore, it is recommended that whenever discarding this type of product, it should be forwarded to recycling plants, which guarantee proper waste management.

It is essential to read and understand the product documentation, such as manuals and technical characteristics before its installation or use.

The examples and figures presented in this document are solely for illustrative purposes. Due to possible upgrades and improvements that the products may present, Altus assumes no responsibility for the use of these examples and figures in real applications. They should only be used to assist user trainings and improve experience with the products and their features.

Altus warrants its equipment as described in General Conditions of Supply, attached to the commercial proposals.

Altus guarantees that their equipment works in accordance with the clear instructions contained in their manuals and/or technical characteristics, not guaranteeing the success of any particular type of application of the equipment.

Altus does not acknowledge any other guarantee, directly or implied, mainly when end customers are dealing with third-party suppliers.

The requests for additional information about the supply, equipment features and/or any other Altus services must be made in writing form. Altus is not responsible for supplying information about its equipment without formal request.

COPYRIGHTS

Nexto, Ponto Series, MasterTool, Grano , WebPLC and BluePlant are registered trademarks of Altus Sistemas de Automação S.A.

Windows, Windows NT and *Windows Vista* are registered trademarks of Microsoft Corporation.

Summary

1. BLUEPLANT TECHNICAL SUPPORT	1
Introduction	1
Documents Related to this Manual	1
Visual Inspection	2
Technical Support	2
Warning Messages Used in this Manual	2
2. BLUEPLANT SERIES	3
BluePlant Characteristics	3
BluePlant Models	3
Available Models	3
Related Products	4
Solution for System Integrators	4
Accessories	5
3. BLUEPLANT TECHNICAL DESCRIPTION	6
Models General Features	6
Common General Features	7
General Features	8
Intrinsic Safety Software	8
Superior Graphical Engine	8
Maintenance Capabilities, Testing and Enhanced Diagnostics	8
Built-in Servers and .NET Extensions	8
Innovative Product Features	9
Real Time Database (Tags)	9
.NET Languages and Scripts	9
Alarms and Security	10
Trend and Historian.....	10
Devices and Networks.....	10
Datasets	11
Reports	12
Client Displays	12
Runtime Objects.....	12
Module Isolation	13
Runtime and Diagnostics Tools	13
Project Test and Deployment	13
Communication Drivers	14
4. BLUEPLANT	15
Installation and Operation Minimum Requirements	15
Installation	16
Licenses and Hardkey	18
Starting BluePlant	19
Project Management.....	19
Getting Started with BluePlant	21
Selecting the Demo Project	21

Creating a New Project	21
User Interface	21
Creating Tags in the Project	22
Associating Tags to a Communication Protocol	24
Including Objects in the Main Screen	26
Running a Project	29
5. BLUEPLANT MAIN MENU	34
Application Editing	34
Application Diagramming	35
Application Execution	35
Application Information	36
6. BLUEPLANT COMPONENTS	38
Edit Menu	38
Editing Tags	38
Editing Security Settings	47
Editing Devices	50
Editing Alarms	58
Editing Databases	61
Editing Scripts	73
Editing Displays	77
Editing Reports	83
Draw Menu	86
Selection Tools	88
Geometric Objects	89
Display Components	89
Input and Output Text Tools	89
Symbols Library	91
Advanced Controls	91
Alarm	91
Trend	93
Data Grid	95
Horizontal Toolbar	97
Dynamic Configuration	98
CodeBehind	105
Symbols	106
Info Menu	109
Project	109
Redundancy	111
Track	113
Notes	115
Application Tools	116
Tstartup	116
PropertyWatch	117
TraceWindow	118
ModuleInformation	119
Runtime Objects	119
Namespace Tag	121
Namespace Security	130
Namespace Alarm	133
Namespace Device	138
Namespace Dataset	140
Namespace Script	151

Namespace Display	153
Namespace Report	154
Namespace Info.....	156
Namespace Server.....	161
Namespace Client.....	163
Advanced Settings	169
Command Lines	169
Running BluePlant as a Windows Service	170
Remote Clients	170
Installing Web Server in IIS.....	171
7. SCENARIOS OF TYPICAL SYSTEMS.....	178
Systems Configurations	178
Standalone System	178
Distributed Inputs/Outputs System	178
Client and Server System	179
Redundant Server System	180
Control System.....	181
Distributed and Redundant Distributed Control System.....	182
Load Sharing System	183
8. GLOSSARY	184

1. BluePlant Technical Support

Introduction

Altus BluePlant is the ultimate solution for supervisory, control and data acquisition systems. Altus reputation stands for excellence in delivering automation systems and process control products, like Programmable Logic Controllers (PLCs) and Remote Terminal Units (RTUs), offering superior performance, technology in the state of the art and features like redundancy, online change, hot-swapping, and high connectivity among other high-end features. The extensive experience in industrial automation systems was the development base for this SCADA/HMI software product. The expertise and portfolio of these many different automation products grant to Altus a key position in delivering complete automation solutions.

Altus BluePlant fulfills requirements like high-performance, enhanced connectivity capabilities, an extremely rich and powerful graphical user interface and superior real-time data acquisition engines. The selection of drivers embedded in BluePlant, the capability of distributed engineering and redundancy, OPC support, brings a new and unique user experience. Created on Microsoft's Windows Presentation Foundation (WPF), BluePlant technology allows users to get the best of current graphic cards, resulting in applications with outstanding performance.

Altus BluePlant also brings the standard functionalities found in this product range, such as interaction with database servers (SQL, PI, Oracle, Sybase, Informix and others), custom network buses, a user-friendly alarm server and event notification module, a logging and reporting component, an advanced historian server, business logic management capabilities as well as support for local and remote clients, either running in computers, web, tablets and smartphones.



Figure 1-1. Altus BluePlant

Documents Related to this Manual

For additional information about BluePlant Series, you can examine other specific documents (user manual and technical characteristics) in addition to this one. These documents are available in its last review on www.altus.com.br.

Each product has a document (Technical Characteristics - CE) which describes its characteristics. Additionally, the product may have User Manuals (manual's codes are mentioned at CEs from the respective modules).

It is advisable to consult the following document as a source of additional information:

- BluePlant Technical Characteristics – CE124000

Visual Inspection

Prior to installation, we recommend performing a careful visual inspection of equipment, by checking if there is damage caused by shipping. Make sure all components of your order are in perfect condition. In case of defects, inform the transportation company and the nearest Altus representative or distributor.

It is important to record the serial number of each item received, as well as software revisions, if any. This information will be necessary if you need to contact Altus Technical Support.

Technical Support

To contact Altus Technical Support in São Leopoldo, RS, call +55 51 3589-9500. To find the existent centers of Altus Technical Support in other locations, see our website (www.altus.com.br) or send an email to altus@altus.com.br.

If the equipment is already installed, please have the following information when requesting assistance:

- The installed system configuration (operational system, data base, used architecture and hardware requirements compliance - computer)
- The BluePlant model in use and additional accessories
- The project content (used modules) from BluePlant programmer

Warning Messages Used in this Manual

In this manual, warning messages will present the following formats and meanings:

DANGER:
Relates potential causes, that if not noted, generate damages to physical integrity and health, property, environment and production loss.

CAUTION:
Relates configuration details, application and installation that shall be followed to avoid conditions that could lead to system fail, and its related consequences.

ATTENTION:
Indicate important details of configuration, application or installation to obtain the maximum operation performance from the system.

2. BluePlant Series

BluePlant Characteristics

BluePlant runs natively on 64 bits machines with .NET Framework 4. There are different product models to allow the choice of the best solution according to the users' needs. BluePlant can meet from large-scale solutions to embedded applications. BluePlant models are compatible with 32 bit computers legacy. The client displays can run on web browsers on Windows computers and on Windows based mobile devices as well.

	Characteristics
BluePlant Enterprise	Designed for Plant Process Management, Business Intelligence (BI), realtime panels, SCADA, advanced HMI, process control and optimization. Allows clients and distributed data acquisition. The application size starts in 150 data points
BluePlant Lite	Designed for panels, industrial computers, embedded devices, and standalone systems. Mainly applied on machine interface and small centralized projects. Application sizes range from 150 up to 1500 communication points
BluePlant Express	This version of BluePlant is for evaluation only. It is not for field installation, and is limited on 75 tags and one hour of runtime execution
BluePlant Student	Designed for schools and universities. Application size range from 150 up to 1500 communication points and one hour of runtime execution

Table 2-1. BluePlant Models

To meet the needs of system integrators were developed solutions that allow the project development as on BluePlant Lite or BluePlant Enterprise. These solutions for system integrators executes only one hour the Runtime that can be performed the project tests and after this period is necessary to restart the Runtime.

All solutions for system integrators communicate fully with all programmable controllers through the main communication drivers integrated in the products.

BluePlant Models

BluePlant is divided in family/models according to the features and requirements of each application. Thus it is possible to use the BluePlant model suitable for the application size to optimize system performance. The total amount of available tags must be taken into consideration and this amount is 10 times higher than the communication point's quantity. The communication points are included in the total amount. The available models are presented on Table 2-1 while on **Erro! Fonte de referência não encontrada.** is the communication point's amount for each BluePlant model.

Available Models

The BluePlant is divided in models according with functionalities and requirements of each application. The models available are: BluePlant Express, BluePlant Student, BluePlant Lite or BluePlant Enterprise. The next option is the application size, which is divided in many models to maximize the system performance. Also should be considered the total tags quantity available, being this quantity 10 times the communication points quantity, where the communication points are included in the total quantity.

BluePlant Enterprise (Communication points)	BluePlant Lite (Communication points)	BluePlant Express (Communication points)	BluePlant Student (Communication points)
-	-	75	-
150	150	-	150
300	300	-	300
500	500	-	500
1,500	1,500	-	1,500
2,500	-	-	-
5,000	-	-	-
15,000	-	-	-
25,000	-	-	-
50,000	-	-	-
100,000	-	-	-
Ultimate	-	-	-

Table 2-2. Amount Communication Points per BluePlant Model

Related Products

The following table has the codes that must be used to purchase the product:

Communication Points	BluePlant Enterprise		BluePlant Lite		BluePlant Express	BluePlant Student
	Engineering Runtime	Runtime	Engineering Runtime	Runtime	Engineering Runtime	Engineering Runtime
75	-	-	-	-	BP6400	-
150	BP1203	BP1103	BP2203	BP2103	-	-
300	BP1205	BP1105	BP2205	BP2105	-	-
500	BP1207	BP1107	BP2207	BP2107	-	-
1,500	BP1209	BP1109	BP2209	BP2109		BP4400
2,500	BP1211	BP1111	-	-	-	-
5,000	BP1213	BP1113	-	-	-	-
15,000	BP1215	BP1115	-	-	-	-
25,000	BP1217	BP1117	-	-	-	-
50,000	BP1219	BP1119	-	-	-	-
100,000	BP1221	BP1121	-	-	-	-
Ultimate	BP1299	BP1199	-	-	-	-

Table 2-3. Related Products per BluePlant Model

Notes:

Ultimate: Applications with more than 100,000 communication points should use this license.

BluePlant Enterprise: This model comes with one BluePlant full client (remote rich client) other accessories must be requested separately.

BluePlant Lite: This model comes with one BluePlant full client (remote rich client) and one Internet Explorer full web client.

Solution for System Integrators

The following table shows the product codes that meet the needs of system integrators.

Code	Description
BP5001	Solution for system integrators with temporary license – 1 year
BP5003	Solution for system integrators with temporary license – 3 years
BP5010	Solution for system integrators – BluePlant Lite
BP5020	Solution for system integrators – BluePlant Lite/Enterprise

Table 2-4. Solutions for System Integrators

Notes:

BP5001, BP5003: Solution that allows to develop projects both BluePlant Lite as BluePlant Enterprise, with temporary license valid for 1 (one) year (BP5001) and 3 (three) years (BP5003) from the date of recording.

BP5010: Solution that allows developing projects only on BluePlant Lite, unlimited license expiration time.

BP5020: Solution that allows developing projects for BluePlant Lite and BluePlant Enterprise, unlimited license expiration time.

Accessories

The following table has the accessories codes that can be used to purchased. These accessories only are available for BluePlant Enterprise model.

Code	Description
BP9501	Internet Explorer viewer web Client
BP9601	Rich full client
BP9701	Internet Explorer full web Client

Table 2-5. Accessories for BluePlant Enterprise

3. BluePlant Technical Description

This chapter presents the technical characteristics of the BluePlant models, addressing the integral parts of the system, its architecture and general characteristics.

Models General Features

	BluePlant Lite	BluePlant Student	BluePlant Express	BluePlant Enterprise
Limited runtime execution	No	Yes	Yes	No
OPC DA server	Yes	Yes	No	Yes
C# language	No	Yes	Yes	Yes
Multi-threading scripts execution	No	No	No	Yes
Array tags (multiple dimensions)	No	No	No	Yes
User types (multiple levels)	No	No	No	Yes
Historian table configuration	No	No	No	Yes
Extension SDK and Toolkits	No	No	No	Yes
Server for iPad and iPhone clients	No	No	No	Yes
Concurrent remote Rich clients	No	Yes	Yes	Yes
Concurrent remote web clients (Full and/or Viewer)	Yes, only 1 full web client	Yes	Yes	Yes
Device node redundancy	No	Yes	Yes	Yes
Server redundancy	No	No	No	Yes
Graphical objects report	No	Yes	Yes	Yes
Extended alarms conditions	No	Yes	Yes	Yes
Project version control	No	Yes	Yes	Yes
Track changes by objects	No	Yes	Yes	Yes
Automatic historian compressing	No	No	No	Yes
WPF controls access	No	Yes	Yes	Yes
Hot Start	No	No	No	Yes
Test mode	No	Yes	Yes	Yes

Notes:

Limited runtime execution: Runtime execution limited in 1 hour. The runtime can be restarted.

C# language: It is possible create scripts using C# language.

Multi-threading scripts execution: This functionality allows create scripts and generate different threads for each script created. When this feature is enabled, then the threads execution is concurrent. When this feature is not enabled the threads execution is sequential.

User types (multiple levels): It is allowed create new data types and it is possible use up to 4 levels chained of data types.

Extension SDK and Integration toolkits: It is possible create proprietary libraries (dlls), with specific functions and use the projects.

Concurrent remote rich clients: BluePlant should be installed on a remote machine and the licenses must be present on a server machine. The number of concurrent remote rich clients depends on the number of purchased licenses.

Concurrent remote web clients (Full and/or Viewer): The number of concurrent remote web clients, both viewer and full are dependent of the number of purchased licenses except for BluePlant Lite that allows only one remote client full and none remote client viewer.

Server redundancy: To use this functionality two servers and two hardkeys with their respective licenses are necessary. Each server must have its own hardkey and then it is possible to configure the servers as a redundant pair.

Track changes by objects: This feature allows track the changes done in the displays, tags, scripts, modules and other modified objects in a project.

Hot start: It is possible to modify the application and overload it without stopping the system.

Common General Features

	BluePlant Lite, BluePlant Express, BluePlant Student and BluePlant Enterprise
Simultaneous protocols	All BluePlant models have at least 4 channels
OPC client	Yes
Open multiple projects	Yes
VisualBasic .NET language	Yes
Scripts for math expressions evaluation	Yes
Integration with external SQL database	Yes
Historian and logging	Yes
Alarm and security	Yes
WPF graphical editor	Yes
Engineering and debugging tools	Yes
Scripts to create .NET classes and tasks	Yes
Report editor	Yes
Track changes by tables	Yes
Localization	Yes

Notes:

Simultaneous Protocols: Simultaneous communication drivers running during runtime execution.

Track changes by tables: Tracking changes by tables informs what was modified, removed or inserted, but not informs where the changes were done.

Localization: This feature translates display texts and alarms in Runtime.

General Features

Intrinsic Safety Software

In order to ensure security and reliability, one of the key foundations for the development of the BluePlant platform is that there is no use of C or C++ code, which entirely eliminates the risk of a “clobbered or compromised” pointer and/or memory exceptions. Each process and execution thread, whether internal or created to run in the BluePlant framework, runs in its own allocated and “protected” space with built-in exception control, memory isolation, multithread control and real-time synchronization. The legacy software development methodology employed with VBA, VBScript and proprietary math and logic is replaced by compiled .NET languages. In the former methodology the potential problems could only be detected during the runtime execution, while in the latter, there is a complete script validation before runtime deployment with built-in protection, which adds both performance and enhanced operational stability and security.

Superior Graphical Engine

The graphics in BluePlant are pure Windows Presentation Foundation (WPF) with full support for XAML. This allows for seamless integration with geospatial maps and 3D models. The 3D models can be directly presented, as well as linked to dynamic data with associated responses and behaviors based on real-time values and events. A powerful WPF Graphical Editor is included with BluePlant. The web clients rely on XBAP (browser-based applications and Silverlight) so there is no requirement for the installation of any external Active-X component. In addition to the support that allows web pages to be presented on smartphones, BluePlant offers support for graphics and reports on Apple’s iPads and iPhones.

Maintenance Capabilities, Testing and Enhanced Diagnostics

The system provides seamless switching of project versions, allowing test mode applications to run side-by-side in the same server with the production mode applications, for validation and quality assurance, with built-in analysis of CPU usage and communication statistics of the runtime modules and networks provided. Built-in hot-standby deployment for redundancy, alternate operating locations, and disaster recovery are also included.

Built-in Servers and .NET Extensions

Besides the built-in modules for realtime database, external SQL and ERP connections, alarm and events server, historian server and reporting, BluePlant allows complete access to the whole Microsoft .NET Framework, for advanced customization and extensibility, without the addition of any kind of third part application or external tool.

In addition to the previously mentioned features, BluePlant was architected from a "green field" and was created entirely without the employ of any legacy code. It is a 100% managed code application which allows users to leverage and take advantage of the full potential of the Microsoft .NET Framework today as well as tomorrow. BluePlant has a configuration interface entirely created on Microsoft’s Windows Presentation Foundation Graphics (WPF) and fully supports software as a service (SaaS) deployment combined with typical on premise (local) installations, allowing BluePlant users to access and to collaborate on development and projects anywhere in the world with just an internet browser.

Innovative Product Features

Real Time Database (Tags)

BluePlant supports the following built-in real-time data point types: Digital (Boolean), Analog (Integer, Double and Decimals), Text Message, Counter, Timer and Date Time variables. Further, can be defined types with multiple levels of inheritance, reference tags and tri-dimensional arrays.

The real-time database guarantees, without requiring any additional programming, the synchronization of data among multiple server processes and multiple client stations. A large set of built-in properties, such as data quality, time-stamp, lock state and locked value, simplifies and empowers the creation of the applications.

Real-time data base (Tags)	
Extensive support to tag types	Digital, Analog Int, Analog Double, Analog Decimal, Text, Timer, Counter, Date/Time (date and time variables)
Built-in data table tag type	Access dataset query results on standard .NET data table object
Reference tags	Use reference tags to switch the tag link on runtime execution
Tag arrays	Define tag arrays (one to three dimensions depending on product version)
User-defined structs and types	Define of own extension types to the real-time database
Tag properties	Extensive set of tag properties accessible on both configuration and runtime

Table 3-1. Real Time Data Base (Tags)

.NET Languages and Scripts

BluePlant is a SCADA system that fully supports the Microsoft .NET languages in complete integration within the Microsoft .NET Framework. The project scripts and logics can be written in C# or VB.NET, and a built-in language converter allows to switch dynamically the created code between the languages.

Inside the BluePlant framework, user can compile, cross-reference the objects and access directly (using the Intellisense) the .NET classes and project objects, including alarms, reports and communication nodes.

.NET languages provide a more powerful environment when compared to VBA or VBScript that are interpreted languages, not compiled. These technologies leave many programming errors that are only found when running a VBA or VBScript project in real-time, resulting many times in undesired results and consequences. The managed environment of the Microsoft .NET Framework gives one the support for finding and recovering from exceptions, thus providing a highly reliable environment for the runtime system and applications.

.NET Languages and Scripts	
Create VB.NET functions and procedures	Access BluePlant objects directly from the code
Run scripts on events and scheduling	Easy connection to tags and process events using scripts
Support for class libraries	Create users' own classes accessible to other scripts and displays
Built-in .NET editor with Intellisense	Intellisense helps the user to select tag names and business objects properties
Support for exceptions and trace messages	The entire .NET Framework, external components and services are easily integrated

Table 3-2. .NET Languages and Scripts

Alarms and Security

These features allow to define multiple alarm levels for each point or tag and set a whole range of behaviors, such as logging, acknowledgement, displaying, etc. (the behaviors are pre-packaged in order to simplify the configuration). The security system defines the access levels to each display object. Both alarm and security conditions are automatically replicated on redundant applications.

Alarms and Security	
Multiple alarm conditions	Hi, HiHi, Lo, LoLo, rate of change and deviation
High resolution	Millisecond range timestamp (when available) using the remote I/O time, not the computer time
Built-in visualization object	Online and historical graphical object, where it runs locally or in the Web
Alarm group and item objects	Access alarms properties directly, e.g., "total alarms active", with no requirements to create application tags

Table 3-3. Alarms and Security

Trend and Historian

Blue Plant allows the creation of historian files on external databases, like Microsoft SQL Server or Oracle, or even the use of the built-in Altus SQL Database. BluePlant enables to save the data based on data change or group triggers and has an exclusive time-span option that prevents logging data with a timestamp smaller than a defined preset, allowing the creation of more compact databases. The access to OSIsoft's PI Server is also a possibility.

The time stamping feature may use the timestamp provided directly from the remote I/O, instead of from the computer, providing increased event accuracy. The organization of the samples allows users to include or remove tags for logging without losing compatibility with past data. A complete trend chart object is also supplied for the visualization of both online and historical data.

Trend and Historian	
Connection with ADO databases	Historian information can be saved in any external database with ADO.NET support
Built-in SQL database	When not defined as an external database, logging is on the internal built-in SQL database
High resolution	Millisecond range timestamp (when available) using the remote I/O time, not the computer time
Trigger by tag or by group	Allows the saving of a record according to tag change, or based on process events
Historian dead band by Tag	Allows the definition of the minimum tag variation to trigger recording
Minimum time span configuration	Allows the definition of a minimum interval for recording, enabling the creation of more compact databases
Database tables with multiple tags	Allows the creation of a group of tags, and stores the tags on the same data table to speed up recording and loading
Built-in trend visualization object	Online and historical graphical object, where it runs locally or in the Web

Table 3-4. Trend and Historian

Devices and Networks

BluePlant is supplied with an OPC DA driver to get information from remote devices. Besides OPC, BluePlant also supports customized communication drivers to directly access PLCs, remote I/O systems, fieldbus standards, single and multi-loops, scanners, barcode readers, RFID devices, and digital displays.

The device configuration tool can import databases from OPC servers, CSV or text files. If the device is compatible, it automatically implements multi-threading on TCP/IP networks or multi-serial scenarios. The addressing syntax follows the naming convention of the remote device, making configuration and maintenance much easier. A complete set of performance and diagnostics tools is also included.

Devices and Networks	
Import data point configurations	Copies and pastes from Microsoft Excel, imports CSV or OPC server databases
Communication runs on isolated process	Full protection for runtime environment and enhanced performance on multi-core CPUs
Easy configuration for multiple channels	Automatically creates multiple threads on multi-serial or TCP/IP protocols
Abstract naming for nodes and stations	Provides an easy way to rename and maintain IP address and I/O network configuration
Dynamic creation of optimized blocks	Simple selection of Read and Write points based on the protocol, and creation of optimized blocks
Points configuration follows device syntax	When addressing device points, the same addressing is used as PLC programming tools
Channels and nodes properties	Access properties directly, e.g., node status, application tags are not required
Customization of write events	Easy setup for commands and events, write all events or only up or down value changes

Table 3-5. Devices and Networks

Datasets

The Dataset Module included in BluePlant provides an easy-to-use interface to exchange data in real-time with external databases, XML, CSV or text files and, as well as the possibility of accessing tables and SQL queries.

For the most popular databases and data sources (Microsoft SQL Server, Oracle, CSV files, Microsoft Access, PI, Firebird, Informix, Excel), BluePlant supplies pre-defined configurations that reduce the settings management to a mouse-click. Any database that supports ODBC, ADO.NET or OLE-DB can also be accessed. A built-in DB SQL Database Engine is supplied as an option of local database for the application.

The data collected with the datasets can be dynamically mapped to real-time points/tags and can be used on scripts or reports, or presented on displays using a powerful Data Grid Visual Object.

Datasets	
Access text, CSV and XML files	Define real-time binding with tags and file contents
Define multiple database sources	Easily manage multiple database connections
Tag mapping with data tables	High level configuration utility to manage the database tables used in the project
Queries and mappings definition	Manage multiple queries triggered by process events and filter conditions using real-time data points
Powerful data grid visualization object	Comprehensive and powerful data grid object to create user interfaces, local and on the Web
Table and queries properties	Access properties (e.g., row count) directly, where creation of application tags not required

Table 3-6. Datasets

Reports

BluePlant supports Web services, XML and other data-exchange interfaces to provide data for external reporting tools. In contrast with other packages, where the reports are necessarily created in another tool, BluePlant has its own built-in report editor.

The Report Editor allows the inclusion of dynamic text, dynamic graphical symbol and charts, dataset and query results, in a complete and easy to use editor. The reports can be saved in HTML, text, PDF or XPS formats and are easily presented in remote clients and Web displays.

Reports	
Built-in editor	User-friendly text editor, allowing the inclusion of tables, images, hyperlinks and text formatting
Text, HTML and XPS support	Save reports in multiple formats, such as XPS format that allows easy deployment in distributed environments
Copy and paste	To edit in Microsoft Word or in HTML or RTF editor, just copy and paste the BluePlant contents
Easily embedded real-time tags	With one click, user can add real-time data values on reports

Table 3-7. Reports

Note:

It is not possible to make reports based on dynamic results from the database, that is, the search should always return an exact number of rows.

Client Displays

The built-in graphics editor of BluePlant uses the Microsoft WPF technology to allow the easy creation of complete user interfaces with real-time mapping for process values and tags. A powerful and complete set of dynamic animations are also included.

The displays are internally saved using XAML, which provides resolution independence, isolation from the code and easy extensibility. A symbol library, where the symbols can also keep a dynamic link with the library, speeds up the synoptic process creation. All client technologies support redundant server scenarios.

Three technologies are used on remote clients:

- BluePlant Visualizer Clients: runs as a desktop application and allows the blocking of the Windows task switch (CTRL+ALT+DEL and ALT+TAB Windows keys). This is ideal for Intranet operators/users with high security requirements
- Web Smart Clients: uses the Microsoft .NET Smart Client technology and installs on remote clients with a single-click and no administration requirements. The application is automatically updated on the remote clients, when it is updated on the server. BluePlant uses all the power of the remote computer and yet retains the advantages of a centralized installation.
- XBAP Partial Trust Clients: the client displays can run directly from web browsers, with no installation of any software required (nor any Active-X controls). The Partial Trust Clients guarantees that the client displays will run in a completely isolated environment. As in the case of the Smart Clients, when the application is updated on the server, it is automatically updated on the clients too

Runtime Objects

More advanced than most systems, where the creation of tags or variables for all internal properties and custom logics for projects are necessary, BluePlant allows the applications to directly access all the runtime objects that were created in the project.

This means that temporary tags are not required to manage the status of PLC network nodes, the total number of alarms in a group, or the number of rows in a dataset. It is possible to access runtime

objects (representing a network node), an alarm group or dataset, and display required information or take action directly through their built-in properties.

Module Isolation

For enhanced performance, security and reliability, the most CPU consuming and sensitive modules, such as scripts, datasets, devices (communication drivers), reports and displays, run in their own processes, or application domain, in their own thread, independently from the server real-time database.

In addition to the previously described advantages, the BluePlant architecture also allows distribution of the data acquisition application, or any CPU intensive application, in different computers in a distributed environment, providing increased flexibility to implement many redundant scenarios and simplification for field maintenance.

Runtime and Diagnostics Tools

The property watch tool allows you to inspect and simulate values in all modules and objects and also start and stop all modules individually.

The trace window tool automatically generates system messages about important runtime events and can be easily extended to issue specific messages connected with script events, data point/tag changes or user actions.

The module information tool is an advanced performance and profiling tool that provides internal information for the entire runtime environment.

Runtime and Diagnostics Tools	
Test mode	Run projects with protections, such as, read-only on external devices or temporary files in the historian
Module information	Advanced tools for performance profiling and internal systems diagnostics
Localization tools	Create the Operator User Interface in any number of languages, and dynamically, switch between them in runtime mode
Trace window	When creating your an application, this tool provides tag monitoring and system diagnostics messages
Property watch	Verify and simulate tag values and properties, start and stop functional modules

Table 3-8. Runtime and Diagnostics Tools

Project Test and Deployment

Before executing an application or project, user may use the exclusive BluePlant “Test Mode” that runs the project or application in a safe testing environment. In “Test Mode”, no commands are sent to the remote controllers (only the read commands are sent); alarms and historian saves data on temporary files and the external real-time databases are accessed in read-only mode.

After successfully completing testing, it is needed to run the “Startup” option for full functionality. When the project or application is ready to be deployed in the field, it should be used the “Publish” feature to set up redundancy options (if applicable), and to create a read-only, version controlled, copy of project for the field installation.

Project Test and Deployment Tools	
Open multiple projects	Simultaneously open multiple projects on the same computer
Remote engineering	Remotely access and edit the project configuration
Run as a Windows service	Run the runtime server installed as a Windows service
Switch applications protection	Protect from unauthorized application switch on operator interfaces using CTRL+ALT+DEL Windows keys or others
Startup shortcuts	Use simple startup shortcuts and parameters for startup customization
Single file project and embedded resources	The entire project configuration is saved on a single secured file including all images and bitmaps used on displays and reports

Table 3-9. Project Test and Deployment Tools

Communication Drivers

There are several drivers available for the main manufacturers of PLCs and automation systems.

Communication Drivers	
Altus	ALNET I
	ALNET II
	FBS
MODBUS	RTU-TCP
	RTU-TCP Slave
Allen Bradley	CIP (Control Logix)
OPC	DA
	XML-DA
	UA
	Xi
Siemens	S7 Ethernet

Table 3-10. Communication Drivers

4. BluePlant

Installation and Operation Minimum Requirements

The BluePlant Student, BluePlant Express and BluePlant Lite models present four available channels while the BluePlant Enterprise model counts with 64. The following tables show the minimum requirements for the installation and operation of BluePlant, using different amounts of channels.

	BluePlant Express, BluePlant Student, BluePlant Lite and BluePlant Enterprise (up to 4 channels)
Platform	PC with Windows XP® (32-bit), Windows Vista® (32-bit), Windows 7® (32-bit or 64-bit), Windows 8® (32-bit or 64-bit), Windows Server2008 or Windows Server2012
Processor	Intel Core 2 Duo (min)
Disk space	1 Gbyte (min), 2 Gbytes (recommended)
RAM	2 Gbytes (min), 4 Gbytes (recommended)
Resolution	1024 x 768 (min), 1280 x 1024 (recommended)
Language	Any language

Table 4-1. 4-Channel Configuration up to 4 channels

	BluePlant Enterprise (up to 8 channels)
Platform	PC with Windows XP® (32-bit), Windows Vista® (32-bit), Windows 7® (32-bit or 64-bit), Windows 8® (32-bit or 64-bit), Windows Server2008 or Windows Server2012
Processor	Intel Core i5 (min)
Disk space	1 Gbyte (min), 2 Gbytes (recommended)
RAM	4 Gbytes (min), 6 Gbytes (recommended)
Resolution	1024 x 768 (min), 1280 x 1024 (recommended)
Language	Any language

Table 4-2. 8-Channel Configuration up to 8 channels

	BluePlant Enterprise (up to 16 channels)
Platform	PC with Windows XP® (32-bit), Windows Vista® (32-bit), Windows 7® (32-bit or 64-bit), Windows 8® (32-bit or 64-bit), Windows Server2008 or Windows Server2012
Processor	Intel Core i7 (min)
Disk space	1 Gbyte (min), 2 Gbytes (recommended)
RAM	6 Gbytes (min), 8 Gbytes (recommended)
Resolution	1024 x 768 (min), 1280 x 1024 (recommended)
Language	Any language

Table 4-3. 16-Channel Configuration up to 16 Channels

Note:

Platform: It is necessary to install Microsoft .NET Framework 4.0.

ATTENTION:

The amount of channels and data acquisition performance are the main reasons to select operation requirements. If the project requires more than 16 channels or any other specific project demands, it is strongly recommended to contact the Altus Technical Support via the website www.altus.com.br or e-mail altus@altus.com.br.

Installation

To run the BluePlant software installation is necessary to perform the download of the installation file from the site www.altus.com.br. After that, the user must close all the programs running on PC, double-click the installation file, and then click Next.

The license agreement screen that appears should be read carefully. If the license terms are accepted, the user should select the corresponding option. The following installation screen will appear. By a click on "Next" button, the installation proceeds.

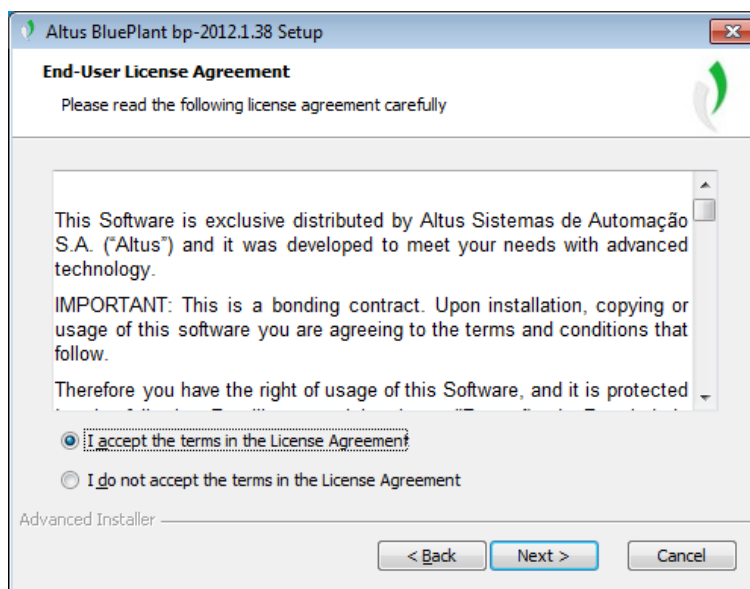


Figure 4-1. Screen of the BluePlant License Agreement

During the installation, another agreement screen will appear, which should be read carefully. It refers to the Advosol OPC Core Components software. This software offers a full support in .NET platform for all major OPC standards, since OPC is a set of standardized specifications. Such patterns resulted from the collaboration of several leading automation suppliers around the world working in cooperation with Microsoft. The specification defined a standard set of objects, interfaces and methods for use in process control and automation applications to facilitate interoperability. Currently, there are hundreds of OPC Data Access servers and clients.

If the terms are accepted, the check box must be selected to continue the installation. The following installation screen will appear. By a click on "Next" button, the installation proceeds.

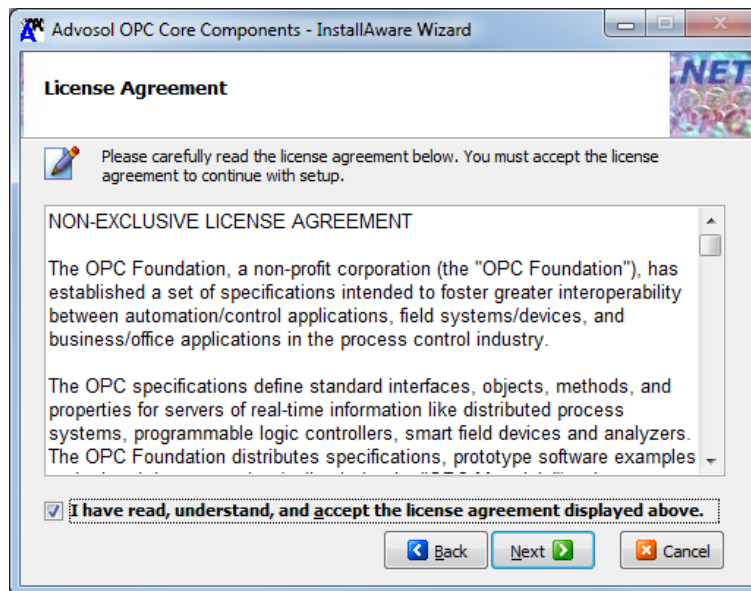


Figure 4-2. Screen of the Advosol License Agreement

The following screen shows the installation progress of the Advosol OPC Core Components. The user must wait until all the required files are completely installed on the computer, which may take several minutes, depending on its configuration.

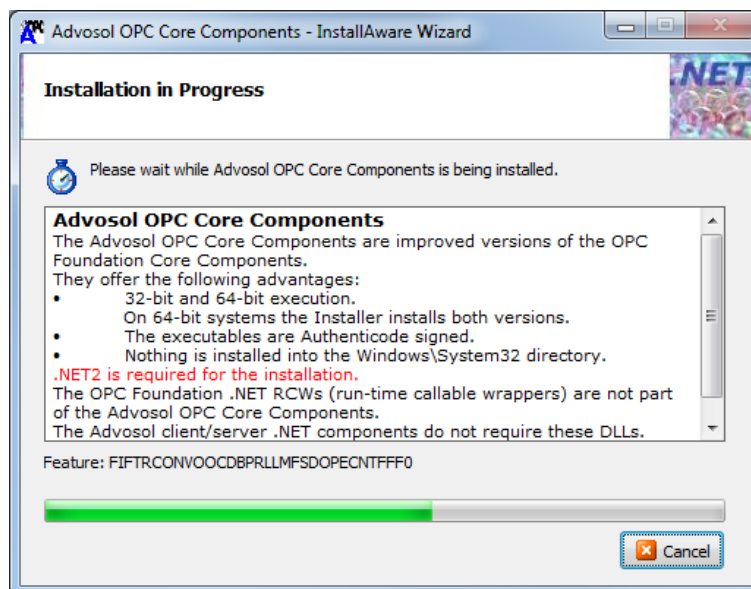


Figure 4-3. Advosol Installation Screen

When the Advosol OPC Core Components is finished, the BluePlant installation gets started. Wait until the required files are installed on the computer. This process also may take several minutes depending on the computer's configuration.

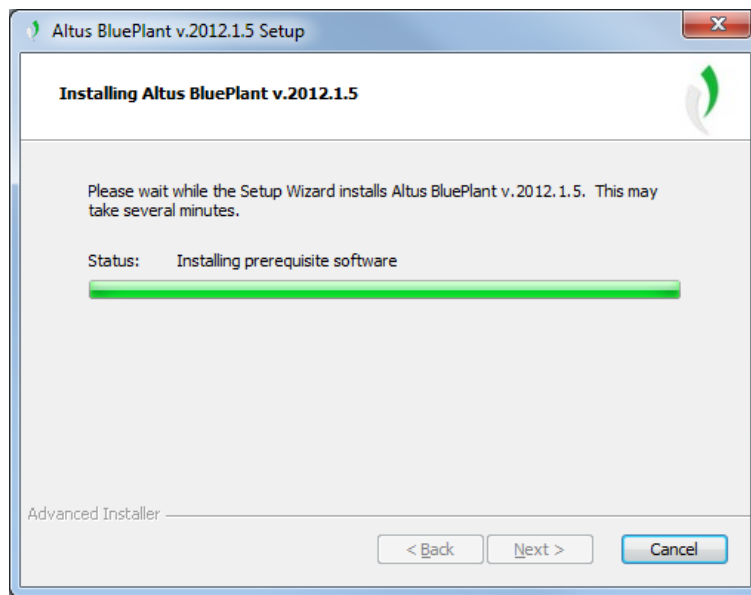


Figure 4-4. BluePlant Installation Screen

After the installation is complete, the following screen will be displayed. The "Finish" button should be selected to finish the installation procedure.



Figure 4-5. Screen of the Closing of the BluePlant Installation

Upon completion of these steps, BluePlant is installed and ready to use. To execute it, user should click on the "BluePlant" shortcut created during the installation process.

Licenses and Hardkey

Some BluePlant models have features which are enabled through licenses present in a hardkey. The hardkey is a physical environment where the required licenses are saved. In order to release the features present in the same hardkey, it must be connected to the server where the user will run the project.

In the table 2 1, can be checked the available BluePlant models and, except for the Express model, all the others are enabled by licenses present in hardkey.

Once the hardkey is connected to the server, the information can be viewed via the tab "License" as shown in Figure 4-9.

ATTENTION:

If the hardkey is damaged, and the access to its data is compromised, the enabled features will no longer be available. In this case the user should contact Altus Technical Support via the website www.altus.com.br or e-mail altus@altus.com.br.

Starting BluePlant

Once BluePlant has been installed on the computer, a double click on the following icon opens the software.



Figure 4-6. BluePlant Icon

Project Management

Once BluePlant has started, the Project Management window appears.

On the center of the initial screen, in its top part, the user visualizes three tabs: "Projects", "Server" and "License".

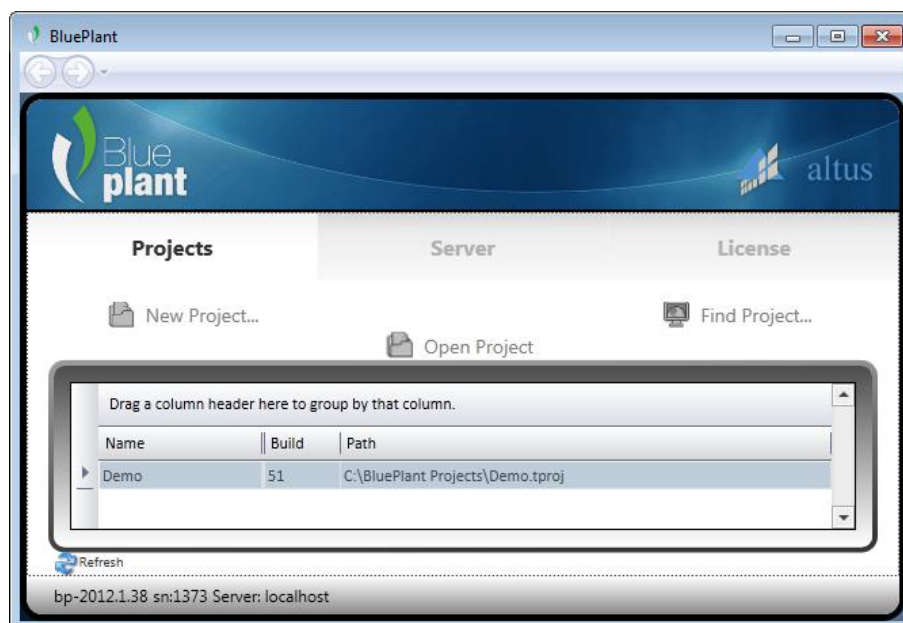


Figure 4-7. Project Management Window

Immediately the Web standard menu structure of BluePlant is shown. This design allows a more efficient access to the tool set by reducing the number of clicks required to access them. It also shows a table indicating the projects which can be accessed.

The "Projects" tab displays the icons used to create, open and find projects ("New Project", "Open Project..." and "Find Project ..."). These icons are not placed in the default BluePlant directory.

The "Server" tab, shown on Figure 4-8, presents two options: Localhost and Remote. The first one means that the project will run locally and the latter that the project will run from a remote server.

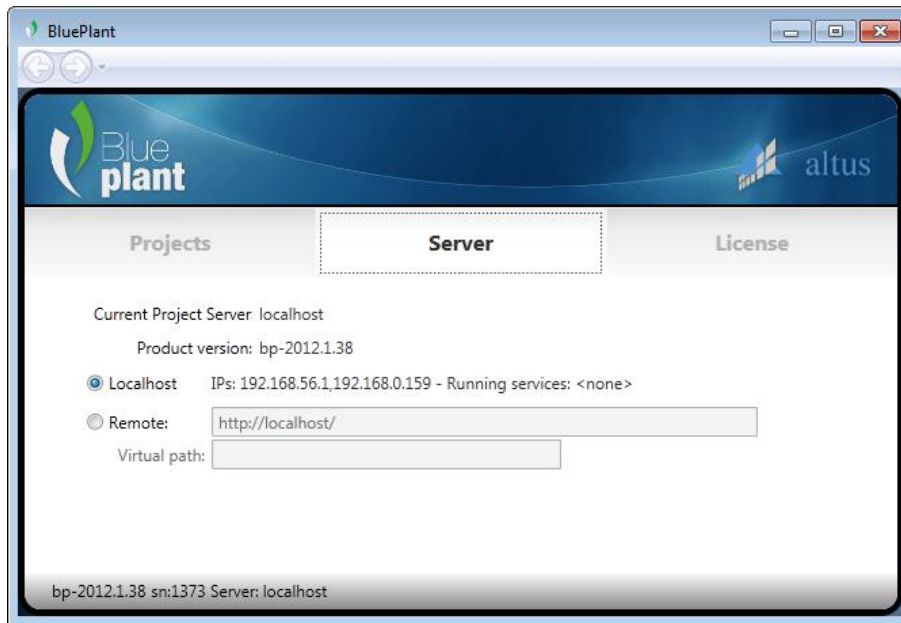


Figure 4-8. Server Tab

The third tab shown on Figure 4-9 refers to the product license, including the model, product version, family, serial number, and product code among other informations available in the hardkey.

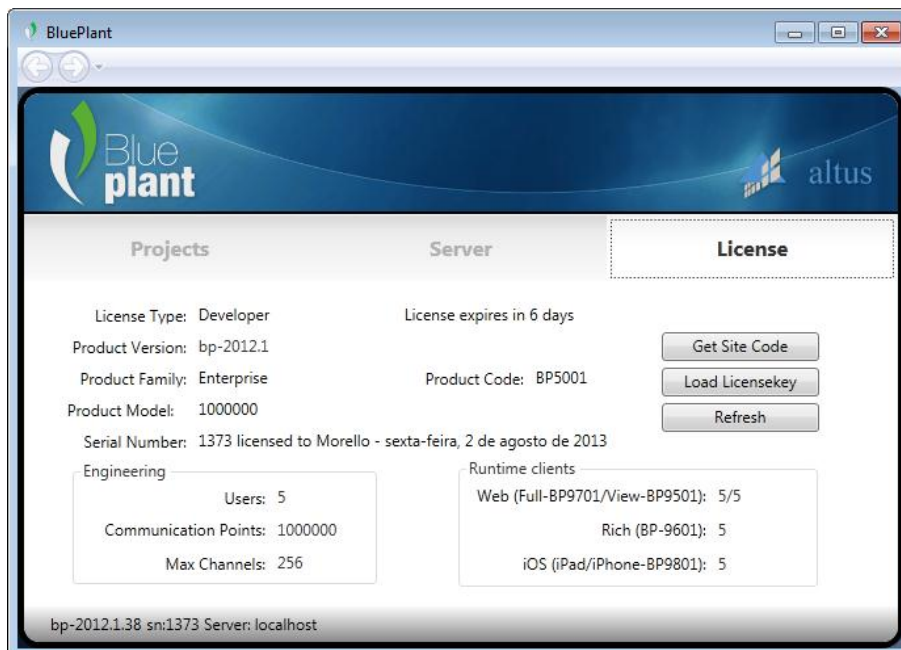


Figure 4-9. License Tab

Getting Started with BluePlant

This section presents a sequence of operations for creating a simple project or opening an existing one. Along with the tool, it is provided a demo which allows a simplified view of the BluePlant available resources. The details of the referred features will be explored throughout this manual.

Selecting the Demo Project

From the tab "Projects ...", in the project editor, the demo project (Demo.tproj) can be selected, which includes an overview of the BluePlant features and resources.

To open the project, the user must select the project available in the list and click "Open Project".

Creating a New Project

The basic settings of the project to be created are defined from the BluePlant opening window by clicking on "New Project ...", as shown on Figure 4-7.

The next window, shown in Figure 4-10, indicates the steps to create a new project.

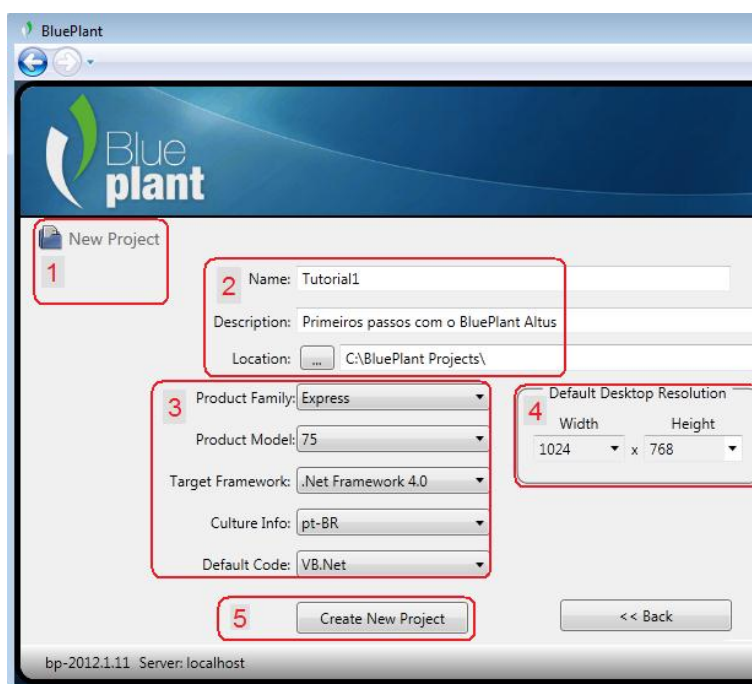


Figure 4-10. Creating a New Project

1. New project indication
2. Definition of the project name, description and directory where the project will be generated
3. Initial settings such as: family and product model, platform, culture and standard code (script)
4. Screen resolution settings
5. Button to create a new project and start the application development

The new project created is referenced in the list of the BluePlant opening window.

User Interface

The user interface configuration operates as a front-end Web page where the user can easily browse between items and edit them. The data are saved automatically on the back end, without having to open/save/close the menus. On the left, the user interface displays the main menu on top and the submenu right below, which will change according to the user's selection in the main menu. The right

part shows the desktop tabs, and at the top, the user can view a bar of quick links and the recent items. Figure 4-11 illustrates the user interface.

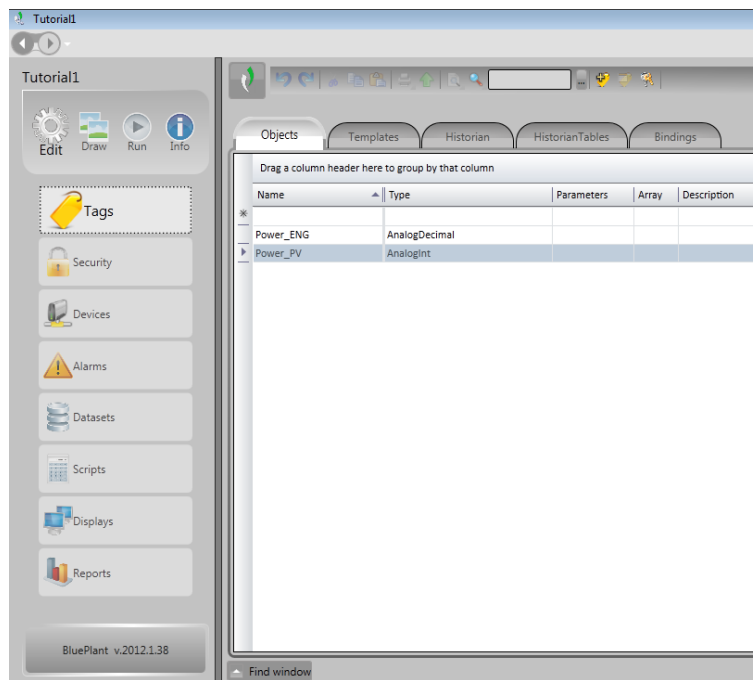


Figure 4-11. User Interface

Creating Tags in the Project



In the editing environment the user can edit objects such as tags. Figure 4-12 shows the way for editing them. In this example, two analog tags have been created: the gross value of an electric power meter to be read from the programmable controller called Power_PV and the engineering value of this measurement called Power_ENG.

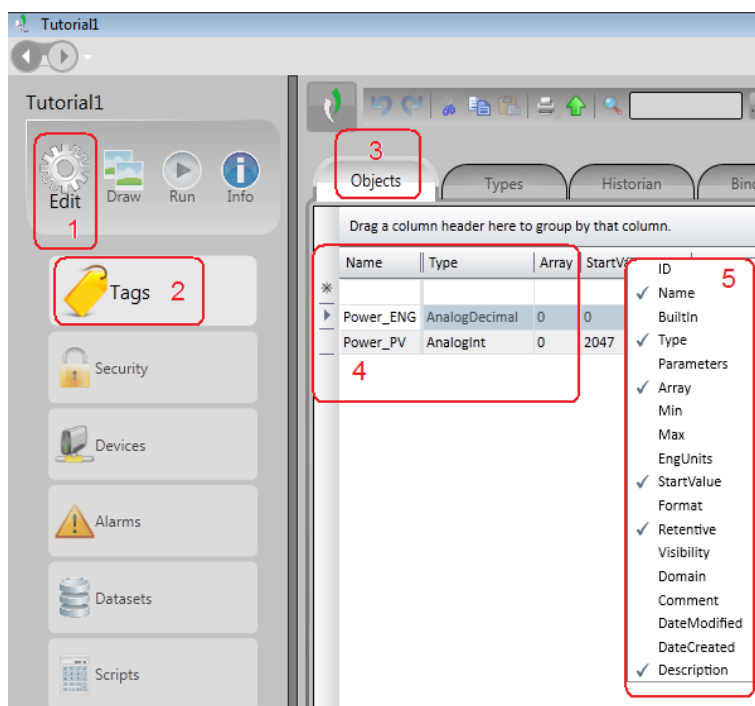


Figure 4-12. Editing Tags in the Project

The steps for creating these tags are numbered in the preceding figure and are described below:

1. Access the Edit menu
2. Select the Tags option
3. Access the Objects tab where will appear the table with the user tags
4. Enter the data related to the new application tag in the row labeled with an asterisk (*). Confirm the insertion in the table by clicking the Enter key
5. Choose the tags configuration options through the desired columns of the table by clicking with the right button of the mouse in the header of a column and marking the items for exhibition

It is also possible to create a new tag through the toolbar at the top of the edition environment. The user should click the icon and set the tag parameters.

Figure 4-13 shows the creation process of a unidimensional array tag with three positions of analog integer type from the Edit Menu. An array tag comprises a set of tags with the same name, whose IDs are given through indexes. In this example, the syntax to access the tag is: TT[0] TT[1], TT[2] and TT[3].

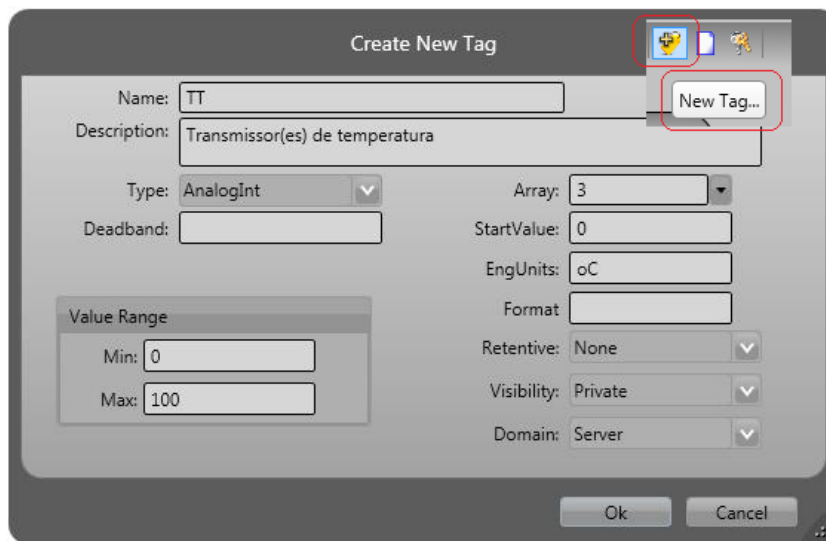


Figure 4-13. Tag Creation Process

Associating Tags to a Communication Protocol

To relate a communication protocol with the created tags, the following steps must be performed.

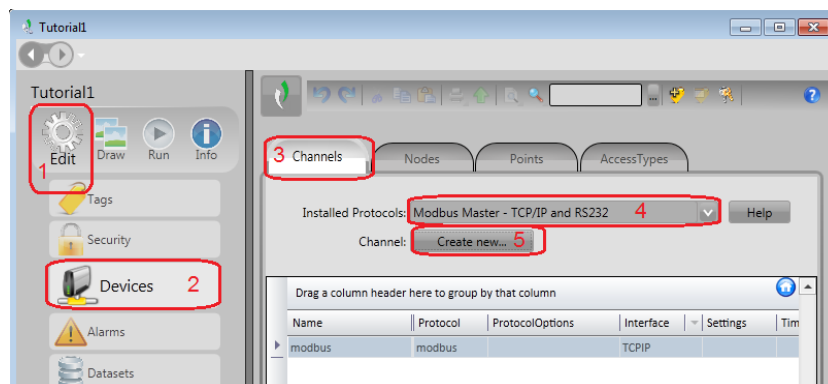


Figure 4-14. Communication Channel Creation Process

1. Access the Edit menu
2. Select the Devices option
3. Access the Channels tab on which should appear the field with the available protocols
4. Select in the field Installed Protocols the communication protocol to be used
5. Click on the button “Create new...” to confirm the creation of the channel

After the protocol selection and channel creation, a window with the properties of the protocol should appear, as can be seen in Figure 4-15.

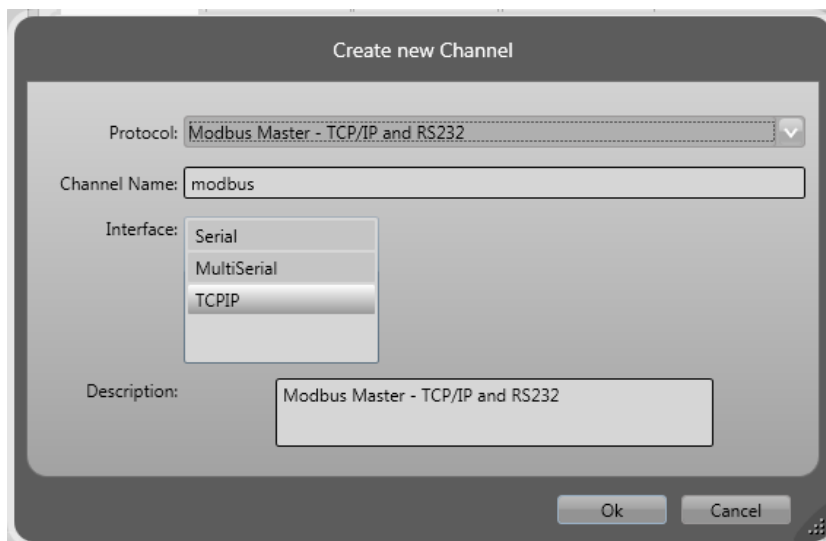


Figure 4-15. Protocol Options for Channel Creation

On the Nodes tab (figure below) the nodes are created according to the existing channels. For more details on how to create and configure nodes, see Editing Devices chapter.

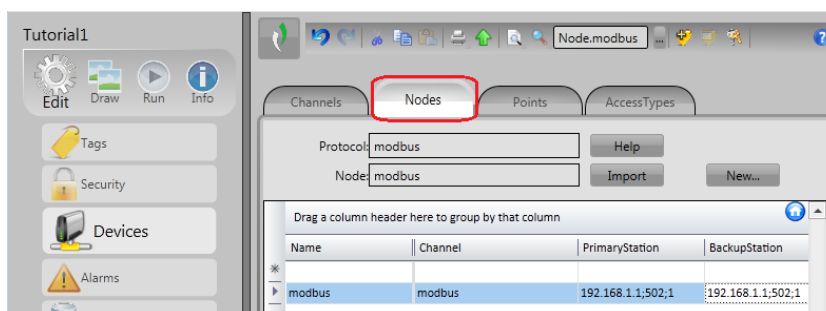


Figure 4-16. Inclusion and Nodes Setting

The Points tab (figure below) allows to create an association between tags, nodes and the protocol addresses, as well as access type settings, ranges and data types.

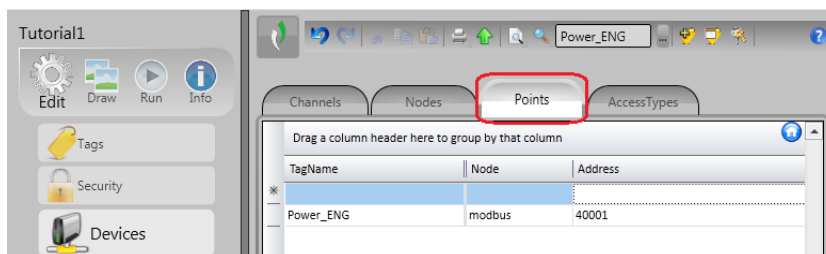


Figure 4-17. Association of Tags with Nodes and Protocol Addresses

As shown in Figure 4-18, by selecting the TagName field (in the Point tab), it is displayed a window listing all the created tags. Then, the selected tag can be associated to the created node.

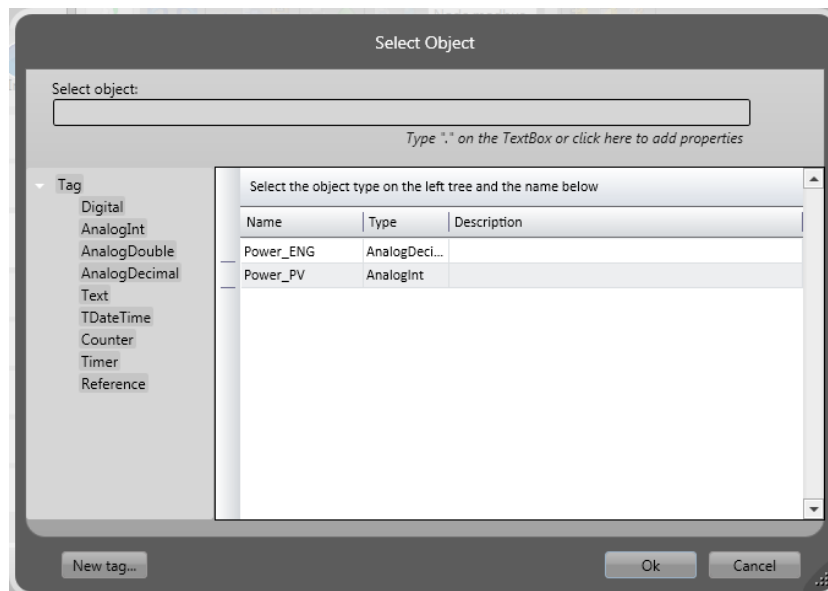
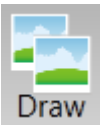


Figure 4-18. Selecting the Tag for Node Association

Including Objects in the Main Screen



The Draw menu allows to create the application screens. The screens are made up of objects, such as symbols and texts.

Let's explore the inclusion of these two types of objects in the default main screen.

Adding Symbols

To add a tachometer symbol type on the main screen (MainPage), the following steps should be considered:

1. Select the Draw menu
2. Access the Drawing tab
3. Open the Symbol Library
4. Select the symbol (the tachometer, in this example), placing it at the desired location
5. Configure the symbol properties

These steps are referenced in Figure 4-19, with the correspondent numeration.

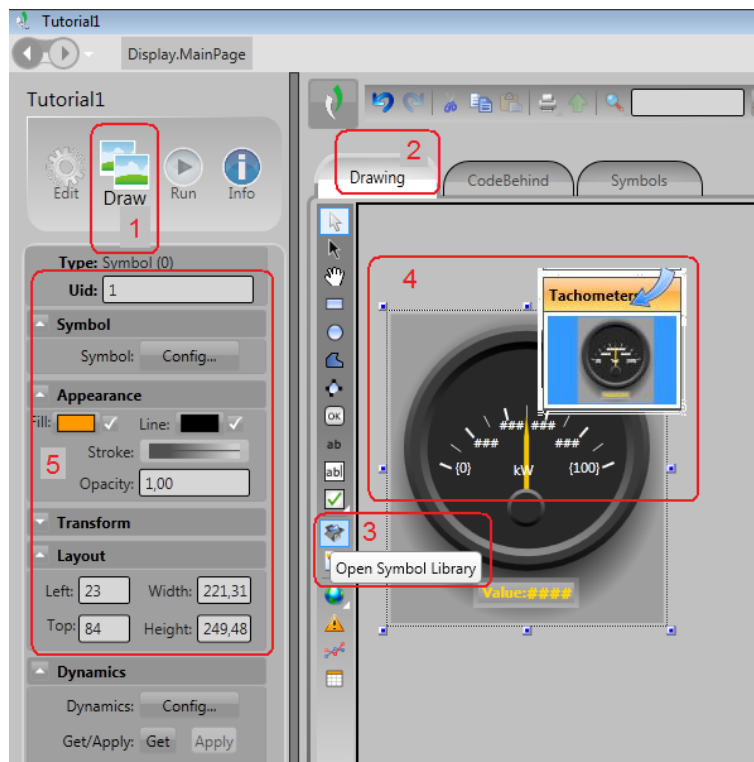


Figure 4-19. Add a Tachometer Symbol Type

To associate a tag to the symbol, the following steps should be considered:

1. Click with the right mouse button on the symbol
2. Select the option Symbol links

Figure 4-20 shows these steps to tag association.

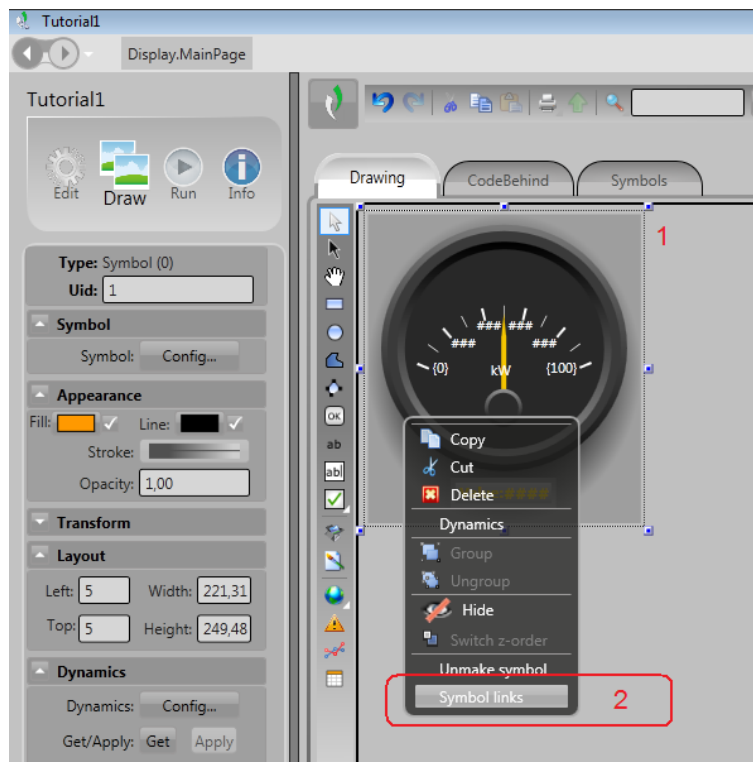


Figure 4-20. Association of Tag to a Symbol

In the next window (Figure 4-21) the user can configure the tag and its parameters as described in the following steps:

1. Configure the symbol properties (Label and Max/Min values)
2. Associate a tag to the symbol in RotateValue field.

In this example it was used the client tag "SimulationAnalog" to simulate the motion of the meter pointer.

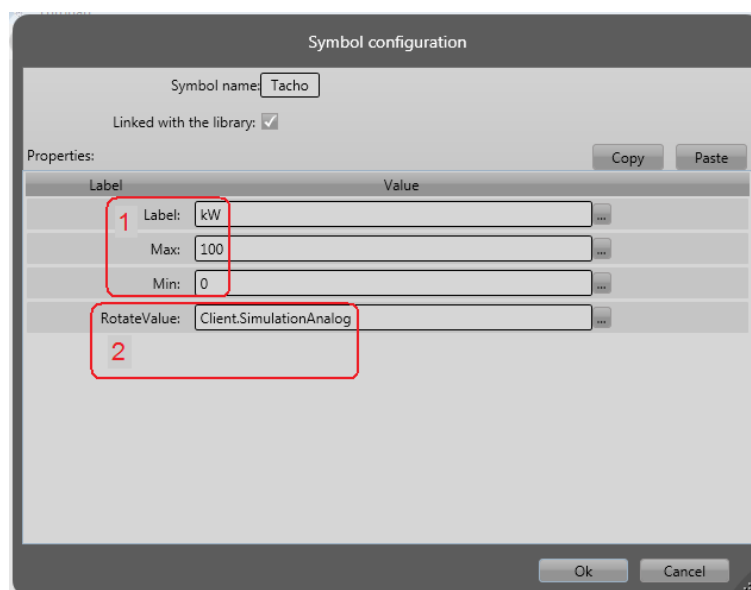


Figure 4-21. Configuration of the Tag Associated to the Symbol

Adding Text

To add a text on the screen (MainPage), the following steps should be considered:

Select the Text Output option:

1. Define the text location
2. Set the text properties

The Figure 4-22 illustrates this sequence.

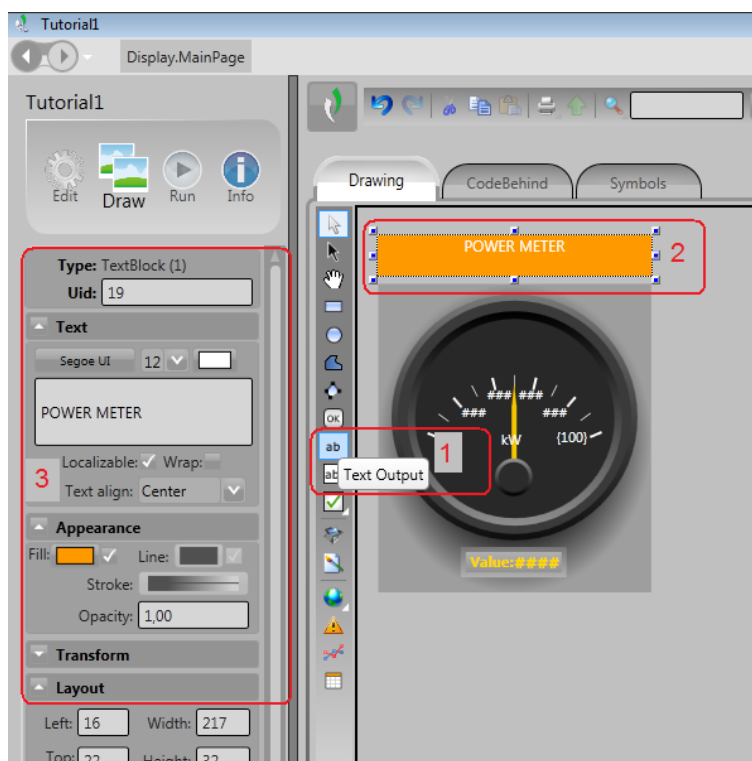


Figure 4-22. Adding Text

Running a Project



The Run environment provides access to all execution resources of the Project.

The following modules are available and are explored in this section: Build, Test, Startup and Publish.

In addition, this menu includes the following tools: UseCount, Localization and Extensions, which will be explored throughout this manual.

Project Build

The compilation checks the project against errors and optimizes the system aiming a fast and efficient operation. The time required to compile a project depends on its size and on the processing capacity of the computer. To build the project the following steps serve as guidance:

1. Select the Run menu
2. Select the Build option
3. Access the Messages tab
4. Press the Build button
5. Confirm the build selection

Figure 4-23 illustrates the project building process.

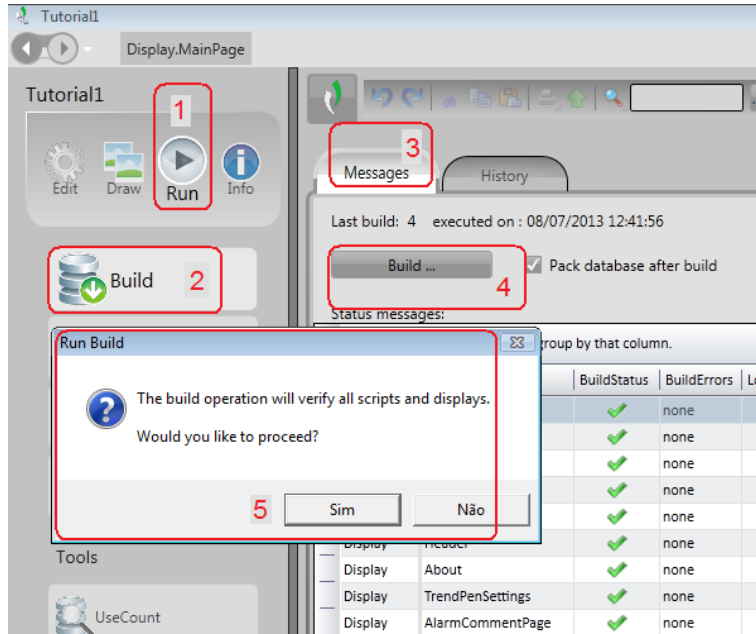


Figure 4-23. Project Building

Figure 4-24 shows the Build progress window.

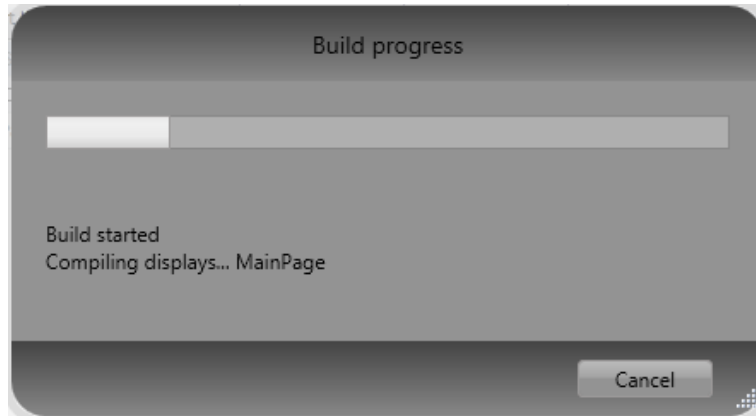


Figure 4-24. Build Progress Window

Figure 4-25 illustrates the result of compilation and errors, if any.

Status messages:

Drag a column header here to group by that column.

Module	Object ID	BuildStatus	BuildErrors
Script	Class.ServerMain	✓	none
Script	Class.ClientMain	✓	none
Display	MainPage	✓	none
Display	SelectPage	✓	none
Display	LogOn	✓	none

Figure 4-25. Build Result

Project Test

Figure 4-26 shows the steps described to test the project.

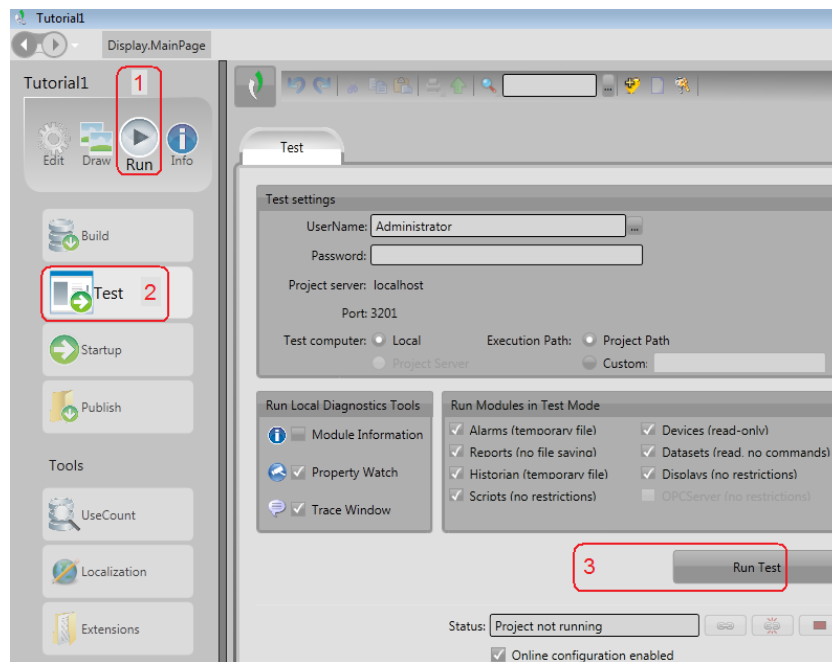


Figure 4-26. Project Test

1. Select the Run menu
2. Select the Test option, which runs the project into test mode. The user can configure the test, diagnostic tools and modules to be tested
3. Start the test with the configured settings via Run Test button

Figure 4-27 shows the project test result.

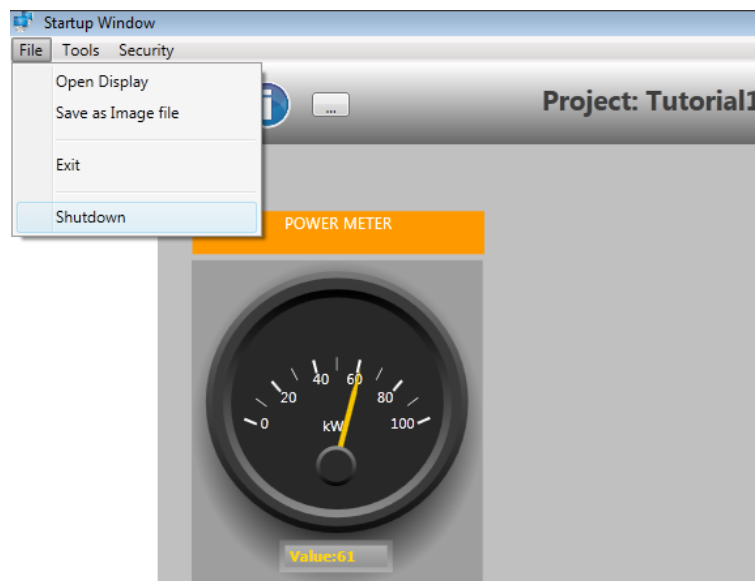


Figure 4-27. Running Project Test

To end the project test, the user can open the File menu and select the Shutdown option, as shown in Figure 4-27.

Project Startup

Figure 4-28 shows the sequence to launch the project (run startup).

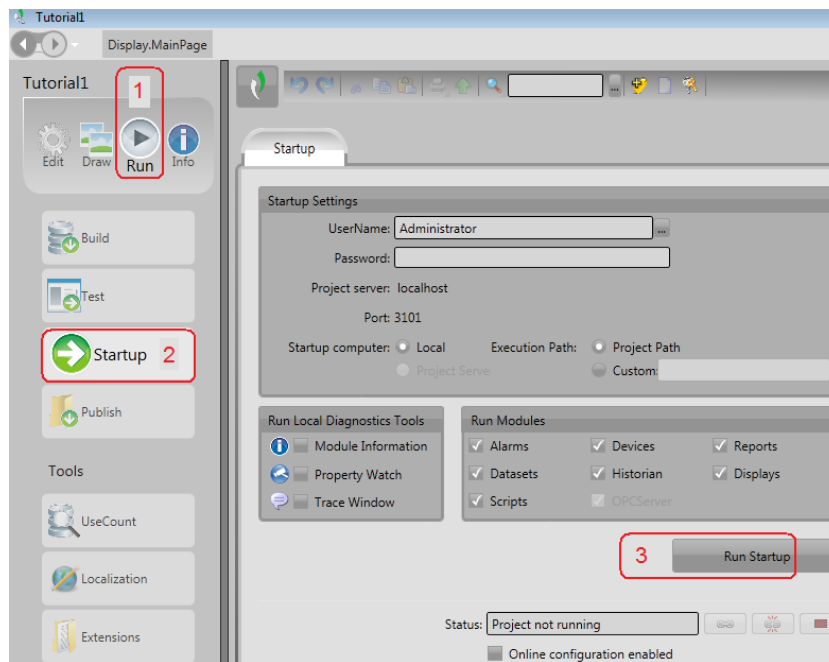


Figure 4-28. Project Startup

1. Select the Run menu
2. Select the Startup option, which allows the user to configure the startup, diagnostic tools and modules to be initialized
3. Start the project execution with the configured settings via Run Startup button

Along with the project main screen, a window will be opened showing the status and startup messages as can be seen in Figure 4-29.

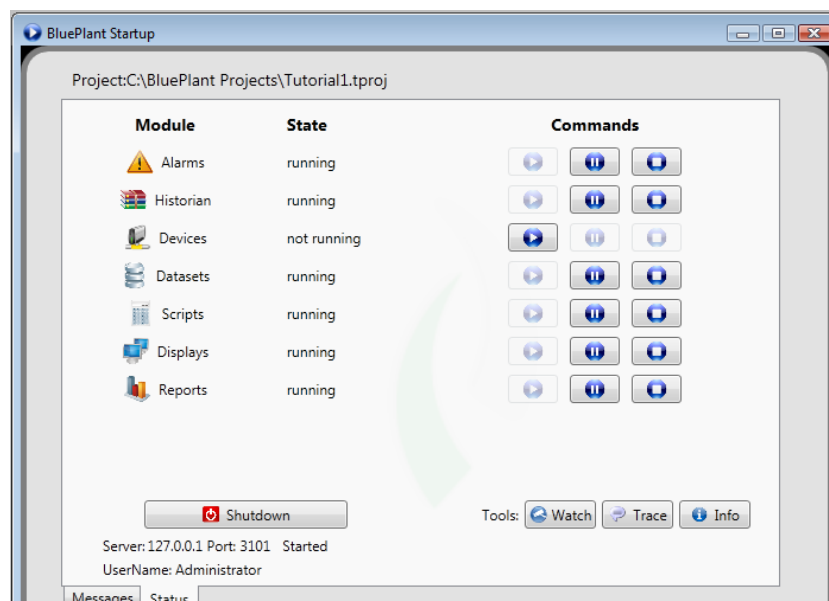


Figure 4-29. Project Startup Status

Project Publication

When the project is ready to be executed in field, the publishing feature must be used to configure the redundancy options (if applicable) and to create a copy of the project in read-only mode with controlled version, i.e. a copy of the project to run in the field. The extension of the generated file on the occasion of the publication is ".teng". Figure 4-30 shows the project publication sequence.

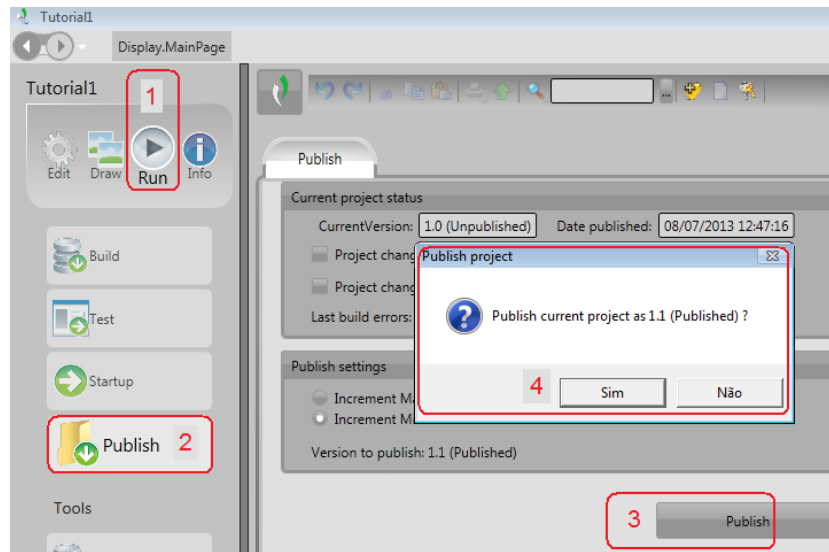


Figure 4-30. Project Publication

1. Select the Run menu
2. Access the Publish option, which allows to access the settings of the project publication
3. Click the Publish button, which triggers the project publication with the configured settings
4. Confirmation window for the project publication

After the publication, a similar window to the one shown in Figure 4-31 appears indicating the path of the published project.

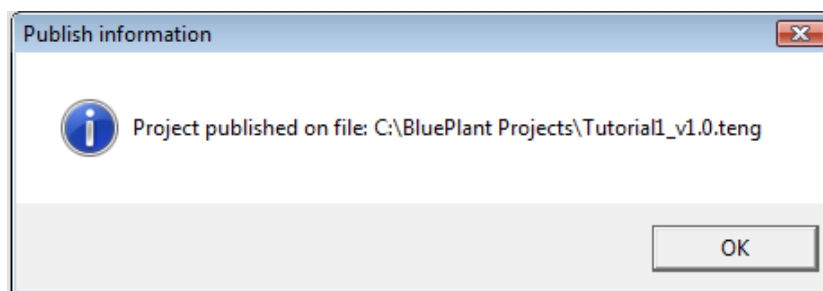


Figure 4-31. Path of the Published Project

5. BluePlant Main Menu

Each supervisory system manages a certain number of objects (also called entities) that describe the controlled process variables and the usual control elements. The configuration of a supervisory usually comprises two steps:

- Define each process variable in the database
- Define synoptic, graphics and reports

In the system there are simple, primitive and composed variables, the latter formed from the first. In this context the variable name is called Tag.

In this chapter are described the four basic BluePlant menus: Edit, Draw, Run and Info, which contain the necessary tools for the development of a supervisory system project comprising the typical steps previously listed. These four menus are described in the sections Application Editing, Application Diagramming, Application Execution and Application Information respectively.

Application Editing

The Edit menu allows the user to access the tools for editing a project in BluePlant. Figure 5-1 shows the menu selection.

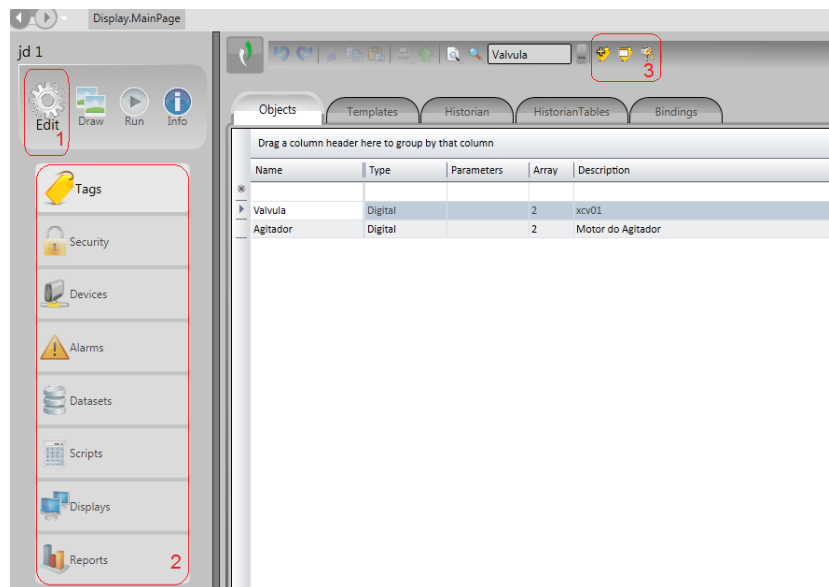


Figure 5-1. Edit Menu

The items that comprise the Edit menu are numbered in the preceding figure and are described below:

1. Selection bar of the Project Editing menu
2. Edit menu toolbar, containing the configuration resources for the following elements: tags, users and their security policies, devices and their communication protocols, alarms, database, script language, displays and reports
3. Shortcuts on top toolbar, including the definitions to create a new tag, its properties and users configurations as well as his access to the application elements

Application Diagramming

In the Draw menu is possible to create the screens and symbols that make up the application. Figure 5-2 illustrates the inclusion of symbols on the main screen (MainPage) with the help of the toolbars associated.

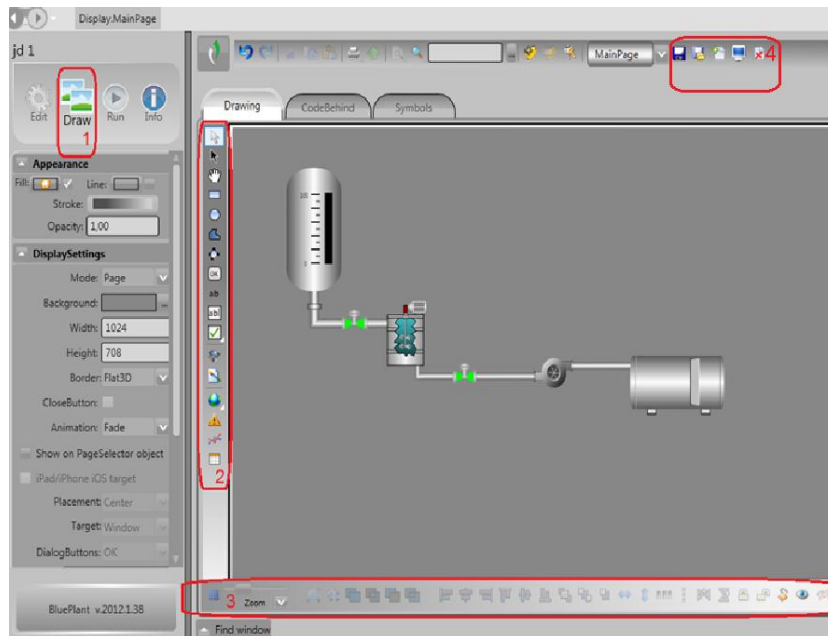


Figure 5-2. Draw Menu

The items that comprise the Draw menu are numbered in the preceding figure and are described below:

1. Selection bar of the Project Drawing menu
2. Vertical toolbar used to include, manipulate and configure the elements that will compose the application screens
3. Horizontal toolbar located below the screen tab, which presents some commands to group, combine, align, lock the selected component(s), etc.
4. TopToolBar located on the top part of the Draw Menu main screen, which have icons with the following functions: create new display(s), delete display(s), save display(s) and view the already existing display(s) in the project

Application Execution

The Run menu provides access to all project execution features. The following menu items are available in the Run environment: Build, Test, Startup, Publish and Tools. Figure 5-3 illustrates the menu.

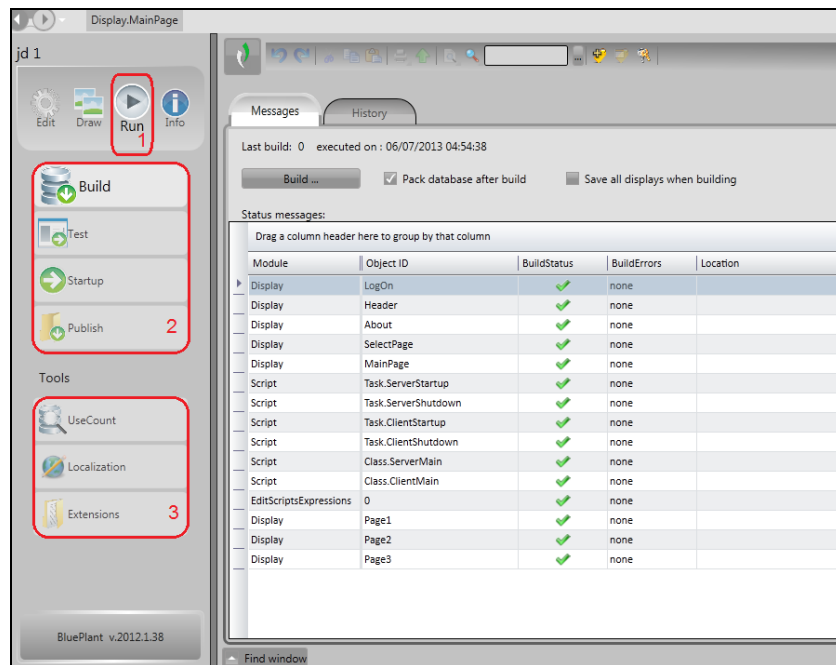


Figure 5-3. Run Menu

The items that comprise the Run menu are numbered in the preceding figure and are described below:

1. Selection bar of the Project Execution menu
2. Project execution features
 - Build command, which prepares a project for implementation and save it in the Project Build History. It is divided into Messages Compilation (displays status, errors, location and module informations) and Historical Compilation (displays compilation, errors, execution date and users informations)
 - Test command, including the settings associated with the project test
 - Startup command to initialize the project settings
 - Publish command that allows the user to access the project publication settings
3. Tools module, including the counting tags and objects functionalities via "UseCount" and "CrossReference" commands, translation settings at Runtime (Localization) and execution tools as well as Add-ons (Extensions)

Application Information

The Info environment provides access to the current project information. Figure 5-4 illustrates the menu.

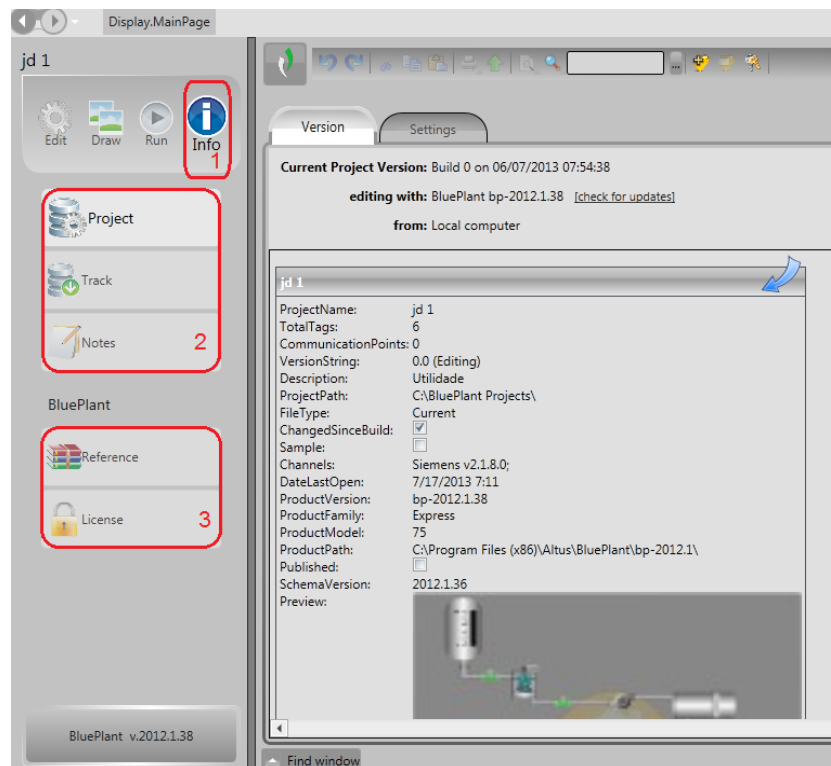


Figure 5-4. Info Menu

The items that comprise the Info menu are numbered in the preceding figure and are described below:

1. Selection bar of the Project Information menu
2. Application information, divided in Project (project settings), Track (tracking of changes) and Notes (user notes)
3. Software information where the user can get information about the software license as well as access the help tool

6. BluePlant Components

This chapter provides information about the configuration tools, runtime and BluePlant applications. It goes according to the following structure:

- Configuration tools: menus Edit, Draw, Run and Info
- Runtime tools: applications tools, runtime objects and command lines
- BluePlant applications: displays in other operational systems and service on Windows

Edit Menu



The Edit environment provides access to all the required functionalities to configure the project. It comprises the following items.







Functionality	Graphical Representation
Tags Edition	 Tags
Security Definitions	 Security
Devices Configuration	 Devices
Alarms Planning	 Alarms
Datasets	 Datasets
Script Language	 Scripts
Displays Project	 Displays
Report Configuration	 Reports

Table 6-1. Edit Menu Functionalities

The following sections detail the functionalities.

Editing Tags

The Edit Tags menu configures the tags database in real time.

Edit Tag Objects

Use the tags (and their pre-defined properties) included in the table below to configure a realtime database. Through the EditTagsUserTypes table, the available types can be extended, and new types can be created as well.

Note:

The term "Tag" in the context of a Project configuration refers to a process variable. Figure 6-1 illustrates the edition of Tags Objects.

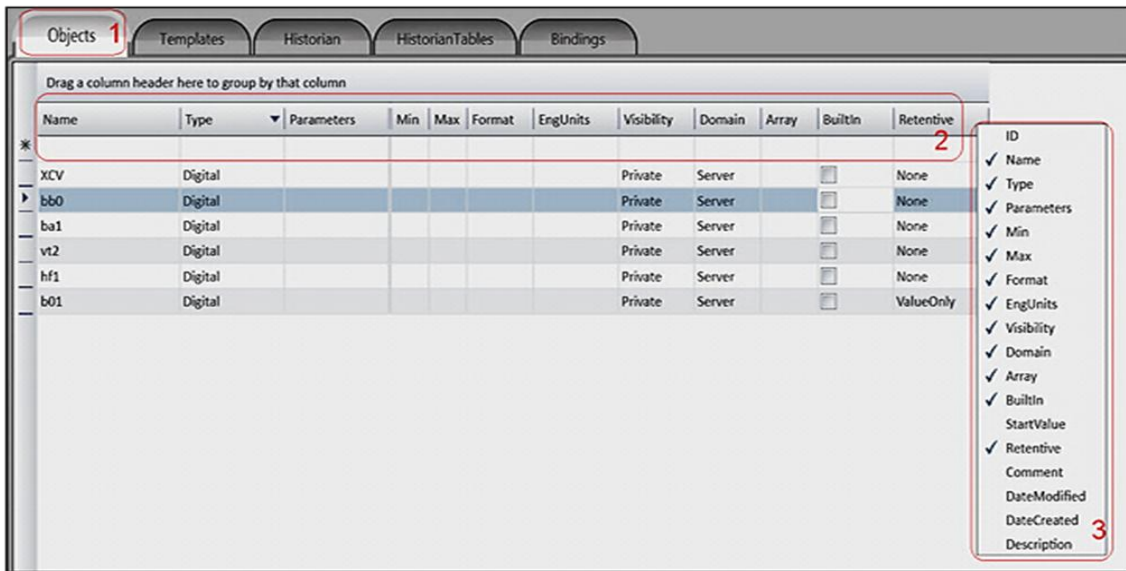


Figure 6-1. Edition of Tag Objects

The Edit menu items numbered on the previous figure are listed below:

1. Select the Objects tab
2. Enter the new application Tag data on the row labeled with an asterisk (*)
3. Choose the tag settings by clicking with the right button of the mouse in the header of a column and marking items for exhibition

As shown on Figure 6-1, each item is detailed below.

Name

This field defines the tag name. In this column it is possible to create or edit tags.

Type

This field sets the type of the tag. Available tags:

Type	Description
Digital	True or False
AnalogInt	Integer
AnalogDecimal	Decimal
AnalogDouble	Floating point
Text	Text
Timer	Time
Counter	Count
Reference	Reference object pointing to another object
DateTime	Date and time
UserTypes	User defined types

Table 6-2. Available Tags Types

Note:

Regarding the Reference type, a reference object must be initialized to point to another object. This is typically accomplished using the following syntax in a script body:

```
@Tag.Reference1.Link = @Tag.TagName.GetName();      (VB)
@Tag.Reference1.Link = @Tag.TagName.GetName();      (C#)
```

Parameters

This field defines the tag parameters according to the type: Deadband for Analog tags and settings for Timer and Counter tag types. When creating "Reference Tags" (or POINTERS) define the target reference tag in the "Reference Type" column.

Min

This field sets the minimum value acceptable for the tag.

Max

This field sets the maximum value acceptable for the tag.

Eng Units

This field defines the base Engineering Unit for the tag.

Format

This field specifies the display value format. For valid numeric formats, refer to Standard Numeric Format Strings. Ex: N1 (number with 1 decimal place). For valid date and time formats, refer to Standard Date and Time Format Strings. Example: d (short date). For a more in-depth discussion of format strings, refer to Formatting Types. Numeric formats example:

Specifier	Description
N0	Number with no decimal places
N3	Number with 3 decimal places
X	Hexadecimal (supported only for integral types)
C	Currency

Table 6-3. Example of Numeric Formats

Examples of DateTime formats:

Specifier	Description
T (only)	Long time pattern (equivalent to "HH:mm:ss")
d (only)	Short date pattern (equivalent to "M/d/yyyy") (month / day / year) (US)
dd	Represents the day of the month as a number from 01 through 31
ddd	Represents the abbreviated name of the day of the week
dddd	Represents the full name of the day of the week
MM	Represents the month as a number from 01 through 12
MMM	Represents the abbreviated name of the month
yy	Represents the year as a two-digit number
yyyy	Represents the year as a four-digit number
hh	Represents the hour as a number from 01 through 12
HH	Represents the hour as a number from 00 through 23
mm	Represents the minutes as a number from 00 through 59
ss	Represents the seconds as a number from 00 through 59
fff	Represents the milliseconds as a number from 000 through 999
tt	Represents the A.M./P.M. designator

Table 6-4. Example of DateTime Formats

Example: long time pattern

Format = HH:mm:ss (hour; minute; second)

Visibility

This field defines the tag value visibility on the OPC server for remote Projects. The following configuration options are available:

- Private: defines a Tag visible only to local project and redundant pair
- Protected: defines a Read-only Tag visible on the OPC DA server to remote Projects and OPC DA clients
- Public: defines a Tag visible on the OPC DA server to remote projects and OPC DA clients

Domain

The domain defines the tag value for the entire project or a specific value to each client display. The following configuration options are available:

- Server: the Tag value is consisted across the entire project and all clients
- Client: the Tag value is local to each remote computer running a client display (Web or display)

Note:

Client Tags should not be used in server modules like Device, Alarm, Historian and ServerScripts because its value is local (restrict to the running computer) and is not propagated to remote clients.

Most tags in a project are defined as "Server". tags "Local" allow different values on each client computer. However, it is possible to use "Local" tags in specific temporary data to individual client computers. The most common situation for using "Local" tags is when temporary data are needed to manage the User Interface on the displays. "Local" tags allow different values in each client computer.

Array

When not defined (blank) the tag is not an Array. When defining an array with value N an array is created from position 0 to N. For example, when creating a Tag array of size "5", the Array is created

from Tag[0] to Tag[5], this means that 6 elements are created. Two programming styles are accommodated by this method: the indexing from 0 and the counting from 1.

Note:

Regarding the array extensibility by account levels, the Express and Lite versions are limited to one dimension array. The Enterprise version can create arrays with up to 3 dimensions.

Example 1

Name field: TagAnalog
Array field: 2

Creates 3 elements: TagAnalog[0] TagAnalog[1] TagAnalog[2]

Syntax to access:

C#: TagAnalog[1]
VB.NET: TagAnalog(1)

Example 2

Name field: Temp
Array field: 1,2

Creates 6 elements: Temp[0][0] Temp[0][1] Temp[0][2] Temp[1][0] Temp[1][1] Temp[1][2]

Syntax to access:

C#: Temp[1,2]
VB.NET: Temp(1,2)

Startup Value

This field sets the Tag startup value. When left blank no startup value is applied.

Retentive

The Retentive column specifies if the Tag properties are retained when shutting down the application and used as startup values on next execution. The following configuration options are available:

- Non-Retentive: Tag value is not save
- ValueOnly: Tag value is saved when modified and can be used as startup value on next execution
- Properties: all Tag properties are saved when modified to be used on next startup

Note:

In the edition process the ENTER key must be pressed in the table cell, in order to confirm the accomplished configurations.

Edition of User Defined Types

An existing type tag can be localized in the "User Custom Type" field.

Click  to delete any existing Struct Type Tag and  to create a new Struct Type Tag.

A UserType Tag can be used in the same manner as the built-in Tags.

Note:

Express and Lite versions allow UserTypes to be used only on the main TagsObjects list. Enterprise version allows creating up to four levels of UserTypes.

Example: When creating a PID UserType with 'setpoint' and 'PV' members, it is possible to create a Tag named 'loop' (PID type). The syntax to access its value is:

loop.setpoint and loop.PV

Figure 6-2 shows the edition of the defined types by user.

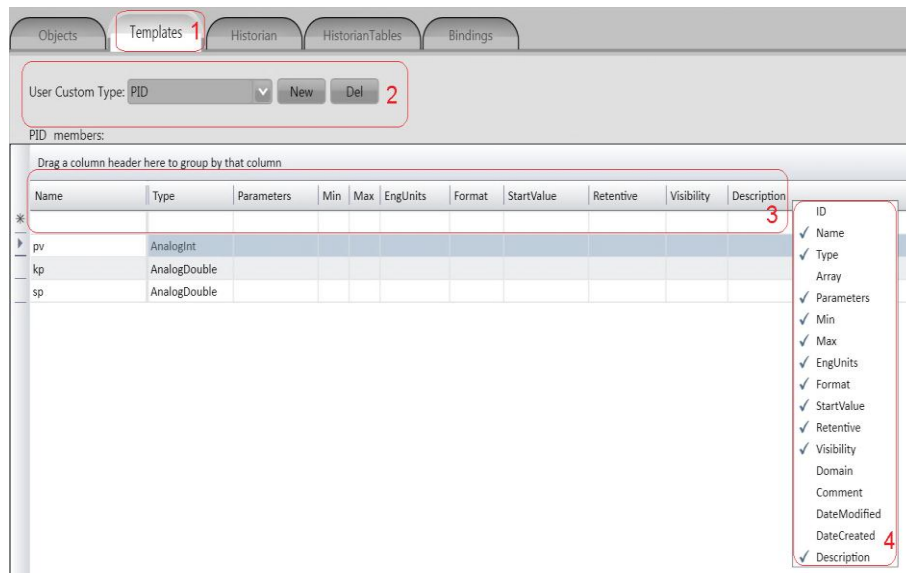


Figure 6-2. Templates Tag Edition

The Edit Type menu items numbered on the previous figure are described below:

1. Select Templates option
2. Select the User Custom Type option and click "New" or "Del" to create a new element or remove it respectively (using the left button of the mouse)
3. Enter the data related to the new type on the row labeled with an asterisk (*)
4. Choose the desired configuration options of the Type column from the table, by clicking with the right button of the mouse in the header of the column and marking the items for exhibition.

As shown on Figure 6-2, each item of the Template Edit Menu is detailed below.

Name

This field sets the element name. In this column is possible to create or edit user types.

Type

This field sets the tag type. The configuration options of the tag component type are:

Type	Description
Digital	True or False
AnalogInt	Integer
AnalogDecimal	Decimal
AnalogDouble	Floating point
Text	Text
Timer	Time
Counter	Count
Reference	Reference object pointing to another reference object
DateTime	Date and time
UserTypes	User defined types

Table 6-5. Base DataType to Generate New Base DataType

Array

When not defined (blank) the tag is not an Array type. Defining an Array with value N, an Array is created from position 0 to N. For example, when creating a tag Array of size 5, the Array is created from tag[0] to tag[5], this means that 6 elements are created.

Lite and Express versions are limited to one dimension arrays. Enterprise version provides arrays up to 3 dimensions.

Reference

This field sets reference type tags. It specifies the target type when a reference tag is created. A reference tag during the Runtime can dynamically point to different tags according to the types defined in this field.

Min/Max

This field sets the minimum and maximum values acceptable for the tag. The range must not be smaller than Min value nor greater than Max value.

Eng Units

This field defines the base Engineering Units for the Tag.

Start Value

This field sets the tag startup value. When left blank no startup value is applied.

Retentive

This field specifies if Tag Properties are retained when shutting down the application and used as startup values on next execution. The available options are:



- Non retentive: Tag Value is not save
- ValueOnly: Tag Value is saved when modified to be used as startup value on next execution

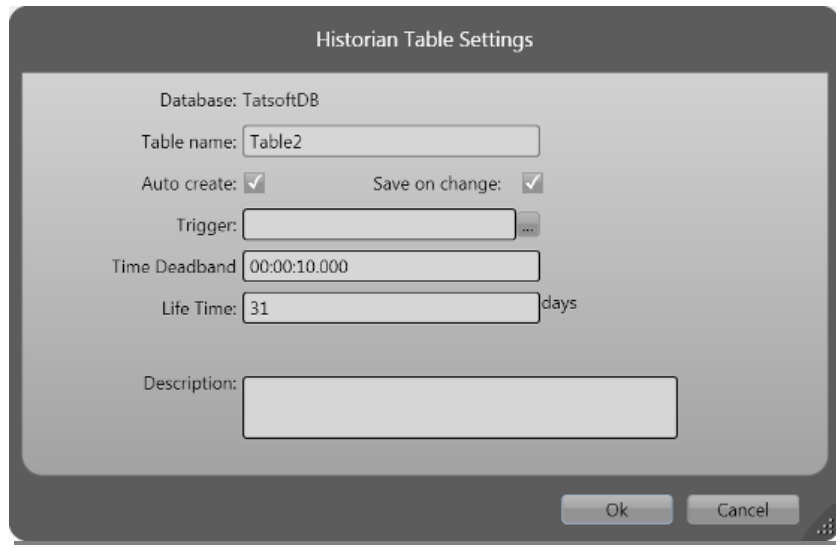
All Tag properties are saved when it is modified to be used on next startup.

When applying retentive properties, the modifications are saved in the database <project>.retentiv. The use of Retentive settings for tags whose values change rapidly is not recommended, once that the execution performance will be decreased. This is the case, for example, of the process variables that are going to be considered critical and/or are associated to security conditions.

Historian Edit

It is possible to configure a Historian database to log Tags changes. To select an existing Historian Table, search in the "Historian Tables" field.

Click  to delete any existing Historian Table or click  to define a new Historian Table. Figure 6-3 shows the Historian Table settings.



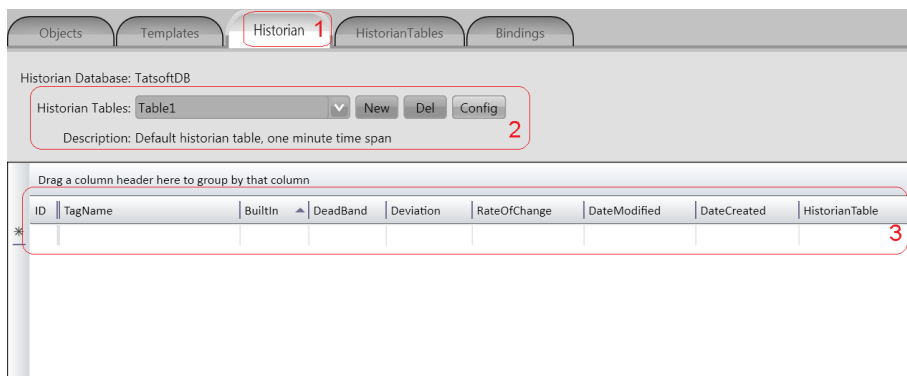
The dialog box titled "Historian Table Settings" is shown. It contains the following fields and controls:

- Database: TatsoftDB
- Table name: Table2
- Auto create: Save on change:
- Trigger: ...
- Time Deadband: 00:00:10.000
- Life Time: 31 days
- Description:
- Buttons: Ok, Cancel

Figure 6-3. Historian Table Settings

The database in which Tags are saved is defined on EditDatasetDBs at DB named "TagHistorian".

Figure 6 4 illustrates the settings which are related to the Tags Historian.



The "Historian Edit" dialog box is shown. It features a tabbed interface with "Historian" selected. The "Historian Tables" section shows "Table1" with "New", "Del", and "Config" buttons. The "Description" field contains "Default historian table, one minute time span". Below is a table with the following columns:

ID	TagName	BuiltIn	DeadBand	Deviation	RateOfChange	DateModified	DateCreated	HistorianTable
*								

Red boxes highlight the "Historian" tab (1), the "New", "Del", and "Config" buttons (2), and the table (3).

Figure 6-4. Historian Edit

The Edit Historian menu items shown on the previous figure are listed below:

1. Select the Historian tab, by clicking with the left button of the mouse
2. On Historian Tables field, click "New" or "Del" to create or remove the table element respectively (using the left button of the mouse). The "Config" button provides access to the historian table configurations shown on Figure 6-3
3. Enter the Tag data whose changes should be registered on the row labeled with an asterisk (*)

As shown on Figure 6-4, each Historian Edit Menu item is detailed below.

TagName

This column indicates the tag name and enables the user to insert or remove historian tags.

DeadBand

This column sets the Historian deadband. The saving event in the log is triggered only if Tag value is changed to a value greater than the one of the DeadBand .

Examples:

Configured DeadBand 10. Current Value 20. All new values 10 (or more) greater than the prior value are added to Historian Log.

Value changed to 35 (difference from previous value of 20:15). Result: Value added to Historian Log.

Value changed to 50 (difference from previous value of 35: 15). Result: Value added to Historian Log.

Value changed to 55 (difference from previous value of 50: 5). Result: Value is not added to Historian Log.

Deviation


If SaveOnChange is set on the table settings and Tag undergoes a greater change than the Deviation parameter (previous value and current > Deviation), the value will be immediately saved (no need to wait for the next TimeSpan).

RateOfChange

If Tag RateOfChange (engineer units per second) is greater than the specified RateOfChange parameter, and SaveOnChange is enabled, the table will be immediately saved (no need to wait for the next TimeSpan).

HistorianTable

The HistorianTable tab defines the Database table where the "Historian" Tags are saved.

Click  to edit Historian Table Settings as shown on Figure 6-3. The available configuration options are:

- TableName: Database table name
- TimeSpan: Minimum time between records
- Trigger: The table is saved at each trigger change
- AutoCreate: Automatically creates a table when this is not found in the database
- LifeTime: When defined (greater than "0") deletes records older than lifetime
- User-defined description for documentation purposes

Binding Tags

It is possible to configure the binding data between tags or properties, so that when the data changes its value, the elements associated with them change automatically. Data binding can also mean that if there is a change in another representation of the data in an element, then the underlying data will also be automatically updated. Figure 6-5 shows this selection.

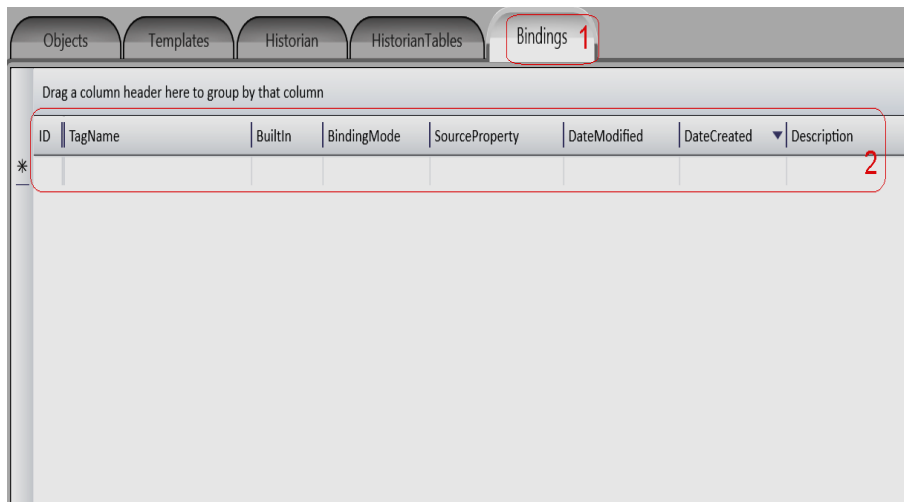


Figure 6-5. Edit Binding Tags

The Edit Binding menu items numbered in the previous figure are listed below:

1. Select the Bindings tab, by clicking with left mouse button
2. Enter the Tag data on the row labeled with an asterisk (*)

As shown on Figure 6-5, each item of the Edit Binding menu is detailed below.

Name

This field specifies the tag name.

BindingMode

This field sets the binding mode with the following configuration options:

- OneTime (at the startup)
- OneWay (tagname changes when sourceproperty change)
- TwoWay (any change in tagname or in the sourceproperty changes the opposite filed)
- OneWayToSource (sourceproperty changes when tagname changes)

Source Property

This field sets the Data source property.

Editing Security Settings

The Edit Security menu defines the user permissions and the project access levels.

Users Edit

This tab defines the Project users and related permissions. The Administrator-level and Guest-level IDs have built-in attributes. Some user permissions remain active regardless of renaming or changing permissions as follows:

Administrator User (ID:2): The Administrator ID remains the same even if the user name or permissions are changed. The Administrator is the only user who can delete or block users and the only user who can define passwords for Database DB interfaces.

Guest User (ID:0): This ID is used for anonymous login users. Guest users do not have passwords assigned to them. However, their permissions can be changed. When the system starts up with no User(s) defined (or there is an anonymous login) the security permissions defined for the Guest user are then applied.

Figure 6-6 shows the users profile edit, regarding the security scope.

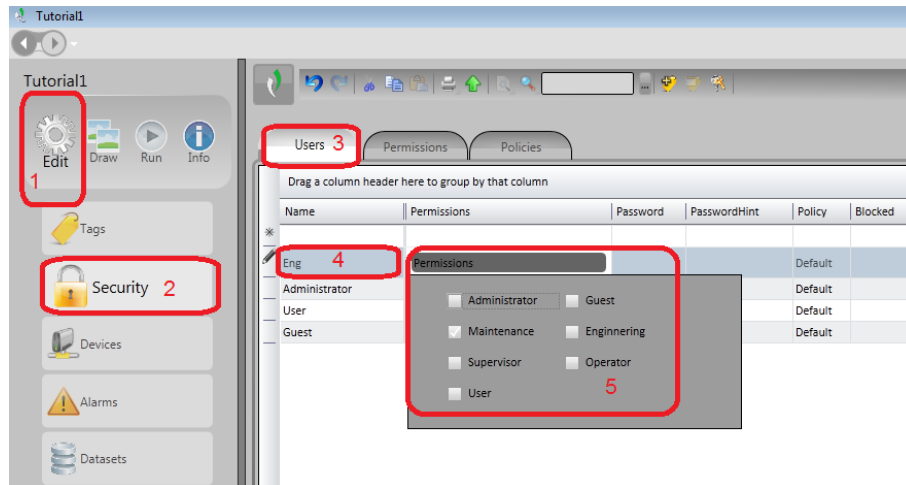


Figure 6-6. Users Profile Edit

The steps for edit/include user profiles are highlighted on Figure 6-6 and described below:

1. Select Edit menu
2. Access Security item
3. Click on Users tab to access the system users
4. Enter the application users data on the row labeled with an asterisk (*)
5. Select the available permissions that the added user or in edition user may have in the system

ATTENTION:

It is possible to set a password and a password tip for each user.

Edit Permissions

This tab defines the settings and execution privileges to each permission group. Figure 6 7 illustrates the selection.

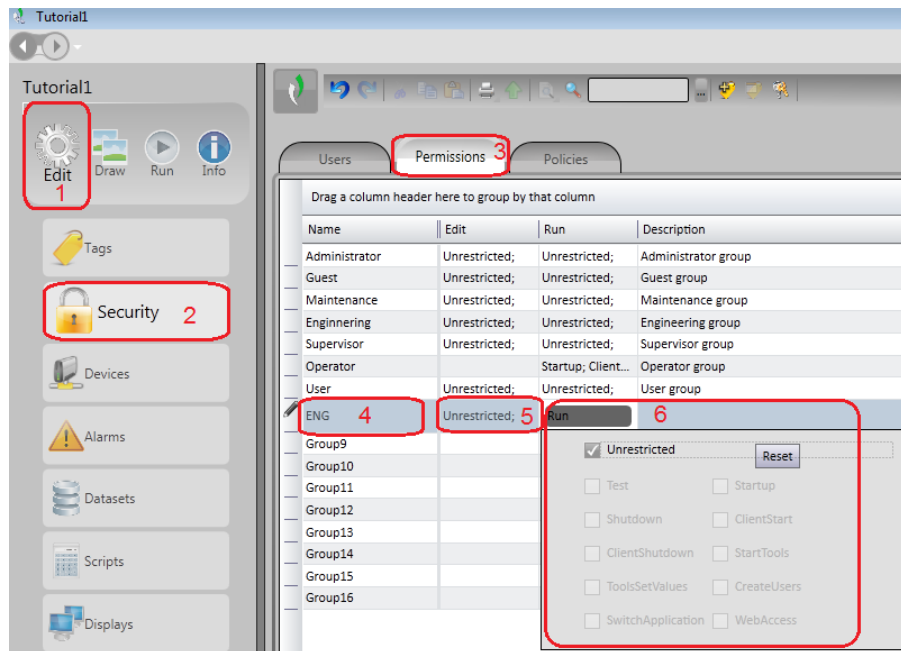


Figure 6-7. Edit of Security Permissions

The highlighted options on Figure 6-7 should be followed in order to edit the permissions and are described as follows:

1. Select Edit menu
2. Access Security item
3. Click on Permissions tab for access to the system permissions
4. Enter the user permissions data on the row labeled with an asterisk (*)
5. Set the Edit permissions on the Edit column
6. Set the Runtime permissions on the Run column

ATTENTION:

The Editing permissions configuration enables users configured with this permission to perform certain types of modifications in the project. The Runtime permissions configurations enable users configured with this permission to perform certain actions during the project execution.

Name

This column sets the permission group name.

Edit

This column defines the Edit and Draw Permission Groups privileges. Available options:

- Unrestricted
- EditTags
- Security
- Scripts
- Datasets
- Reports
- Publish
- Notes
- Historian
- Alarms

- Devices
- Displays
- Startup
- Settings
- CreateTags

Run

This column defines the Permission Groups Execution (Runtime) privileges. Available options:

- Unrestricted
- Test
- Startup
- Shutdown
- ClientStart
- ClientShutdown
- StartTools
- ToolsSetValues
- CreateUsers
- SwitchApplication
- WebAccess

Editing Devices

BluePlant is provided with an OPC DA driver for gathering information from remote devices. In addition to the OPC, BluePlant also supports custom communication drivers to directly access CPs, I/O remote systems, standardized fieldbuses, single and multiple loops, scanners, barcode readers, RFID devices and digital displays. See the Device support manual for more information about Protocol and programming of each device.

ATTENTION:

For more information about device settings (channels, nodes and communication points) use the Help button on the Channels tab.

Channels

This field configures the protocols and communication Channels. Figure 6 8 illustrates the selection.

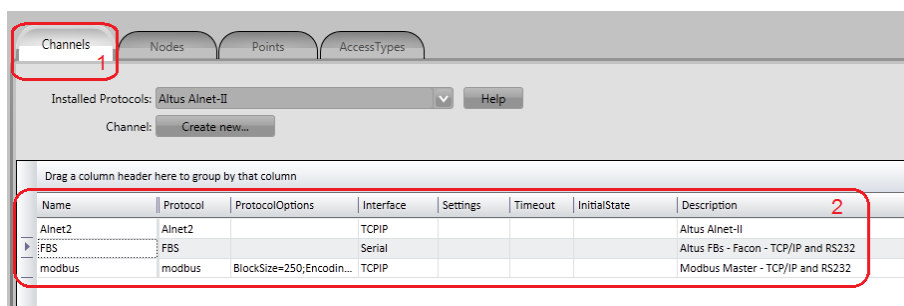


Figure 6-8. Edit Communication Channels

The Edit Communication Channels menu items numbered in the previous figure are listed as follows:

1. Select the Channels option
2. Observe that installed protocols data are indicated on the row labeled with an arrow (►)

As shown on Figure 6-8, each item of the Edit Communication Channel menu is detailed below.

Name

This column sets the channel name.

Protocol

This column sets the communication protocol running on a specific channel.

ProtocolOptions

This column comprises the protocol specific configuration.

Interface

This column indicates the communication interface for the channel. The available interfaces depend on the chosen protocol. Table 6-6 shows the available interfaces for each protocol.

Protocol	Available Interfaces
Altus ALNET I	Serial and Multiserial
Altus ALNET II	TCP/IP
ControlLogix - Rockwell – Protocol CIP	TCP/IP
Altus FBs - Facon	Serial, Multiserial, and TCP/IP
MODBUS Master - TCP/IP e RS-232C	Serial, Multiserial, and TCP/IP
MODBUS Slave - TCP/IP e RS-232C	Serial and TCP/IP
OPCxmIDA - OPC Xml/DA Client	OPC
Siemens –S7 Protocol	TCP/IP

Table 6-6. Protocols and Interfaces

Settings

This column comprises the communication interface configuration settings. Each interface type presents specific settings, such as communication port, speed, data bits, stop bits, parity and control signs. A double click on the Settings cell allows access to the menu with the assigned settings interface.

Timeout

This column defines the timeout settings for the communication interface.

Create a Communication Channel

The  button is used to create a new channel, as indicated on Figure 6.9.

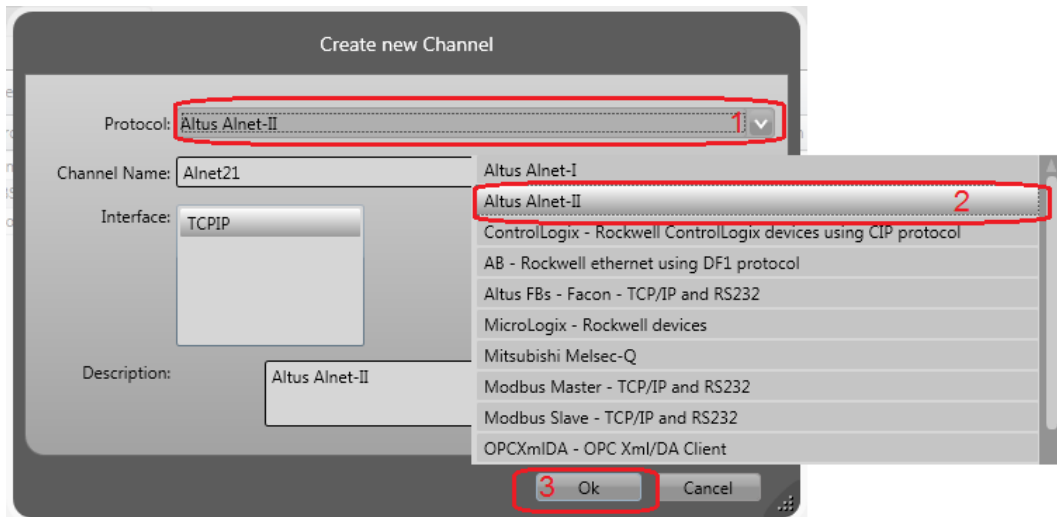


Figure 6-9. Edit of a New Communication Channel

The following items detail the step by step procedure to the creation of a new communication channel. See Figure 6-9.

1. Select Protocol by clicking with the left button of the mouse
2. Select one among the available options. In this example: Altus ALNET II
3. Conclude the procedure by clicking Ok with the left button of the mouse

Nodes

In the computing context, a node is a point or network terminal where a message can be created, received or transmitted. In the context of supervisory systems operating in network, the node can be an active electronic device connected to a network and it is able to send, receive or transmit information via a communication channel. Figure 6-10 shows the Edit nodes menu for a given communication channel. This communication channel, as indicated previously, is associated with a specific Protocol and can contain one or more nodes.

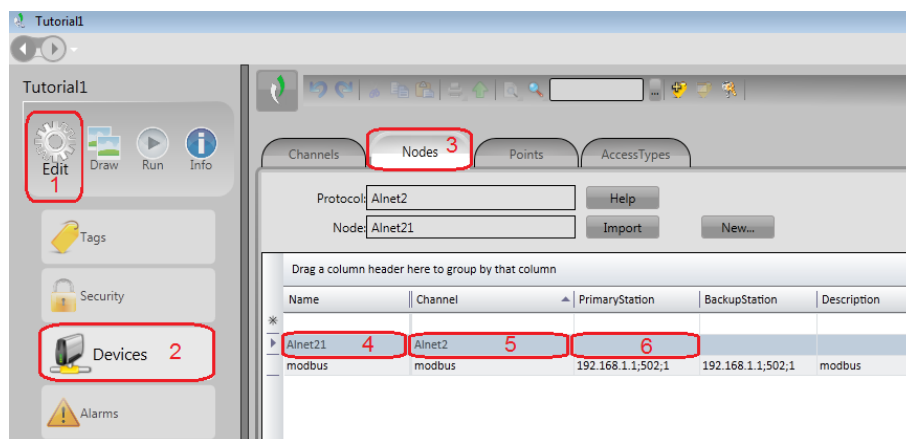


Figure 6-10. Edit Node for Communication Channel

The required steps to edit or include existing nodes are listed on Figure 6-10 and explained below:

1. Select Edit menu
2. Select Devices
3. Click on Nodes to access the node configurations
4. Include or edit the node name on the row labeled with an asterisk (*)

5. Select Channel (protocol) previously added to the project
6. Indicate, in the PrimaryStation column, the equipment address that will be associated to the node

Name

This column indicates the node name.

Channel

This column informs the communication channel associated with this node.

Primary Station

The Primary Station column refers to the node station. It defines the IP Address, Port e SlaveID. The station field syntax is dependent on the protocol. Next figure shows an example of a parameterization (ALNET protocol).

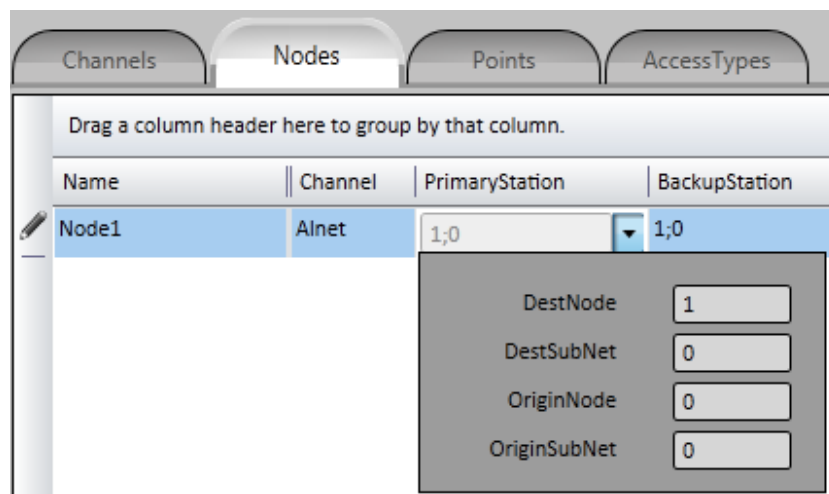


Figure 6-11. Primary Station Parameterization

Backup Station

The Backup Station column defines IP Address, Port number and SlaveID. When defined, and a communication failure occurs on the primary station, the system automatically attempts to establish communication with the backup station.

Points

This tab sets the data acquisition values of the field devices and maps the values in the tags.

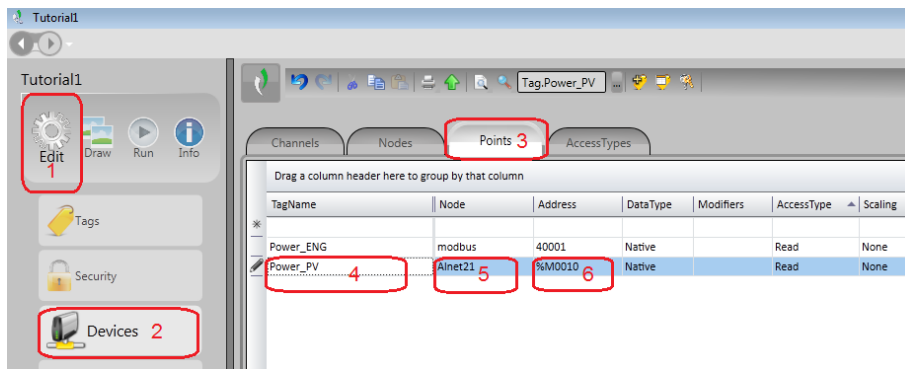


Figure 6-12. Edit Points

The required steps to edit or include communication points are listed on Figure 6-12 and described as follows:

1. Select Edit menu
2. Select Devices item
3. Click on Points tab to access the communication points configurations
4. Include or edit the tag name (selectable) on the row labeled with an asterisk (*)
5. Select the node (configured device) previously added to the project
6. Inform in the Address column the data address of the specified equipment that may be received or sent

Name

The Name column defines the TagName value to be read or written on the identified device.

Nodes

This column defines the Communication Node associated with the Device Point.

Address

This column indicates the Device Point Address. The address field syntax depends on the protocol. Figure 6-13 shows an example of address parameterization (operand) in the ALNET protocol case.

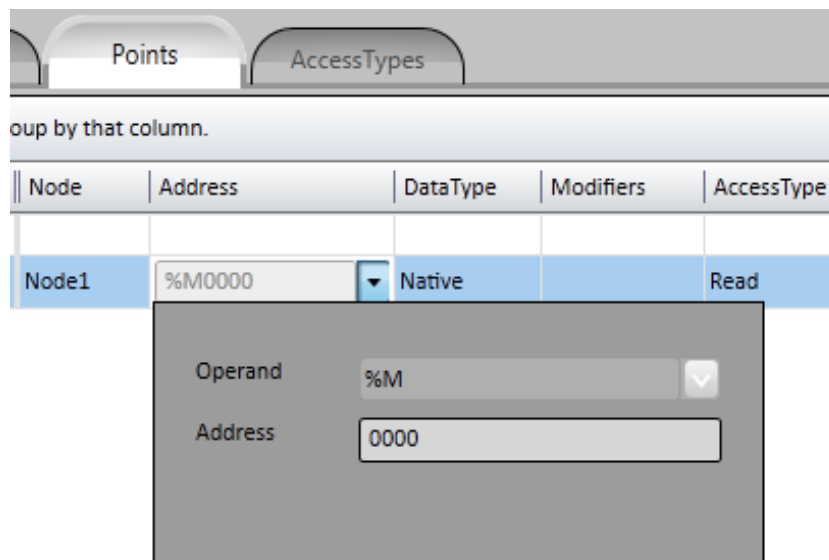


Figure 6-13. Example of Address Parameterization

Date Type

The Data Type column defines the data conversion applied to communication data. Most protocols should use the NATIVE option. When NATIVE is used the protocol will automatically handle the data conversion. If a DataType different than NATIVE is selected, the default protocols are overwritten. The possible DataTypes are:

- Native (automatic)
- Bit (binary – 1 bit)
- Byte (octet – 8 bits)
- Char (smallest addressable unit of the machine that can contain basic characters – 8 bits)
- Short (integer number with signal - 16 bits)
- Dword (integer number – 16 bits)
- Integer (number without decimal point)
- Long (integer number with signal – 32 bits)
- ULong (integer number without signal – 32 bits)
- BCD (decimal number encoded as binary)
- LBCD (Long BCD)
- Single (single floating point number)
- Real (real number – floating point)
- ASCII (7-bit characters code based on English alphabet)
- Unicode (text representation and manipulation)
- OPCDateTime (standard OPC date and time)
- Timer (time code)
- Counter (count code)
- Control (control data type)

Modifiers

The Modifiers column provides bit selection and other communication data settings, changes may happen according to the protocol. The following fields can be defined:

- Bit
- ByteSwap
- WordSwap
- Stringlength

AccessType

The AccessType column defines the Read and Write behavior for each Point.

Scaling

This field sets the scaling conversion applied using the communication data. Scaling conversions settings are:

- None
- TagMinMax
- Liner
- Equation

Figure 6-14 shows the scaling conversion parameterization based on TagMinMax option.

Property	Value
Scaling	TagMinMax
TagMin	0
TagMax	100
DeviceMin	0
DeviceMax	32767

Figure 6-14. Scaling Conversion

Access Type

This tab defines the access type characteristics that are common to the Device Points. Figure 6-15 shows the three types of default access types:

- ReadWrite
- Write
- Read (read only)

Property	AccessType1	ReadWrite	Write	Read
Name	AccessType1	ReadWrite	Write	Read
ReadOnStartup	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ReadPolling	Always	Always	Never	Always
ReadPollingRate	one second	00:00:00.500	00:00:00.500	00:00:00.500
ReadTrigger				
ReadStatus				
ReadCompleted				
WriteEnable	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
WriteEvent	Changed	Changed	Changed	Changed
WriteTrigger				
WriteStatus				
WriteCompleted				
AcceptUnsolicited	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
UseStaticBlocks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BlockCommand				
Description				

Figure 6-15. Access Types

Click  to create a new AccessType.

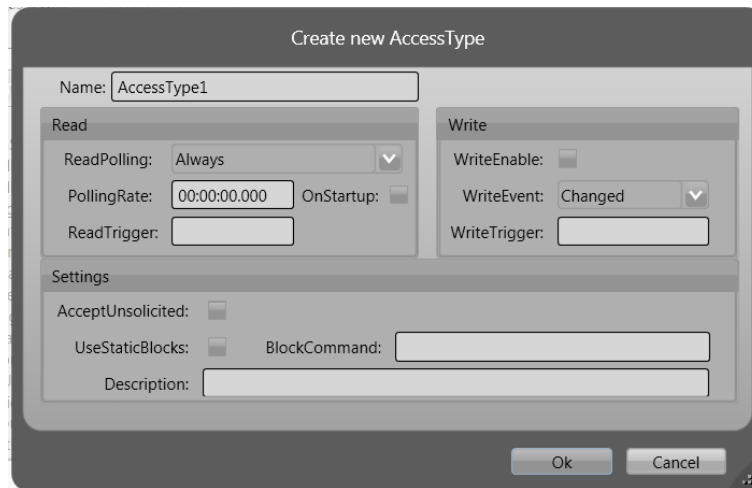


Figure 6-16. Access Type

As shown on Figure 6-16, the main items of the AccessType menu are detailed below.

Name

This field indicates the AccessType name.

ReadPooling

This field defines the read by pooling. Assigned attributes:

- Checked = always = read by pooling on
- Unchecked = never = read by pooling off

PoolingRate

This field defines the read pooling rate for each AccessType if ReadPooling is checked.

OnStartup

This field defines the reading point on startup. Assigned attributes:

- Checked = true = Enables reading on startup
- Unchecked = false = Disables reading on startup

WriteEnable

This field enables or disables the writing when an event occurs. Assigned attributes:

- Checked = true = Enables writing
- Unchecked = false = Disables writing

WriteEvent

This field sets the type of writing access to a point. Assigned attributes:

- Changed - Writes when the linked tag is changed
- ChangedUp - Writes when the value of the linked tag is increased
- ChangedDown - Writes when the value of the linked tag is decreased


AcceptUnsolicited

Defines the acceptance attributes of an unsolicited message. Assigned attributes:

- Checked - Enables unsolicited message
- Unchecked - Disables unsolicited message

Editing Alarms

An alarm can be configured through the association of a Tag with a specific Alarms group and a threshold value.

To display a previously configured Alarm item draw an alarm object on the Draw environment and insert the symbol  (Alarm Window) through the left toolbar.

The methods for Alarms recognition include alarm objects on the screen, Tag properties, alarm groups or alarm properties, as well as the fields "AckAll" (global) and "AckMostPriority".

Notes:

Recognize all alarms: use the property <Alarm.AckAll> that recognizes all the configured alarms in a Project with <Edit.Alarms.Items>.

Recognize single alarm or high-priority alarm: the property {Alarm.PriorityItem.UnAck} allows the recognition of the high priority alarm configured on <Edit.Alarms.Items> in the "Priority" column. If this is the only alarm or if this is a high priority one, it will be recognized, as long as it is Active or Normalized.

Recognize specific alarm: use the property <Alarm.Items.IDxx.Unack> to recognize a specific alarm.

Alarm Groups

The Alarm Groups define the alarm handling behavior which is common to a Group of Alarms.

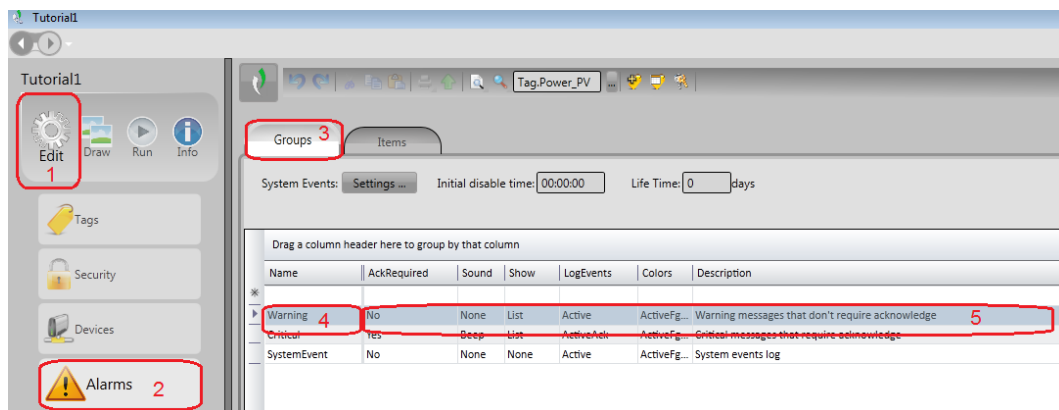


Figure 6-17. Alarm Groups

Figure 6-17 shows the required steps to edit or include the alarm groups. The following procedure should be performed.

1. Select Edit menu
2. Select Alarms
3. Click on Groups to access the alarm groups configurations
4. Include or edit the alarm group name on the row marked with an asterisk (*)
5. Select the group configuration specified on the columns presented on the figure above

Name

The field indicates the name defined by the user for the Alarm group. The Alarm Groups "Warning", "Critical" and "SystemEvents" are built-in and cannot be deleted, however their names and settings can be changed.

AckRequired

The field sets the acknowledge options for the Alarm group. Assigned attributes:

- No = 0 - The alarm points set in the group do not require acknowledgement
- Yes = 1 - The alarms set in the group require acknowledgement

Sound

The field enables or disables the Alarm Sounds when there are active Alarms in Group. Assigned attributes:

- None = 0: No sound
- Beep = 1: A regular beep will be played at each client computer while there are alarms without acknowledgement

Show

The field configures to display/not display the Alarm (includes display on the online Alarms object).

Log Events

The field defines the type of Historian archiving on Alarm events. Assigned attributes:

- None
- Active (log when the event is active)
- ActiveAck (log when the event is active and recognized)
- ActiveNorm (log when the event returns to the normal state)
- All (log in all conditions above)

Note:

The database in which the Tags are saved is defined on EditDatasetDBs at DB named "AlarmHistorian".

Colors

The field defines the Alarm display customization for each Alarm line according to its main Group.

Alarm Items

The field configures Tags to generate Alarms under defined conditions.

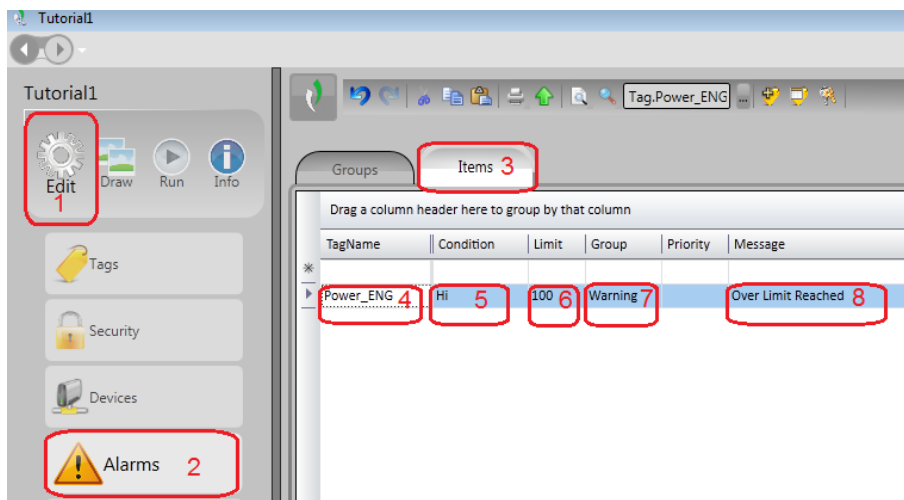


Figure 6-18. Alarm Items

Figure 6 18. shows the required steps to edit or include the alarm groups. The following procedure should be performed:

1. Select Edit menu
2. Select Alarms
3. Click on Items to access the alarms configurations
4. Include or edit the tag name which generates the alarm on the row labeled with an asterisk (*)
5. Select the alarm generation condition
6. Set the limit which triggers the alarm
7. Select the previously created group with its alarm features
8. Insert the message to be displayed in case of alarm trigger

Name

This column defines the TagName which will be evaluated to generate the Alarm.

Condition

This column refers to the evaluation condition to generate alarms. Options:

- Hi: Tag \geq limit
- HiHi: Tag \geq limit (when acknowledged automatically, acknowledges Hi alarm to the same Tag)
- Lo: Tag \leq limit
- LoLo: Tag \leq limit (when acknowledged automatically, acknowledges Lo alarm to the same Tag)
- RateOfChange: Tag rate of change \geq limit
- DeviationMinor: Absolute value (Tag - Setpoint) $>$ limit (Setpoint defined on the Setpoint column)
- DeviationMajor: Absolute value (Tag - Setpoint) $>$ limit (Setpoint defined on the Setpoint column)
- Equal: Tag = limit
- GreaterThan: Tag $>$ limit
- GreaterEqual: Tag \geq limit
- LessThan: Tag $<$ limit
- LessEqual: Tag \leq limit
- Changed: Tag value changed
- ChangedUp: Tag value increased
- ChangedDown: Tag value decreased

Limit

This column defines the Value to evaluate the Alarm conditions.

Group

This column defines the Group name on EditAlarmGroups where the behavior of the Alarm item is specified. The pre-defined groups are:

- Critical (Critical Messages that require acknowledgment)
- SystemEvent (System Events log)
- Warning (Warning messages that do not require acknowledgment)

Priority

This column defines the Alarm priority. "0" is the highest priority.

Message


This column defines a Message to display when Alarm occurs. Message fields can contain Tag values evaluated when generating the Alarm. Add the TagName in the message using the XAML binding notation between curly brackets as illustrated below:

```
TagValue = {TagName.Value}.
```

Editing Databases

As mentioned previously, the Datasets module included in BluePlant provides an easy operation interface for real-time data exchange with external databases, XML, CSV or text files, as well as the possibility to access tables and SQL queries. The real-time database ensures data synchronization across multiple processes on the server as well as multiple client stations, without the need of additional programming. A wide range of internal properties, such as data quality, time stamp, block state and blocked value simplify the applications creation.

DBs

The DBs tab allows to access the Database configurations. Click  to select the database provider and create a new connection.

Name

This field sets the name for the DB Object.

Provider

This field selects the provider to the DB when creating a new database connection. The default Providers are:

- Odbc Data Provider - Allows the access to an ODBC database through a native ODBC driver
- OleDb DataProvider - Provides applications to access the stored data in several information sources
- SqlClient Data Provider - It is a collection of classes that can be used to access SQL Server databases
- Microsoft SQL Server Compact Data Provider - Provides access to the Microsoft SQL Server Compact databases
- TatsoftDB 4 direct connection – Allows the access to the TatsoftDB 4 database

Additional Providers can be created by adding XML templates to the product in the "DBProviders" folder, that is located in the installation folder of the BluePlant software. Examples of data providers:


- OdbcDataProvider
- Firebird database
- Microsoft Access Database
- Microsoft Excel Database
- ODBC using DSN
- ODBC using fileDSN
- Oracle Database
- SQL Server Database

Database

This field selects the Database when creating the DB object. The list of available databases is dynamically created based on Providers found in the "DBProvider" subfolder which is located at the installation folder of BluePlant software. The most common databases:

- Microsoft Access Database
- Microsoft Excel Database
- ODBC DSN
- ODBC FILEDSN
- SQL Server Database

ConnectionString

This field defines the Connection string used to connect with database. Type the database source path where the database file is located. Click  to test if the Date Source is located and valid.

NameLogon

This field defines the Logon Name to connect with the database.

Password

This field refers to the required Password to connect with the database. This field can be edited only by the Administrator user (user ID: 2).

Excel Connection

This field connects to Excel databases using an ODBC, an ODBC DSN, or OleDb driver. For an ODBC connection, the following steps must be performed:

1. Select and name a required area in the spreadsheet. This allows BluePlant to read the information as a table
2. Choose one of the following naming processes for your version of Microsoft Excel

For Microsoft Office 2007: Right-click the selection and then choose "Name a Range". Figure 6-19 illustrates this option.

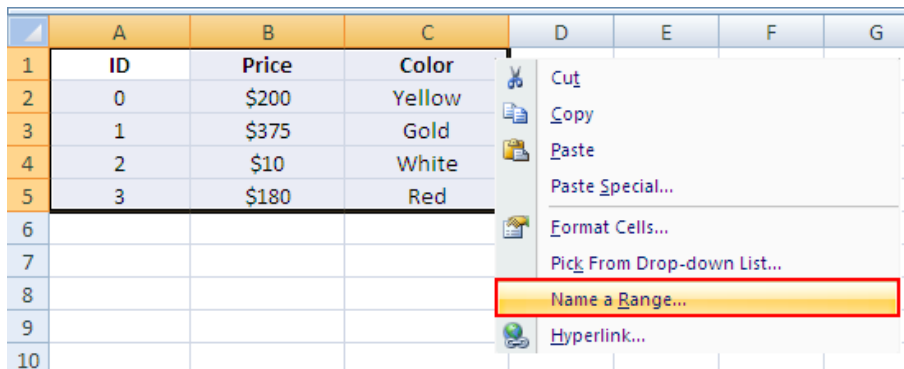


Figure 6-19. Connection with Excel 2007

For Microsoft Office 2003: In Microsoft Excel go to "Insert > Name > Define ...". Figure 6-20 illustrates this option.

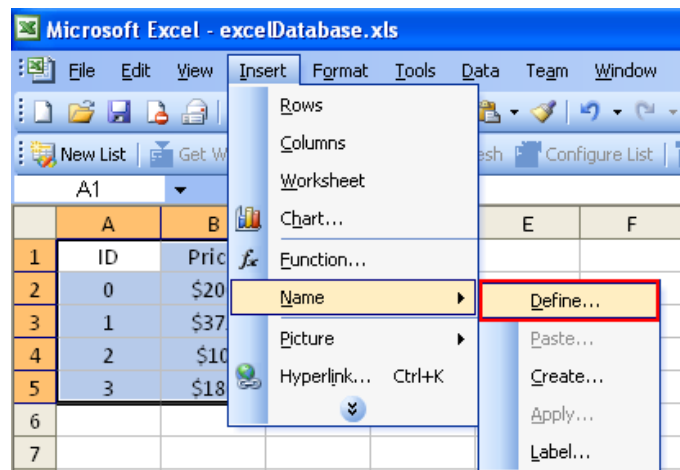


Figure 6-20. Connection with Excel 2003

Name the selection (e.g., "itemsTable"). The Excel file is now ready for use.

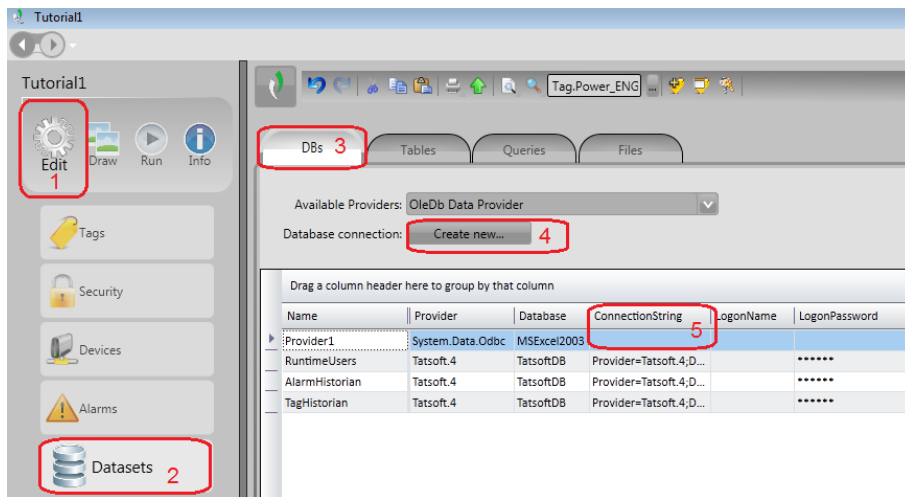


Figure 6-21. Configuration of the Connection with Provider Excel 2003

The required steps to create a new connection with the database are numbered on Figure 6-21 and listed below:

1. Select Edit menu
2. Select Datasets
3. Click DBs to access the connections created with database
4. Select Create new to create a new connection with Excel 2003
5. The Connectionstring is the place to configure the connection and test with the database. Figure 6-23 illustrates the procedure

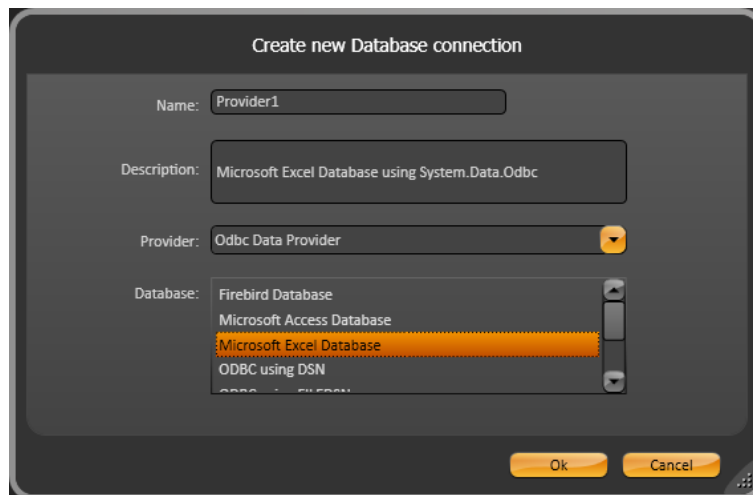


Figure 6-22. Create a New Connection with Database

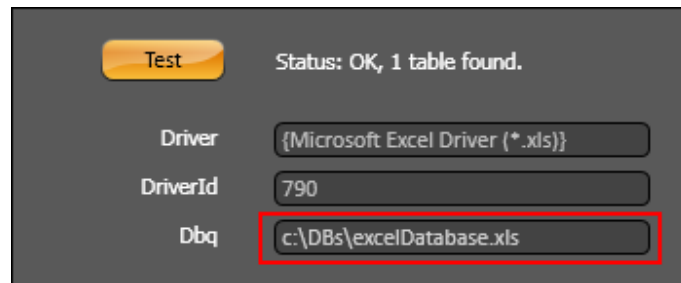


Figure 6-23. Connection with Database Test

If the database is an ODBC DSN one, proceed as follows:

In the computer control panel select "Administrative Tools" and double click "Data Sources (ODBC)", as shown on Figure 6-24. If the operational system is 64 bits Windows, the access is via C:\Windows\SysWOW64\odbcad32.exe file.

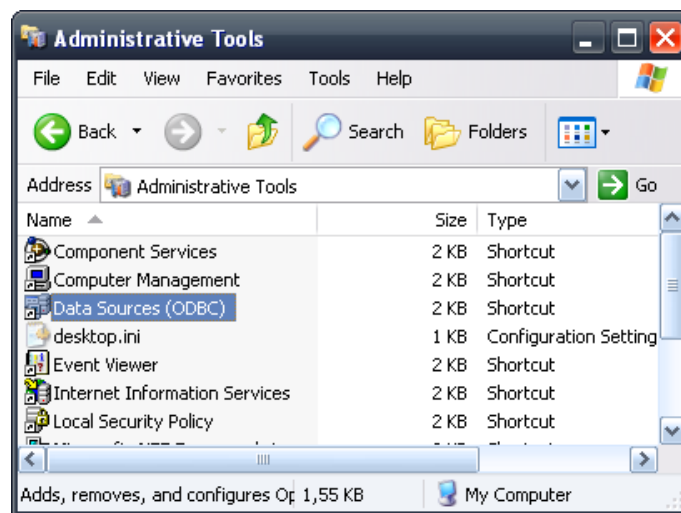


Figure 6-24. ODBC with DSN

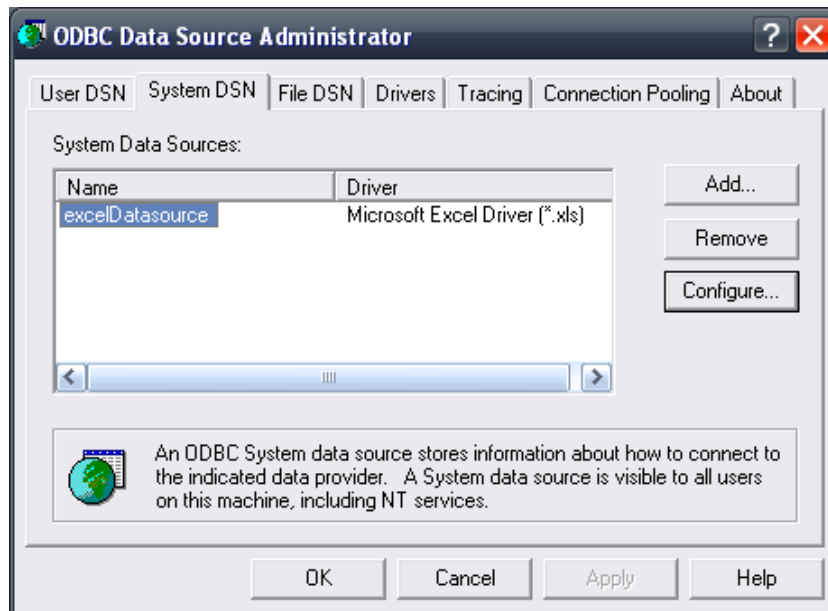


Figure 6-25. "ODBC Data Source Administrator"

In the "ODBC Data Source Administrator" window click "Add". You are prompted to select a driver. Select the "Microsoft Excel Driver (*.xls)" and Figure 6-26 will show up.

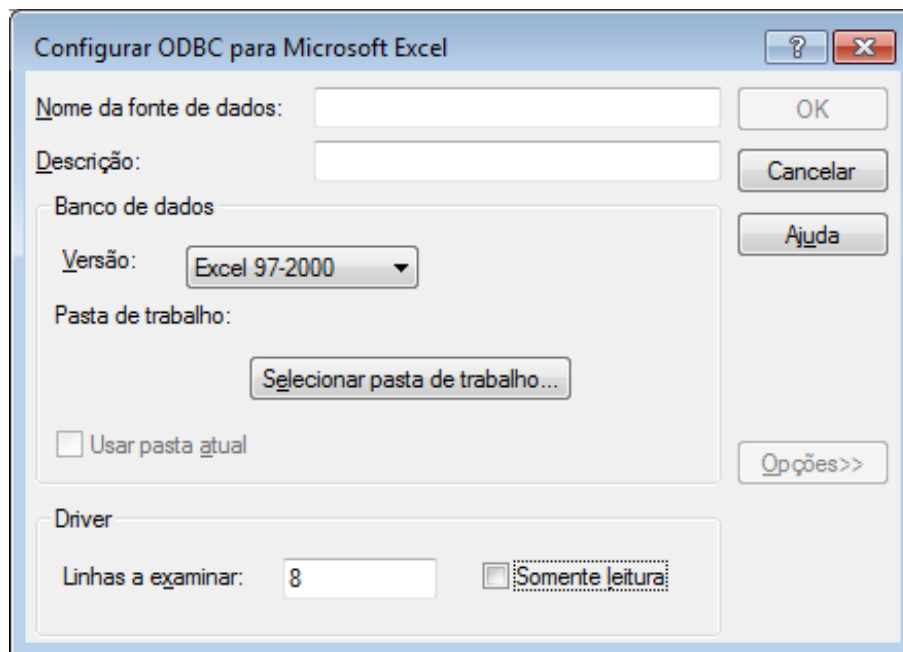


Figure 6-26. Database Configuration

Click "Select Workbook" then select the name of the Excel file created previously (e.g., "excelDatasource"). For Write access uncheck the "ReadOnly" checkbox.

In the Datasets namespace, choose "DBs" tab and create a new Provider on the correspondent option. Under "Odbc Data Provider" options, choose "ODBC using DSN" then click Ok. On the ConnectionString column of the new row, enter with the DSN in the "DSN" field and click in the button "Test" to verify the connection with Excel.

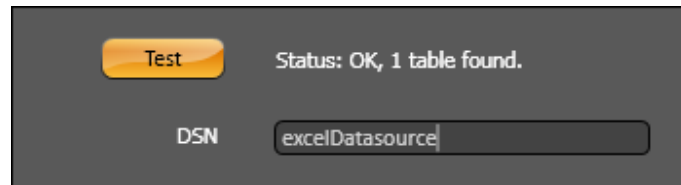


Figure 6-27. ODBC Database Connection Test

To connect the OLEDB provider with Excel, the steps below must be followed:

In BluePlant Datasets namespace, choose "DBs" tab and select the option "OleDb data provider". Then create a new Provider by clicking the corresponding option.

Choose "Microsoft Excel Database" and then click Ok. Click the ConnectionString column of the new inserted Provider, then enter the path and the name of the Excel (.xls) file in the "DataSource" field as shown on Figure 6-28.

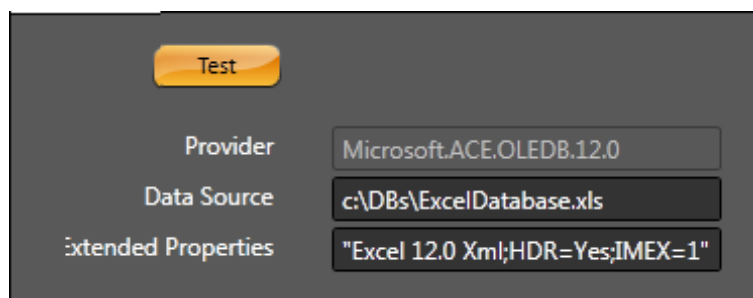



Figure 6-28. OLEDB Database Connection Test

Connection with Oracle Database

The Oracle.DataAccess.Client provider used on the example of the Connection with Oracle Database is available for free download at:

<http://www.oracle.com/technology/software/tech/windows/odpnet/index.html>

To create a new connection with the Oracle database click on  and select the installed provider as shown on Figure 6-29.

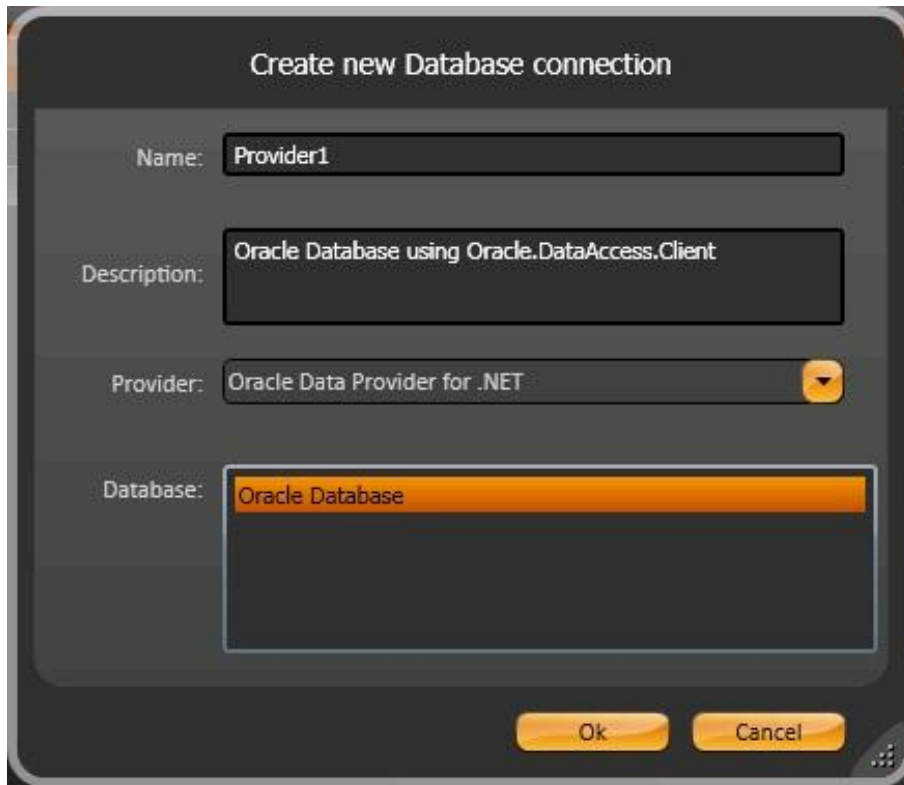


Figure 6-29. Insert Oracle Provider

Enter the user and database password to assure the connection. In order to do that use the LogonName and Logon Password which were previously registered in the database.



Figure 6-30. User and Password to Connection with Oracle Database

In ConnectionString, enter the IP address, port and bank SID to establish the connection, as shown on Figure 6-31.

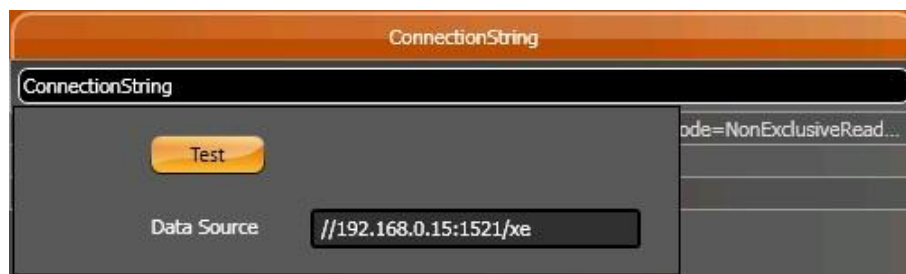


Figure 6-31. Oracle Database IP Address, Port and SID

Check if the connection with the Oracle Database is working by clicking on "Test" and a similar result to the one of the Figure 6-32 must appear.



Figure 6-32. Connection with Oracle Database Test

ATTENTION:

The provider used on the example above, requires Oracle clients (version 9.2 or previous). Oracle Provider that comes with BluePlant is the System.Data.OracleClient, and requires Oracle clients (version 8.1.7 or later).

Connection with SQLServer Database

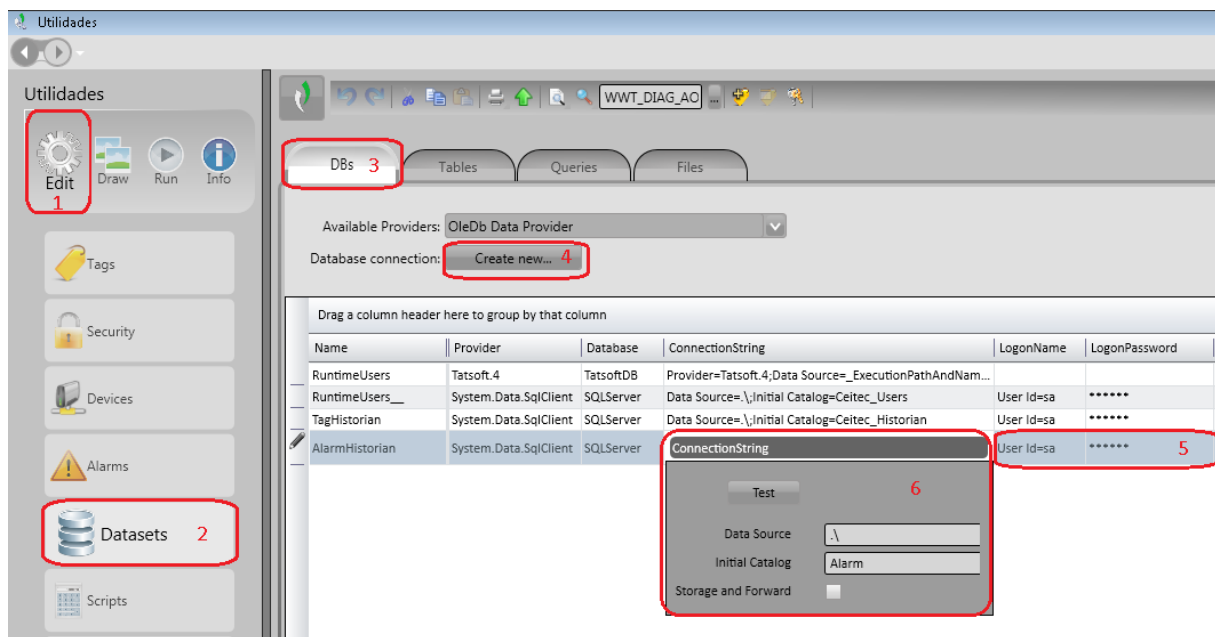


Figure 6-33. Insert SQLServer Provider

The required steps to create a new connection with a SQLServer database are numbered on Figure 6-33 and listed below:

1. Select Edit menu
2. Select Datasets
3. Click DBs to access the connections created with database
4. Click Create new... to create a new connection with SQLServer Database
5. Configure the database user and password in the columns LogonName and LogonPassword
6. The Connectionstring is the place to configure the connection and test with the SQLServer database

Tables

This tab allows to access data tables from configured Databases (DBs). Figure 6 27 illustrates this selection.

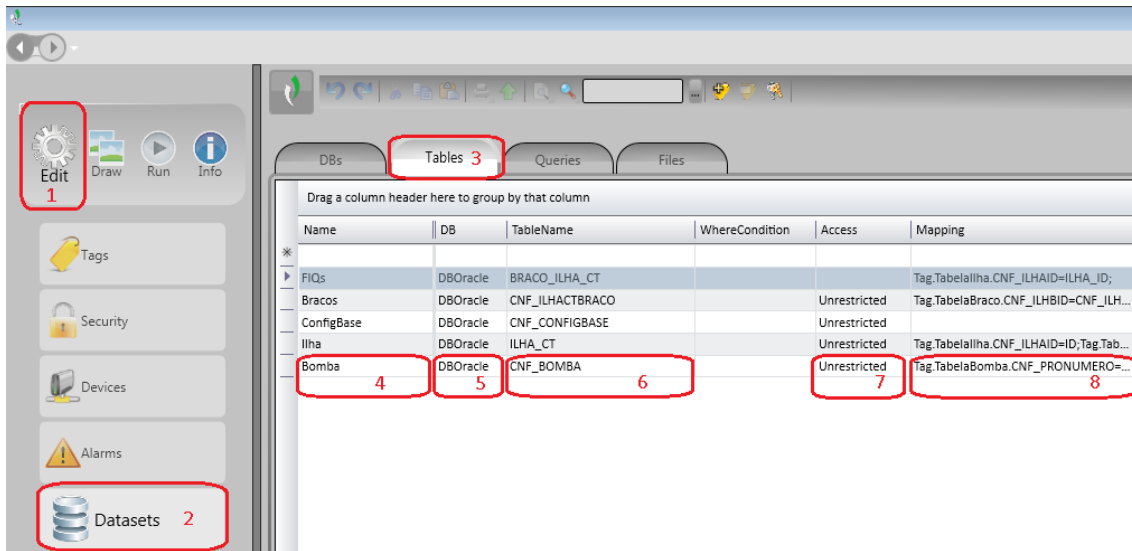


Figure 6-34. Edit Data Tables

The required steps to edit or include tables are shown on Figure 6-34 and described below:

1. Select Edit menu
2. Select Datasets
3. Click Tables tab to access the tables configurations
4. Type or edit the table name on the row labeled with an asterisk (*)
5. Select which connection with a previously created database will be used
6. Select which database table will communicate
7. Select which is the access permission to the database
8. On column Mapping configure the tags related with the database table columns

Name

This column indicates the Table name used on Runtime objects.

DB

This column indicates the name of DB (Database connection) where the DataTable belongs.

TableName

This column indicates the table name as it is in the database.

WhereCondition

The WhereCondition indicates the condition in which the SQL statement is executed.

Access

The Access defines the Security policy with allowed operations for this DataTable. Possible options:

- Read
- Insert
- ReadWrite
- Unrestricted

Mapping

This column maps the contents of the data table with Tag objects. When executing Select and Next commands the first row in the selected result is applied to the Tags. Then, when executing an Update command the contents of the Tag are written to the row.

Queries

This tab maps the configured databases (DBs) as shown on Figure 6-35.

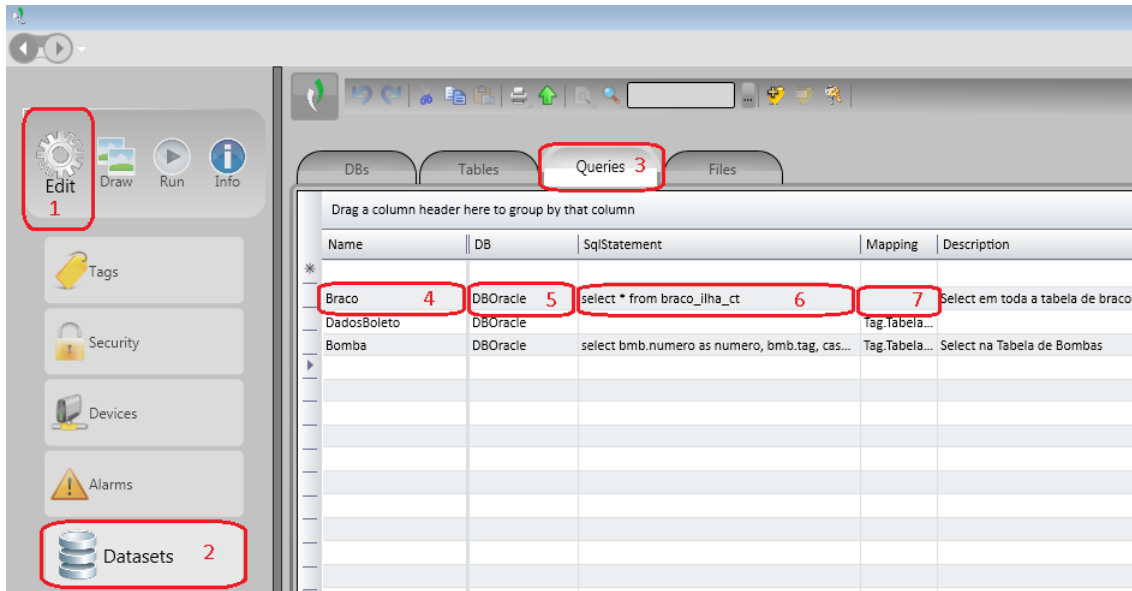


Figure 6-35. Edit Queries

The required steps to edit or include SQL(queries) instructions shown on Figure 6-35 are listed below:

1. Select Edit menu
2. Select Datasets
3. Click Queries to access configuration of the SQL instructions
4. Enter or edit the name of the instruction on the row labeled with an asterisk (*)
5. Select which connection with the previously created database will be used
6. Configure the SQL instruction on the SqlStatement column
7. On column Mapping configure the tags related with the database table columns

Name

This column refers to the name of the Query object used on Runtime objects.

DB

This column refers to the name of the Database connection (DB) used for the storage of queries.

SQLStatement

This column refers to the SQL statement used to consult database.

Mapping

This column maps the contents of the data table in Tags. When executing Select command, the first row in the result is applied to the Tags.

Files

This tab defines the files to the information exchange with the databases, as shown on Figure 6-36.

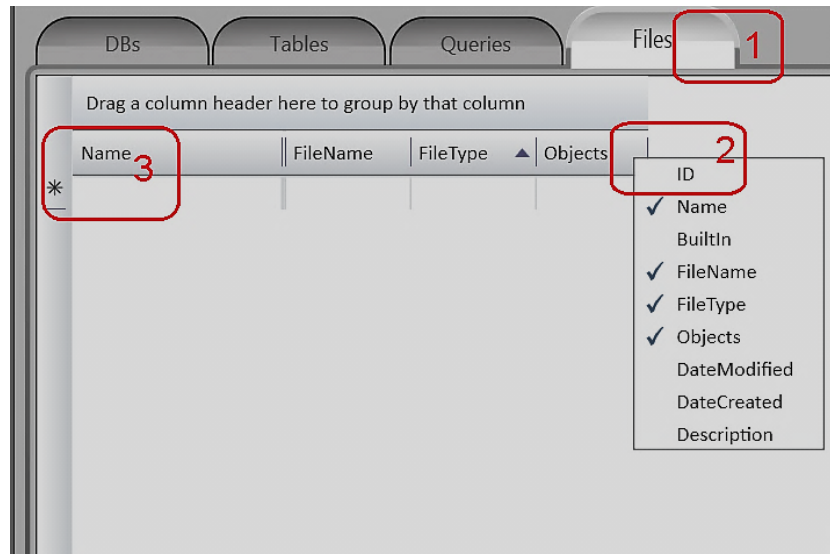


Figure 6-36. Database File Editing

The items that comprise the Database Edit menu are numbered in the preceding figure and are described below:

1. Select the Files tab, then click with the left button of the mouse
2. Right-click on the column header to view the available items
3. The data related to the files will show up on the table rows

The edit items of the database files are described below, according to the illustration on Figure 6-36.

Name

This column defines the file name used on Runtime objects.

FileName

This column refers to the file name and path. Example: C:\BDs\file1.txt.

FileType

This column features the file types. Possible options:

- ASCII (7-bit characters code based on English alphabet)
- Unicode (text representation and manipulation)
- XML (standard OPC date and time)

Objects

This column maps the contents of the File with Tag objects.

Editing Scripts

The Script is executed either when the Trigger event occurs, or cyclically, on every interval set on the Interval column. The options that comprise the Edit Scripts menu are detailed below:

Tasks

The Figure 6-37 emphasizes the Tasks tab for edition of script tasks. There are four built-in tasks:

- ServerStartup – script executed when the Project starts running. Runs on the Server machine (TServer.exe)
- ServerShutdown - script executed when running shutdown. Runs on the Server machine
- ClientStartup - script executed on each client machine when the TVisualizer.exe (Displays module) starts running
- ClientShutdown - script executed on each client machine when the Display module is closed

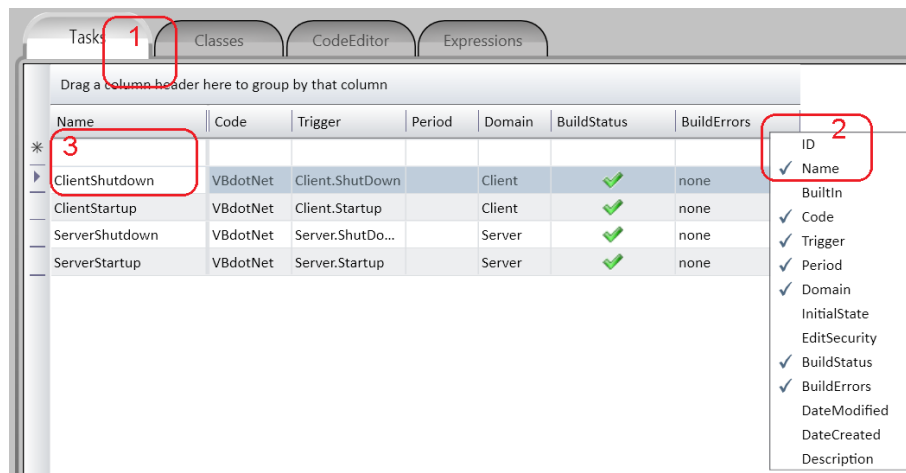


Figure 6-37. Edit Script Tasks

The items that comprise the Script Tasks Edit menu are numbered in the preceding figure and are described below:

1. Select the Tasks tab
2. Select the configuration options of the desired columns from the table, right-clicking on the column header and marking the desired items for exhibition
3. The script task data will show up in the table rows

The items of the script task edit menu are described below, according to the illustration on Figure 6-37.

Name

This field indicates the name for the Script object.

Code

This field defines the Script language. The user can choose VBdotNet or CSharp.

Trigger

This field refers to the Tag or Object that triggers the Task execution. The task is executed when the value of the object changes.

Period

This field represents the period of time to execute a Task.

Domain

This field defines if the Script is executed on the Server station or on each Client station.

BuildStatus

This field shows the last compilation status: green indicates a successful compilation; red indicates an error.

BuildErrors

This field shows the error count status from the last Script compilation. Assigned attributes: ReadOnly.

Classes

This tab lists the User-defined classes via Methods library. Figure 6-38 shows it on the Script menu. There are two built-in UserClasses:

- ServerMain: methods library available for all server and client tasks
- ClientMain: methods library available for all client scripts, including displays scripts

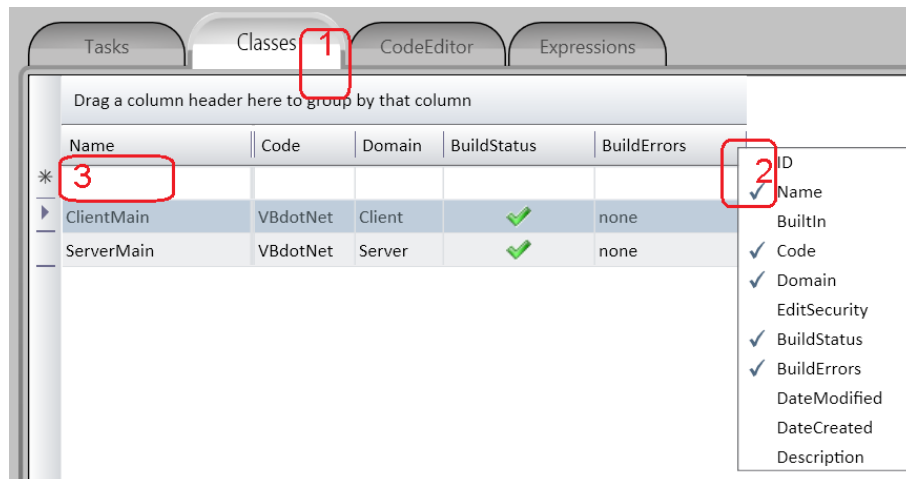


Figure 6-38. Script Classes

The items that comprise the Script Classes Edit menu are numbered in the preceding figure and are described below:

1. Select the Classes tab
2. Select the configuration options of the desired columns from the table, right-clicking on the column header and marking the items for exhibition
3. The data related to the script classes will show up in the table rows

The items of the script classes edit menu are described below, according to the illustration on Figure 6-38.

Name

This column indicates the name of the user class.

Code

This column indicates the Script language. The user can choose VBdotNet or CSharp.

Domain

This column defines if the methods in this user class will be visible to Server scripts or Client scripts.

CodeEditor

Figure 6-39 shows the panel of the task code and user class editor (Item 1). Available programming languages: VBdotNet our CSharp. Figure 6-39 (Item 2) also shows the language selection. Information on the syntax of the programming languages can be found at <http://www.microsoft.com/net>.



Figure 6-39. CodeEditor

Using Script Tasks

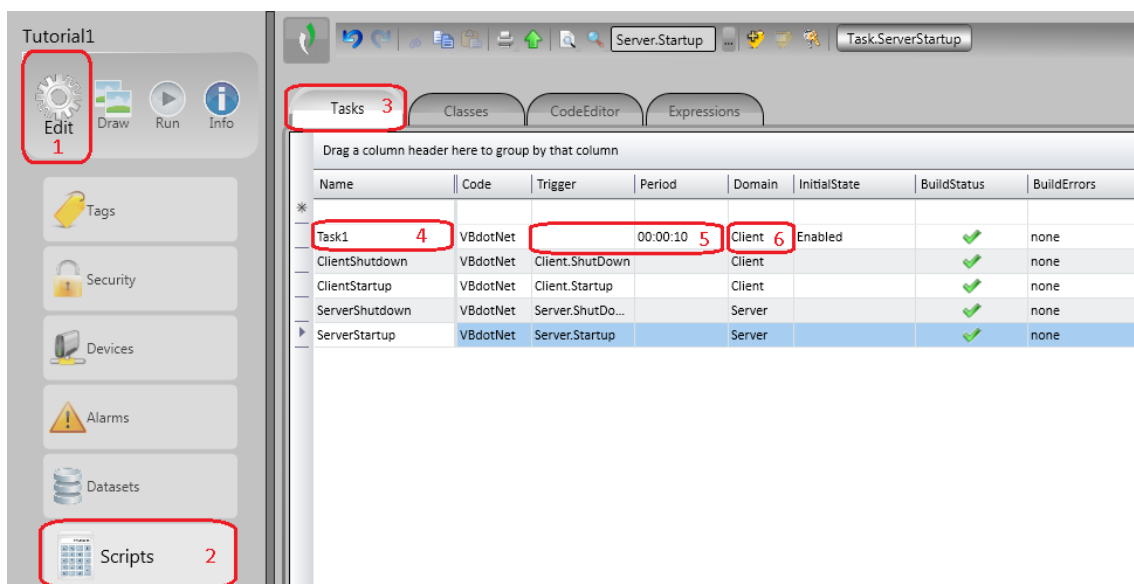


Figure 6-40. Script Tasks

The required steps to edit or include a script task shown on Figure 6-40 are described below:

1. Select Edit menu
2. Access the Scripts
3. Click on Tasks to access the script tasks configured on the system
4. Enter or edit the task name on the row labeled with an asterisk (*)
5. Configure the trigger variable or period in which the script task will be executed, through the columns Trigger or Period
6. Select the environment to execute the task (client or server)

Figure 6-41 presents the required steps to encode the script task using the method of an existing class.

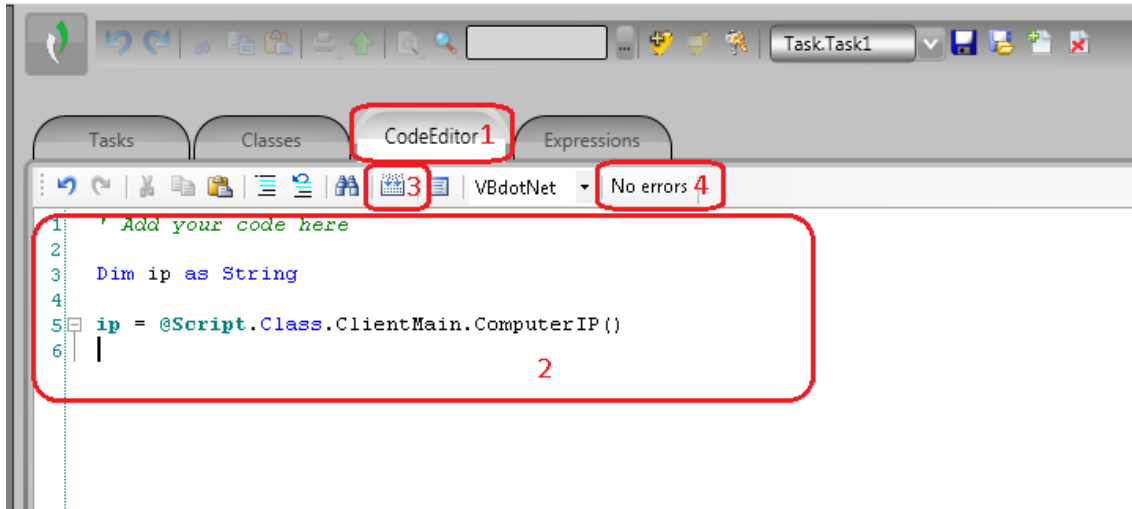


Figure 6-41. Script Tasks CodeEditor

1. Click on the CodeEditor tab to access the encoding environment
2. Type the code in the proper place
3. Click the compilation icon
4. Check if there are errors on the encoded script

ATTENTION:

To use namespaces in the coding environment is required to type the character @ before the namespace, otherwise it will be used as a local variable generating possible errors in the compilation.

Editing Displays

The Displays tab in the Edit menu allows the configuration of the module associated with the displays. This module includes the repository of Displays, their modes of operation (Layouts) and images (Resources).

Displays

This tab lists the project displays. The visualization of the display list can either be on table or card format. The Card View/Table View selector allows to switch the modes. Figure 6-42 illustrates the environment on table format.

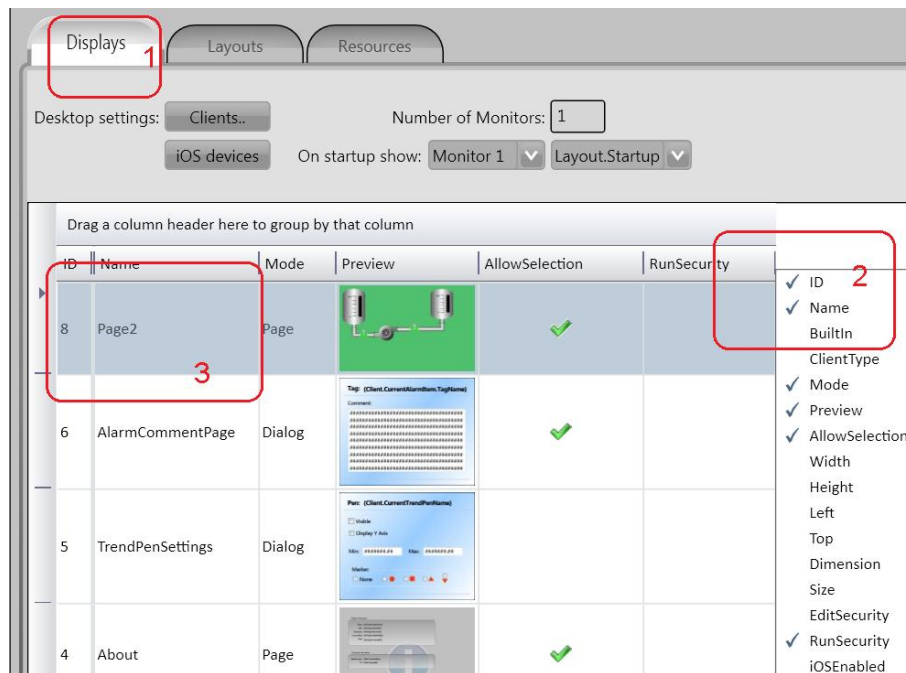


Figure 6-42. Edit Displays

The items that comprise the Displays Edit menu are numbered in the preceding figure and are described below:

1. Select the Displays tab
2. Right-click on any column header to select the configuration options that will be displayed
3. Click on the display rows to change the configurations

Name

This field indicates the display name.

Mode

This field refers to the display operation mode (Page, Dialog or Popup). They are detailed below:

- Page: This is the default Display mode. When a Page is opened automatically, it closes the last page on the current layout
- Dialog: A Dialog display opens as a "Window Modal Dialog". This means that it disables the commands on all other open displays. When clicking OK on a dialog, the OnOK method on display CodeBehind is executed
- Popup: A popup display opens on the top of all other displays. When a new Page is opened, all Popup displays are closed by default

Preview

This field displays the image Preview.

AllowSelection

This field refers to a flag indicating if the display is listed on the DisplaySelection object during runtime. AllowSelection is a function used by the application/project creator to remove the page selection, test and other displays from the default operator and assure that the removed items are available only for specific application contexts as well. The built-in method for setting display selection when running the application is the "PageSelector" function which is set in Edit>Displays>Displays tab. To block this selection from the default operator remove the check from the page or from the "AllowSelection" option on objects column.

RunSecurity

This field refers to the security permissions groups required to open this display during Runtime. The user must click on the "Run Security" field to open the "Run Security Selection" window and set the desired options. A click anywhere in the table will close the selection window. The settings will be displayed in the "RunSecurity" field of the table row.

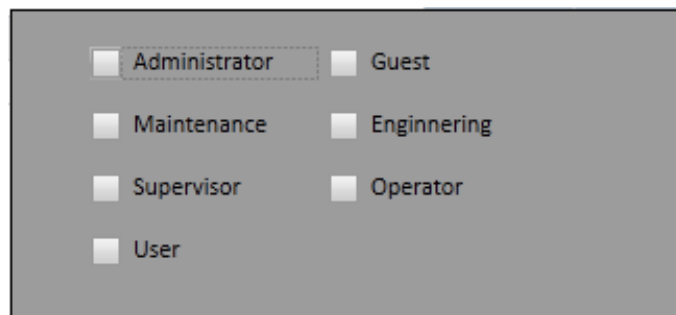


Figure 6-43. Runtime Permissions

Configuring Multiple Monitors

The required steps to configure multiple monitors shown on Figure 6-44 are described below:

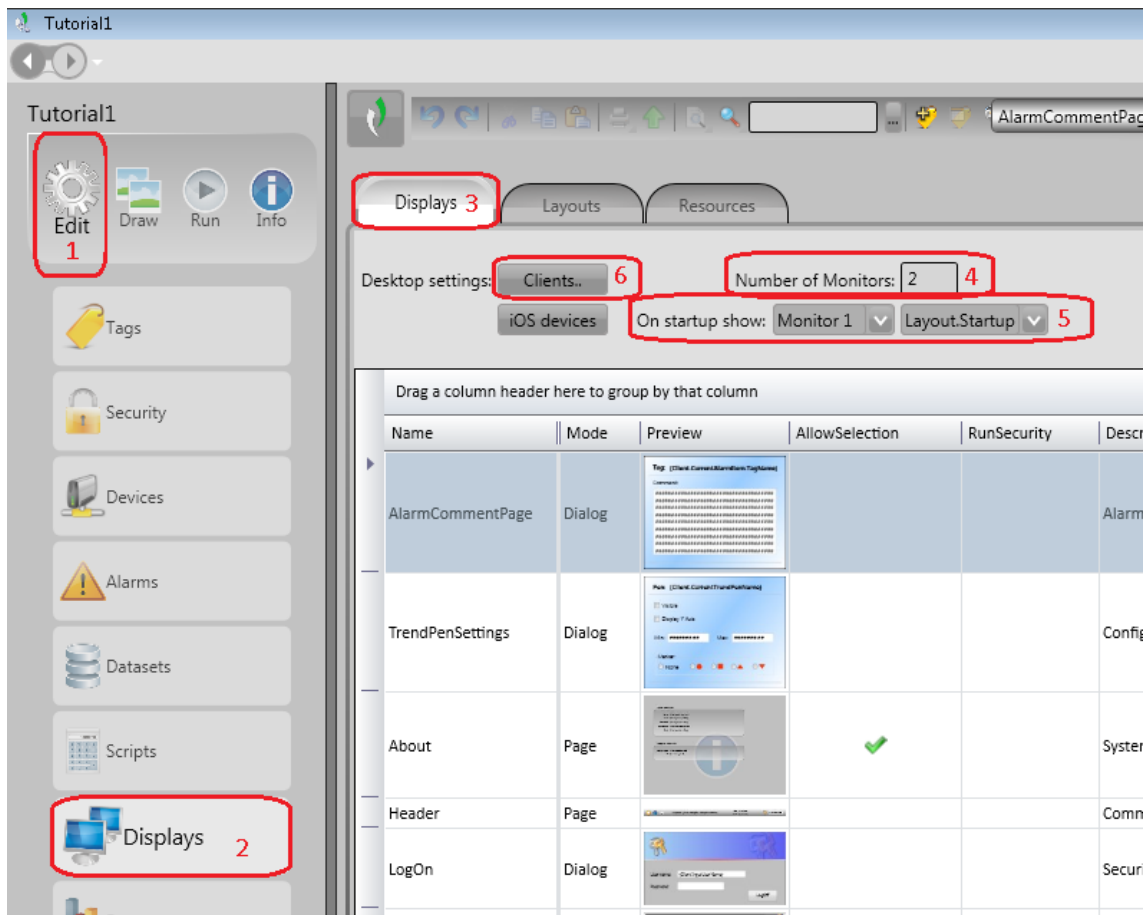


Figure 6-44. Multiple Monitors

1. Select Edit menu
2. Select Displays
3. Click on Displays tab to access the already existing display configurations of the project
4. Set the amount of monitors to be used
5. Select the corresponding layout to each monitor that will be opened when running the project
6. Click Clients button to make the configuration of the Rich Clients on Runtime (a single configuration for all rich clients). See Figure 6-45

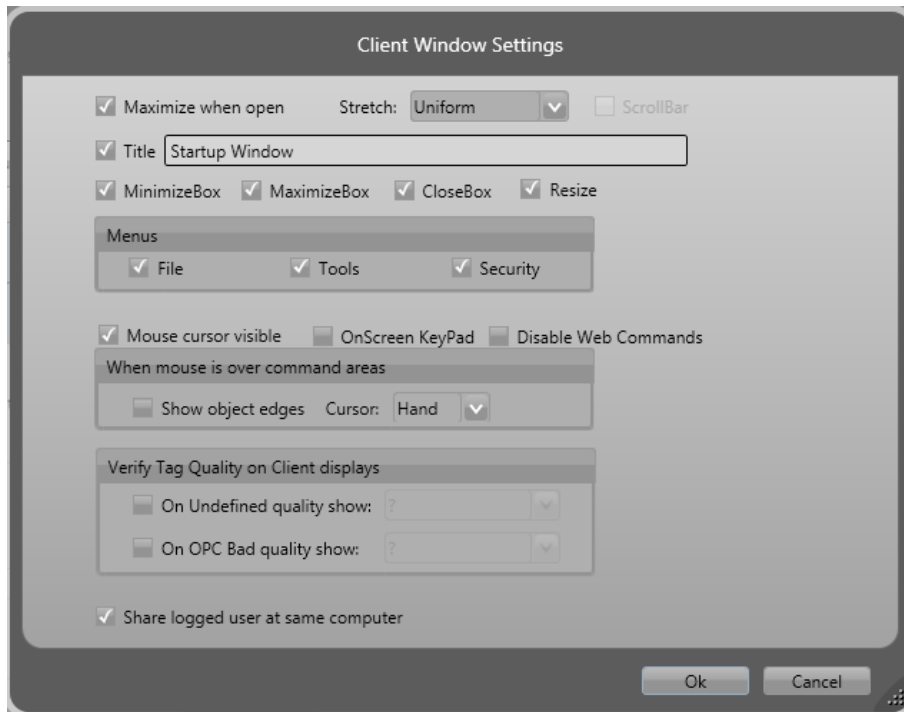


Figure 6-45. Rich Clients Configurations

Layouts

This tab edits the project layouts. Layouts are containers (DockPanels) that define Display arrangement during the runtime execution. The last Display listed on the layout is the one that will be changed when executing the `Client.OpenDisplay()` command. The other Displays that remain open typically contain information like navigation menus, alarm lines and global application information. The Project can have only one layout, or it can dynamically change the layout using the `Client.OpenLayout()` command.





To create a new layout, or add/delete pages on the selected layout, the user should click the following buttons: ,  and . The  button enables the user to navigate up and down in the page list. Layout dimensions and background color can be defined using Width, Height and Background fields.

Figure 6-46 illustrates the layouts settings.

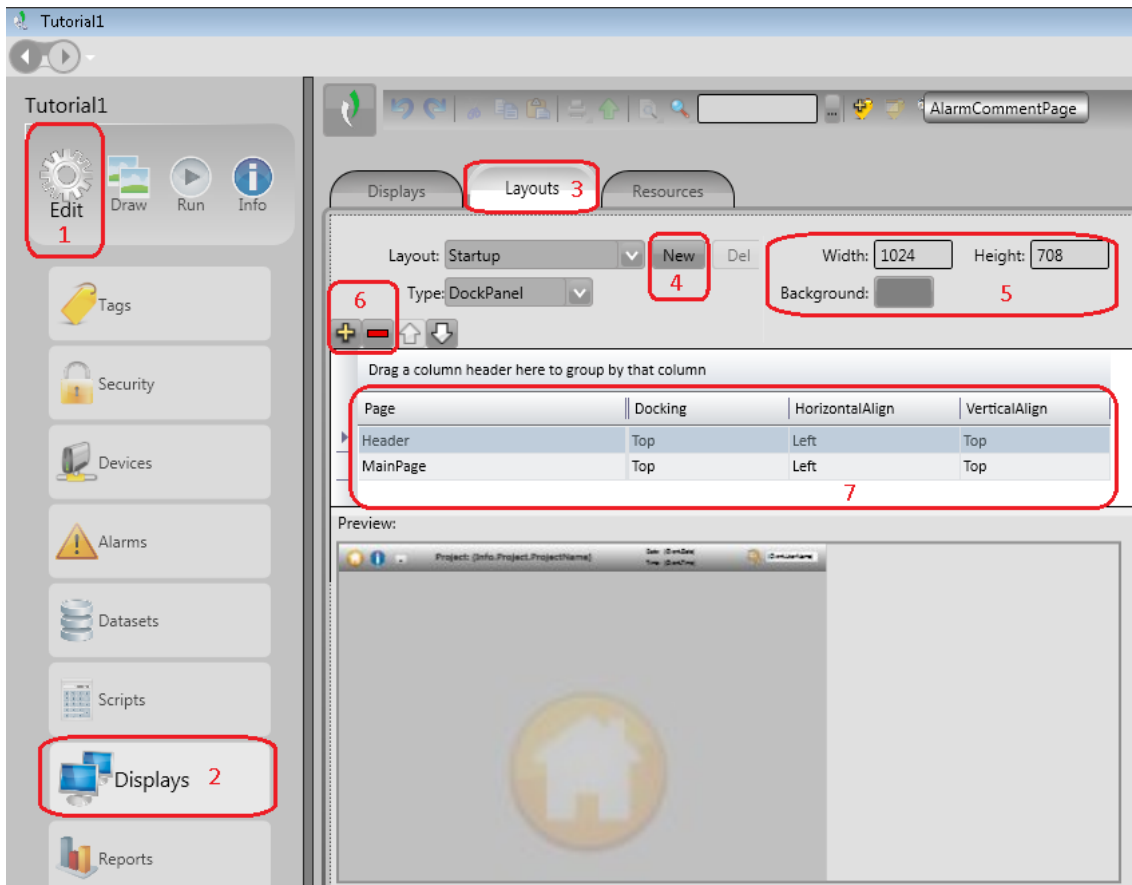



Figure 6-46. Edit Layouts

The required steps to edit and include a new layout shown Figure 6 46 are described below:

1. Select Edit menu
2. Select Displays
3. Click on Layouts to access the layout configurations already existing in the system
4. Click on New to add a new layout
5. Set the dimensions and background color of the corresponding layout
6. Set the amount of screens to be presented via the  button
7. Select the screens and its positions on the layout

Page

This column refers to the name of the display used in this Layout. Only displays on PAGE mode can be included on layouts.

Docking

This column defines the Display Docking position. Docking positions:

- Left
- Top
- Right
- Bottom

The docking position can be changed by a click in the table field and selecting the desired position.

HorizontalAling

This column defines the Horizontal alignment. Possible options:

- Left
- Center
- Right

VerticalAling


This column defines the Vertical alignment. Possible options:

- Top
- Center
- Bottom

Margins

This column defines the Display margins inside the DockPanel. All the project display measures (size, width, etc.) are in WPF Units (Windows Presentation Foundation).

Resources

This tab lists the Displays resources. In order to enable a distributed execution of the Runtime displays and Web Clients (when using images on displays and reports), the user should import the image(s) through the corresponding button () instead of selecting the file name. The Resources tab presents the additional benefit that, in case an image is replaced in the ResourceDictionary (not changing the Resource Name), it automatically updates all the references to that specific resource on Displays and Reports. Figure 6-47 illustrates the three stages of the Resource Edit menu.

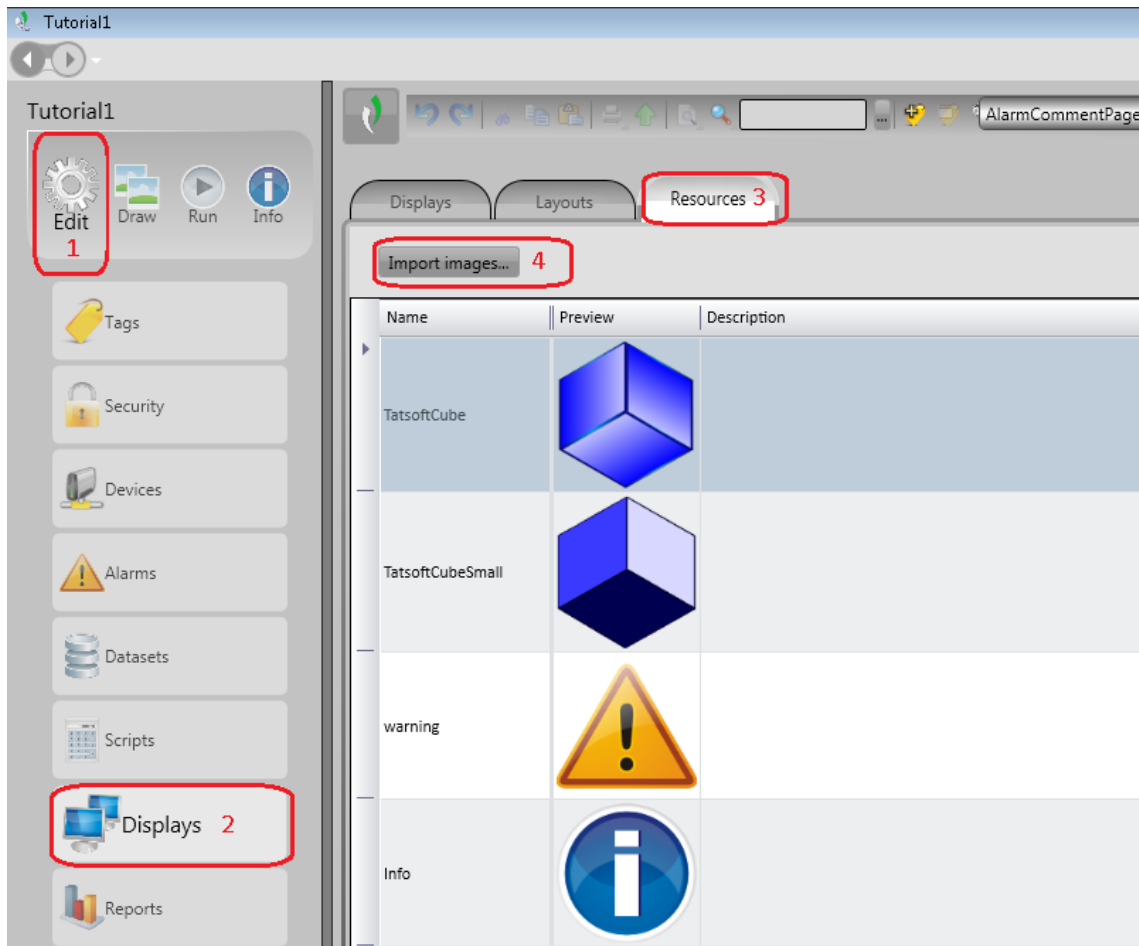


Figure 6-47. Resources Edit

The required steps to import new images shown on Figure 6 47 are described below:

1. Select Edit menu
2. Select Displays
3. Click on Resources tab all images already imported in the project
4. Click on Import Images... button to select and import the image

Name

This field refers to the name of the Resource object. The name can be changed by a click in the corresponding table field.

Preview

This field allows the Resource image Preview.

Editing Reports

The report editor allows the inclusion of dynamic text, dynamic symbol, graph, data sets as well as the results view in a complete and easy-to-use editor. The Reports tab configuration is shown on Figure 6-48.

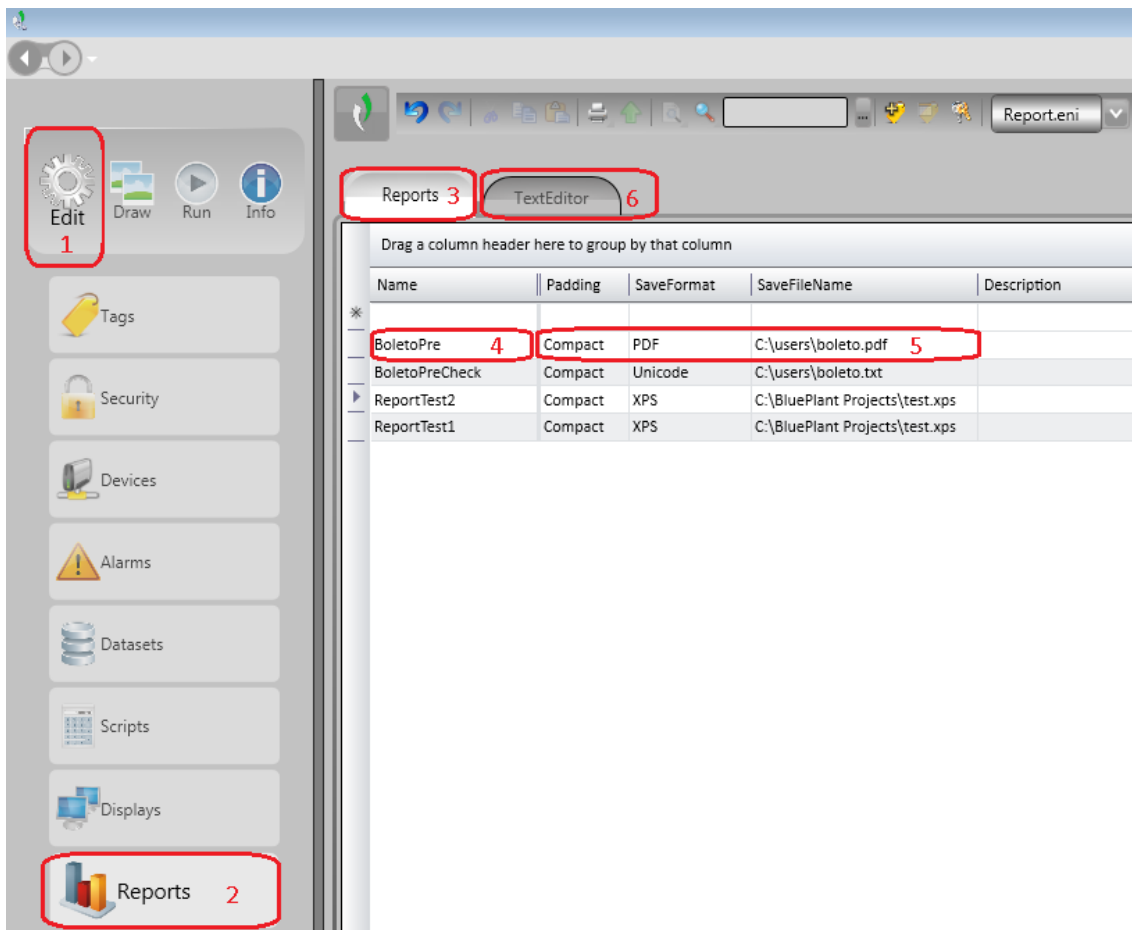


Figure 6-48. Reports Configuration

The required steps to edit or include a new report shown on Figure 6-48 are described below:

1. Select Edit menu
2. Select Reports
3. Click on Reports tab to access all the already created reports of the project
4. Enter or edit the report name on the row labeled with an asterisk (*)
5. Set the report features such as data alignment, format and name of the file where will be created the report
6. Click on TextEditor to edit the report format.

The Figure 6-49 details the last step.

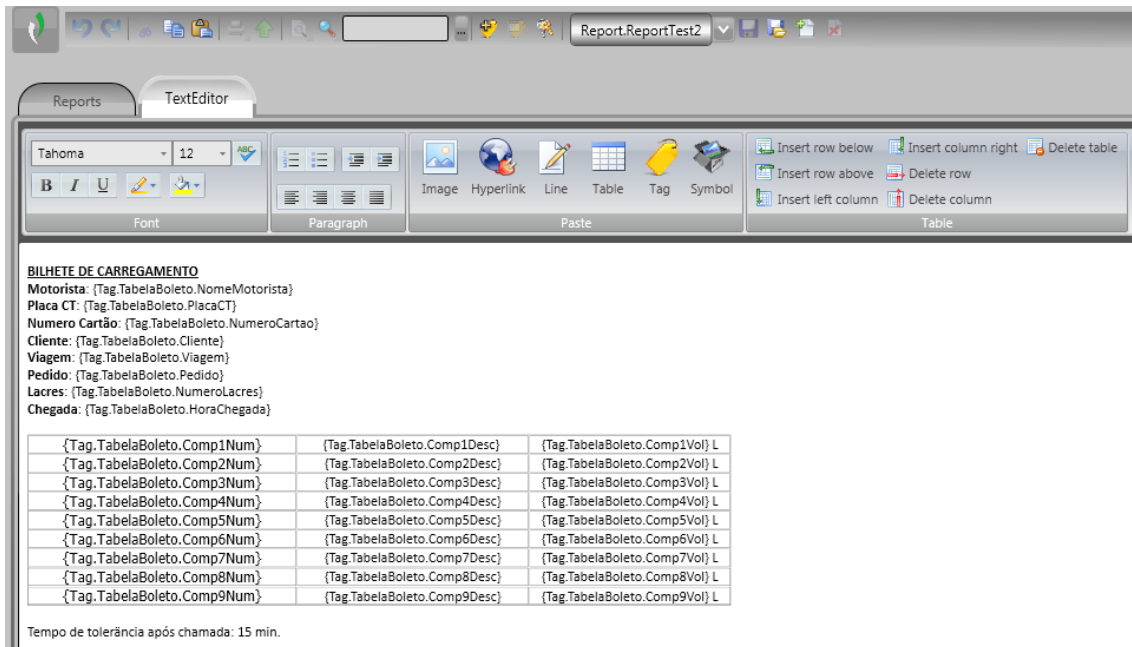


Figure 6-49. Report Edit

Name

This field refers to the name of the Report object.

Padding

This field defines the Padding when replacing a TagName by its values. Possible options: Compact, PadRight and PadLeft. The Padding uses the exact value number that is configured in the report template when creating the Report in the Runtime. The left or right align value inside the space is also included.

SaveFormat

This field defines the File format used to save the Report. Possible options: XPS, HTML, Unicode, PDF and ASCII.

Save File Name

This field defines the filename used and where the report will be saved. This field can have Tags values evaluated when generating the Alarm, for example. Add the Tag name in that table cell using the XAML binding notation between curly brackets. Example:

C:\MYREPORT.XPS

C:\MYREPORT-MONTH{SERVER.MONTH}-DAY{SERVER.DAY}.XPS

TextEditor

This tab comprises a Text editor for creating reports, as illustrated on Figure 6-50.



Figure 6-50. Text Editor

Figure 6-51 shows the edit and saving procedures of a report using the text editor built in the software.

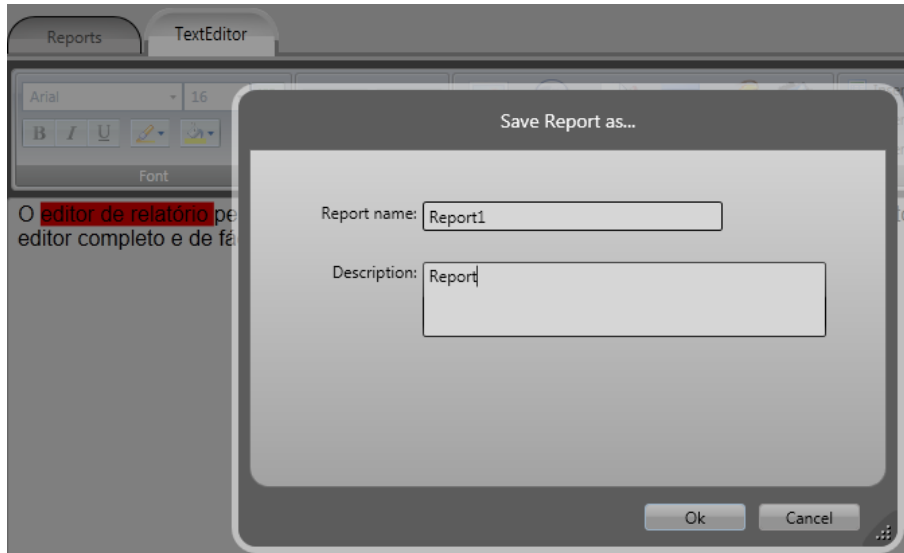


Figure 6-51. Edit and Saving on Text Editor

Figure 6-52 shows the consolidation of the edited form on the preceding figure with some settings available in the report edit menu.

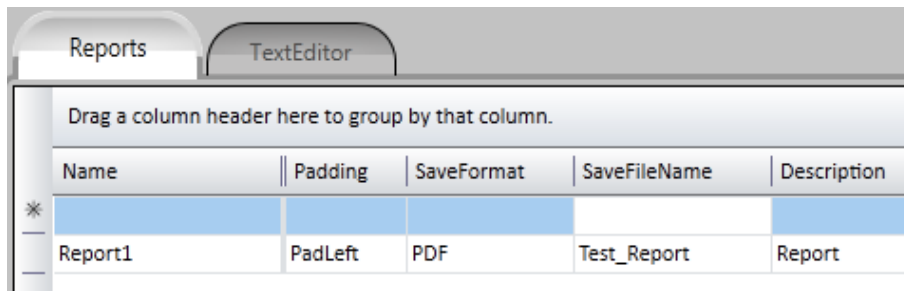
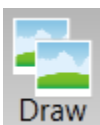


Figure 6-52. Report Configuration

Draw Menu



The Draw environment provides access to all construction tools for building the application screens. This menu affords the options: screens, codes and symbols. The option Draw allows the drawing of Screens and Symbols. The vertical toolbar enables the user to select a specific component or use the selection tool to move, group and resize this component. Figure 6-53 presents the vertical toolbar.

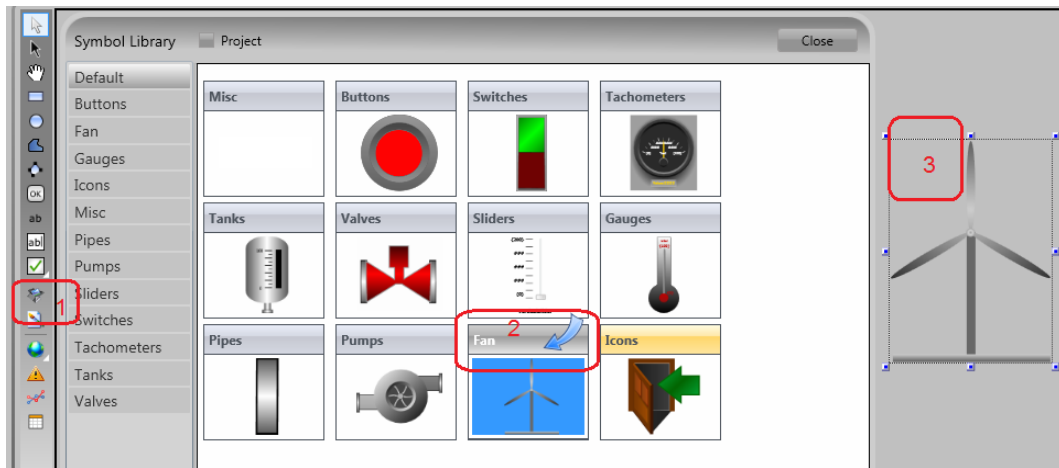


Figure 6-53. Draw Menu and Vertical Toolbar

The required steps to include a symbol on the screen are highlighted on the preceding figure and are described below:

1. Select the symbol to be included by clicking on the icon placed on the vertical toolbar
2. Click and drag the symbol to the screen
3. Release the left button of the mouse to end the inclusion of the symbol

The horizontal toolbar (on the bottom part of the screen) provides commands to group, combine, align and block the selected symbol(s). The usage of the horizontal toolbar, as well as some context menu commands, is shown on Figure 6-54.

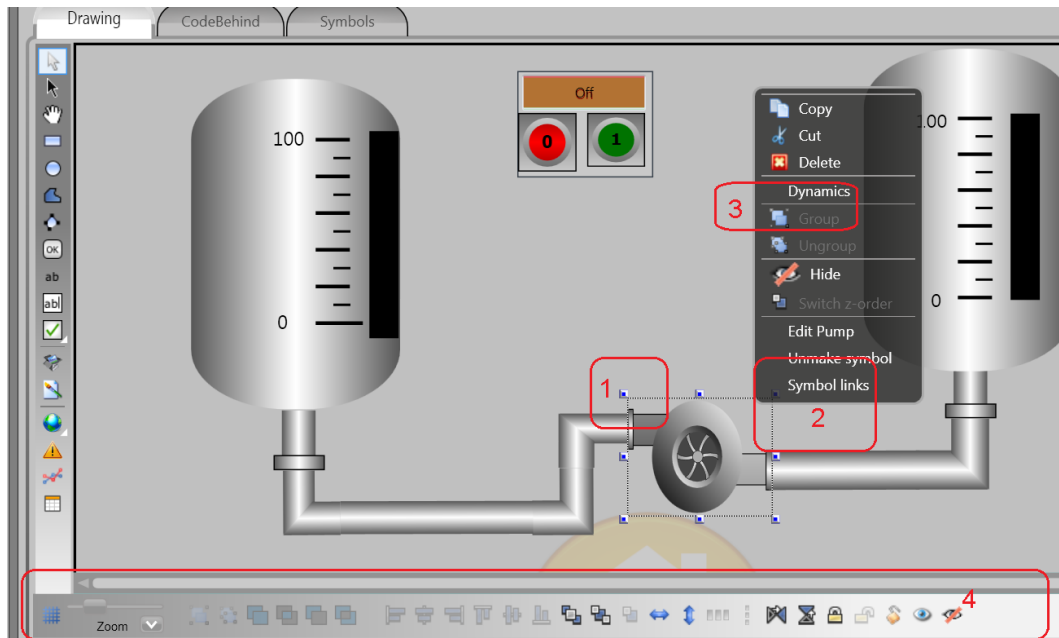


Figure 6-54. Draw Menu and Horizontal Toolbar

The required steps to set the symbol properties are numbered below:

1. Select the desired symbol with the left button of the mouse. To obtain a multiple selection, press SHIFT + CLICK WITH THE LEFT BUTTON in each desired component
2. Click on the symbol with the right button of the mouse to open the context menu
3. Select the desired option on context menu
4. Apply the desired properties to the symbol(s) using the toolbar

The items of the vertical toolbar shown on Figure 6-55 are detailed in the following.

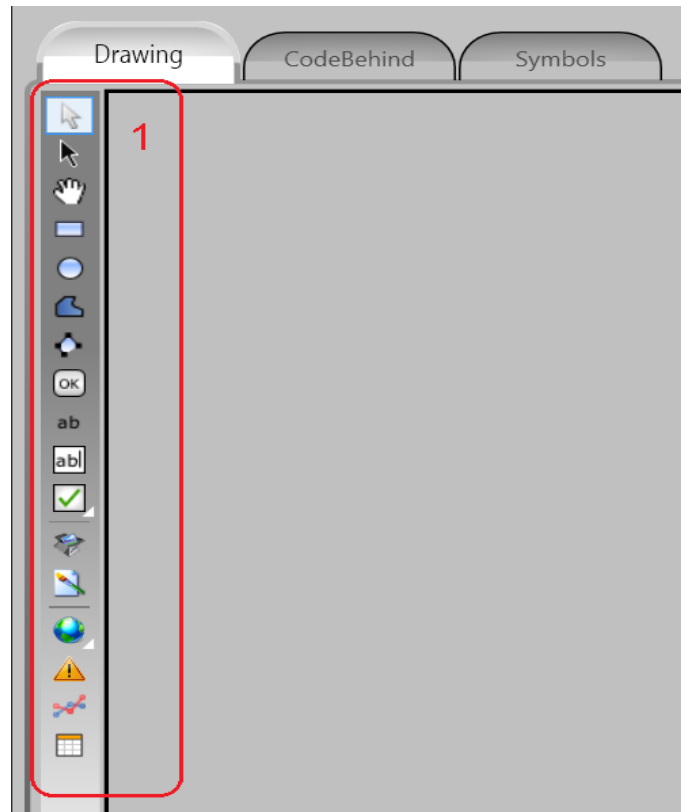


Figure 6-55. Vertical Toolbar

Selection Tools

The Selection Tools are used to select an object and modify the drawing view area.

Selection Tool

Click once on an object to select it. Press CTRL+MOUSE CLICK to select multiple objects (and object groups), at the same time. Hold down the CTRL key as you click on each object. Press SHIFT+MOUSE CLICK to toggle from among the selected objects. Click in an open area of the display and then select a group of elements by highlighting the desired elements while holding down the left mouse button. Double-click on an object to open the dynamic configuration window which provides settings for dynamic object properties.

Direct Selection Tool

This tool is used to select an object inside a group and modify its properties. Click once on the object to select it. The user can also add, remove and modify the points in a Polyline by this tool. To move the point, select by clicking on it while holding the left mouse button down and then, drag it to the

new position. To add a new point (adjacent to the selected one), double-click on a point. To delete a point, right-click on it.



Hand Tool

The Hand tool is used to modify the view window. By clicking once on the display background and holding down the left mouse button, the display is shifted to the desired position.

Geometric Objects



Creates a Rectangle object



Creates an Ellipse object



Creates a Polygon object



Creates a Polyline object

A right-click of the mouse finalizes the use of each tool.

To obtain further information on how to add, modify and remove points after the creation of a polygon/polyline check the Selection Tools section.

Display Components

These objects create screen components.



Creates a Button object



Creates a CheckBox object

A right-click on the component icon allows the user to access the tools in a horizontal popup menu. Once a tool is selected in this menu it becomes the default one for that block in the vertical bar.



Creates a RadioButton object



Creates a ComboBox



Creates a ListBox



Creates a PasswordBox



Creates a DatePicker Control



Creates a DateTimeTextBox

Input and Output Text Tools

The text Input/Output tools are used to create input/output text objects

ab Creates a TextOutput (TextBlock) object. See Figure 6-56.

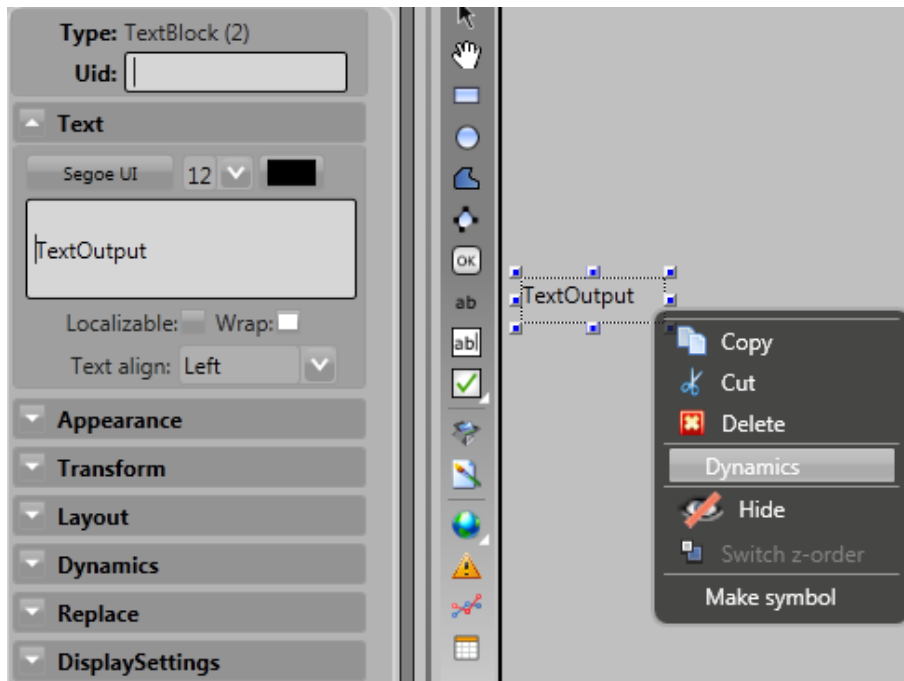


Figure 6-56. TextOutput Object

One right-click of the mouse on the object allows the access to the suspended edit menu associated to the text output, including its Dynamic Configuration, that can also be accessed through a double click on the object.

abl Creates a TextIO (TextBox) object. See Figure 6-57.

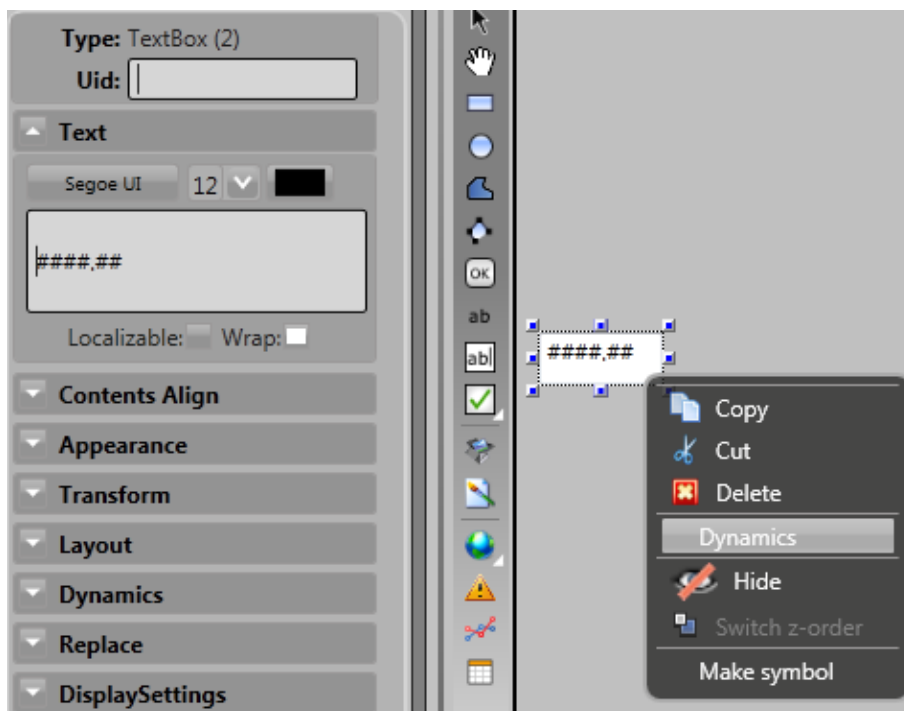


Figure 6-57. TextIO Object

One right-click of the mouse on the object allows the access to the suspended edit menu associated to the text input/output, including its Dynamic Configuration that can also be accessed through a double

click on the object. For example, to link the TextIO with a Tag, double click the TextIO object, and, on the dynamic configuration window, choose the TextIO dynamic.

Symbols Library

This item opens the Symbols Library.



This Library includes both built-in and user-defined symbols.

Advanced Controls



This item creates a WebBrowser object.

A right-click on the component icon allows the user to access the tools in a horizontal popup menu. Once a tool is selected in the horizontal menu, it becomes the default one for that position in the vertical bar.



Creates a PageSelector object



Creates a ReportViewer object



Creates a XpsViewer object



Creates a CircularPanel object



Creates a Calculator object



Includes aWPF control component

Alarm

The Alarm Object is used to control Warnings.



Creates an Alarm Window.

This object enables the user to create and position an Alarm Window. See Figure 6-58.

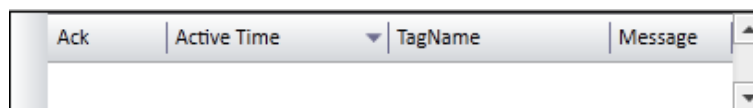


Figure 6-58. Alarm Window



Figure 6-59. Alarm Window Configuration

Control Name: Defines a name for the control. It is used via CodeBehind to link the graphical object to a .Net class (TAlarmWindow).

MaxLines: Maximum number of rows that will be displayed in the object. The filter is done on the server (no data sent from the server to the client with the purpose of improving performance).

List: Types of alarms that appear on the object.

- OnlineAlarms
- AlarmsHistory
- Events
- AlarmsHistory+Events

History Interval: DateTimeOffset that represents the beginning of the history period.

History Interval (To): DateTimeOffset that represents the end of the history period.

Ack by Page: Object configuration that, when triggered, recognizes all visible alarms in the alarm object.

Show Column Titles: Enables to show the columns titles.

Show Group Control: Enables the area where the user can drag the column names to create groups

Show Group Control (Label): Text that will be displayed within the "Group control" area.

AllowSort: Enables the sort functionality on the right side of each column.

Allow Column Reorder: Enables the functionality of dragging the column to another column place, swapping the order between them.

Display Value Column as String: Enables replace in column "Value" the value by its corresponding string, configured in the dictionary.

Filter: This field is a "where" of a sql statement, so the valid syntax is anything valid in a "where" sql statement, taking into account the existing column names in the object. Ex:

```
[TagName] = 'Tag.Tag1' AND [Group] = 'Alarm.Group.Critical'
```

Refresh: Object configuration that, when triggered, refreshes the alarm object.

Display Millisecond: Enables milliseconds visualization in date formatting.

Ack Selected Line: Configures the hotkey or combination of hotkeys to acknowledge the selected row in the alarm object.

Columns: Through the up and down arrows, the user can change the order in which the columns will be displayed in the alarm object.

For each column the following settings are possible:

Visible: Defines if the column is visible or not.

Allow Filter: Enables the filter option in the column.

Show in column chooser: Allows the user to select and drag the column.

Title: Sets the title that appears in the column header.

Width: Sets the column width.

Sort: Configures the type of sort sue that will be applied automatically in the column.

Trend

The Trend Object shows the historical record of a Tag.



Creates a Trend Window.

The user should place the Trend window on the screen and double-click on it to open the configuration window. Fields:

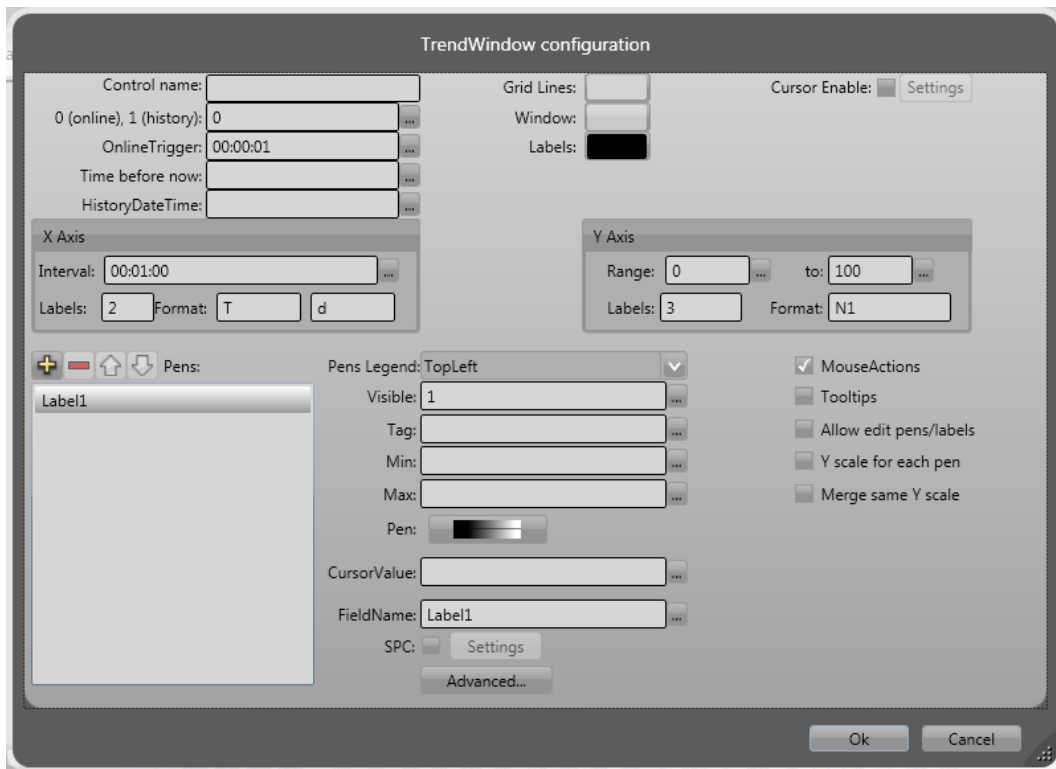


Figure 6-60. Trend Window Configuration

ControlName: Defines a name for the control, so it can be accessed in the associated code script. See: Access to the Display object in the associated script.

0(online) 1(history): 0 - the control shows the online data. 1 - the control shows the history data. Ex:
`Tag.onLineHist`

OnlineTrigger: The refresh rate of the online trend control. Ex:
`00:00:01` or `{Tag.trendTimeSpan}`

HistoryDateTime: The initial point for the history trend control. Ex:
`{Tag.initialTrendHistory}`

Max Samples: Sets the maximum amount of samples that will be collected from the database.

Window: Sets the background color of the trend control.

Labels: Sets the color of the trend control labels.

Cursor Enable: Enables (checked) or Disables (unchecked) the vertical cursor.

CursorPosition (%): Indicates the cursor position, where 0 means initial position and 100 means end position. Ex:
`Tag.cursorPos`

Cursor Output: Indicates the X axis value for the current cursor position. Ex:
`Tag.cursorOut`

Cursor Color: Defines the cursor color.

Pens Legend: Defines the position of the pens legend in the trend control.

Y Axis

Range: Defines the minimum and maximum values for the Y Axis.

Labels: Defines the quantity of horizontal grid lines.

Format: The format of the values in the Y Axis. For valid numeric formats, see the item that refers to tag format. Ex: N1 (number with 1 decimal place).

X Axis

Interval: Defines the X Axis TimeSpan.

Labels: Defines the quantity of horizontal grid lines.

Format: The X Axis format is defined by two fields: first line format and second line format. This is especially useful to represent label marks that require two levels of information. For valid date and time formats, see the item that refers to Tag Format. Ex:

T (Time) for the first line format, d (short date) for the second line format.

Pens

Visible: Shows (1) or hides (0) the selected pen. Ex:

1 or {showPen1 }

Tag: Sets the tag that will provide the value for the pen.

Min: Linear scale reference for the tag value, according to the Y Axis range.

Max: Linear scale reference for the tag value, according to the Y Axis range.

Pen: Sets the style, the color, and the thickness of the pen line.

Mark: Sets the mark for each point in the trend line.

CursorValue: Sets the tag that will receive the real value of the Y Axis, according to the cursor position. Ex:

Tag.pen1CursorValue.

FieldName: Sets the pen name.

Y Axis Scaling: Consider Y Axis Min = 0 and Y Axis Max = 100.

The trend control allows many pens to be displayed together. When your pens are not on the same range, you can use the tips below to fit your data in the same chart, for better visualization. If some pen has a lower range, 0 to 1 for example, you can set the Max property of the pen to 1, so when the real tag value is 1, the value 100 will be displayed in the chart (100/1 scale). If some pen has a higher range, 0 to 1000 for example, you can set the Max property of the pen to 1000, so when the real tag value is 1000, the value 100 will be displayed in the chart (1/10 scale).

Data Grid



Creates a DataGrid window.

The user should place the DataGrid window on the screen and double-click on it to open the configuration window.

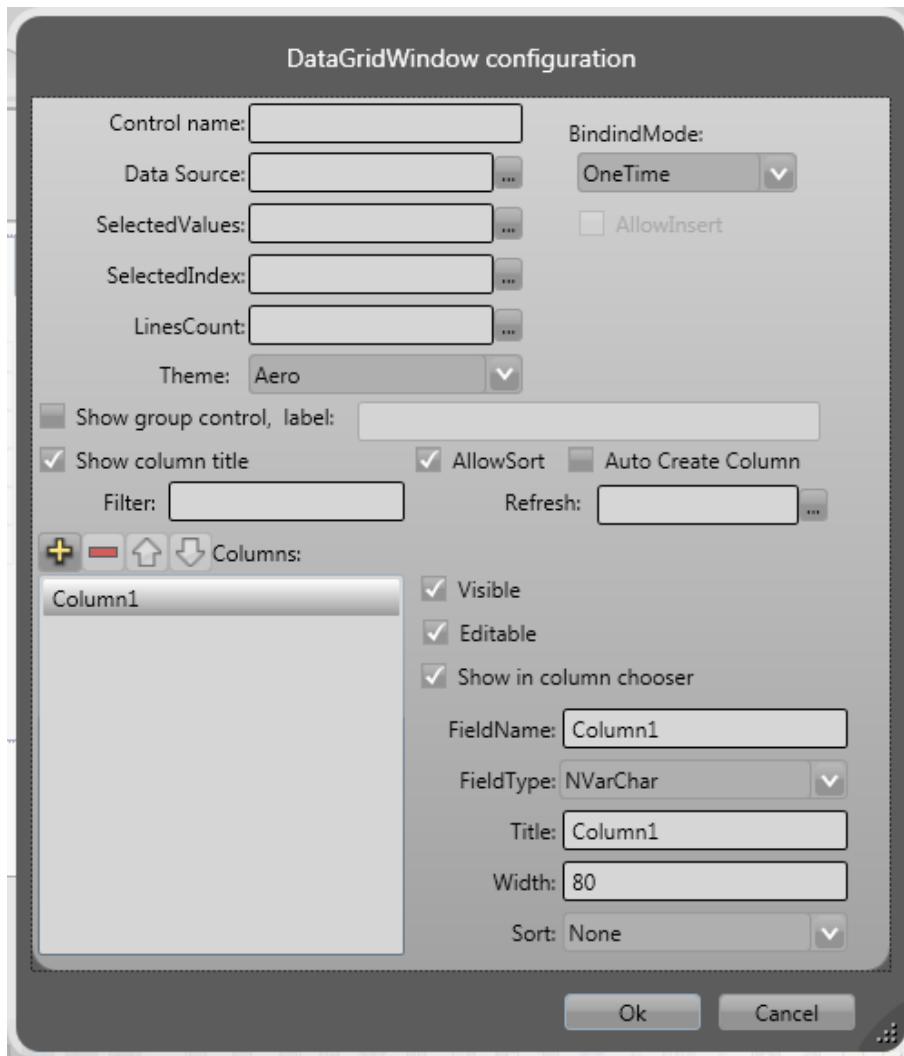


Figure 6-61. DataGrid Window Configuration

Control Name: Defines a name for the control. It is used via CodeBehind to link the graphical object to a .Net class (TDataGridWindow)

Data Source: Dataset object in which the user wants to display the contents in the DataGrid. It can be either Dataset.Table as Dataset.Query.

Selected Values: Refers to the tag array of the Text type, where the contents of each column will be placed in each position of the array.

Selected Index: It is the object that receives the selected line.

Lines Count: It is the object that receives the number of rows in the DataGrid.

Theme: Selection of the visual theme to be used.

Binding mode: Selects the data flow shape:

OneTime: The grid object is populated only once.

OneWay: The GRID is populated only from the database.

TwoWay: The GRID is populated from the database and, when modified in the GRID, the data is also updated in the database.

OneWayToSource: The data is only updated from the GRID to the database.

Allow Insert: Enables insertion of new rows in the DataGrid object.

Show group control: Enables the area where the user can drag the column names for grouping.

Show group control (Label): The text that will be displayed within the "Group control:" area.

Show Column Titles: Enables columns titles.

AllowSort: Enables the sort functionality on the right side of each column.

Auto Create Column: Enables the columns to be created automatically based on existing columns in the datatable of the Dataset.Table or Dataset.Query.

Filter: This field is a "where" of a sql statement, so the valid syntax is anything valid in a "where" sql statement, taking into account the existing column names in the object. Ex:

```
[Col1] = 'ABC' AND [Col2] = 'DEF'
```

Refresh: Object configuration that, when triggered, refreshes the GRID object.

Columns List: Through the up and down arrows, you can change the order in which the columns will be displayed in the object, and with the buttons + and - you can add or remove columns.

For each column the following settings are possible:

Visible: Enable if the column is visible or not.

Editable: Enables if column content can be changed.

Show in column chooser: Allows the user to select and drag the column.

FieldName: Column name at database. If this information is incorrect the user will not be able to bind the column to the datatable from the database.


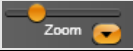










FieldType: Type corresponding to the column in the database.

Title: Sets the title that appears in the column header.

Width: Sets the column width.

Sort: Configures the type of sort sue that will be applied automatically in the column.

Horizontal Toolbar

Graphical Representation	Description
	Open Grid Settings: Grid configuration and adjustment
	Zoom: Zoom in or out adjustment
	Group: Makes grouping of the selected objects
	Ungroup: Makes ungrouping of the selected objects
	Union: Makes the union of geometrical objects which generates only one object
	Intersect: Makes the intersection of geometrical objects which generates a new object
	Exclude: Makes the exclusion of the frontal geometric form between the selected objects generating a new object
	Exclusive-Or: Makes the exclusion of the intersection of the selected geometric forms which generates a new object
	Align Left: Makes the objects alignment at the left of the last selected object
	Align Horizontal Center: Centralizes the objects horizontally, being reference the last selected object
	Align Right: Makes the objects alignment at the right of the last selected object
	Align Top: Makes the objects alignment at the top of the last selected object














	Align Vertical Center: Centralizes the objects vertically, being reference the last selected object
	Align Bottom: Makes the objects alignment at the bottom of the last selected object
	Move to The Front: Moves the selected objects to the front
	Move to The Back: Moves the selected objects to the back
	Resize Width: Makes the width adjustment of the selected objects having as reference the last selected object
	Resize Height: Makes the height adjustment of the selected objects having as reference the last selected object
	Space Evenly Horizontally: Makes the horizontal spacement equal between the selected objects
	Space Evenly Vertically: Makes the vertical spacement equal between the selected objects
	Flip Horizontally: Makes the horizontal inversion of the selected objects
	Flip Vertically: Makes the vertical inversion of the selected objects
	Lock Element: Lock all the selected objects
	Unlock Element: Unlock the selected object through the Direct Selection Tool
	Unlock All Elements: Unlcock all the display elements

Table 6-7. Horizontal Toolbar Elements

Dynamic Configuration

This option provides the access to the Dynamics Configuration window. Check a dynamic in order to enable it, or uncheck some dynamic to disable it. See Figure 6-62.

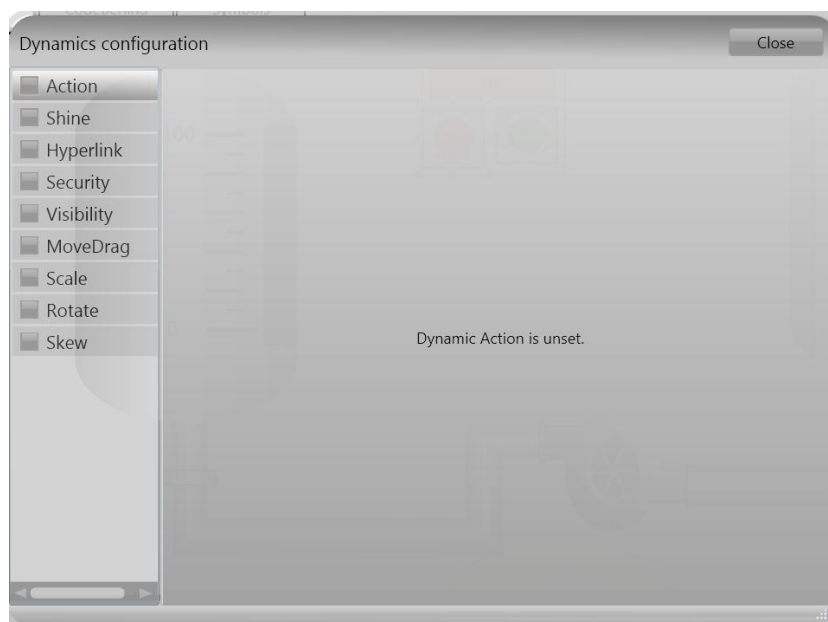


Figure 6-62. Dynamic Configuration

The items that comprise the Dynamic Configuration shown on Figure 6-62 are described below:

Action

This entry executes the actions triggered by the user interface. Available configurations:

- **Event:** Choose one of the Mouse events. More than one event can be selected for each action.
 - **Ex:** One action for `MouseButtonDown` (press down left mouse button) and another action for `MouseButtonUp` (release left mouse button)
- **Action:** Choose one action for the given event. None: No action
- **SetValue:** Set a value to the object
- **Object:** The object that will receive the value
- **Value:** The value that will be passed to the object
- **ToggleValue:** Toggles the object value. If the object current value is zero, the value will be 1. If the object current value is different than zero, the value will be 0. The "Object" field refers to the object that will be toggled.
- **OpenDisplay:** Opens a display. The "Display" field refers to the name of the display that will be opened
- **CloseDisplay:** Closes a display.
- **OpenLayout:** Opens a Layout. The "Layout" field refers to the Layout name that will be opened
- **RunScript:** Runs a script that is placed in the Code Associated to the Display. Write the new method name and click on New Button, or select one of the existing methods in the ComboBox
- **RunExpressions:** Runs the given expression
- **Expression:** enter the expression. Ex: `Tag.a + 1`, or `Tag.a + Tag.b`, or `Math.Cos(Tag.angle) * Math.PI`
- **Result (optional):** Enter the Tag or the property that will receive the expression value

Examples:

1. Run Expressions:

Sum two values and pass the result to another tag.

Expression:

```
Tag.quantity1 + Tag.quantity2
```

Result:

```
Tag.totalQuantity
```

2. Add one Tag

Expression:

```
TagCounter + 1
```

Result:

```
TagCounter
```

3. Add one Tag (0 - 10)

Expression:

```
If(TagCounter < 10 , TagCounter + 1 , 0)
```

Result:

```
TagCounter
```

These fields related to the expressions are illustrated on Figure 6-62.

Shine

This entry changes the object appearance dynamically. Available configurations:

IsMouseOver: Enter a tag that will receive the `OverValue` or the `NotOverValue`.

OverValue: Refers to the IsMouseOver value when the mouse is over the object.

NotOverValue: Refers to the IsMouseOver value when the mouse is NOT over the object.

Mouse Over Appearance: Refers to the object appearance when the mouse is over it.

Opacity: Refers to the object opacity (0 = transparent, 1 = opaque).

Scale: Refers to the object size (0.5 = half, 1 = the same size, 1.5 = one and a half, 2 = double size).

OuterGlow: Defines the OuterGlow color, the checkbox enables or disables it.

TextColor: Defines the text color, the checkbox enables or disables it.

Mouse Not Over Appearance: Refers to the object appearance when the mouse is not over it.

Opacity: Defines the object opacity (0 = transparent, 1 = opaque).

Scale: Defines the object size (0.5 = half, 1 = the same size, 1.5 = one and a half, 2 = double size).

IsSelected Appearance: Refers to the object appearance when it is selected.

IsSelected: Defines whether the object is selected or not.

Opacity: Refers to the object opacity (0 = transparent, 1 = opaque).

Scale: object size (0.5 = half, 1 = same size, 1.5 = 1,5 times, 2 = double size).

Scale Reference: Defines the reference for the scale dynamics.



Center



Left



Up



Right



Down

TextIO

This entry refers to the Text Input and Text Output Dynamics. It is compounded by the following settings:

Binding Mode: There is two ways: input and output are allowed; in the first, only input is allowed (the current tag value is not shown, but new values can be entered); in the second, only output is allowed.

Text: Indicates the text that will be shown in the object. If the text is a tag value or a property, it must be between curly brackets, for example: {Tag.analogInt1}.

DesignModeCaption: The value shown in design mode.

- ShowObjectNames: the content of the Text field is shown exactly as it is.
- ShowPlaceHolders: the characters ### are shown and the quantity of characters is defined by the MaxLength field.

Input Range: Defines the numeric range for the entered value.

CampoMaxLength: Defines the maximum characters amount.

Note:

If text is a tag value or a property, it must be between curly brackets, for example:

```
{Tag.analogInt1}
```

HyperLink

This entry opens a hyperlink.

HyperLink type: Choose one among the available options (http, ftp, file, mailto, telnet).

Url: Set the url to be opened.

FillColor

This entry changes the object fill color dynamically.

Expression: represents the value used for the FillColor dynamic.

ChangeColor: changes the object fill color through the following settings:

- **UsingLimits:** the resulting color will be given when the value is equal or higher than one of the limits.
- **AbsoluteValue:** the color will be the expression value; in this case the value must be a valid color. Ex: "White" or "#FFFFFFF"

Using Limits Example:

1 - Red

10 - Blue

When the value is 0, the object will have its own color (fill color dynamic will not act)

When the value is 1 to 9, the object will have the red color.

When the value is greater than 10, the object will have the blue color.

LineColor

This entry changes the object line color dynamically.

Expression: Represents the value used for the LineColor dynamic.

ChangeColor: Changes the object line color trough the following settings:

- **UsingLimits:** The resulting color will be the given when the value is equal or higher than one of the limits.
- **AbsoluteValue:** The color will be the expression value; in this case the value must be a valid color. Ex: "White" or "#FFFFFFF"

Using Limits Example:

1 - Red

10 - Blue

When the value is 0, the object will have its own color (fill color will not do anything).

When the value is 1 to 9, the object will have the Red color.

When the value is greater than 10, the object will have the Blue color.

TextColor

This entry changes the object Text color dynamically.

Expression: Represents the value used for the TextColor dynamic.

ChangeColor: Changes the object Text color trough the following settings:

- **UsingLimits:** The resulting color will be given when the value is equal or higher than one of the limits

- **AbsoluteValue:** The color will be the expression value; in this case the value must be a valid color. Ex: "White" or #FFFFFF " "

Using Limits Example:

1 - Red

10 - Blue

When the value is 0, the object will have its own color (fill color dynamic will not act).

When the value is 1 to 9, the object will have the red color.

When the value is greater than 10, the object will have the blue color.

Bargraph

This entry represents the Bar Graph Dynamic.

Expression: Represents the value used for the bar graph dynamic.

Range Value: The minimum and maximum values that will correspond to the minimum and maximum fill percentage.

Fill (%): The minimum and maximum bar graph fill percentage.

Bar Color: The bar graph color.

Orientation: The bar graph orientation.







Elements Graphical Representation	Description
	Movement of the graph bars from the bottom part to the top
	Movement of the graph bars from the center to the horizontal extremities
	Movement of the graph bars from the top part to the bottom
	Movement of the graph bars from the left to the right
	Movement of the graph bars from the center to the vertical extremities
	Movement of the graph bars from the right to the left

Table 6-8. Bargraph

Visibility

This entry changes the object visibility and opacity dynamically.

Visible: Enter a tag, a property or an expression returning a value. If the resulting value is zero, the object will be visible; if it is greater than zero the object will be hidden.

Tooltip: The string that will appear as a tooltip.

Opacity: Configurations.

- **Object Value:** the value used to set the opacity
- **Range:** the minimum and maximum values that will correspond to the minimum and maximum opacity
- **Opacity:** the minimum and maximum opacity (0 - invisible, 0.5 - a bit transparent, 1 - opaque)

Move and Drag

This entry moves the object dynamically.

BindingMode: Configurations.

- Two Ways: input and output moving
- Input Only: input movement only, the object does not move when its value changes
- Output Only: output moving only, the object does not move with user interaction

Horizontal Move: Configurations.

- Object Value: represents the value used for the horizontal moving
- Range: the minimum and maximum values that will correspond to the minimum and maximum horizontal position
- Position : the minimum and maximum horizontal position

Vertical Move: Configurations.

- Object Value: represents the value used for the vertical moving
- Range: the minimum and maximum values that will correspond to the minimum and maximum vertical position
- Position : the minimum and maximum vertical position

Scale

This entry changes the object size dynamically.

Width Scale: Configurations.

- Expression: represents the value used for the width scaling
- Range: the minimum and maximum values that will correspond to the minimum and maximum width scale percentage
- Scale (%) : the minimum and maximum width scaling percentage

Height Scale: Configurations.

- Expression: represents the value used for the height scaling
- Range: the minimum and maximum values that will correspond to the minimum and maximum height scale percentage
- Scale (%) : the minimum and maximum height scaling percentage
- Scale Reference:



Center: Scale with reference at the center of the object



Left: Scale with reference at the left of the object



Up: Scale with reference on the top part of the object



Right: Scale with reference at the right of the object



Down: Scale with reference on the bottom part of the object

Rotate

This entry rotates the object dynamically.

Object Value: Represents the value used for the rotation.

Value Angle: The minimum and maximum values that will correspond to the minimum and maximum angle. Ex: 0 to 100.

Angle: The minimum and maximum rotation angle. Ex: 0 to 360°.

Center Reference:

Center: Rotation with reference on the center of the object



Left: Rotation with reference at the left of the object



Up: Rotation with reference on the top part of the object



Right: Rotation with reference at the right of the object



Down: Rotation with reference on the bottom part of the object

Skew

This entry skews the object dynamically.

X axis skew: configurations.

- Object Value: represents the value used for the X axis skewing
- Range: the minimum and maximum values that will correspond to the minimum and maximum X axis skewing angle. Ex: 0 to 100
- Skew (°): the minimum and maximum values that will correspond to the minimum and maximum X axis skewing angle. Ex: 0 to 180°

Y axis skew: configurations.

- Object Value: represents the value used for the Y axis skewing
- Range: the minimum and maximum values that will correspond to the minimum and maximum Y axis skewing angle. Ex: 0 to 100
- Skew (°): The minimum and maximum Y axis skewing angle. Ex: 0 to 180°
- Scale Reference:



Center: Skewing with reference on the center of the object



Left: Skewing with reference at the left of the object



Up: Skewing with reference on the top part of the object



Right: Skewing with reference at the right of the object



Down: Skewing with reference on the bottom part of the object

TextOutput

Dynamic text output.

Text: Indicates the text that will be displayed in the object. If the text is a Tag value or property it must be enclosed in braces. Example:

```
{Tag.analogInt1}.
```

Localizable: Indicates whether text should be translated in case of dictionary change.

DesignModeCaption: The value shown in design mode:

- ShowObjectNames: the content of the Text field is shown exactly as it is
- ShowPlaceHolders: the characters ### are shown. The quantity of characters is defined by the MaxLength field.

MaxLength: Sets the maximum number of characters.

Note:

If the text is a tag value or property it must be enclosed in braces, for example:

```
{Tag.analogInt1}.
```

CodeBehind

Use DrawCodeBehind to define a set of functions related with the displays as illustrated in Figure 6-63.

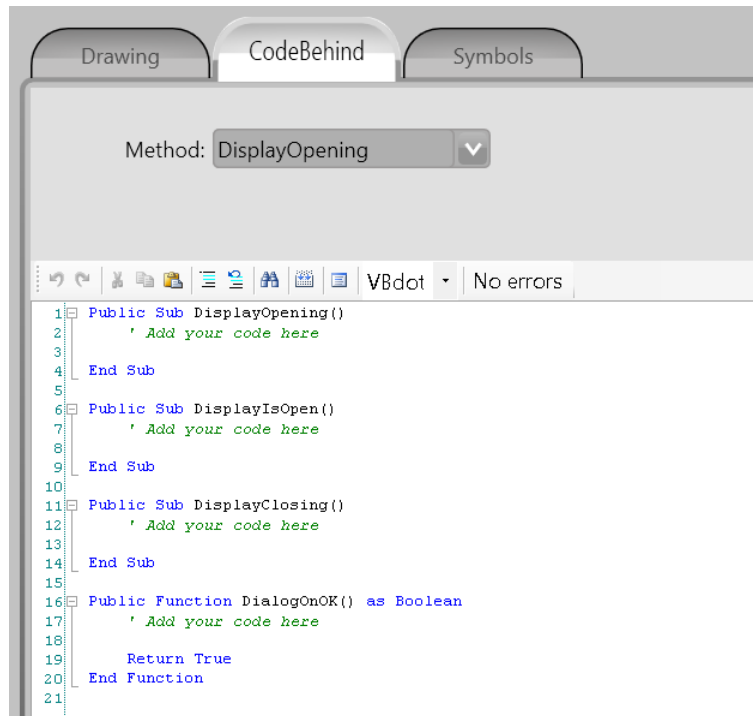


Figure 6-63. CodeBehind

These functions can be executed when opening or closing, or while the Display is opened, depending on the configured code. You can use the associated code to define mouse and input command handling methods to be executed on a specific display.

For DIALOG type displays use the built-in DialogOnOK method which is called when the built-in OK button is pressed. If "TRUE" is returned on that method, the dialog is closed; if "FALSE" is returned the dialog remains open. This method is commonly used to ensure data validation on the dialog (which prompts the user to correct incorrect entries prior to the closing of the dialog). Options:

- DisplayOpening(): executed when display is opening
- DisplayIsOpen(): called in a regular interval while the display is open
- DisplayClosing(): executed when display is closing
- DialogOnOK(): called when OK button on a dialog display is pressed.

Returning "1" allows the dialog to close. Returning "0" prevents the dialog from closing. The user can add .NET variables and methods on this page.

Note:

Because the client displays are designed to run on distributed and web environments it is recommended to avoid the use of functions that do not allow partial trust execution or that refer to physical file paths.

Symbols

Symbols are user-defined objects containing drawing and dynamic properties. The Figure 6-64 shows a symbol and its context menu.

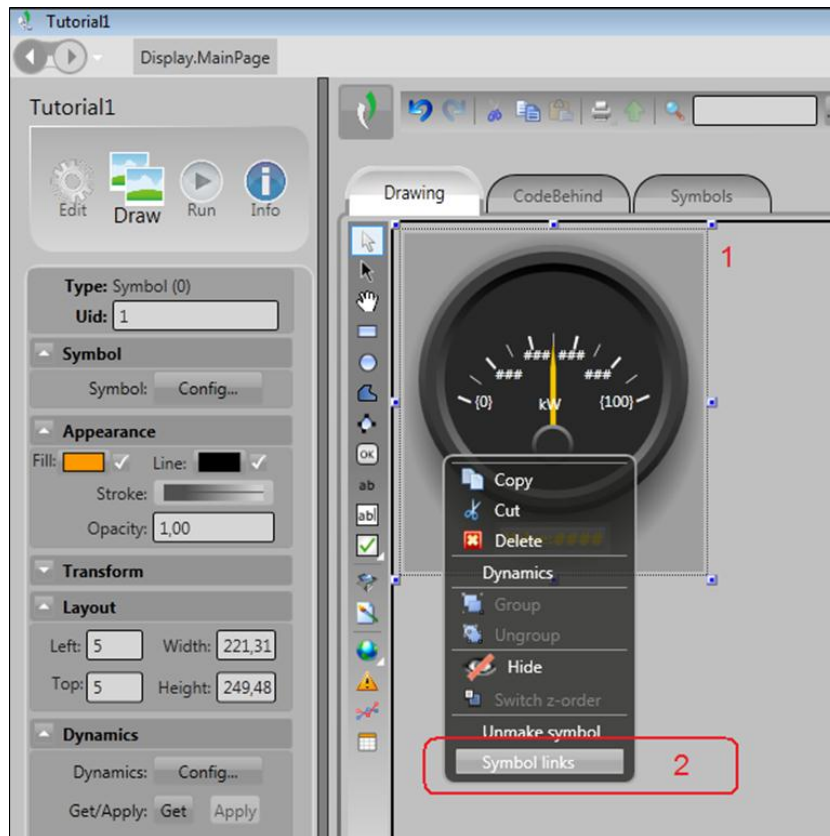


Figure 6-64. Symbol Context Menu

To simplify the use of symbols in other displays and applications, the user can define labels on dynamic properties where a TagName is expected. Use the syntax:

#LabelName: or #LabelName:DefaultValue.

When using symbols on displays or reports, select new tag names on the configuration window to be applied on defined label fields. A set of Symbols that is visible to all Projects is incorporated to the BluePlant framework. Those symbols are defined in the SymbolLibrary.tproj file located at the Product binary installation folder. The global Symbol Library file can be edited as a project file. The user can also create and save his own Symbols.

Creating Symbols

The images of Figure 6-65, Figure 6-66 and Figure 6-67 present the required steps to create a symbol. The procedures are also described below:

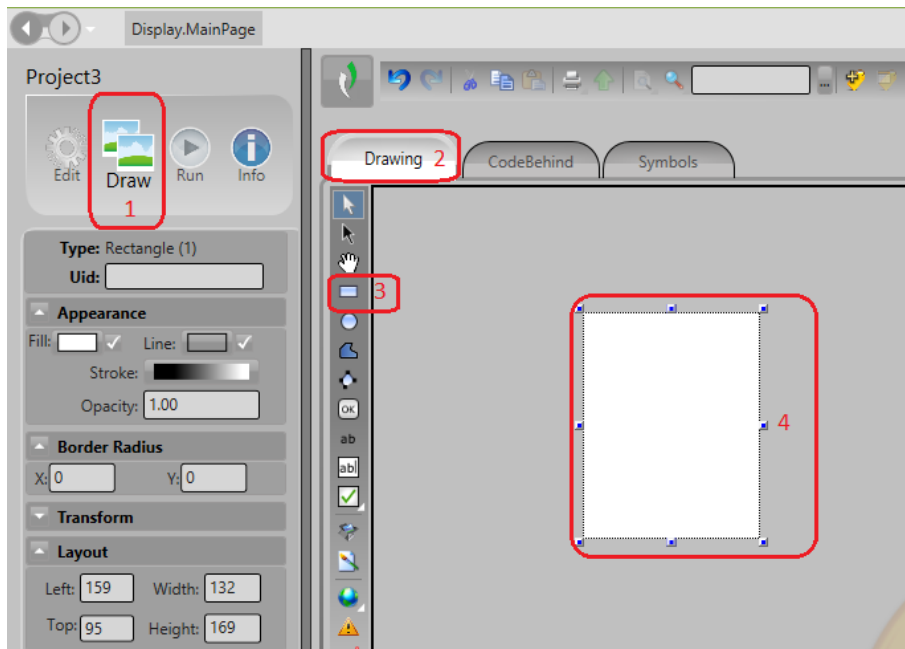


Figure 6-65. Creating Symbols

1. Select the Draw menu
2. Click on Drawing
3. Select the Rectangle object
4. Draw a rectangle on the screen and double-click the object created. The following screen will appear

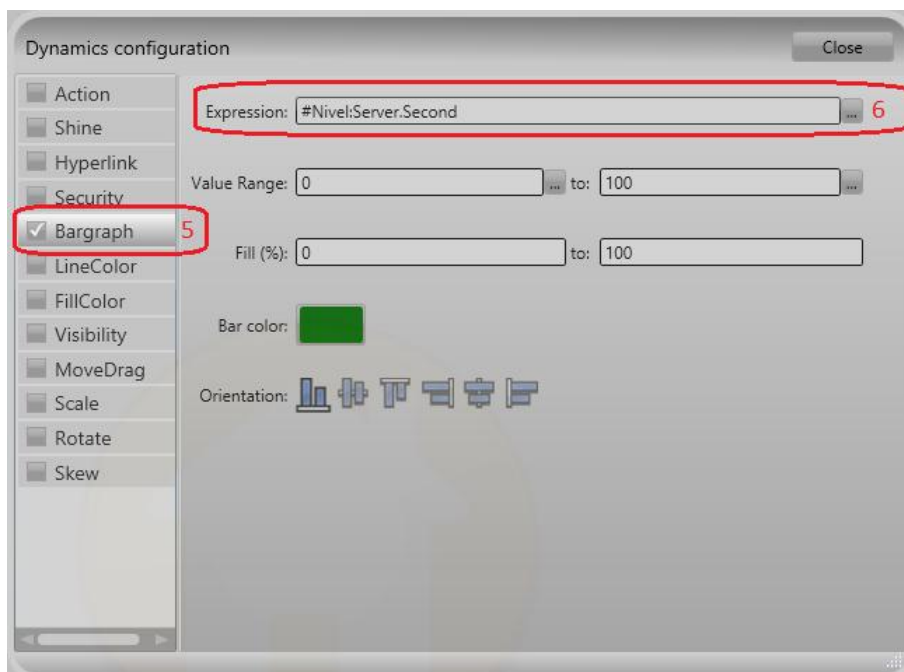


Figure 6-66. Dynamic Configuration

5. Select the Bargraph option.
6. In order to expose the Level, type the following syntax in the Expression field.
#Nivel:Server.Second

Note:

The symbol properties are exposed through the use of mnemonics, which are:

#<name>:<value> or #<name>:(<expression>).

7. Close the Dynamics Configuration screen.
8. Select all objects that make up the symbol (in this case, the rectangle only).
9. Right-click to open the context menu.
10. Select the option "Make new Symbol".
11. Fill out the information: symbol name, category, description and click YES.
12. This symbol will be available to be inserted at any point in the project through the Symbol Library.

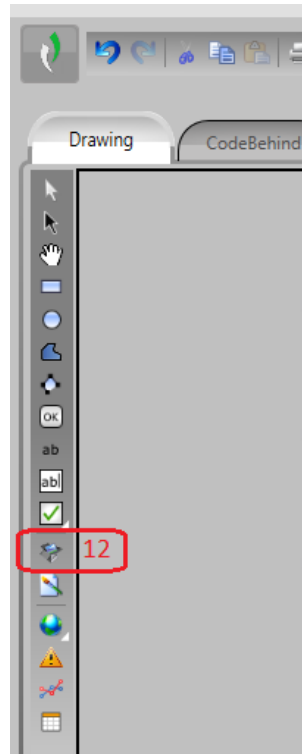


Figure 6-67. Dynamic Configuration

13. Go to the Symbol Library icon as shown in the figure above and insert the symbol created on the screen.
14. Double click on the symbol to open the symbol configuration screen. The property "Level" with the value "Server .second" will be shown and it can be changed to the desired tag. Each symbol can present its independent level value.

Changing a Symbol

The following steps describe the symbol modification procedure.

1. Select the symbol to be modified
2. Right-click to open the context menu
3. Select the option "Edit <symbol name>"
4. Make all desired changes to the symbol
5. Select all objects that make up the symbol
6. Right-click to open the context menu
7. Select the option "Update <symbol name>"

- All the symbols with this current screen name will be changed. A build operation saving all screens is necessary in order to change all symbols with this name in the project.

Info Menu

Project

Information submenu of the current project.

Version

Includes information about the current Project (Figure 6-68). Some information fields are also available during Runtime through the Info Namespace.

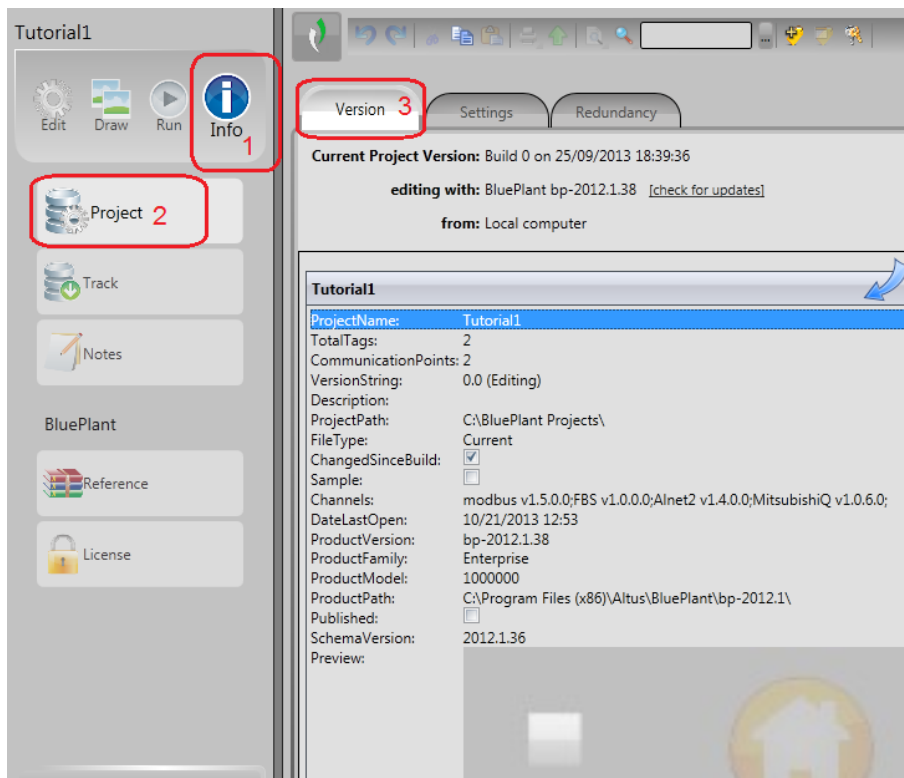


Figure 6-68. Version Tab

The items that comprise the information menu are numbered in the preceding figure and are described below:

- Select the Info option
- Select the Project option
- Select the Version option for project data exhibition

Settings

The Settings tab items are explained below, as Figure 6-69 illustrates.

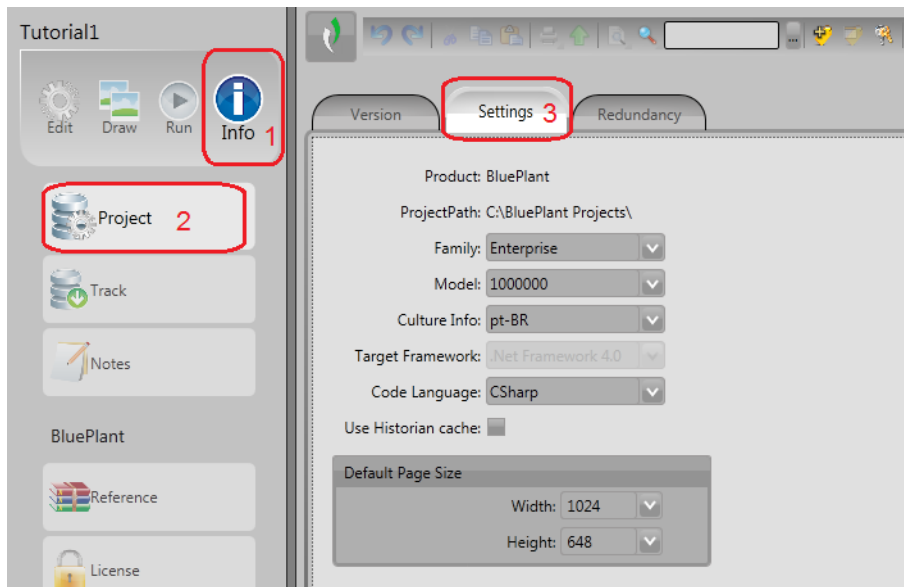


Figure 6-69. Settings Tab

The items that comprise the Settings tab are numbered in the preceding figure and are described below:

1. Select the Info option
2. Select the Project option
3. Select the Settings option

Project Settings

This tab defines global settings for the Project. Some information fields are also available during Runtime through the Namespace Info.

BluePlant Model

Select the corresponding runtime model. The following options are available:

- Express
- Enterprise
- Student
- Lite

Model

Select the runtime model. The following Runtime models are available:

- 75
- 150
- 300
- 500
- 1.500
- 2.500
- 5.000
- 15.000
- 25000
- 50.000
- 100.000

- 1.000.000

Culture Info

Select the project language among the available ones.

Target Framework

Select the framework, platform that BluePlant executes.

Code Language

Select the Script language. The following scripts code languages can be selected:

- VBdotNet
- CSharp

Default Page Size

Define the default page size. Figure 6-70 illustrates this option.

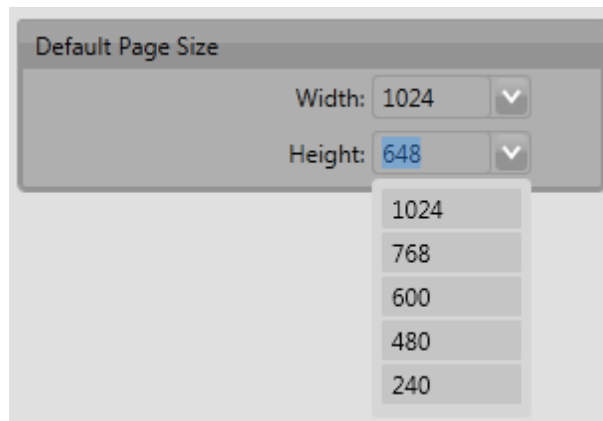


Figure 6-70. Page Size Adjustment

Redundancy

The Redundancy tab items are explained below, as Figure 6-71 illustrates.

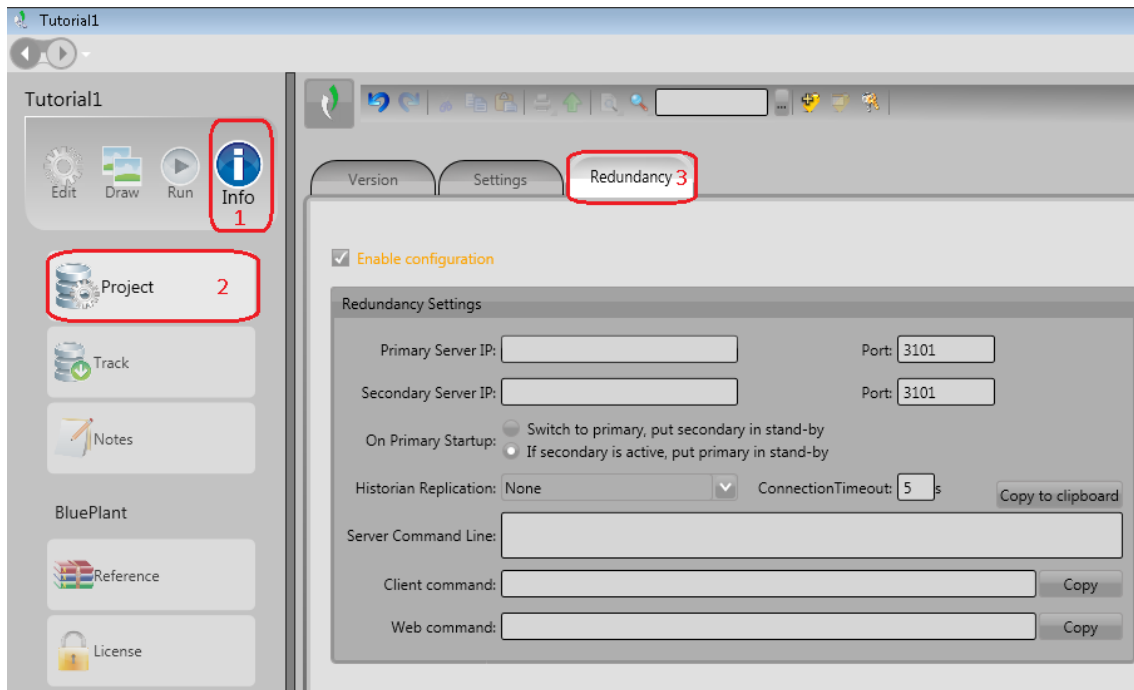


Figure 6-71. Redundancy Tab

Enable configuration: Option to enable redundancy in the project.

Primary Server IP: Field to set the IP address of the primary server.

Secondary Server IP: Field to set the IP address of the secondary server.

Port: Port number that the server will use to communicate with other modules and servers.

On Primary Startup: Defines the behavior of the primary server on its startup:

- Switch to primary, put secondary in stand-by
- If secondary is active, put primary in stand-by

Historian Replication: Defines which historical data will be replicated automatically.

- None
- Alarm Historian
- Tag Historian
- Alarm and Tag Historian

Connection Timeout: Specified spare time that server waits in case of communication failure between redundant servers and assumes as active.

Server Command Line: Command line used to startup the servers.

Client Command: Command line used to startup the clients.

Web command: URL used to access the project through Internet Explorer.

More information about the behavior of redundant servers can be found on Scenarios of Typical Systems.

Track

Tables

This tab of the Info menu, option Track, tracks the status of the configuration tables. The Figure 6-72 illustrates this selection.

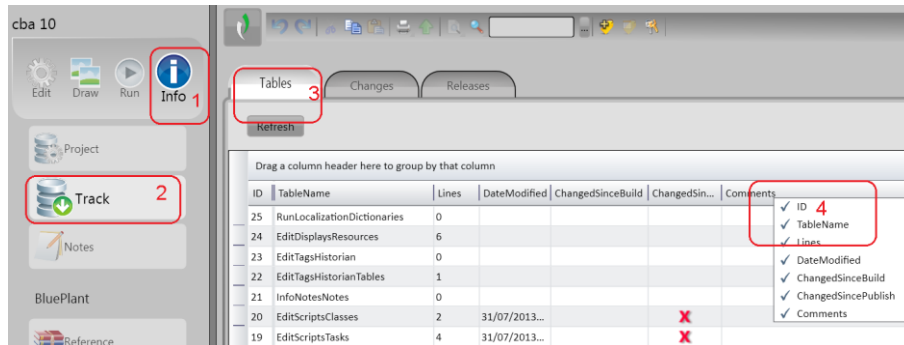


Figure 6-72. Tables Tab of the Info Menu, Track Option

The items that comprise the Info menu – Track – Tables are numbered and described below:

1. Select Info option
2. Select Track option
3. Select Tables option
4. Right-click on any of the column headers and select the options to be displayed

The items of the Table tab are explained as follows and can be seen on Figure 6-72. TableName

TableName

Project Table Name. Attributes: ReadOnly.

Lines

Current number of Lines in the configuration table. Attributes: ReadOnly.

DateModified

Last modification Date. Attributes: ReadOnly.

ChangedSinceBuild

Changes since the last build command execution. Attributes: ReadOnly.

ChangedSincePublish

Changes since the last publish command execution. Attributes: ReadOnly.

Comments

User-defined comments. Attributes: Editable.

Changes

This tab of the Info menu, option Track, enables the user to track the project modifications. Figure 6-73 illustrates this selection. The Tracking field defines when the changes are going to be saved.

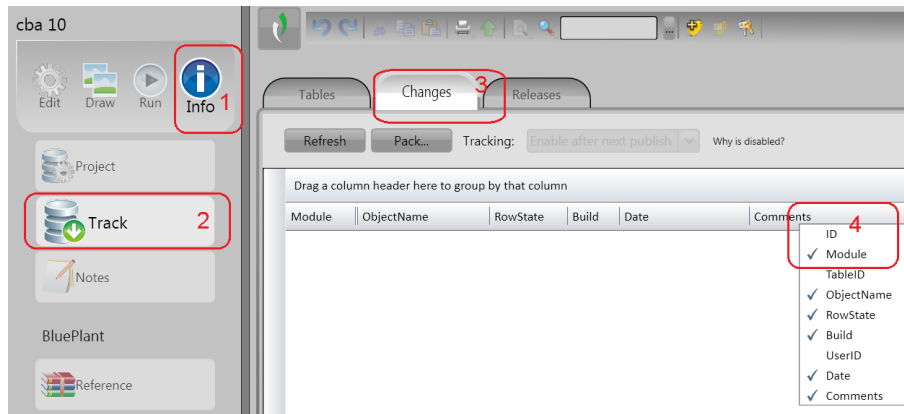


Figure 6-73. Changes Tab of the Info Menu, Track Option

The items that comprise the Info menu – Track - Changes are numbered in the preceding figure and are described below:

1. Select the Info option
2. Select the Track option
3. Select the Changes option
4. Right-click on any of the column headers and select the options to be displayed

Module

Module name where the object is defined. Attributes: ReadOnly.

ObjectName

Object name or row index. Attributes: ReadOnly.

RowState

Operation executed on the Object. Attributes: ReadOnly.

Build

Project Build number when operation is executed. Attributes: ReadOnly.

Date

Modification execution date. Attributes: ReadOnly.

Comments

User-defined comments. Attributes: Editable.

Releases

This tab of the Info menu, option Track, enables the user to track the published versions of the project. Figure 6-74 shows this selection.

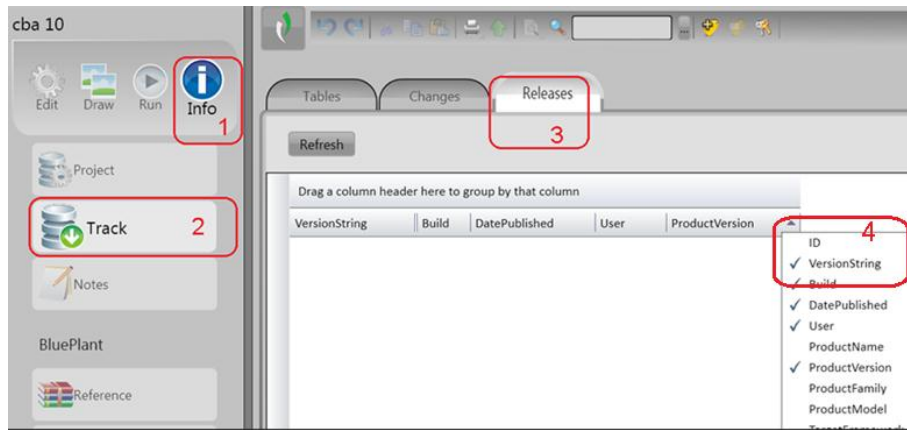


Figure 6-74. Releases Tab of the Info Menu, Tack Option

The items that comprise the menu Info – Track - Releases are numbered in the preceding figure and are described below:

1. Select the Info option
2. Select the Track option
3. Select the Releases option
4. Right-click on any of the column headers and select the options to be displayed

The Releases entries viewed in Figure 6-74 are detailed below.

Versionstring

Published version. Attributes: ReadOnly.

Build

Build number when publishing this version. Attributes: ReadOnly.

DatePublished

Version publishing date for this project. Attributes: ReadOnly.

User

User name who published this version. Attributes: ReadOnly.

Notes

This entry allows the user to create notes and "PostIt" type messages that can be visualized by all the Project design team members. Notes are visible on the desktop when editing the Project.

This interface provides a simple "message board" table for the creation of critical reminders.

When a user clicks on a "closed" message, it remains closed but is not deleted. In order to delete a message, the user must select the note on the table, double-click on it and then select the corresponding option ("Delete Selected Row"). One click in "IsOpen" on message row, opens it.

To implement messages to the Project Operators during its execution use the `OpenPopupNote()` function available in the options of the client runtime objects.

This tab of the Info menu, option Notes, edits the user notes concerning project configuration. Figure 6-75 illustrates it.

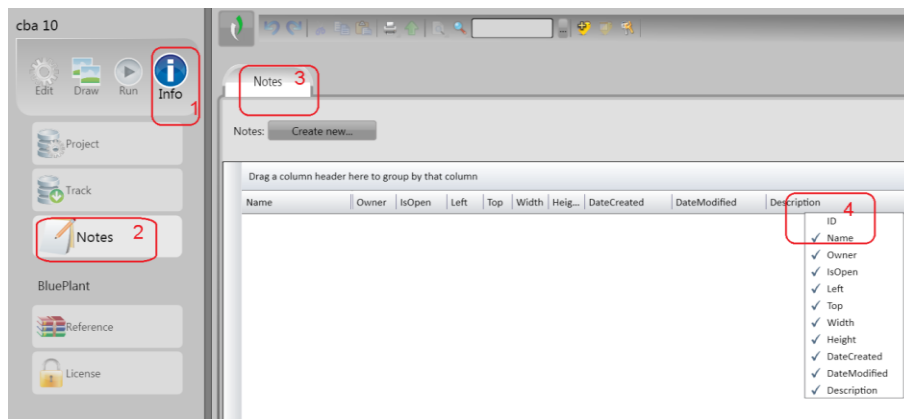


Figure 6-75. Notes Tab of the Info Menu, Notes Option

The items that comprise the Notes tab are numbered in the preceding figure and are described below:

1. Select the Info option
2. Select the Notes option
3. Select the Notes option
4. Right-click on any of the column headers and select the options to be displayed

Application Tools

This section explores the available tools in BluePlant.

- TStartup
- PropertyWatch
- TraceWindow
- ModuleInformation

Tstartup

When the user runs the project using the Manager or welcome interfaces, the TStartup Window pops up according to Figure 6-76.

Content: information about modules status (running, stopped, paused). The associated commands are:

- Play Button: starts a module execution
- Stop Button: stops a module execution, close the connections and releases the resources
- Pause Button: pauses a module execution, usually used by the server redundancy feature
- Shutdown Button: stops the BluePlant Server (TServer) and all the modules
- Watch Button: starts the PropertyWatch
- Trace Button: starts the TraceWindow
- Info Button: starts the ModuleInformation

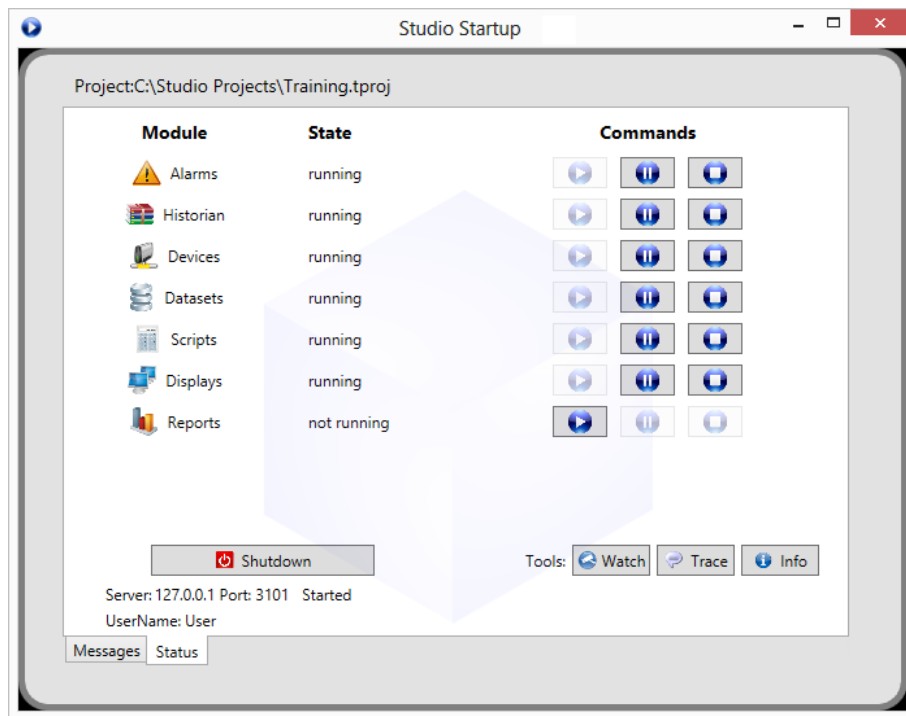


Figure 6-76. Startup Window

The user can call the Startup window using the command line. The available commands can be checked at the Command Lines (Tstartup) section.

PropertyWatch

The PropertyWatch can be used to access tags and Properties Domain server, and change its values as illustrated in Figure 6-77.

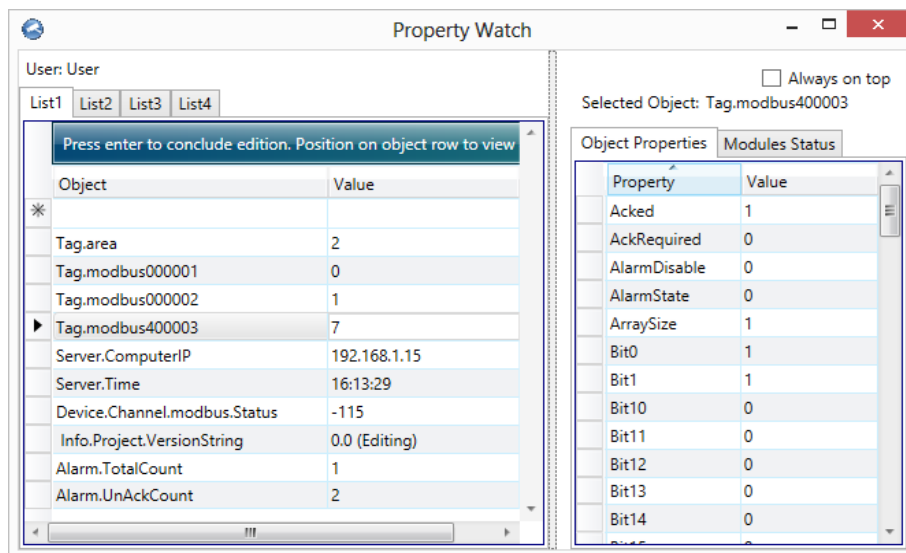


Figure 6-77. PropertyWatch

The user can call the PropertyWach using the command line. The available commands can be checked at the Command Lines (PropertyWatch) section.

TraceWindow

The TraceWindow is used to capture messages from the BluePlant Runtime modules (Figure 6-78). These messages are helpful to understand the runtime behavior.

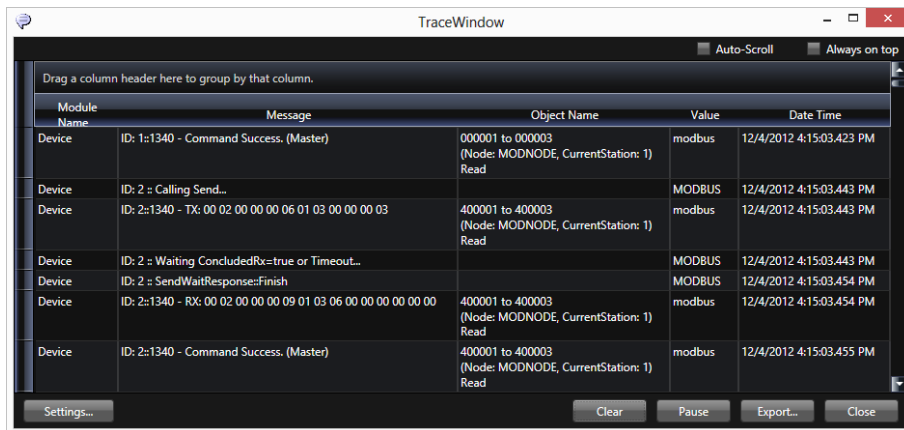


Figure 6-78. TraceWindow

Adjustments:

Settings according to Figure 6-79:

- Modules: determines what modules the TraceWindow receives messages from
- Type: determines the message types that the TraceWindow receives

The user can add a Tag or property in the ObjectName field, and click the Add button, so a notification will be sent to TraceWindow always the object changes its value.

Outputs:

- Grid: indicates the maximum number of files that appear in the grid
- File: a filename can be defined to save the messages

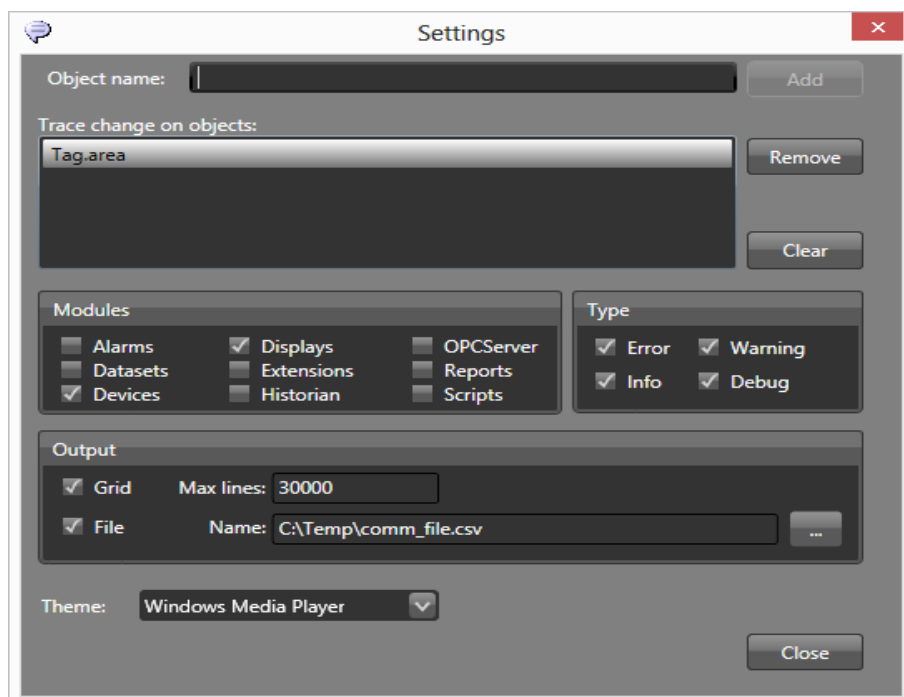


Figure 6-79. Configurations

The user can call the TraceWindow using the command line and specific parameters. The user can send messages to the project TraceWindow through the Info.Trace (string str) funcion.

TraceWindow Message

Selects the message types of the trace window. Available types:

- Error
- Information
- Warning
- Debugging

ModuleInformation

ModuleInformation contains advanced information about the modules execution as shown in Figure 6-80.

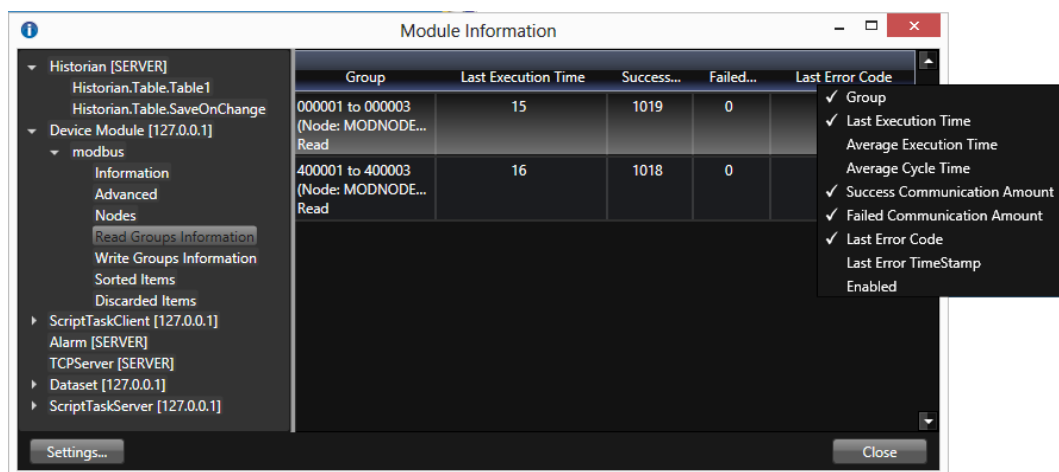


Figure 6-80. Module Informations

The user can call the ModuleInformation using the command line and specific parameters.

Runtime Objects

This feature allows the user to view all the opened elements, including Displays, Scripts and Reports. Runtime objects are in groups containing information about their specific functionalities, being called Namespaces. The runtime available Namespaces list:

- Namespace Tag
- Namespace Security
- Namespace Alarm
- Namespace Device
- Namespace Dataset
- Namespace Script
- Namespace Display
- Namespace Report
- Namespace Info
- Namespace Server
- Namespace Client

During project configuration, type directly in the field. The "Intellisense" will guide the user to the valid Namespaces for that field and the available members.

On the Code Editor (ScriptCodeEditor and DrawCodeBehind interfaces) and on TextOutput dynamic it is necessary to prefix the namespace with "@" symbol in order to avoid conflict with the .NET Namespaces.

Example:

On the dialogs and grids (Figure 6-81), use:

```
Tag.Analog1
Alarm.Group.Warning.TotalActive
Device.Node.Node1.Status
```

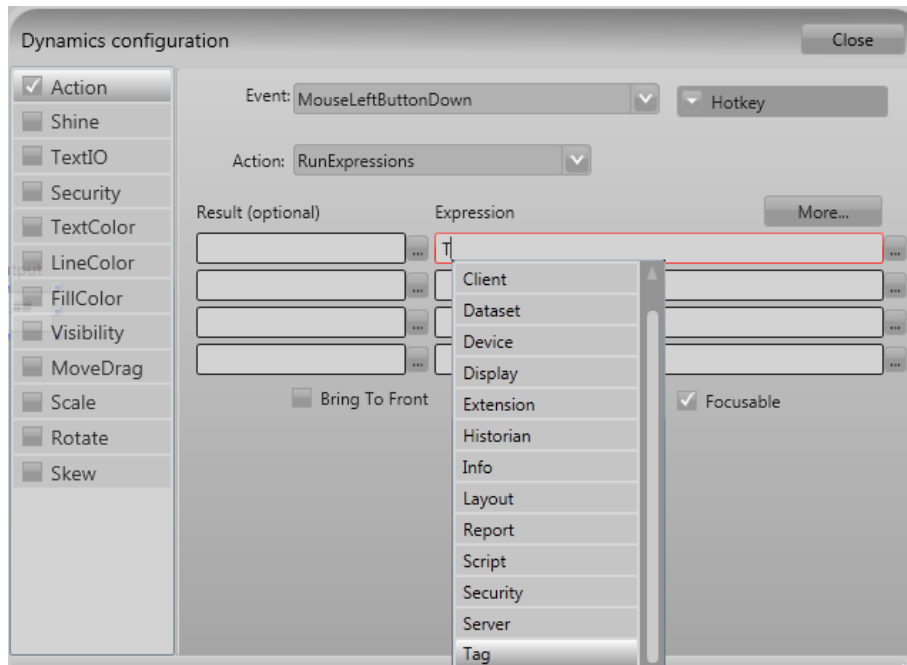


Figure 6-81. Namespaces in Dialogs

On the script code (Figure 6-82), use:

```
@Tag.Analog1
@Alarm.Group.Warning.TotalActive
@Device.Node.Node1.Status
```

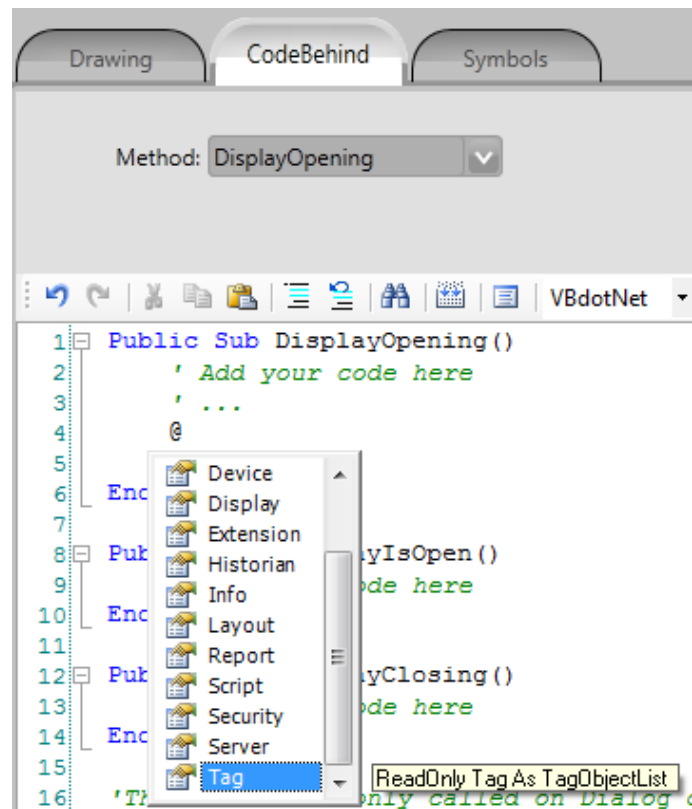


Figure 6-82. Namespaces in Scripts

Some fields on the Grids and Dialogs are allowed for one type of object (one namespace) only, e.g., "Tag" or "Display".

When the user type a Namespace in this field the Intellisense will guide him to the available objects.

Namespace Tag

All project realtime variables, or "Project Tags" as they usually are referred to in process automation contexts, are available at this Namespace. The Tags are created at the EditTagsObjects interface. Use the syntax Tag.Namespace to refer to a created Tag and Tag.namespace.Min to access Min Property. For each tag a type is defined for its value. The following built-in types are available:

- Digital
- AnalogInt
- AnalogDouble
- AnalogDecimal
- Text
- TDateTime
- Counter
- Timer
- Reference

The syntax Tag.<TagName> is available for all tag types. To know the type of a particular tag, it is possible to access additional properties and methods using the syntax: Tag.<TagName>.<PropertyName>. The TagObj element shows properties and methods available for all Tag Types. On the other hand, Analog shows common properties to all Analog tags (refers to the tag type for specific properties). The types created by user combined with built-in types are allowed.

ClassTagObj

Base classes to all tag objects.

ForceValue

Forces the object's value property to the value passed as a parameter. Parameters: object value.

Example:

```
@Tag.AnalogInt1.ForceValue(10);    (VB)
@Tag.AnalogInt1.ForceValue(10);    (C#)
```

Tostring

Returns a string that represents the current object. Example:

```
Dim s as string;                    (VB)
@Tag.AnalogInt1 = 33;
s = Tag.AnalogInt1.ToString();
string s;                            (C#)
@Tag.AnalogInt1 = 33;
s = Tag.AnalogInt1.ToString();
```

In this example, "s" will be evaluated as "33".

Alarm Disable

Gets or sets the Tag's alarm disable state. If 0 (zero), the AlarmDisable property is not active. In the case of a different value - not 0 (zero) - the AlarmDisable property is active. When the AlarmDisable property is not active, all the alarms associated to the current Tag are not treated. Example:

```
@Tag.AnalogDecimal1.AlarmDisable = 1;    (VB)
@Tag.AnalogDecimal1.AlarmDisable = 1;    (C#)
```

In this example, the alarms associated to the object AnalogDecimal1 will not be treated.

Alarm State

Gets or sets the Tag's alarm state. If 0 (zero): not in alarm state. A value not 0 (zero): in alarm state.

Example:

```
Dim alarmState as Int;                (VB)
alarmState = @Tag.AnalogDecimal1.AlarmState;
int alarmState;                        (C#)
alarmState = @Tag.AnalogDecimal1.AlarmState;
```

GetName

Gets the Tag's name. Example:

```
@Tag.ReferenceTagName.Link = @Tag.TagName.GetName();    (VB)
@Tag.ReferenceTagName.Link = @Tag.TagName.GetName();    (C#)
```

AlarmUnack

Gets or sets the current Tag's alarm acknowledgement state. If 0 (zero): current alarm acknowledged. If not 0 (zero): current alarm unacknowledged. Example:

```
Dim alarmStatus as Int;                (VB)
alarmStatus = @Tag.AnalogDecimal1.AlarmUnack;
int alarmStatus;                        (C#)
alarmStatus = @Tag.AnalogDecimal1.AlarmUnack;
```


Changed

Gets or sets the Tag's value changed state. If True: value changed. If False: value not changed.

Domain

Gets or sets the Tag's Domain property. If 0 (zero): Server. If 1: Client.

EngUnits

Gets or sets the engineering units used to quantify the Tag.

Format

Gets or sets the format of the Tag's value for display purposes.

Historian

Gets the HistoryItem object where the current tag is configured as TagName. If duplicated tags are allowed in the historian tables, then Historian will point to the last HistoryItem object where the current tag is configured as TagName. Example:

```
@Tag.AnalogInt1.Historian.Deadband = 3;      (VB)
@Tag.AnalogInt1.Historian.Deadband = 3;      (C#)
```

Locked

Gets or sets the Tag's locked state. When a Tag is locked, the value used in the processing originates from the LockValue property and not from the value property. If 0 (zero): not locked. A not 0 (zero) value: locked. Example:

```
@Tag.AnalogDouble1.Locked = 1;              (VB)
@Tag.AnalogDouble1.Locked = 1;              (C#)
```

Quality

Gets or sets the Tag's quality state. Default values:

Value	Quality
0	Bad
192	Good

Table 6-9. Quality Default Values

Example:

```
@Tag.AnalogInt1.Quality = 192;              (VB)
@Tag.AnalogInt1.Quality = 192;              (C#)
```

Retentive

Gets or sets the Tag's retentive property, which specifies if the tag's value property should be saved when the application shuts down. The saved value is then used as the startup value on the next application execution. If 0 (zero): not retentive. A value not 0 (zero): retentive. Example:

```
@Tag.AnalogInt1.Retentive = 1;              (VB)
@Tag.AnalogInt1.Retentive = 1;              (C#)
```

TimeStamp

Gets or sets the tag's timestamp property. Example:

```
Dim dt coo DateTimeOffset;           (VB)
dt = @Tag.AnalogDouble1.Timestamp;
DateTimeOffset dt;                 (C#)
dt = @Tag.AnalogDouble1.Timestamp;
```

ValueType

Gets the tag's ValueType property. Example:

```
Dim doubleType as Integer;           (VB)
doubleType = @Tag.AnalogDouble1.ValueType;
int doubleType;                     (C#)
doubleType = @Tag.AnalogDouble1.ValueType;
```

Visibility

Gets or sets the tag's visibility state. If 0 (zero): private, 1: protected and 2: public. Example:

```
Dim visibilityState as Integer;      (VB)
visibilityState = @Tag.AnalogDouble3.Visibility;
int visibilityState;                 (C#)
visibilityState = @Tag.AnalogDouble3.Visibility;
```

Class Digital

Runtime properties for digital class. Possible values: 0 = false and 1 = true. Equivalent in the script:

- C#: int32
- VB: int
- .NET: int

Toogle

Toggles the tag's value property between 0 (zero) and 1. If current value is 0 (zero), then new value is 1. If current value is 1, then new value is 0 (zero). Example:

```
Dim newValue as Byte;               (VB)
newValue = @Tag.Digital1.ToggleValue();
byte newValue;                      (C#)
newValue = @Tag.Digital1.ToggleValue();
```

Lock

Gets or sets the tag's LockedValue property. When a tag is locked, the value that is used for processing originates from the LockValue property and not from the Value property. Example:

```
@Tag.Digital1.LockValue = 1;        (VB)
@Tag.Digital1.LockValue = 1;        (C#)
```

State

Gets the digital tag's state property. If Value = 0 corresponds to State = False (VB) or State = false (C#). If Value = 1 corresponds to State = True (VB) or State = true (C#).

Value

Gets or sets the digital tag's value property. Valid values are 0 (zero) or 1. Example:

```
@Tag.Digital1.Value = 1;    (VB)  
@Tag.Digital1.Value = 1;    (C#)
```

Class Analog

Runtime properties for Analog class.

Member	Description
Bit0	Gets or sets the bit 0 of the tag's value
Bit1	Gets or sets the bit 1 of the tag's value
Bit10	Gets or sets the bit 10 of the tag's value
Bit11	Gets or sets the bit 11 of the tag's value
Bit12	Gets or sets the bit 12 of the tag's value
Bit13	Gets or sets the bit 13 of the tag's value
Bit14	Gets or sets the bit 14 of the tag's value
Bit15	Gets or sets the bit 15 of the tag's value
Bit16	Gets or sets the bit 16 of the tag's value
Bit17	Gets or sets the bit 17 of the tag's value
Bit18	Gets or sets the bit 18 of the tag's value
Bit19	Gets or sets the bit 19 of the tag's value
Bit2	Gets or sets the bit 2 of the tag's value
Bit20	Gets or sets the bit 20 of the tag's value
Bit21	Gets or sets the bit 21 of the tag's value
Bit22	Gets or sets the bit 22 of the tag's value
Bit23	Gets or sets the bit 23 of the tag's value
Bit24	Gets or sets the bit 24 of the tag's value
Bit25	Gets or sets the bit 25 of the tag's value
Bit26	Gets or sets the bit 26 of the tag's value
Bit27	Gets or sets the bit 27 of the tag's value
Bit28	Gets or sets the bit 28 of the tag's value
Bit29	Gets or sets the bit 29 of the tag's value
Bit3	Gets or sets the bit 3 of the tag's value
Bit30	Gets or sets the bit 30 of the tag's value
Bit31	Gets or sets the bit 31 of the tag's value
Bit4	Gets or sets the bit 4 of the tag's value
Bit5	Gets or sets the bit 5 of the tag's value
Bit6	Gets or sets the bit 6 of the tag's value
Bit7	Gets or sets the bit 7 of the tag's value
Bit8	Gets or sets the bit 8 of the tag's value
Bit9	Gets or sets the bit 9 of the tag's value
Hi	Gets or sets the tag's high limit
HiHi	Gets or sets the tag's highhigh limit
Lo	Gets or sets the tag's low limit
LoLo	Gets or sets the tag's lowlow limit

Table 6-10. Runtime Properties for Analog Class

Class Analog<T>

Runtime properties for Analog <T> class.

Deadband

Gets or sets the analog tag's deadband. Example:

```
@Tag.AnalogDouble1.Deadband = 5;      (VB)
@Tag.AnalogDouble1.Deadband = 5;      (C#)
```

LockValue

Gets or sets the analog Tag's lock value. Example:

```
@Tag.AnalogDouble1.LockValue = 50;      (VB)
@Tag.AnalogDouble1.LockValue = 50;      (C#)
```

Analog.Min

Gets or sets the analog Tag's maximum value. Example:

```
@Tag.AnalogDouble1.Max = 100;          (VB)
@Tag.AnalogDouble1.Max = 100;          (C#)
```

Analog.Min

Gets or sets the analog tag's minimum value. Example:

```
@Tag.AnalogDouble1.Min = 0;            (VB)
@Tag.AnalogDouble1.Min = 0;            (C#)
```

StartValue

Gets or sets the analog tag's start value. Example:

```
@Tag.AnalogDouble1.StartValue = 50;    (VB)
@Tag.AnalogDouble1.StartValue = 50;    (C#)
```

State

Gets or sets the analog Tag's state. If Value equals 0 (zero), then state is FALSE. If Value is not zero, then state is TRUE. Example:

```
Dim state as Boolean;                    (VB)
state = @Tag.AnalogDouble1.State;
bool state;
state = @Tag.AnalogDouble1.State;      (C#)
Value
@Tag.AnalogDouble1.Value = 55;          (VB) or
@Tag.AnalogDouble1 = 55;
@Tag.AnalogDouble1.Value = 55;          (C#) or
@Tag.AnalogDouble1 = 55;
```

Class AnalogInt

Runtime properties for AnalogInt class.

Class AnalogDecimal

Runtime properties for Analog Decimal Class.

Class AnalogDoble

Runtime properties for AnalogDouble Class.

Class Text

Runtime properties for Text class.

LockValue

Gets or sets the text tag's lock value. Example:

```
@Tag.Text1.LockValue = "Welcome";           (VB)
@Tag.Text1.LockValue = "Welcome";           (C#)
```

Value

Gets or sets the text tag's value. Example:

```
@Tag.Text1.Value = "My text";               (VB) or
@Tag.Text1 = "My text";
@Tag.Text1.Value = "My text";               (C#) or
@Tag.Text1 = "My text";
```

Class TDataTime

Runtime properties for TDataTime class.

LockValue

Gets or sets the TDataTime tag's lock value. Example:

```
@Tag.Text1.LockValue = "Welcome";           (VB)
@Tag.Text1.LockValue = "Welcome";           (C#)
```

Value

Gets or sets the TDataTime tag's value. Example:

```
@Tag.Text1.Value = "My text";               (VB) or
@Tag.Text1 = "My text";
@Tag.Text1.Value = "My text";               (C#) or
@Tag.Text1 = "My text";
```

Class Counter

Runtime properties for Counter class.

Event

Gets the Counter Tag's event. Possible values: "Change", "ChangeUp" and "ChangeDown".

Example:

```
Dim counter1Event as string;                 (VB)
counter1Event = @Tag.Counter1.Event;
string counter1Event;                         (C#)
counter1Event = @Tag.Counter1.Event;
```

Model

Gets the Counter Tag's model. Possible values: "Up" and "Down". Example:

```
Dim counter1Model as string;                 (VB)
counter1Model = @Tag.Counter1.Model;
string counter1Model;                         (C#)
counter1Model = @Tag.Counter1.Model;
```

Trigger

Gets or sets the Counter Tag's trigger. Example:

```
@Tag.Counter1.Trigger = "Tag.Digital1";      (VB)
@Tag.Counter1.Trigger = "Tag.Digital1";      (C#)
```

Class Timer

Runtime properties for Timer Class.

Interval

Gets or sets the Timer Tag's interval. The interval is a string that represents a time interval and is displayed in the format "hh:mm:ss.mmm". Example:

```
@Tag.Timer1.Interval = "0:0:10";           (VB)
@Tag.Timer1.Interval = "0:0:10";           (C#)
```

Model

Gets the Timer Tag's model. Possible values: "SquareWave", "Pulse" and "Comparer". Example:

```
Dim timerModel as string;                   (VB)
timerModel = @Tag.Timer1.Model;
string timerModel;                           (C#)
timerModel = @Tag.Timer1.Model;
```

Class Reference

Runtime properties for Reference Class.

Link

Gets or sets the Reference Tag's link. Example:

```
@Tag.Reference1.Link = @Tag.TagName.GetName(); (VB)
@Tag.Reference1.Link = @Tag.TagName.GetName(); (C#)
```

Class TDataTable

Runtime properties for TDataTable Class.

Initialize

Sets a new reference to the Table object. This method is only used internally.

Table

Gets a copy of the DataTable object. Example:

```
Dim dt as New TDataTable(parent, id);        (VB)
Dim table As DataTable;
table = dt.Table;
TDataTable dt = new TDataTable(parent, id);  (C#)
DataTable table;
table = dt.Table;
```

OverwriteOnUpdate

Gets or sets the OverwriteOnUpdate operand. This property is only used internally.

Update

Updates the Table object. This method is only used internally. Parameters: DataTable table

Class UserType

Runtime properties for UserType Class.

Namespace Security

Class ModuleSecurity

Runtime properties for ModuleSecurity objects.

GetPasswordHint

Returns (displays) the password hint for the selected user name. Parameter: string userName.

Example:

```
Dim pswHint As string = @Security.GetPasswordHint("User"); (VB)
string pswHint = @Security.GetPasswordHint("User"); (C#)
```

AddRuntimeUser

Adds a user to the Runtime user list. Parameters: string name; string permissionsStr; string password; string passwordHint; string policyStr; string profileEmail; string profilePhone and string profileCompleteName. Example:

```
@Security.AddRuntimeUser("User", "User", "psw", "remember", "Default",
"a@b.com", "67521855", "Alfred Burns"); (VB)
@Security.AddRuntimeUser("User", "User", "psw", "remember", "Default",
"a@b.com", "67521855", "Alfred Burns"); (C#)
```

GetListOfUserNames

Returns the list of Runtime user names separated by \n (newline). Example:

```
Dim userList as string; (VB)
userList = @Security.GetListOfUserNames();
string userList; (C#)
userList = @Security.GetListOfUserNames();
```

RemoveRuntimeUser

Removes a Runtime user from the Runtime user list. Parameter: string name. Example:

```
@Security.RemoveRuntimeUser("User"); (VB)
@Security.RemoveRuntimeUser("User"); (C#)
```

Permission

Gets the permission list. Refers to SecurityPermission objects. Example:

```
Dim permissionList as SecurityPermissionList; (VB)
permissionList = @Security.Permission;
SecurityPermissionList permissionList; (C#)
permissionList = @Security.Permission;
```

User

Gets the user list. Provides access to SecurityUser objects. Example:


```
Dim userList as SecurityUserList; (VB)
userList = @Security.User;
SecurityUserList userList; (C#)
userList = @Security.User;
```

Policy

Gets the Policy list. Example:

```
Dim policyList as SecurityPolicyList; (VB)
policyList = @Security.Policy;
SecurityPolicyList policyList; (C#)
policyList = @Security.Policy;
```

Class *SecurityPermission*

Runtime properties for SecurityPermission objects.

DateCreated

Gets the date and time that the SecurityPermission was created. Example:

```
Dim permissionDate as DateTime; (VB)
permissionDate = @Security.Permission.Administrator.DateCreated;
DateTime permissionDate; (C#)
permissionDate = @Security.Permission.Administrator.DateCreated;
```

DateModified

Gets the date and time that the SecurityPermission was modified. Example:

```
Dim permissionDate as DateTime; (VB)
permissionDate = @Security.Permission.Administrator.DateModified;
DateTime permissionDate; (C#)
permissionDate = @Security.Permission.Administrator.DateModified;
```

Description

Gets the description of the SecurityPermission. Example:

```
Dim permissionDescription as string; (VB)
permissionDescription = @Security.Permission.Administrator.Description;
int permissionDescription; (C#)
permissionDescription = @Security.Permission.Administrator.Description;
```

Edit

Gets the Edit permission. Example:

```
Dim editPermission as Integer; (VB)
editPermission = @Security.Permission.Administrator.Edit;
int editPermission; (C#)
editPermission = @Security.Permission.Administrator.Edit;
```

Name

Gets the SecurityPermission name. Example:

```
Dim permissionName as string; (VB)
permissionName = @Security.Permission.Administrator.Name;
string permissionName; (C#)
permissionName = @Security.Permission.Administrator.Name;
```

Run

Gets the run Permission. Example:

```
Dim runPermission as Integer;          (VB)
runPermission = @Security.Permission.Administrator.Run;
int runtPermission;                   (C#)
runPermission = @Security.Permission.Administraytor.Run;
```

Class SecurityUser

Runtime properties for SecurityUser objects.

DateCreated

Gets the date and time the SecurityUser was created. Example:

```
Dim userDate as DateTime;             (VB)
userDate = @Security.User.Guest.DateCreated;
DateTime userDate;                   (C#)
permissionDate = @Security.User.Guest.DateCreated;
```

DateModified

Gets the date and time the SecurityUser was modified. Example:

```
Dim userDate as DateTime;             (VB)
userDate = @Security.User.Guest.DateModified;
DateTime userDate;                   (C#)
userDate = @Security.User.Guest.DateModified;
```

PolicyName

Gets the SecurityUser's policies name. Example:

```
Dim userPolicyName as string;         (VB)
userPolicyName = @Security.User.Guest.PolicyName;
string userPolicyName;               (C#)
userPolicyName = @Security.User.Guest.PolicyName;
```

SecurityUser.Blocked

Gets the SecurityUser's blocked state. Example:

```
Dim blockedState as Boolean;          (VB)
blockedState = @Security.User.Guest.Blocked;
bool blocked;                         (C#)
blocked = @Security.User.Guest.Blocked;
```

SecurityUser.Deleted

Gets the SecurityUser's deleted state. Example:

```
Dim deletedState as Boolean;          (VB)
deletedState = @Security.User.Guest.Deleted;
bool deleted;                         (C#)
deleted = @Security.User.Guest.Deleted;
```

SecurityUser.Name

Gets the SecurityUser's name. Example:

```
Dim userName as string;              (VB)
```

```
userName = @Security.User.Guest.Name;  
string userName;           (C#)  
userName = @Security.User.Guest.Name;
```

SecurityUser.PasswordHint

Gets the SecurityUsers password hint. Example:

```
Dim pswHint as string;      (VB)  
pswHint = @Security.User.Guest.PasswordHint;  
string pswHint;            (C#)  
pswHint = @Security.User.Guest.PasswordHint;
```

SecurityUser.Permissions

Gets the SecurityUser's permissions. Example:

```
Dim userPermissions as Long; (VB)  
userPermissions = @Security.User.Guest.Permissions;  
long userPermissions;       (C#)  
userPermissions = @Security.User.Guest.Permissions;
```

SecurityUser.PermissionsName

Gets the SecurityUser's permissions name. Example:

```
Dim permissionsName as string; (VB)  
permissionsName = @Security.User.Guest.Permissions;  
string permissionsName;       (C#)  
permissionsName = @Security.User.Guest.Permissions;
```

SecurityUser.Policy

Gets the SecurityUser's policies. Example:

```
Dim userPolicy as Long;      (VB)  
userPolicy = @Security.User.Guest.Policy;  
long userPolicy;            (C#)  
userPolicy = @Security.User.Guest.Policy;
```

SecurityUser.Profile

Gets the SecurityUser's profile. Example:

```
Dim userProfile as string;   (VB)  
userProfile = @Security.User.Guest.Profile;  
string userProfile;         (C#)  
userProfile = @Security.User.Guest.Profile;
```

Namespace Alarm

Runtime objects and methods related to the alarm module.

Class AlarmGroup

Runtime properties for AlarmGroup objects.

AckAll

Acknowledge all alarm items that belong to this group. Toggle the set property to AckAll. Allowed values: 0 (acknowledgement if the value was 1) and 1(acknowledgement if the value was 0).

Example:

```
@Alarm.AckAll = (@Alarm.AckAll==0) ? 1 : 0;
```

AckRequired

Get the acknowledgment of the defined required operator. Allowed values: 0 (no Acknowledgement required) and 1 (acknowledgement required). Example:

```
int AckReq = @Alarm.Group.Critical.AckRequired;  
se (AckReq == 1) @Alarm.AckAll = (@Alarm.AckAll==0) ? 1 : 0;
```

Colors

Represents the foreground and background colors for each alarm state. This property represents the value that was configured in the Colors column.

Description

AlarmGroup description configured in EditAlarmsGroups. Example:

```
@Tag.string = @Alarm.Group.Critical.Description;
```

Disable

Disable/Enable an alarm group. Allowed values: 0 (enable an Alarm Group) and 1 (disable an Alarm Group). Example:

```
@Alarm.Group.Critical.Disable = 1;
```

Id

Gets the ID of an Alarm Group. Example:

```
@Tag.Int = @Alarm.Group.Critical.Id;
```

LogEvents

Get the type of Historian archiving on Alarm events. Allowed values:

- 0 - None
- 1 - Active
- 2 - ActiveAck
- 3 - ActiveNorm
- 4 - All

Example:

```
@Tag.Int = @Alarm.Group.Critical.LogEvents;
```

PriorityItem

Return the highest priority Alarm item of an Alarm Group. Example:

```
@ALARM.PRIORITYITEM PITEM = @ALARM.GROUP.CRITICAL.PRIORITYITEM;
```

Show

Gets a configured value to display/not display an alarm. Allowed values:

- 0 - None
- 1 - List

Example:

```
@Tag.Int = @Alarm.Group.Critical.Show;
```

Sound

Property to describe if the Alarm Group Sound is enabled or disabled. Default values:

- 0 - None
- 1 - Beep

Example:

```
@Tag.Int = @Alarm.Group.Critical.Sound;
```

TotalCount

Get the number of Active Alarms. Example:

```
@Tag.Int = @Alarm.Group.Critical.TotalCount;
```

UnAckCount

Get the number of Unacknowledge Alarms. Example:

```
@Tag.Int = @Alarm.Group.Critical.UnAckCount;
```

Class AlarmItem

Runtime properties for Alarm Items objects.

AckTime

Time that an Alarm is set at "Acknowledge".

ActiveTime

Time that an Alarm Item started.

Alarm

Property to check if an Alarm Item is active.

ColorBG

Configured Background Color of an Alarm Item.

ColorFG

Configured Foreground Color of an Alarm Item.

Condition

Configures evaluation condition to generate an Alarm Item. Allowed values:

- Hi
- HiHi
- Lo
- LoLo
- RateOfChange
- DeviationMinor
- DeviationMajor
- Equal
- GreaterThan
- GreaterEqual
- LessThan
- LessEqual
- Changed
- ChangedUp
- ChangedDown

Deadband

Deadband of a defined alarm item. Configures the inactivity time of the alarm item. The value is chosen by user.

Disable

Property to disable/enable an alarm item. Allowed values:

- 0 - Enable
- 1 - Disable

Group

A group to which an Alarm Item belongs.

Id

Object ID (Internal Use).

Limit

Get defined value to evaluate the Alarm item conditions.

Message

Get the configured message to display when Alarm occurs.

NormTime

Time that an Alarm takes to get back to "Normal".

Priority

Get an Alarm Item Priority. The value is chosen by user.

Setpoint

Get the defined value to evaluate Alarm Item conditions.

SetPointDeadband

Represents the dead band for the SetPoint property. Used for the following alarm conditions: DeviationMinor and DeviationMajor.

State

Get an alarm item state. It can be: Active, Acknowledge, Normalized and TagName. The last one gets the defined TagName that will be evaluated to generate an alarm item.

UnAck

Property set/get an unacknowledged alarm item. Allowed values:

- 0 - Unacknowledge alarm item
- 1 - Acknowledge alarm item

Class ModuleAlarm

Runtime methods and properties for the alarm module.

AckAll

Trigger this property to acknowledge all Alarms.

BeepState

Indicates if beep is set to ON (value = 1) on the client computer.

Group

Provides access to Alarm Group objects.

InitializationMessage

Get/set initial message to display.

Item

Access to Alarm Item objects.

LastErrorMessage

Contains description message of the most recent error occurred in the Alarm Module.

LastStoredTimeStamp

Contains the TimeStamp of the most recent error that occurred in the Alarm Module.

PriorityItem

Reference to the highest priority online Alarm Item.

QueryActive

TDataTable object with the current active Alarm list.

TotalCount

Total count of active alarms.

UnAckCount

Total count of Alarms that require acknowledgment.

Namespace Device

Runtime objects and methods related to the Device.

Class DeviceAccessType

Runtime properties for DeviceAccessType objects.

AcceptUnsolicited

When set to true, the device points are allowed to receive unsolicited messages. The device channel must have the AcceptUnsolicited property set to "true" to allow unsolicited messages.

ReadOnStartup

When set to true, the device point will be read on the startup of the Devices Module.

ReadPooling

When set to true, it indicates that the Pooling read is enabled for this device point.

ReadPoolingRate

Indicates the Pooling rate for the group of points.

WriteEnable

When set to true, the device point is written to the device when its value changes.

Class DeviceChannel

Runtime properties for DeviceChannel objects.

Activity

Activity indication. The device module toggles this value to each operation completed in this Channel.

LastErrorCode

Last (most recent) Error Status Code that occurred in this channel. Allowed values:

Value	Description
0	Success
-1	BuildCommandException
-2	ParseCommandUnsolicitedException
-3	ParseReplyException
-4	BuildReplyUnsolicitedException
-5	ChannelException
-6	NodeException
-100	Base Send Error
-101	Base SendAndWait Error
-102	TCP Create Error 1
-103	TCP Create Error 2
-104	TCP Create SocketError
-105	TCP Connect Callback Error
-106	TCP Receive Error
-107	UDP Create Error
-108	UDP Receive Error
-109	Serial Create Error
-110	Serial Receive Error
-111	TCP NotConnected
-112	Start message timeout
-113	Receiving bytes timeout
-114	End message timeout
-115	Connect timeout
-200	ProtocolError
-201	InvalidProtocol
-202	InvalidStation
-203	InvalidCommand
-204	InvalidMsgSequence
-205	InvalidChecksum
-206	InvalidAddress
-207	InvalidModifiers

Table 6-11. Error Status Code for Channel or Node

Note:

Positive values are specific protocol error codes.

LastErrorDateTime

TimeStamp of the last (most recent) error in this channel.

Status

Current status for this channel according to Table 6-11.

Class DeviceNode

Runtime properties for DeviceNode objects.

Activity

Activity indication. The device module toggles this value to each operation completed in this node.

BackupStation

Current backup station for this Node.

IsBackup

Indication of active Backup station .

IsPrimary

Indication of active PrimaryStation.

IsRedundancyEnabled

Node redundancy indication.

LastErrorCode

Last Error Code status (most recent) that occurred in this node according to Table 6-11.

LastErrorDateTime

DateTime for the last (most recent) error in this Node.

PrimaryStation

Current primary station for this Node.

Status

Current status for this Node according to Table 6-11.

Class ModuleDevice

Runtime methods and properties for the Device Module.

AccessType

Access to DeviceAccessType objects.

Channel

Access to DeviceChannel objects.

Node

Access to DeviceNode objects.

Namespace Dataset

Runtime objects and methods related to the database.

Class DatasetDB

Runtime properties for DatasetDB objects.

ConnectionString

String used to connect with the database.

Database

Name of the DB object database.

Id

Object Identification (internal use).

LogonName

Logon name used to connect with the database.

Provider

Selected Database provider.

Class DatasetFile

Runtime properties for DatasetFile objects.

LoadCommand

Loads the values of the tags configured in the Objects property from the file indicated by the FileName property. Parameter: string statusMessage (message with the status of the load command).

SaveCommand

Saves the values of the tags configured in the Objects property from the file indicated by the FileName property. Parameter: string statusMessage (message with the status of the save command).

Completed

The value of this property is incremented when an operation is concluded.

Description

Gets the description of the configured DatasetFile.

Disable

Disables the commands to the DatasetFile when the value is greater than zero. Default values:

- 0 = Enables the commands to the DatasetFile
- 1 = Disables the commands to the DatasetFile

FileName

Complete path of the file that will be created or loaded. This property represents the value that was configured in the FileName column.

FileType

Indicates the file format. Allowed values:

- ASCII = 0
- Unicode = 1
- XML = 2

This property represents the value that was configured in the FileType column.

Id

Object Identification (internal use).

LastStatus

Gets the status of the last (most recent) asynchronous operation. Default values:

- Zero = Success
- Different than zero = Error code

LastStatusMessage

Gets the status message of the last (most recent) asynchronous operation.

Load

Sends a Load asynchronous command when the value is changed. The value of the LoadExecuted property is changed when the operation is concluded. Example:

```
if (@Dataset.File.File1Unicode.Load.Equals(0))
    @Dataset.File.File1Unicode.Load = 1;
else
    @Dataset.File.File1Unicode.Load = 0;
```

LoadExecuted

The value of this property is changed when the Load asynchronous command is completed.

Objects

Contains the tags and indexes to be saved or loaded. The default values are: TagName (represents the tag name and the start index can also be specified) and Index (indicating the index - if the Tag is an array and the start index is specified). Example:

TagName	Index
Tag.doubleArray[0]	10
Tag.textArray[2]	5
Tag.SelectStatusMsg	

Table 6-12. Example of Objects Option

Save

Sends a Save command when the value is changed. The value of the SaveExecuted property is changed when the operation is concluded. Example:

```
if (@Dataset.File.File1Unicode.Save.Equals(0))
    @Dataset.File.File1Unicode.Save = 1;
```

```
else
    @Dataset.File.File1Unicode.Save= 0;
```

SaveExecuted

The value of this property is changed when the asynchronous save command is completed.

Class *DatasetQuery*

Runtime properties for *DatasetQuery* objects.

ExecuteCommand

Execute a synchronous command according to the *SqlStatement*. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

ExecuteCommandWithStatus

Execute a synchronous command according to the *SqlStatement* (displays a status message). Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

Parameters: out string *statusMessage* (message indicates the status of the command Next).

NextCommand

Executes a synchronous Next command that increments the value of the *CursorIndex* property. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

The tags configured in the mapping column will receive the value of the next row.

NextCommandWithStatus

Executes a synchronous Next command that increments the value of the *CursorIndex* property, and displays a status message. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

Associated parameter: out string *statusMessage* (message that indicates the status of the command Next). The tags configured in the mapping column will receive the value of the next row.

SelectCommand

Executes a synchronous Select command, according to the *SqlStatement*. Returned values: *DataTable*, if success; otherwise, null. The tags configured in the mapping column will receive the value of the first row. Example:

```
DataTable dataTable = @Dataset.Query.query1.SelectCommand();
if (dataTable != null && dataTable.Rows.Count > 0)
{
    @Tag.firstItem = dataTable.Rows[0]["Item"].ToString();
}
```

```
}
```

SelectCommandWithStatus

- Executes a synchronous Select command according to the SqlStatement, and displays status information. Returned values: DataTable, if success; otherwise, null. The first associated parameter is: out in status. The operation status is:
- Zero = Success
- Different than zero = Error code

The second associated parameter is: out string statusMessage (message that indicates the status of the Select command). The tags configured in the mapping column will receive the value of the first row.
Example:

```
int status;
string statusMessage;
DataTable dataTable;

    dataTable = @Dataset.Query.query1.SelectCommandWithStatus(out status,
    out statusMessage);
    if (status == 0 && dataTable != null && dataTable.Rows.Count > 0)
    {
        @Tag.firstItem = dataTable.Rows[0]["Item"].ToString();
    }
}
```

AsyncContents

Contains the TDataTable that resulted from one of the asynchronous commands, such as Select or Update.

Completed

The value of this property is incremented when an asynchronous operation is concluded.

CursorIndex

Defines the current row position in the resulting DataSetTable.

DB

Gets the DB configured in EditDatasetsTables.

Description

Gets the description of the DataSetQuery.

Disable

Disables the commands to the DatasetQuery when the value is greater than zero. Return values:

- 0 = Enables the commands to the DatasetQuery
- 1 = Disables the commands to the DatasetQuery

Execute

The value of the ExecutedCompleted property is changed when the operation is concluded. Example:

```
if (@Dataset.Query.Query1.Execute.Equals(0))
    @Dataset.Query.Query1.Execute = 1;
else
    @Dataset.Query.Query1.Execute = 0;
```

ExecuteCompleted

The value of this property is changed when the operation is completed.

Id

Object Identification (internal use).

LastStatus

Gets the status of the last (most recent) asynchronous operation. Returned values:

- Zero = Success
- Different than zero = Error code

LastStatusMessage

Gets the status message of the last (most recent) asynchronous operation, where an empty string indicates success.

Mapping

Gets the mapping of the resulting DataTable columns with the tags. Returned values:

TagName	Column
int_Id	ID
txt_Name	Name
txt_Description	Description

Table 6-13. DataTable Columns Mapping

Result:

```
Tag.int_Id=ID;
Tag.txt_Name=Name;
Tag.txt_Description=Description;
```

Next

Sends an asynchronous Next command when the value is changed. The value of the NextExecuted property is changed when the operation is concluded. Example:

```
if (@Dataset.Query.Query1.Next.Equals(0))
```

```
@Dataset.Query.Query1.Next = 1;  
else  
    @Dataset.Query.Query1.Next = 0;
```

NextExecuted

The value of this property is changed when the asynchronous Next command is completed.

RowCount

Gets the total number of rows in the resulting DataSetTable.

Select

Sends an asynchronous Select command when the value is changed. The value of the SelectExecuted property is changed when the operation is concluded. Example:

```
if (@Dataset.Query.Query1.Select.Equals(0))  
    @Dataset.Query.Query1.Select= 1;  
else  
    @Dataset.Query.Query1.Select= 0;
```

SelectExecuted

The value of this property is changed when the asynchronous Select command is completed. The value of the SelectExecuted property is changed when the operation is concluded.

SqlStatement

Defines the SQL command to be executed.

Class DataSetTable

Runtime properties for DataSetTable objects.

DeleteCommand

Deletes the current row of the DataSetTable. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

The index of the current row is defined by the CursorIndex property.

DeleteCommandWithStatus

Deletes the current row of the DataSetTable and provides a status message. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

Parameters: out string statusMessage (message indicates the status of the delete command, where an empty string signifies success). The index of the current row is defined by the CursorIndex property.

InsertCommand

Inserts the values of the tags configured in the mapping column into the DataSetTable at the position indicated by the CursorIndex property. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

InsertCommandWithStatus

Inserts the values of the tags configured in the mapping column into the DatasetTable at the position indicated by the CursorIndex property, and provides a status message. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

Associated parameters: out string statusMessage (message that indicates the status of the insert command, where an empty string signifies success).

NextCommand

Executes a synchronous Next command that increments the value of the CursorIndex property. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

The tags configured in the mapping column will receive the value of the next row.

NextCommandWithStatus

Executes a synchronous Next command that increments the value of the CursorIndex property, and provides a status message. Returned values (status of the operation):

- Zero = Success
- Different than zero = Error code

Associated parameters: out string statusMessage (message that indicates the status of the Next command, where an empty string signifies success). The tags configured in the mapping column will receive the value of the next row.

SelectCommand

Executes a synchronous Select command on the DatasetTable. Returned values: DataTable, if success; otherwise, null. The tags configured in the mapping column will receive the value of the first row. Example:

```
DataTable dataTable = @Dataset.Table.table1.SelectCommand();
    if (dataTable != null && dataTable.Rows.Count > 0)
    {
        @Tag.firstItem = dataTable.Rows[0]["Item"].ToString();
    }
```

SelectCommandWithStatus

Executes a synchronous Select command on the DatasetTable, and provides status messages. Returned values: DataTable, if success; otherwise, null.

- Parameter 1: out int status. Status of the select operation:
- Zero = success
- Different than zero = error code

Parameter 2: out string statusMessage (message indicates the status of the Select command, where an empty string signifies success). The tags configured in the mapping column will receive the value of the first row. Example:

```
int status;
string statusMessage;
DataTable dataTable;
dataTable = @Dataset.Table.table1.SelectCommandWithStatus(out status, out
statusMessage);

    if(status == 0 && dataTable != null && dataTable.Rows.Count > 0))
    {
        @Tag.firstItem = dataTable.Rows[0]["Item"].ToString();
    }
}
```

UpdateCommand

Updates the current row of the DataSetTable with the values of the tags configured in the mapping column. Returned values: DataTable, if success; otherwise, null. The index of the current row is defined by the CursorIndex property.

UpdateCommandWithStatus

Updates the current row of the DataSetTable with the values of the tags configured in the mapping column, and provides status information. Returned values: DataTable, if success; otherwise, null.

Parameter 1: out int status. Status of the update operation:

- Zero = success
- Different than zero = error code

Parameter 2: out string statusMessage (message indicates the status of the update command, where an empty string signifies success). The index of the current row is defined by the CursorIndex property.

Access

Access Type of the DataSetTable. Returned values:

- 0 - Read
- 1 - Insert
- 2 - Read/Write
- 3 - Unrestricted

AsyncContents

Contains the TDataTable resulting from one of the asynchronous commands, such as <i>Select or Update.

Completed

The value of this property is incremented when an asynchronous operation is concluded.

CursorIndex

Defines the current row position in the DataSetTable.

DB

Gets the DB configured in EditDatasetsTables.

Delete

Sends an asynchronous Delete command when the value is changed.

The value of the DeleteExecuted property is changed when the operation is concluded. Example:

```
if (@Dataset.Table.table1.Delete.Equals(0))
    @Dataset.Table.table1.Delete= 1;
else
    @Dataset.Table.table1.Delete = 0;
```

DeleteExecuted

The value of this property is changed when the asynchronous Delete command is completed.

Description

Gets the Description of the DataSetTable.

Disable

Disables the commands to the DataSetTable when the value is greater than zero. Returned values:

- 0 = Enables the commands to the DataSetTable
- 1 = Disables the commands to the DataSetTable

Id

Object identification (internal use).

Insert

Sends an asynchronous Insert command when the value is changed. The value of the InsertExecuted property is changed when the operation is concluded. Example:

```
if (@Dataset.Table.table1.Insert.Equals(0))
    @Dataset.Table.table1.Insert= 1;
else
    @Dataset.Table.table1.Insert = 0;
```

InsertExecuted

The value of this property is changed when the asynchronous Insert command is completed.

LastStatus

Gets the status of the last (most recent) asynchronous operation. Returned values:

- Zero = success
- Different than zero = error code

LastStatusMessage

Gets the status message of the last (most recent) asynchronous operation, where an empty string signifies success.

Mapping

Gets the mapping of the DataTable columns with the Tags. Example:

TagName	Column
int_Id	ID
txt_Nome	Name

txt_Description	Description
-----------------	-------------

Table 6-14. Mapping**Result:**

```
Tag.int_Id=ID;
Tag.txt_Name=Name;
Tag.txt_Description=Description;
```

Next

Sends an asynchronous Next command when the value is changed. The value of the NextExecuted property is changed when the operation is concluded. Example:

```
if (@Dataset.Table.table1.Next.Equals(0))
    @Dataset.Table.table1.Next= 1;
else
    @Dataset.Table.table1.Next = 0;
```

NextExecuted

The value of this property is changed when the asynchronous Next command is completed.

RowCount

Gets the total number of rows in the DataSetTable.

Select

Sends an asynchronous Select command when the value is changed. The value of the SelectExecuted property is changed when the operation is concluded.

SelectExecuted

The value of this property is changed when the operation is completed.

Example:

```
if (@Dataset.Table.table1.Select.Equals(0))
    @Dataset.Table.table1.Select = 1;
else
    @Dataset.Table.table1.Select = 0;
```

TableName

Gets the name of the DataTable.

Update

Sends an asynchronous Update command when the value is changed. The value of the UpdateExecuted property is changed when the operation is concluded. Example:

```
if (@Dataset.Table.table1.Update.Equals(0))
    @Dataset.Table.table1.Update = 1;
else
    @Dataset.Table.table1.Update = 0;
```

UpdateExecuted

The value of this property is changed when the asynchronous Update command is completed.

WhereCondition

Defines the Where condition for the DatasetTable. The SQL WHERE clause is used to select data conditionally. Example:

Type	Price(\$)	DateAdded
1	27	10/02/2008
1	120	07/10/2010
2	50	12/01/2009

Table 6-15. SQL WHERE

```
@Dataset.Table.table1.WhereCondition = "DateAdded > '01/01/2010'";
```

The command will return only the items where the DateAdded is greater than 01/01/2010.

Class ModuleDataset

Runtime methods and properties for the Dataset Module.

DB

Provides access to DatasetDB objects.

File

Provides access to DatasetFile objects.

Query

Provides access to DatasetQuery objects.

Table

Provides access to the DatasetTable objects.

Namespace Script

Runtime objects and methods related to the Script.

Class ModuleScript

Runtime methods and properties for the Script module.

RunTasksSimultaneous

Allows simultaneous execution of tasks (Multi-Threading). Allowed values:

- 0 - Simultaneous execution NOT allowed
- 1 - Simultaneous execution allowed

Task

Access to ScriptTask objects.

Syntax:

```
Script.Task.<TaskName>.
```

UserClass

Access to ScriptUserClass objects.

Syntax:

```
Script.UserClass.<UserClassName>.
```

Class ScriptTask

Script Tasks class.

Description

Gets the ScriptTask Description configured in EditScriptsTasks.

Disable

Disables the ScriptTask execution when the value is greater than zero. Default values:

- Zero = success
- Different than zero = error code

Domain

Indicates the ScriptTask Domain. Allowed values: 0 - Server and 1 - Client. If Server: the ScriptTask runs in the Server context, it cannot access client objects i.e., displays or client domain tags. If Client: the ScriptTask runs in each client where the ModuleScript is running.

ErrorMessage

Contains the message with the last (most recent) error that occurred in the script execution.

Event

Reserved for future use.

ExecutionsCount

Gets the amount of executions since the module has started.

LastCPUTime

Gets the CPU time used during the last (most recent) execution of the script.

LastExecution

Contains the TimeSpan of the last (most recent) execution of the script.

Period

Period of time required to execute a Task. This property represents the value configured in EditScriptsTasks.

Running

Indicates if the current script is running. Default values:

- 0 - ScriptTask is NOT running
- 1 - ScriptTask is running

StopExecutionOnError

Indicates if the script execution must be stopped if an error occurs.

Trigger

Gets the Tag or Object that triggers the Task execution. This property represents the value configured in EditScriptsTasks.

Namespace Display

Runtime objects and methods related to the application displays.

Class Display

Runtime properties related to the Display objects. Syntax: Display.<DisplayName>

Close

Closes the selected display.

Open

Opens the selected display.

Description

Gets the description of the display, configured in EditDisplaysDisplays.

Id

Object identification (internal use).

IsOpened

Indicates if the selected display is opened.

Class Layout

Runtime properties for Layout objects. Syntax: Layout.<LayoutName>.

OpenCommand

Opens the selected Layout.

Description

Gets the description of the layout, configured in EditDisplaysLayouts.

Id

Object identification (internal use).

IsOpened

Indicates if the selected layout is opened.

Namespace Report

Runtime objects and methods related to the Report Module.

Class ModuleReport

Access to <ReportItem> objects.

Class ReportItem

Runtime properties for Report objects.

OpenCommand

Opens the selected report. Return values:

- True = success
- False = failed

The ReportViewer can be used to view the content of the report.

SaveCommand

Saves the selected report into the path indicated by the SaveFileName property. Return values:

- True = success
- False = failed

Append

Indicates whether the report will be overwritten or appended on Save commands. Allowed values:

- 0 – Report overwritten on Save commands
- 1 – Report appended on Save commands

Completed

The value of this property is incremented when a Save or Load operation is concluded.

Description

Gets the ReportItem description configured in EditReportsReports.

Disable

Disables the ReportItem operations when the value is greater than zero. Returned values:

- Zero = Enables the ReportItem
- Greater than zero = Disables the ReportItem

Id

Object identification (internal use).

LastStatus

Gets or sets the status of the last ReportItem processed. Allowed values:

- Success = 0
- InvalidMode = 1
- Disabled = 2
- NoObjectsConfigured = 3
- ReportException = 4
- InvalidContent = 5
- ModuleStopped = 6
- ModulePaused = 7

LastStatusMessage

Gets or sets the status message of the last ReportItem processed. Default values:

- "Success"
- "InvalidMode"
- "Disabled"
- "NoObjectsConfigured"
- "ReportException"
- "InvalidContent"
- "ModuleStopped"
- "ModulePaused"

OpenExecuted

Gets or sets the OpenExecuted state in report. Allowed values:

- 0 (zero) - Open Command not executed
- 1 - Open Command executed

Padding

Gets the padding value in the Report. Allowed values:

- Compact = 0
- PadRight = 1
- PadLeft = 2

SaveExecuted

Gets or sets the SaveExecuted state in the report. Allowed values:

- 0 (zero) - Save Command not executed
- 1 - Save Command executed

SaveFileName

Gets or sets the complete path used when saving the Report.

SaveFormat

Gets or sets the save format Report. Allowed values:

- XPS = 0
- Html = 1
- Unicode = 2
- ASCII = 3
- PDF = 4

Namespace Info

Runtime objects and methods related to project informations.

Class ModuleInfo

Trace

Traces a system object. The references to this object are displayed in the trace window. Parameter: string text. Example:

```
@Info.Trace("Digital1")           (VB)  
@Info.Trace("Digital1");         (C#)
```

License

Gets the InfoLicense object that represents the current license.

ExecutionPath

Gets the execution path which points to the folder where the system executable modules are installed.

Module

Gets the InfoModuleList object that represents the modules list.

OnlineConfig

Gets the online configuration state. Valid values:

- TRUE - online configuration is enabled
- FALSE - online configuration is disabled

Project

Gets the InfoProjectVersion object that represents the information of the project version.

ProjectSettings

Gets the InfoProjectSettings object that represents the information of the project settings.

TestMode

Gets the project's test mode state. Valid values:

- TRUE - The project is in test mode
- FALSE - The project is not in test mode

TestModeSync

Gets the test mode synchronization state. If 0 (zero): test mode synchronization disabled. If not 0 (zero): test mode synchronization enabled. Test mode synchronization allows Test Mode to work with the available values of the actual tags from the startup execution.

Note:

TestModeSync: This feature is available only in Enterprise version.

Class InfoprojectVersion

CurrentBuild

Gets the current build number.

DateCreated

Gets the DateTime object that represents the date that the Project version was created.

DateModified

Gets the DateTime object that represents the date that the Project version was modified.

Description

Gets the Project version description.

ProductFamily

Gets the product family of the project version. Possible values:

- "Student"
- "Express"
- "Lite"
- "Enterprise"

ProductModel

Gets the product model of the Project version. Possible values:

- 75
- 150
- 300
- 500
- 1500
- 2500
- 5000
- 15000
- 25000
- 50000
- 100000
- 1000000

ProductName

Gets the product name of the project version. Possible value: "BluePlant".

ProductVersion

Gets the product version of this project version.

ProjectName

Gets the project name of this project version.

ProjectPath

Gets the project version path.

TargetFramework

Gets the target framework of this Project version.

Versionstring

Gets the project version string. Example: "1.0 (Editing)".

Class InfoProjectSettings**CultureInfo**

Gets the project's language.

Class InfoModuleList**Alarm**

Gets the InfoModule object for the Alarm module.

DataSet

Gets the InfoModule object for the DataSet module.

Device

Gets the InfoModule object for the Device module.

Display

Gets the InfoModule object for the Display module.

Historian

Gets the InfoModule object for the Historian module.

ModuleInformation

Gets the InfoModule object for the ModuleInformation module.

OPCServer

Gets the InfoModule object for the OPCServer module.

PropertyWatch

Gets the InfoModule object for the PropertyWatch module.

Report

Gets the InfoModule object for the Report module.

ReportServer

Gets the InfoModule object for the ReportServer module.

Script

Gets the InfoModule object for the Script module.

Security

Gets the InfoModule object for the Security module.

Server

Gets the InfoModule object for the Server module.

TraceWindow

Gets the InfoModule object for the TraceWindow module.

*Class InfoLicense***AllowedRichClients**

Gets the AllowedRichClients state. Valid values:

- Zero - rich clients not allowed
- Not zero - rich clients allowed

AllowedRunInstances

Gets the AllowedRunInstances state. Valid values:

- Zero - run instances not allowed
- Not zero - run instances allowed

AllowedTagElements

Gets the AllowedTagElements state. Valid values:

- Zero - Tag elements not allowed
- Not zero - Tag elements allowed

AllowedWebClients

Gets the AllowedWebClients state. Default values:

- Zero - web clients not allowed
- Not zero - web clients allowed

DateCreated

Gets the date and time that the license was created.

DateModified

Gets the date and time that the license was modified.

ExpirationDate

Gets the date and time that the license expires.

LicenseMedia

Gets the information about license media. Possible values:

- "Hardkey"
- "Softkey"

LicenseType

Gets the information about the license type. Possible values:

- "None"
- "Run"
- "Eng_Run"
- "Developer"

ProductFamily

Gets the license product family. Possible values:

- "Student"
- "Express"
- "Lite"
- "Enterprise"

ProductModel

Gets a numeric value that features the license product model. Possible values: 75, 150, 300, 500, 1.500, 2.500, 5.000, 15.000, 25000, 50.000, 100.000, 1.000.000.

ProductVersion

Gets the license product version.

SerialNumber

Gets the license serial number.

ServerConnected

Gets the server full address.

AllowedEngineeringUsers

Gets the AllowedEngineeringUsers state. Default values:

- Zero - engineering users not allowed
- Not zero - engineering users allowed

AllowedDevices

Gets the AllowedDevices state. Default values:

- Zero - devices not allowed
- Not zero - devices allowed

Class InfoModule

IsPaused

Gets or sets the module's IsPaused state. Possible values:

- TRUE - module is paused
- FALSE - module is not paused

IsRunning

Gets or sets the module's IsRunning state. Possible values:

- TRUE - module is running
- FALSE - module is not running

Namespace Server

Runtime objects and methods related to the Server.

ServerStation

The ServerStation class contains information about the computer on which the Runtime Server (TServer.exe program) is running.

LoadProjectVersion

Loads the project pointed by the specified path. The path must point to a valid project in the Server. Returned values: true if success and false if failure. Parameter: string projectName.

SwitchToStandby

Switches the execution to the standby computer if redundancy is enabled. Returned values: TRUE if success and FALSE if failure.

HttpAddress

Gets the server's http address.

IsPrimary

Gets the server's IsPrimary state. Possible values:

- TRUE - server is primary
- FALSE - server is not primary

IsRedundancyEnabled

Gets the server's IsRedundancyEnabled state. Default values:

- TRUE - redundancy is enabled
- FALSE - redundancy is not enabled

IsSecondary

Gets the server's IsSecondary state. Default values:

- TRUE - server is secondary
- FALSE - server is not secondary

IsStandByActive

Gets the server's IsStandByActive state. Default values:

- TRUE - the standby server is active
- FALSE - the standby server is inactive

IsSwitchToPrimaryEnabled

Gets the server's IsSwitchToPrimayEnabled state. Valid values:

- TRUE - switch option to primary enabled
- FALSE - switch option to primary not enabled

ServerStation.ComputerIP

Gets or sets the server's computer IP.

ServerStation.ComputerName

Gets or sets the server's computer name.

ServerStation.Date

Gets the server's date.

ServerStation.Day

Gets the server's day of the month.

ServerStation.DayOfWeek

Gets the server's day of the week.

ServerStation.DayOfYear

Gets the server's day of the year. Default values: 1 to 366.

ServerStation.Hour

Gets the hour of the server's day.

ServerStation.Minute

Gets the minute component of the server's date.

ServerStation.Month

Gets the month component of the server's date.

ServerStation.Now

Gets the server's local date and time offset.

ServerStation.Second

Gets the "second" component of the server's date.

ServerStation.ShutDown

Gets or sets the server's shutdown state. Possible values:

- TRUE - server is being turned off
- FALSE - server is not being turned off

ServerStation.Startup

Gets the server's startup state. Default values:

- TRUE - server has started up
- FALSE - server has not started up

ServerStation.Ticks

Gets the number of ticks that represent the server's date and time.

ServerStation.Time

Gets the server's time of day.

ServerStation.Year

Gets the year component of the server's date.

TimeMs

Gets the server's time of day (including milliseconds).

Namespace Client

Runtime objects and methods related to the Client Namespace.

Class ClientStation

The ClientStation class contains information about the computer on which the Client is running (TVisualizer.Exe or Web clients).

ChangeUserPassword

Changes the password of the referenced user. Returned value: true if success and false if failure. Parameters: string username, string oldPassword and string newPassword.

CloseDisplay

Closes the referenced display. Parameter: string displayName.

GetPasswordHint

Gets the password hint for the referenced user. Returned value: the password hint. Parameter: string userName.

Locale

Returns the localized text. Returned value: localized text. Parameter: string text.

LogOn

Performs the logon of the specified user with the specified password. Parameters: string username and string password. Returned values:

- OK = 0
- ServerNotAvailable = 1
- InvalidLogon = 2
- ServerNotConnected = 3
- UserBlocked = 4
- UserDeleted = 5
- CannotStartModule = 6
- InvalidUserName = 10
- InvalidPassword = 11
- PermissionsRestrictions = 12
- UnknownError = 99

LogOnGuest

Performs the logon of the user as "Guest".

OpenDisplay

Opens the display at the layout's last page. Parameter: string displayName.

OpenDisplayAtIndex

Opens the display at the layout's page specified by the index. Parameters: string displayName and int index.

OpenLayout

Opens the referenced layout. Parameter: string layoutName.

OpenPopupNote

Opens a popup note with the specified title and description. Parameters: string title, string description, bool isReadOnly, double left and double top.

OpenPreviousPage

Opens the previously displayed page. Default values:

- TRUE if success
- FALSE if failure.

SaveLayoutAsImage

Saves the layout as an image. The path for the file will be defined in the subsequent dialog.

SaveLayoutAsImageFile

Saves the layout as image. The file path is defined in the next dialog.

SwitchToStandby

Switches the Server to Standby mode. Default values:

- TRUE if success
- FALSE if failure

BeepOff

Gets or sets the client's beep off state. Default values:

- TRUE - beep off
- FALSE - beep on

BlinkFast

Gets the client's blink fast property. The blink fast property is a digital variable that toggles from 0 to 1 and from 1 to 0, remaining 500 milliseconds in each state.

BlinkSlow

Gets the client's blink slow property. The blink slow property is a digital variable that toggles from 0 to 1 and from 1 to 0, remaining 250 milliseconds in each state.

ComputerIP

Gets the computer IP.

ComputerName

Gets the computer name.

CultureInfo

Gets or sets the client's language selection.

CurrentPage

Gets the name of the client's page currently displayed.

CurrentUser

Gets the client's current user.

Date

Gets the client's date.

DateTime

Gets the client's date and time.

Day

Gets the client's day of the month.

DayOfWeek

Gets the client's day of the week.

DayOfYear

Gets the client's day of the year.

Dictionary

Gets or sets the client's dictionary.

Hour

Gets the hour component of the client's date.

InputPassword

Gets or sets the client's input password. This is an auxiliary variable used in the system's default logon window.

InputUserName

Gets or sets the client's input user name. This is an auxiliary variable used in the system's default logon window.

IsWebBrowser

Gets the client's IsWebBrowser state.

LayoutName

Gets or sets the client's layout name.

Millisecond

Gets the millisecond component of the client's date.

Minute

Gets the minute component of the client's date.

Month

Gets the month component of the client's date.

Now

Gets the client's local date and time offset.

OnScreenKeyboard

Gets or sets the client's screen and keyboard state.

PreviousPage

Gets the client's previous page name.

Second

Gets the “second” component of the client's date.

SelectedPage

Gets or sets the client's selected page name.

ServerHttpAddress

Gets the server's http address.

ShutDown

Gets or sets the client's shutdown state. Possible values:

- TRUE - client is shutdown
- FALSE - client is not shutdown

SimulationAnalog

Gets the analog [int] simulation variable, which varies from 0 to 100 (in steps of 1); returns to 0 in one cycle and then repeats the same pattern (sawtooth waveform).

SimulationDigital

Gets the digital simulation variable, which toggles from 0 to 1 and from 1 to 0, remaining 3 seconds in each state.

SimulationDouble

Gets the analog [double] simulation variable, which varies from 0 to 100; returns from 100 to 0, and then repeats the same pattern.

Startup

Gets the client's startup state. Valid values:

- TRUE - client has started up
- FALSE - client has not started up

Ticks

Gets the number of ticks that represent the client's date and time.

Time

Gets the client's time of day.

TimeMs

Gets the client's time of day including milliseconds.

Tomorrow

Gets the day component of the client's tomorrow date.

UserName

Gets the client's user name.

UtcNow

Gets the client's UTC date and time offset.

Year

Gets the year component of the client's date.

Yesterday

Gets the day component of the client's yesterday date.

CurrentPage

Gets the name.

DateTime

Gets the date and hour of the currently displayed client's page.

IsWebBrowser

Gets the client state IsWebBrowser. The allowed values are:

- TRUE – the client is running with a Web browser
- FALSE – the client is not running with a Web browser

OnScreenKeyboard

Gets or set the client's display and keyboard state. Associated parameter: public bool OnScreenKeyboard { get; set; }. The allowed values are:

- TRUE – keyboard and display functionality is active
- FALSE - keyboard and display functionality is not active

PreviousPage

Gets the client's previous page name.

ServerHttpAddress

Gets de http adress of the client.

SimulationAnalog

Gets an analog [int] simulation variable, which varies from 0 to 100 (in steps of 1); returns to 0 in one cycle and then repeats the same pattern (sawtooth waveform)

SimulationDouble

Gets an analog [double] simulation variable, which varies from 0 to 100; returns from 100 to 0, and then repeats the same pattern.

Advanced Settings

This section contains additional information about the BluePlant applications including:

- Command line
- Running BluePlant as a Windows service
- Remote clients
- Installing Web Server in IIS

Command Lines

The information about the BluePlant toolbars and executables that can be called using command lines and specific parameters are described below.

TStartup

Starts a BluePlant project.

TVisualizer

Starts the BluePlant Visualizer Module.

TraceWindow

Starts the BluePlant TraceWindow tool.

PropertyWatch

Starts the PropertyWatch diagnostic tool.

ModuleInformation

Starts the Module Information diagnostic tool.

DisableTaskSwitchProtection

The user needs to run the batch file in order to install the device driver that can block the CTRL + ALT + DEL while the TVisualizer is running.

RegServer

Register the BluePlant OPC Server.

UnRegServer

Unregister the BluePlant OPC Server.

Running BluePlant as a Windows Service

This procedure describes how to install BluePlant applications from the Runtime server to be ran as a Windows Service.

In order to enable the Distributed Engineering and also to serve pages to WEB clients, the user needs an enabled Web server too. Further information about this procedure can be found at the Installing Web Server in IIS section.

In order to run the application as a Windows Service, enter the following command:

```
<.NET Framework Install Path>\installutil <Install Path>\<BluePlant Version>\TStartupAsService.exe.
```

On the DOS prompt (Run as Administrator), execute the following command:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319>installutil "C:\Program Files\Altus\bp-2012.1\TStartupAsService.exe"
```

Then, enter the command line for the service, which is the same of the tStartup.exe program: /project:<projectNameAndPath>.

There is no available plugin to setup the register, so the user has to do it manually. The plugin and its parameters require to be set on the Windows Registry. The user must open the Registry Editor (regedit), and go to the following key:

```
"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\TStartup\ImagePath"
"C:\Program Files\Altus\bp-2012.1\TStartupAsService.exe"
"/project:C:\BluePlant Projects\Project1.tproj"
```

On the examples shown above, the BluePlant installation path must be changed to install in your computer.

On the "Services" Windows (Administrative Tools), the "TStartup Service" must be set.

It is possible to set the "Automatic" option, so the selected project will open when the computer starts.

Remote Clients

The only pre-requisite to run the application in remote clients is the .Net Framework installation. The application does not need to be installed in the client computers. When the BluePlant Server presents the WebServer (IIS or TWebServer) running, enter the following address from the Internet Explorer Browser:

Rich Client:

```
http://<server ip or name>/<BluePlant Version>/tvisualizerremote.application
```

Web Client:

```
http://<server ip or name>/<BluePlant Version>/tvisualizerweb.xbap
```

Depending on the Internet Explorer version, it may be required the following configuration:

Disable the protected mode on Internet or Intranet Zone (Figure 6-83).

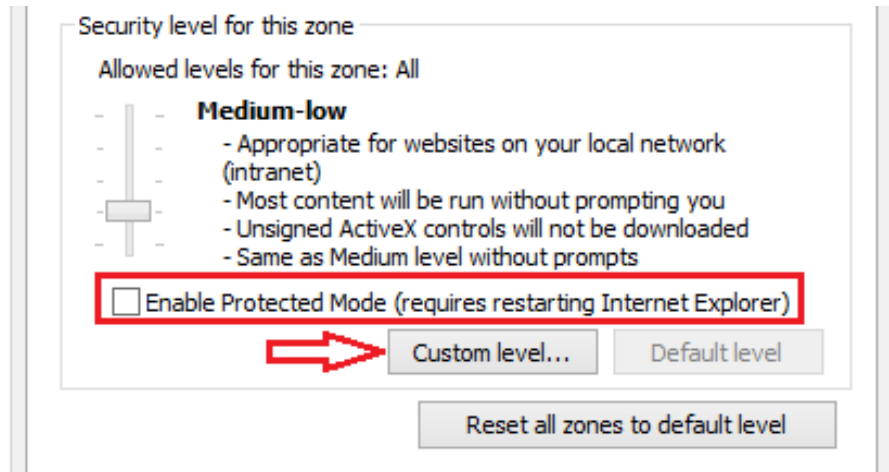


Figure 6-83. Disable the Protected Mode

Enable XAML Browser Applications (Figure 6-84).

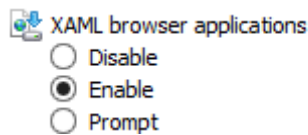


Figure 6-84. Enable XAML Browser Applications

Example:

Remote Rich Client:

<http://127.0.0.1/bp-2012.1/TVisualizerRemote.application>

Web Client:

<http://127.0.0.1/bp-2012.1/TVisualizerWeb.xbap>

Installing Web Server in IIS

When installing the application, if IIS is located, the user will not be able to install our embedded web server (TWebServer.exe).

The program will run without the Web server, remotely accessing project configurations and serving Web pages in Runtime. All the others BluePlant engineering or executing tools will run without installing the TWebServer or IIS configuration.

If the user wants to enable the remote engineering access and web clients through IIS, it is required to install some services in IIS. This manual explains how to install BluePlant services in IIS. This description refers to IIS 7.x., yet other versions are quite similar.

The IIS is also required when the user wants to run the RUNTIME as a Windows Service, and still wants to be able to provide Web pages, iPad clients and remote engineering.

Three services can be installed inside IIS. The first one is the TProjectServer which allows remote access to the project configurations. The second one is the TVisualizerWeb/TVisualizerRemote which allows remote access to interfaces in Runtime. The third and last one is the iDataPanel which is the service to provide data for the iOS devices. The following step-by-step instructions should be followed to configure the IIS.

Installation Procedure

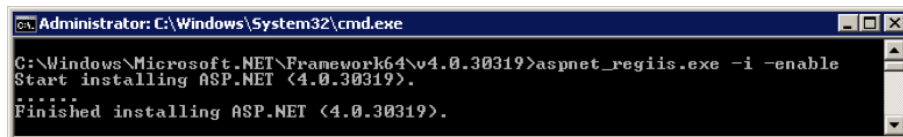
This procedure may be executed on Windows 7, x64, with 7.5 IIS. In case of older versions of the operating system and IIS, the procedure can be slightly different, such as the configuration of a “Virtual Directory” instead of “Add Application”.

- Check if the IIS is installed and functioning properly (trying to open HTML pages, for example)
- Enable ASP.Net and .Svc Handlers for the WCF Service

Installation in IIS 7.x

If IIS was installed after the .NET Framework installation, it is necessary to run the procedure described on <http://msdn.microsoft.com/en-us/library/ms752252.aspx> in order to run the following programs using the prompt command as administrator (see next figures):

"aspnet_regiis -i -enable" (from .NET Framework* installation directory)

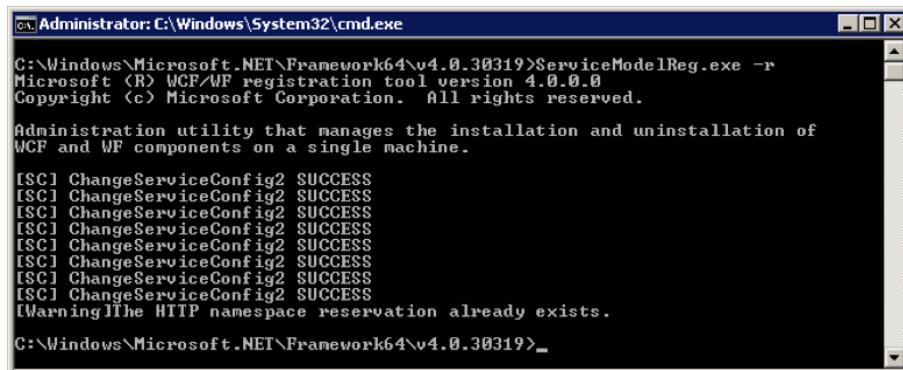


```

Administrator: C:\Windows\System32\cmd.exe
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>aspnet_regiis.exe -i -enable
Start installing ASP.NET (4.0.30319).
*****
Finished installing ASP.NET (4.0.30319).
  
```

Figure 6-85. IIS 7.x Installation Procedure

"ServiceModelReg.exe" -r (from .NET Framework* installation directory)



```

Administrator: C:\Windows\System32\cmd.exe
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>ServiceModelReg.exe -r
Microsoft (R) WCF/WF registration tool version 4.0.0.0
Copyright (c) Microsoft Corporation. All rights reserved.

Administration utility that manages the installation and uninstallation of
WCF and WF components on a single machine.

[SC] ChangeServiceConfig2 SUCCESS
[SC] ChangeServiceConfig2 SUCCESS
[SC] ChangeServiceConfig2 SUCCESS
[SC] ChangeServiceConfig2 SUCCESS
[SC] ChangeServiceConfig2 SUCCESS
[SC] ChangeServiceConfig2 SUCCESS
[SC] ChangeServiceConfig2 SUCCESS
[SC] ChangeServiceConfig2 SUCCESS
[Warning]The HTTP namespace reservation already exists.

C:\Windows\Microsoft.NET\Framework64\v4.0.30319>_
  
```

Figure 6-86. ServiceModelReg.exe

.NET Framework installation directories is:

C:\Windows\Microsoft.NET\Framework\v4.0.30319, or
 C:\Windows\Microsoft.NET\Framework64\v4.0.30319 (Windows x64)

Installation in IIS 8.0

For IIS 8.0, follow the procedure described below.

Go to "Program and Features", and choose "Turn Windows features on or off".



Figure 6-87. Switching Windows Features

Select the option: “Internet Information Services - World Wide Web Services - Application Development Features”- ASP.NET 4.5.

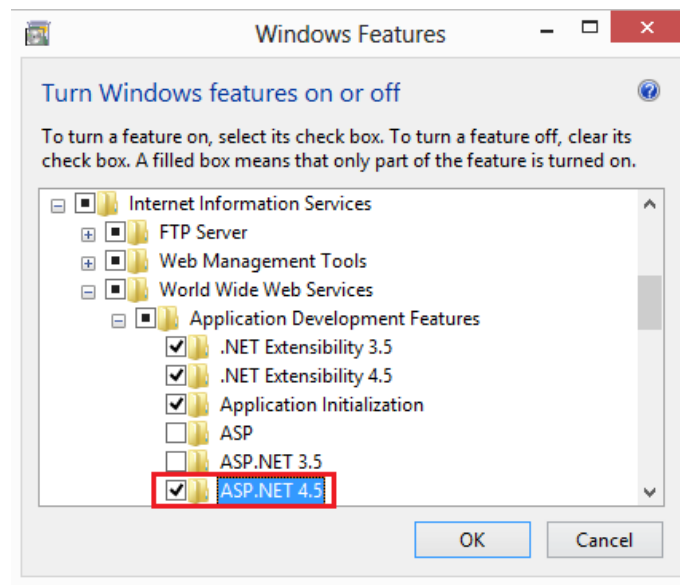


Figure 6-88. Windows Features

Select the Net Framework 4.5 Advanced Services check box, and then activate the “HTTP Activation” option.

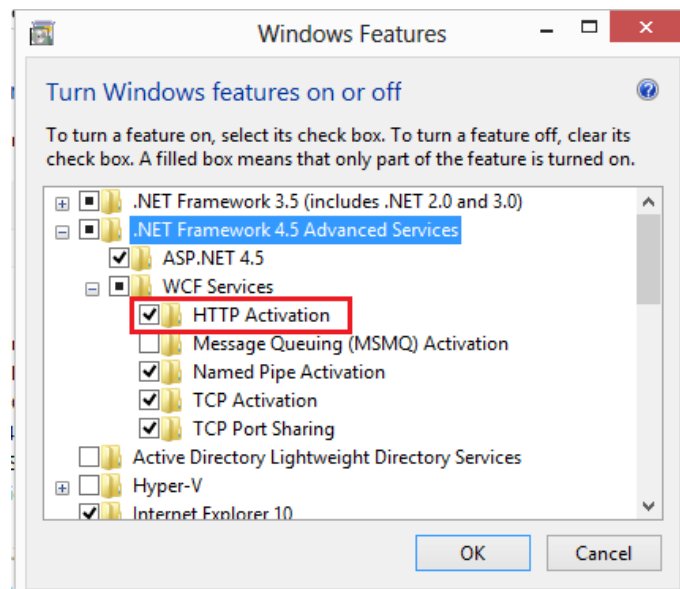


Figure 6-89. HTTP Activation

Check if the extension ".svc" is mapped to "aspnet_isapi.dll". The previous link (msdn.microsoft site) describes how to check this for several IIS versions.

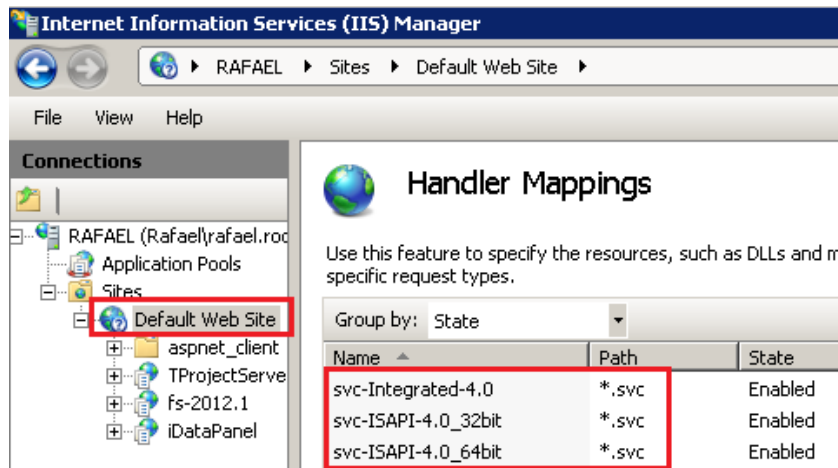


Figure 6-90. Extension Mapping

In the BluePlant installation folder, run the following plugin: `InstallTWebServer.exe /uninstall`, which will remove the current TWebServer installation (see Figure 6-91).

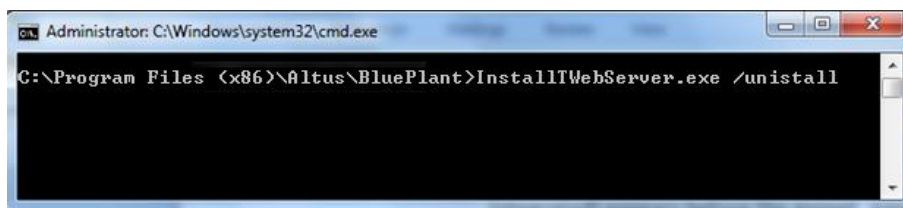


Figure 6-91. InstallTWebServer Plugin

Check if the TWebServer is set to start automatically. Open the Windows Register Editor (`regedit.exe`) and go to the following Key:

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run.`

If there is any reference to the "TWebServer" entry, it should be deleted.

In IIS 7.x, open "Sites/Default Web Site/Add Application" and configure the information for the services (Figure 6-92).

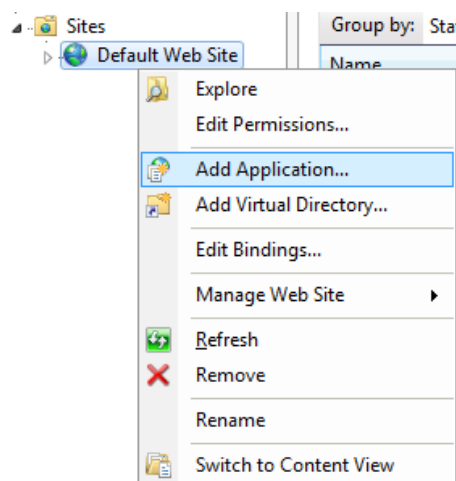


Figure 6-92. Information for Services

The settings for the TProjectServer presented in Figure 6-93 are:

- Alias: TProjectServer
- Physical Path (1): verify and set to the BluePlant installation folder
- Application Pool: configure to any set based on .NET 4.0.

Example: ASP.NET v4.0.

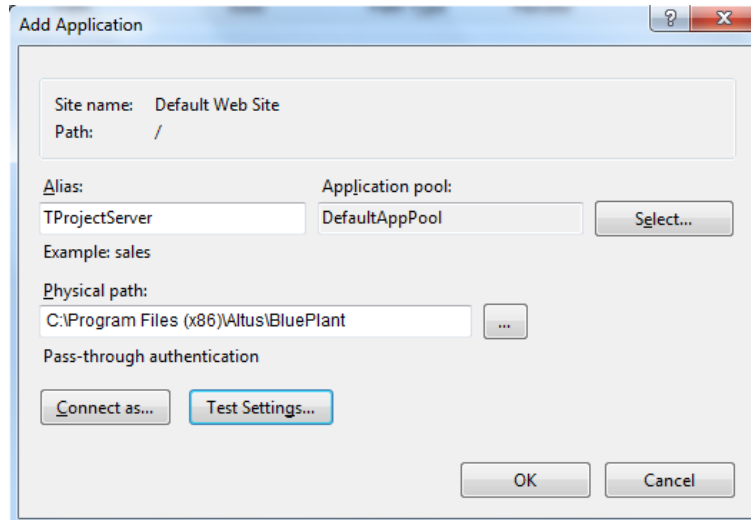


Figure 6-93. ProjectServer

Set the access as “Everyone”, at least for reading. On IE, use the following URL: "http://localhost/tprojectserver/service.svc". This is necessary in order to check if the service was correctly installed. It will show up a page with informations about the service.

Settings for the TVisualizerWeb/TVisualizerRemote:

- Alias: BluePlant version, example: bp-2012.1
- Physical Path: BluePlant installation folder for the specific version
- Application Pool: configure to any set based on .NET 4.0.
- Example: ASP.NET v4.0

Note:

Security: This option must be set as “Everyone”, at least for reading. On IE, use the following URL: http://localhost/fs-2012.1/service.svc. This is necessary in order to check if the service was correctly installed. It will show up a page with informations about the service.

Settings for the iDataPanel Service (Figure 6-94):

- Alias: iDataPanel
- Physical Path (1): BluePlant installation folder for the specific version
- Application Pool: configure to any set based on .NET 4.0.

Example: ASP.NET v4.0

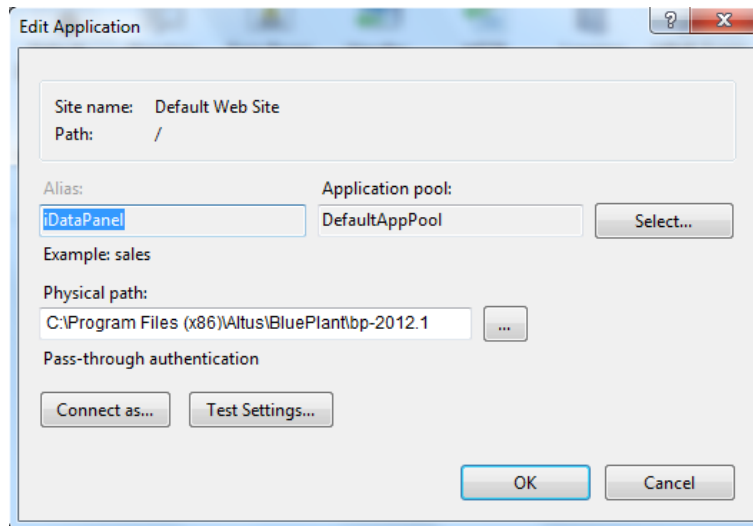


Figure 6-94. iDataPanel Service

Note:

Security: This option must be set as “Everyone”, at least for reading. On IE, use the following URL: <http://localhost/iDataPanel/iDataPanelService.svc>. This is necessary in order to check if the service was correctly installed. It will show up a page with informations about the service.

Settings for the iDataPanelImages (Virtual Directory) and illustrated in the following images are:

- Alias: iDataPanelImages
- Physical Path: C:\BluePlant Projects\iDataPanelImages

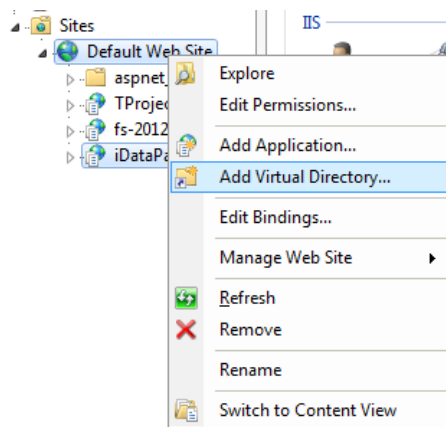


Figure 6-95. Virtual Directory

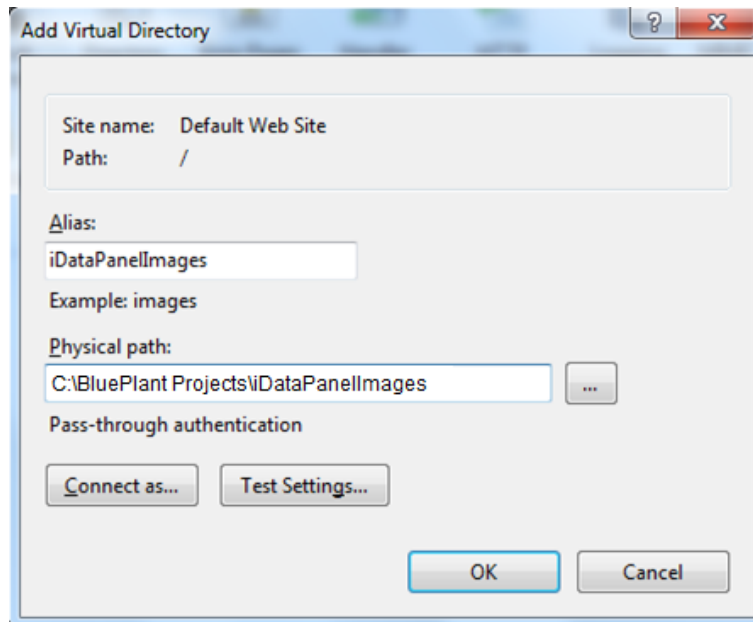



Figure 6-96. Add Virtual Directory

Notes:

Security: This option must be set as “Everyone”, at least for reading.

The “Anonymous Authentication” field must be enabled for services (Figure 6-97).

 Authentication

Group by: No Grouping

Name	Status	Response Type
Anonymous Authentication	Enabled	

Figure 6-97. Authentication

IIS must be restarted after the services configuration.

7. Scenarios of Typical Systems

This chapter describes the typical BluePlant scenarios to meet the production, utilities and manufacturing demands.

Moreover, the complete application project may include a combination of these scenarios in order to satisfy the client customization demands.

Regardless of the considered scenario, it should be kept in mind that BluePlant is made up of a single package, so the server is always a BluePlant. However, all BluePlant modules such as Alarm, Historian, Device, Database, etc. may be placed on different computers. As they are managed by the BluePlant Server the result is a distributed system context.

Based on these assumptions there are some possible configurations, which are described below.

Systems Configurations

Standalone System

This system is characterized by a BluePlant installation running the SCADA server and the client in the same computer.

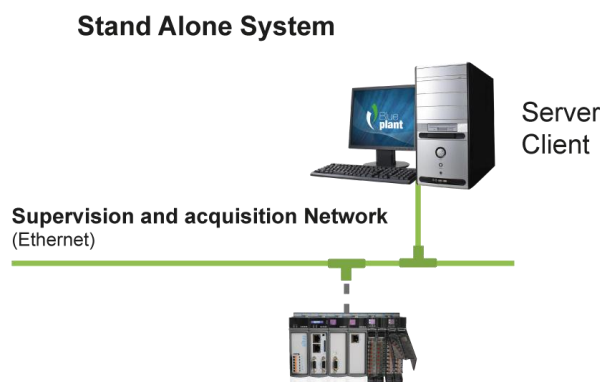


Figure 7-1. Stand Alone System

Distributed Inputs/Outputs System

This system is characterized by a BluePlant server machine and device modules running on computers dedicated to the communication with the process. In this case the SCADA client can be placed in the same server computer or in another one. Figure 7-2 illustrates this configuration.

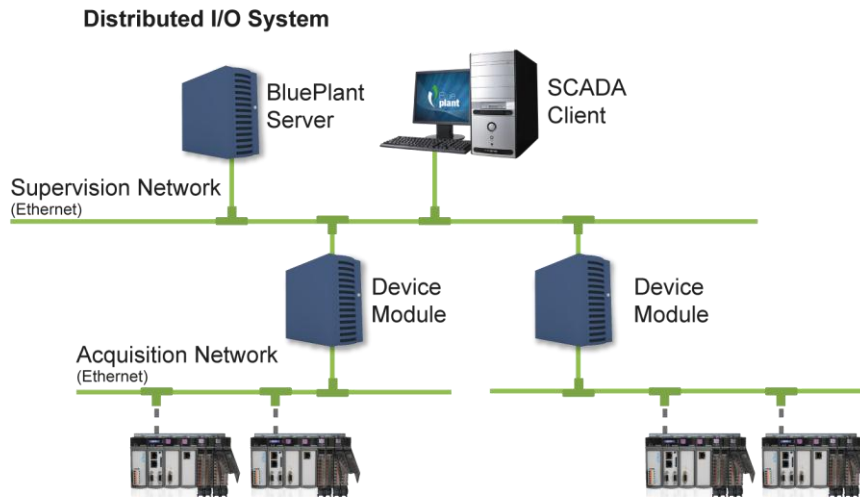


Figure 7-2. Distributed Inputs/Outputs System

This model is also useful in plants that have devices with serial port or limited capability communication. By placing I/O servers in the factory to interact with these devices, it is possible to optimize the communication on slow or with low bandwidth networks and achieve a better global performance.

Despite the geographical distribution of I/O servers in several plants, this type of architecture can be configured as a single cluster system, as long as it is able to support multiple I/O servers.

Client and Server System

This system complies a server BluePlant where the alarm, historian, database modules are being executed and the SCADA clients are being executed in other computers in the LAN. The Figure 7-3 shows this system.

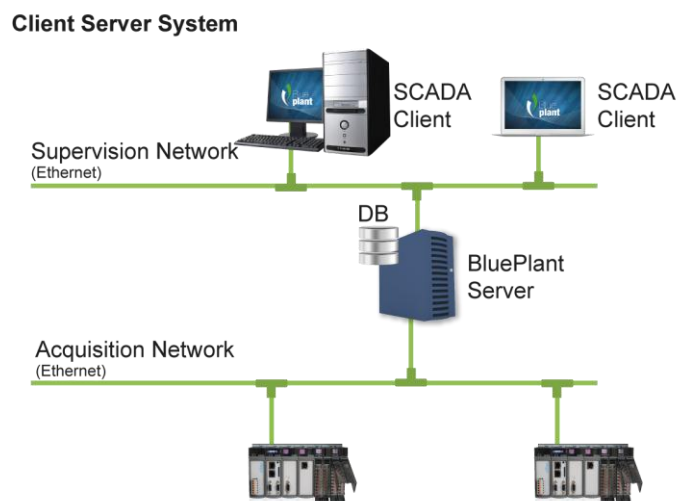


Figure 7-3. Client and Server System

The architecture client server allows that the clients can be distributed in many computers in the LAN creating a system that offers benefits in flexibility and performance. This type of architecture also can be configured as a unique cluster system.

Redundant Server System

The Redundant Server System presents two different computers running BluePlant servers, where the redundancy is done automatically by the supervisory itself. Thus it is necessary only to specify the IP addresses of the primary and secondary stations. The following redundancy configurations are available:

- The Alarm and/or Historian database running on a third machine dedicated to historical data, according to Figure 7-4;
- The embedded databases in the primary and secondary servers are used to store the historical data of Alarm and/or Historian modules, with automatic data synchronization between them. Figure 7-5 shows that system.

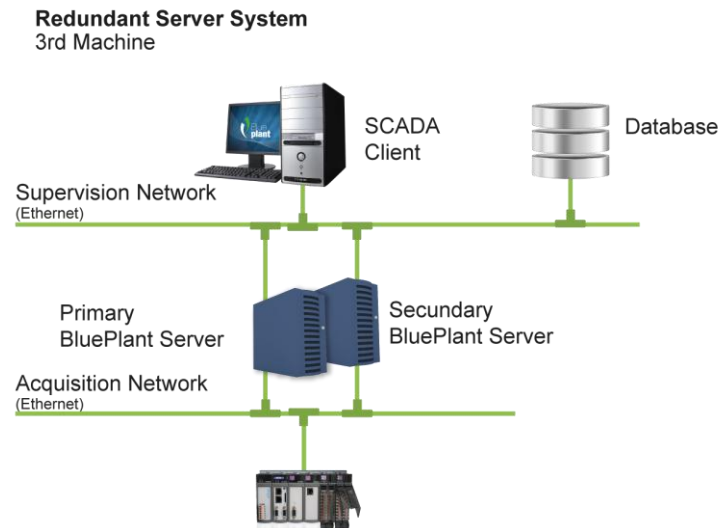


Figure 7-4. Redundant Server System

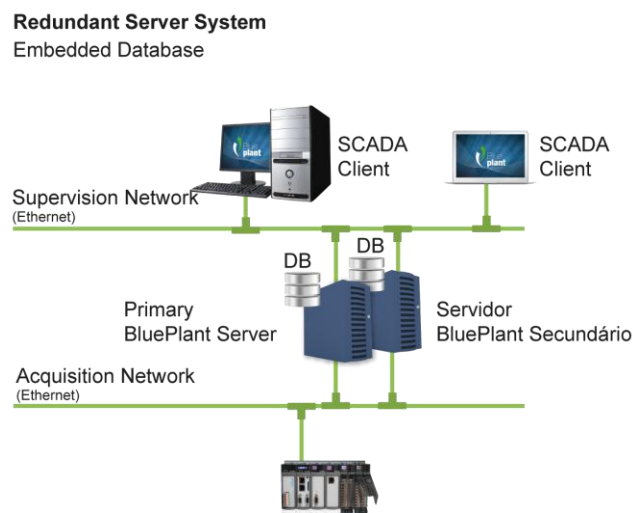


Figure 7-5. Redundant Server System Embedded Database

The possibility to define the primary and secondary servers within a project allows the integration of hardware redundancy in the system infrastructure. This helps to avoid situations where an error

occurs in a server in the global system, causing it to be inoperable. Systems of this kind are interesting to ensure uninterrupted operation and reliability in data collection.

Other possible configuration is the device communication redundancy, where the server(s) have the option to communicate with the primary or the backup device. This redundancy is enabled configuring the IP addresses of the devices. The BluePlant is responsible to manage the switch between the primary and backup device when a failure in communication happens with BluePlant and the device.

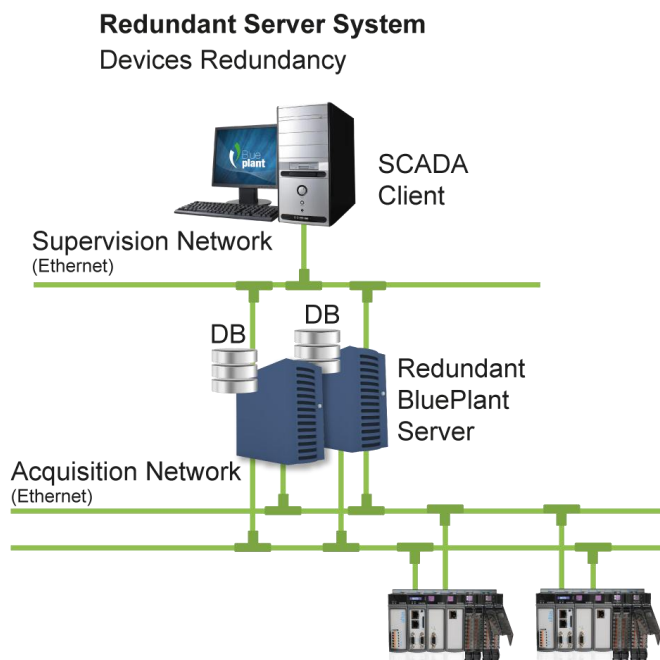


Figure 7-6. Redundant Server System with Devices Redundancy

Control System

In the Control System it is possible to have multiple servers in different plants (different projects) in a distributed architecture. This configuration allows to access, in a control room, any of these plants through specific clients. It is important to mention that the clients of the plants will not be integrated in one machine, so it is necessary to specify which plant user wants to watch. Figure 7-7 illustrates this system.

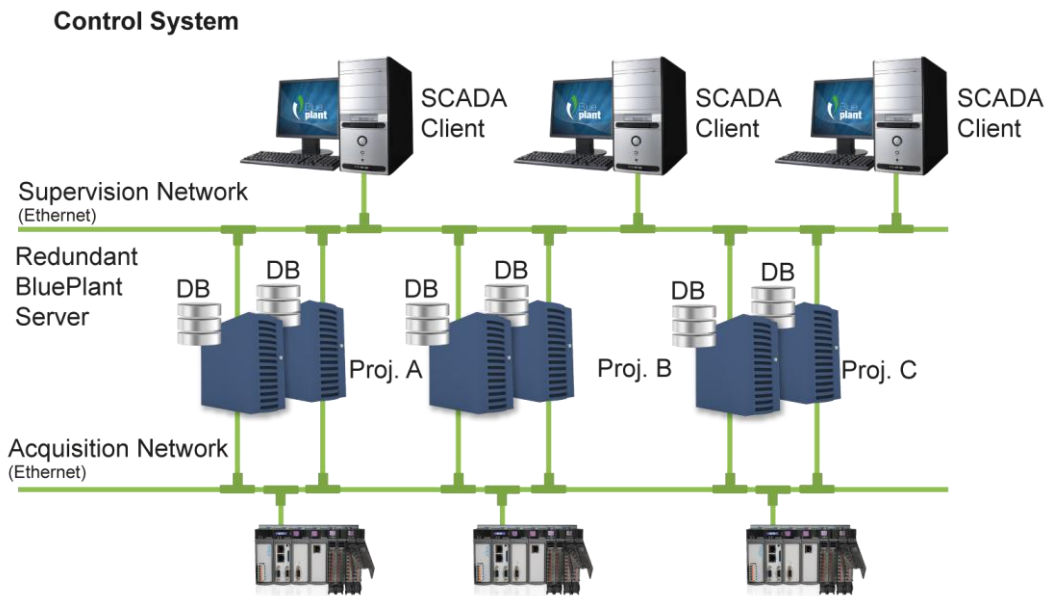


Figure 7-7. Control System

In this scenario, the system is organized in discrete locations controlled by local operators supported by local redundant servers. At the same time, a management level in a central control room can be characterized to monitor all sites simultaneously. Each site is represented in the project as a separate cluster, grouping their primary and standby servers.

Distributed and Redundant Distributed Control System

This system includes a BluePlant Server machine and Alarm modules, Historian, Database and SCADA Clients in other computers of the LAN network. Thus, each module runs in a separate computer. Figure 7-8 shows that system.

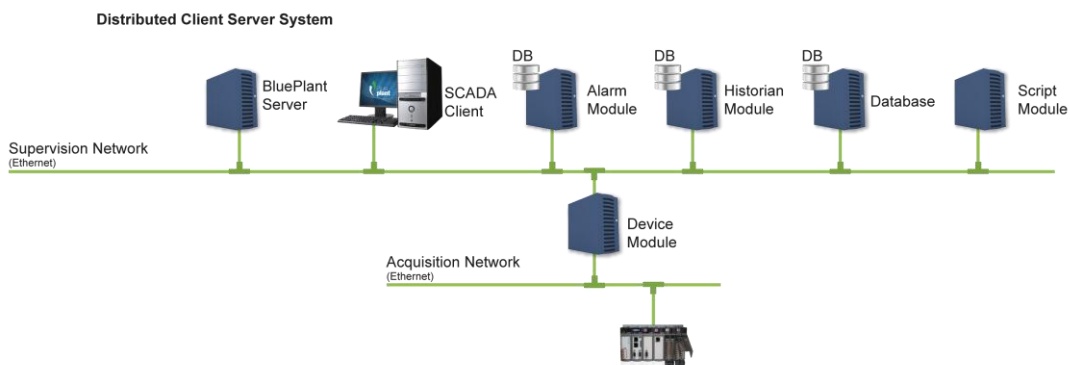


Figure 7-8. Distributed Control System

The client-server architecture allows the modules to be distributed across multiple computers on a LAN, creating a system that provides benefits of flexibility and performance. Each component is identified, in the project scope, by an address, which allows their location and hardware requirements to be considered as independent ones. This type of architecture can also be configured as a single cluster system.

This is a redundant system, but with its pair in a different location. This configuration requires a network with high transmission rates between localities, since it has an expressive amount of data exchanged among the pairs of servers. Figure 7-9 shows this scenario.

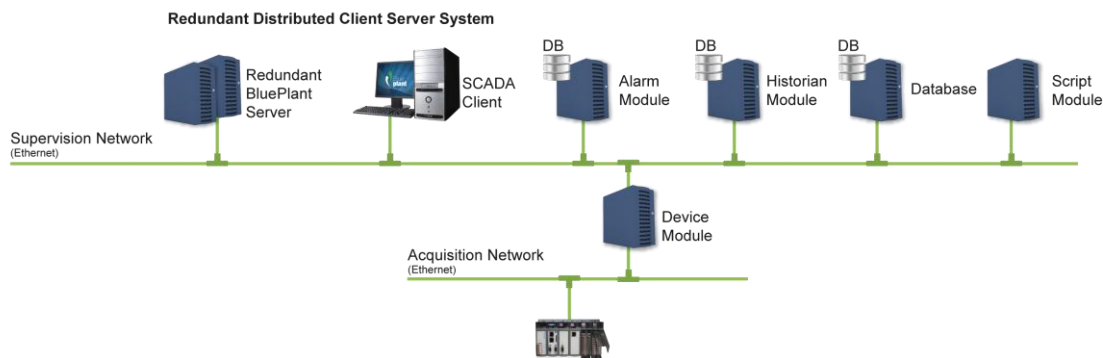


Figure 7-9. Redundant Distributed Control System

In this scenario, the project presents sites which are operated locally. Each one contains its own set of servers and clients.

In case the site becomes inoperative, the uninterrupted monitoring is guaranteed by distributing the primary and secondary servers between different sites, or putting up the secondary servers in a central location.

A cluster is used to define the role of servers in each site, and can be viewed on a common project, running on each client.

Load Sharing System

This system is similar to the client and server one mentioned previously, but the redundancy is only in the device module. This system allows a better use of the available infrastructure, since it enables the sharing of workload between different computers and CPUs.

This approach can be used to improve network performance, data access times and the overall stability of the system.

Moreover, through clusters is possible to run multiple servers of the same type in a single computer.

On the other hand, the distribution of servers in two clusters allows them to operate as redundant units between them, which reduces the required number of computers.

8. Glossary

AppDomain	The limit provided by CLR (Common Language Runtime) around objects created from the same application scope. The application domains help to isolate objects created in one application from those created in other applications. The TRuntime.exe runs on a different AppDomain than other modules (Device Module, Visualizer). Term related to Microsoft dotNet.
Assembly	"Assembly" refers to an executable file (.EXE) or Library (.DLL) created using managed code and the Microsoft .NET framework. Term related to Microsoft dotNet.
Designer	Development component of the program manager used for the design of synoptic screens. Synoptic screens are graphical representations of industrial processes that are generated in the draw environment.
Device Module	Module that allows the implementation of communication protocols (Devices) on the remote machine to the server that is running the Project and maintains the Runtime.
Domain, Server Domain, Client Domain	Refers to values and the location of objects in Runtime. "Server Domain objects running on the server" signifies object(s) running on the Server. Values associated with that object are system-wide; "Client Domain" signifies that the object is running on the Client station and each client machine may have different values.
Internal Module	Features and programs that implement internal system functions that run without any user configuration. Examples are the Network Synchronization task and the Background Report Generator. Some BluePlant tools e.g., ModuleInformation.exe, may display status information of those internal Modules, however that information is required only for advanced system optimizations. Example: The synchronization module.
Main Project File	"Main Project File" refers to the BluePlant-encrypted SQL database that contains the Project configuration. The extension .TPROJ denotes the current Project under development. The extension .TENG denotes Published Read-Only projects.
Modifier (Device/Points)	Auxiliary parameter to effect read and write points on a device, the treatment of Arrays, bit masks, strings, swap and other operations in which a DataType definition cannot define them completely. (additional parameters are required to define the DataType).
Module	A Program that accesses the database in real time (RtDB) and can be composed of one or more Assemblies. Example: Manager.XBAP is a Project Configuration Module that provides to the User, the access to its online name via the URL of the browser.
Namespace	An address space. All objects created within a Namespace have unique names. Namespace can be understood also as a setting to create a hierarchy among objects. Example: All process variables are grouped in the namespace Tag, all reports created are grouped in the namespace report.
ObjectType (RunObj type)	Determines the type of Runtime objects. An object, for example, can have the following Runtime types: display, report e ScriptFunction.
Objects, RunTime Objects	Runtime Objects are all the visible objects (via their names) which can be accessed via scripts or exhibited in the project. Examples: Tags, Reports, Displays and others. Also must be used by module developers. For example, the Alarm Module will start and update the values of Runtime objects associated with the Alarms.
ObjectValues, PropertyValue	RuntimeObjects (e.g., Tags, device Nodes, etc.) can have one or more defined properties. The TAG object (and its Value), presents properties such as: Min, Max, and Description. During the specific configuration of project, "PropertyValue" or "ObjectValue" entries are required. The complete name must be specified until the final property value, i.e., TAG.mytag.Min or Tag.MyTag.Description For some configurations, like alarm items, historian or device points, that specify the main OBJECT (Tag.MyTag, in this example), the system will assume the VALUE property by default to execute the operation.
ObjectWatch	Diagnostic utility to verify and modify values of objects at Runtime.
Parameters (TagConfiguration)	Definition of behavior parameters and values of the processing Tags. Example: CounterUP for accountants, or DeadBand smoothing Tags for integers
Project	A set of BluePlant configurations, Display drawings, Reports, documents and user Notes that are created and edited as a single entity.
projectDB or ProjectDatabase	ProjectDB (or ProjectDatabase) is a proprietary database that contains configuration information. Corresponds to files with extension <project>.Tproj (e.g., Current) or <project> _ VERSION.Teng (type Release).
Property	Property (value) associated with a Tag or a Runtime object.
RunDB, Runtime Database	A real time Database created when the Project is running. All Runtime objects accessible via script (such as Tags, Reports) are generated internally, with encapsulated code and accessible through the class RunDB. The RunDB corresponds to the first level of the tree of Runtime objects.

Runtime	A Project with loaded and running Runtime Modules.
Runtime Startup	Operation that allows the execution of a Project. This operation can be executed from the program TStartup.exe in the current version of BluePlant, or from TServer.exe in the published version of the project.
RuntimeDB	RuntimeDB is a database copy of the ProjectDB that contains specific information about the RuntimeDB when a Runtime version is Published. Corresponds to the files with <project> _ <version>.trun.
Tag	Process variable. Name of a Namespace that includes all the variables created by user in a Project configuration.
Tag type	Defines the type of objects in the Namespace Tag. Example: Digital, Analog and Text. These tags are a class of Properties accessed directly, such as: Min, Max, Value, and Quality. Each property is internally created as a ValueType.
Task (Script.Task)	Task Program written in VB.NET (or C #) that runs on the server or the client during Runtime of a Project. Execution will be in the server or client, depending upon the Domain property set in the script.
TManager	Program that performs the configuration of a project.
Toggle	Inverts the value of a variable. Values greater than zero are converted to zero; zero is converted to the value "1".
Partial Trust, Full Trust	Partial Trust: Environment in which an application has limited access to resources (restricted access to files of particular folders, running of other applications, etc.). XBAP applications which run within a browser (e.g., Internet Explorer) should run in "Partial Trust". Full Trust: Environment in which an application has access to all system resources. Applications installed on a computer usually run in "Full Trust". Term related to Microsoft dotNet.
TWelcome	Initial BluePlant and project selection screen
Visibility (Tag Visibility)	Refers to the tagging system; Tags can be Private, Public or Protected. Public: The value of the Tag during the execution is available for access of external Programs via TCP / IP or OPC Server. Also, the value of Tag is necessarily global or shared in all the client stations (Server Domain). Protected: Read-only. Private: A Tag set to "Private" cannot be accessed by external applications (OPC Server, TCP / IP) and will run in Client or Server according to the application configuration with the following characteristics: <ul style="list-style-type: none"> • Tags used only in modules called by the client (such as displays and Reports) and run in the scope of the client. May have different values on each client machine (Client Domain); • Tags used in server modules, such as Devices and Alarms. Have a unique value in the system (Server Domain).
Xbap	XAML browser application (XBAP) is an application that runs inside a browser (IE). Term related to Microsoft dotNet.