# Advantech Studio

Tutorial

## Advantech Studio

## for Windows NT and Windows CE

ADVANTECH Industrial Automation

eAutomation Series

E-mail: support@advantech.com

# 1. Introduction

Section 1 – Course Introduction

This section will familiarize you with the objectives and agenda of the Advantech Studio course.

## Course Overview

Advantech Studio course is a four-day instructor-led course designed to teach all the basic functionality of the Advantech Studio product as well as an overview of the underlying architecture. The purpose of this course is to give students the ability to develop Human-Machine Interface (HMI) and SCADA (Supervisory Control And Data Acquisition) applications on Windows NT or Windows 95 and run them on Windows CE runtime workstations.

### Main Course Objective

Upon completion of this course, students will be able to develop an application using Advantech Studio.

The following modules are included:

I.1. Company Overview and Available Products

I.2. Advantech Studio Overview

I.2.1. Advantech Studio Internal Structure

I.2.1.1. Development Modules (Database, Comm, Tasks, Graphics)

I.2.1.2. Runtime Modules (Viewer, BGTasks, Driver Runtime, OPC Client, TCP/IP Client/Server)

I.2.1.3. Utilities Modules (Database Spy, LogWin)

I.2.2. Advantech Studio Tasks

I.2.1. Recipes

I.2.2. Reports

I.2.3. Math

I.2.4. Alarms

I.2.5. Trend

I.2.3. Tag Definition

I.2.3.1. Types of Tags

I.2.3.2. Class, Pointer and Array

I.2.3.3. Fields

I.2.3.4. Internal Database, Application Database and Shared Database

I.2.4. Communication Tasks

I.2.4.1. Drivers

I.2.4.2. OPC

I.2.4.4. TCP/IP

I.2.5. Development / Run-Time concepts

I.2.6. Advantech Studio Environment

I.3. Tutorial Application

## System Requirements

To develop an application with the Advantech Studio software, we recommend the following hardware and software:

- IBM-compatible computer with an Intel® Pentium II-compatible processor
- Windows 95/98NT/2000 operating system
- 64 MB of random-access memory (RAM)
- MS Internet Explorer 4.0 or install **`40comupd.exe`** version 4.71 or higher
- 150 MB of free hard disk space is required for the program without any application programs; more is recommended
- 3.5" floppy drive
- CD-ROM drive (can be on a different computer)
- Standard keyboard with function keys **F1** through **F12**
- Parallel printer port (optional)
- 100% IBM-compatible VGA or SVGA display adapter with 2 MB Video RAM (VRAM)
- Microsoft-compatible pointing device (e.g., mouse, trackball, joystick, touchscreen)
- One or two COM ports and adapters for downloading applications (optional)
- Ethernet connection for downloading applications (optional)

To run a developed application, you must have an CE device runtime workstation with a runtime license. The operating system must be Windows CE v2.11 or higher and the files MFCCE211.DLL, ATLCE2311.DLL and OLECE211.DLL should be previouly sotred in the \Windows directory.

# 2. Installing Advantech Studio

The installation CD is used to install Advantech Studio. The Advantech Studio development environment runs on Microsoft Windows 95 or Windows NT 4.0. The CEView runtime environment, which comes preinstalled on the runtime workstation, runs on Windows CE 2.11.

**Section 1 – Introduction to Advantech Studio**

**What is Advantech Studio?**

Advantech Studio is a powerful software product designed specifically for the development of applications used in process supervision, automation, and control. Advantech Studio applications span the globe in a multitude of vertical markets including automotive; machine manufacturers; chemical; pharmaceutical; water and wastewater; food processing; glass manufacturing; biomedical manufacturing; fabric manufacturing; building automation; steel; oil and gas; pulp and paper; transportation; utilities; and more. The flexibility of Advantech Studio allows you to design and implement applications for:

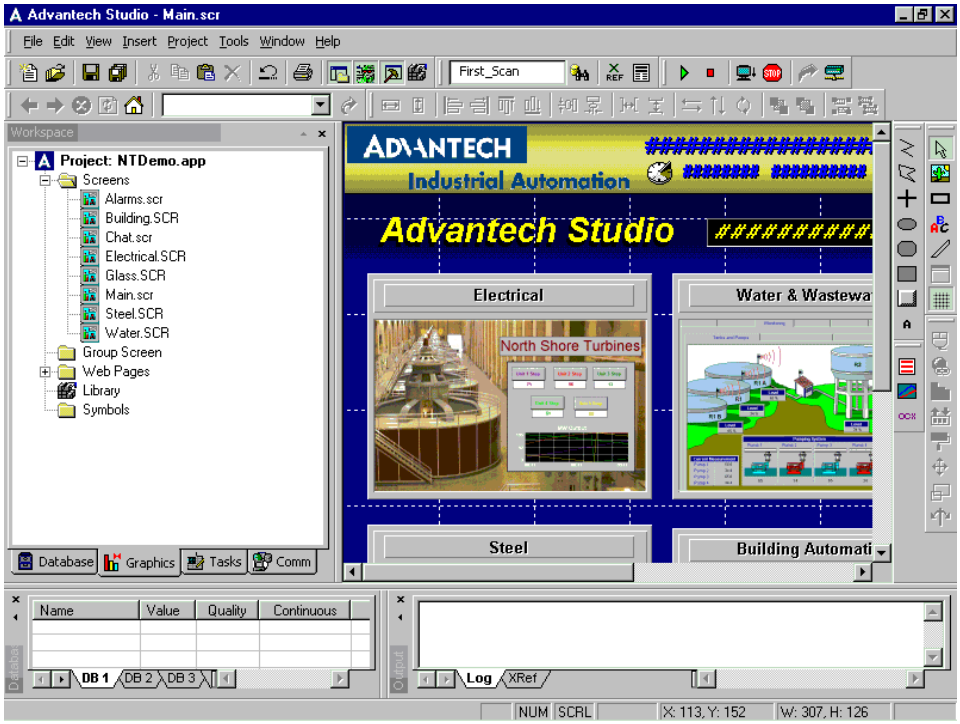
- Data acquisition
- Human-machine interfaces
- Local supervisory stations
- Data concentrators on distributed processes
- Remote supervisory stations
- Data communications with corporate systems

Advantech Studio process automation applications run on microcomputers connected in real-time to machinery or processes via programmable controllers, remote I/O devices, or other data acquisition equipment.

Advantech Studio is a powerful collection of automation tools that includes all the building blocks required to develop a modern human-machine interface (HMI) and a Supervisory Control and Data Acquisition System (SCADA).

Advantech Studio takes advantage of the resources available to Microsoft Windows to provide a powerful and versatile automation package that encompasses a variety of resources for the manipulation of data.

## Advantech Studio Development Environment




## Features and Functionality

Advantech Studio provides the tools you need to create a complete supervisory and process control system, with many innovative features.

### Powerful and simple object-oriented screen editor

The Advantech Studio screen editor allows you to create a variety of windows and dialogs that feature user input by screen selection and keyboard, output of control data to your process, and automatic updates to screen displays based on data input from your process. Other screen editor features include:

- Object grouping that preserves the construction steps of the individual objects.
- Editing without ungrouping internal object components and groups.
- Complete handling of bitmap objects and background bitmaps.
- Support for status lines in application windows and dialogs.

### Object-oriented database

**Array tags:** Any tag (variable) in the database can be defined as an array of up to 512 entries. Any place in the software where a name of a variable is used, it is possible to use *Tag[number]* or *Tag[another tag]*. Arrays simplify many configurations and permit the use of multiplexing in screens, recipes, and communication interfaces. Also, it save development time during the tag declaration.

**Indirect tags (pointers):** Use of the *@Tag* construct causes an indirect read or write. For example, if the X tag has the value "Setpoint" and you use the @X construct, then you are

reading or writing value of the tag Setpoint. You can use @Tag wherever you use a tag name, such as the name of a pen in a trend graph.

**Classes:** You can define a data structure such as: ClassPID { PV, CV, SET, KP, KI, KD } and declare tags or even arrays of the type ClassPID that will have groups of values instead of a single value.

You can even combine all tags features (array, pointer and classes) - For example: @Tag[another tag] or Tag[another tag].SP.

### Mathematical functions

Advantech Studio has an internal programming language used for writing logical relationships and mathematical calculations needed in applications programs.

### Online configuration

The runtime tasks accept new configurations immediately and without the need to restart or recompile the programs. You can change any element in the configuration, including mathematical calculations, reports, addresses in the PLC, and the type of a tag in the database. All these changes can take place on the fly, without halting the application or your process. You can also run an emulation of the application on your development computer and test it before downloading to the target runtime station.

### Easy addition of symbols

Reusable objects or groups of objects that you store for reuse are known as symbols. In a few seconds you can add a new symbol or modify an existing symbol, which allows you to quickly reuse symbols as you build your application.

### Report generation

Advantech Studio has all the tools you need to generate and save to disk reports that contain text and graphics, without requiring the use of other software packages such as Microsoft Excel.

### Recipes in ASCII

The Advantech Studio database supports direct access to recipe files written in ASCII.

### DDE, NetDDE, ODBC and OPC (on Windows NT, Windows 95)

The product provides DDE, NetDDE, and OPC as well as the ODBC interface for access to relational databases on the Windows NT and Windows 95 runtime. The interface OPC is also supported by Windows CE.

### Security system

The internal security system allows you to assign permissions and capabilities to individual users and user groups, each with its own password. You can apply security restrictions to the applications you create with Advantech Studio and you can also limit access to the development tools provided by Advantech Studio. The security system is also applied to the remote clients, connected by Web (Intranet/Internet).

### Batch historical files

Advantech Studio data collection features allow your applications to save and retrieve historical data files, using archives based on time or user-defined names. This is an essential feature for batch systems.

### Alarms

Your Advantech Studio applications can include the following alarm features:

- Free formatting of alarm messages.
- Use of a secondary search key.
- Access alarms through groups or by tags.

### Development support

Advantech Studio contains many tools to help you development applications, such as an easy-to-navigate user interface, message register, error codes, and event codes used during execution. Development tools also handle database access and the creation and verification of variables.

### Advantech Studio application programming interface (API)

All Advantech Studio software modules are developed using an open API. This permits easy growth since the development "kernel" and the application tasks are maintained independently. The API also allows you to create new software modules in any language with .DLL support.

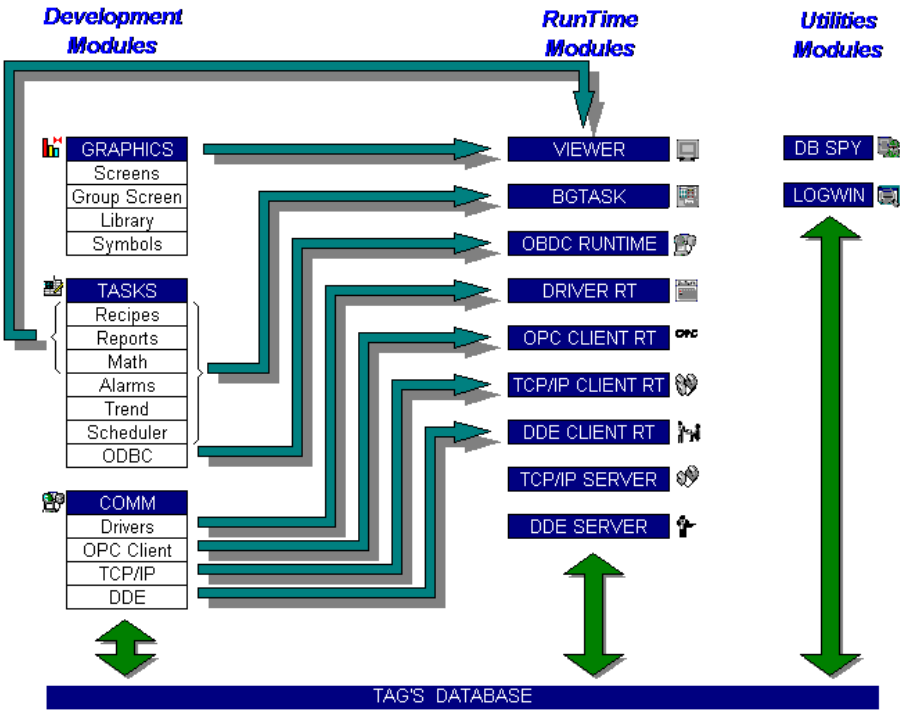
### Dynamic support for multiple languages

To create an application that supports multiple languages, create your application in your primary language, create text files in other languages, then use a table for translation. This feature allows you to create applications that dynamically change languages during execution.

# 3. Advantech Studio Overview

## Advantech Studio Internal Structure

This section describes the Advantech Studio internal structure:




## Tag Database

The tag database is the heart of Advantech Studio product. In Advantech Studio, you use the same tag names in the worksheets and the displays, and Advantech Studio manages tag values among the modules. All modules share information through the application database. The values of the application tags and Advantech Studio internal tags are stored in this database during system execution. The application database is a memory area accessed by all modules to read or write values.

Configuring an application consists of defining which tags must be accessed by each module. This means that application development follows the same logical sequence, regardless of the number of tags involved in a particular application.

# Development Modules

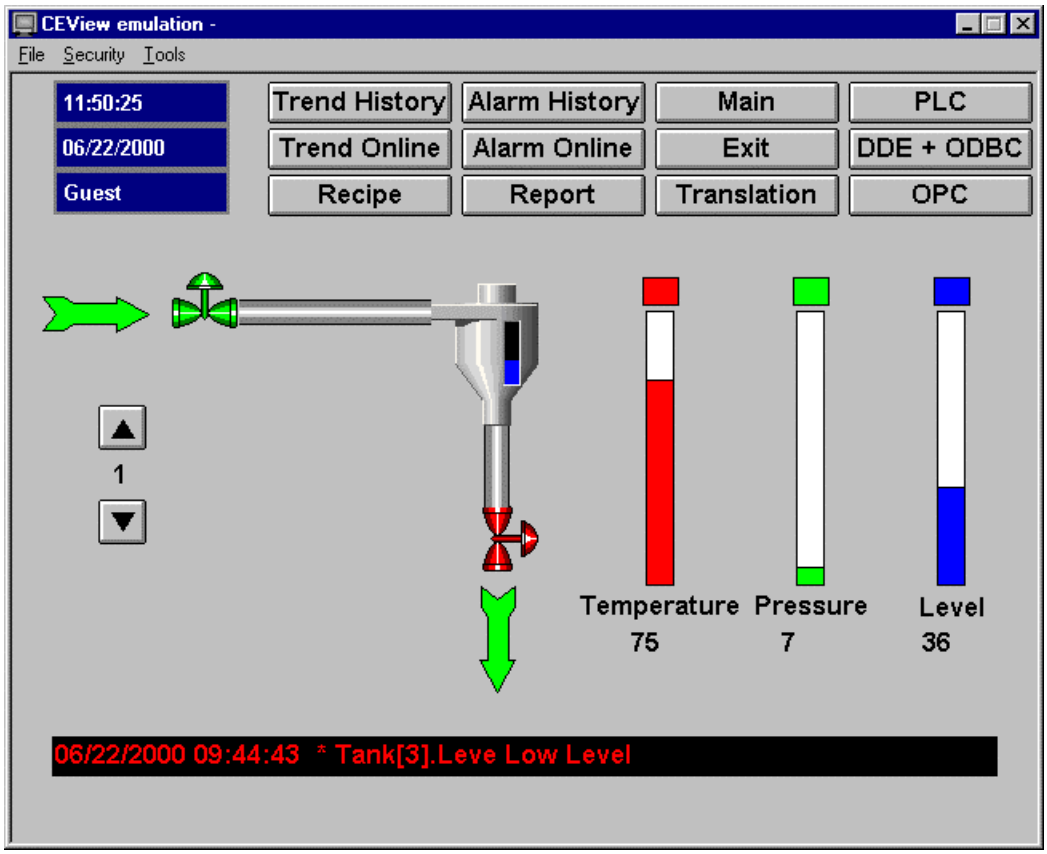
## Graphics

The most basic function performed by Advantech Studio is to provide a window into the process. This ability to display the status of the process, by interacting with instrumentation or computers, is described as the Human-Machine Interface (HMI). Note that if a system only works with sensors, controllers, and other process equipment, the HMI is often limited in scope.

Applications created by Advantech Studio monitor processes using high-resolution color screens. The Advantech Studio graphic tools consist of two modules: the worksheet editor in the Advantech Studio desktop and the application runtime Viewer.

The worksheet editor is what you use to create or import graphics. Graphic objects or symbols can be dynamic objects by using animation links. Animation links can make an object or symbol change appearance to reflect changes in the value of a tag or an expression. Each screen is an association of static and dynamic objects.

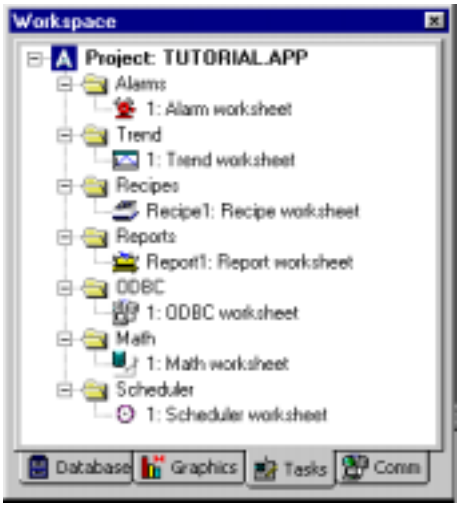
Screens can also have an optional bitmap that acts as a background in the object window. For example, the static images in the screen below can be part of a bitmap in the background object, and objects with animation in the dynamic object layer can reflect the changes in the plant. In turn, the screen has the illusion of being three-dimensional.




The Viewer enables you to preview the runtime screen in an emulation mode.

All of the Advantech Studio configuration tasks require a Windows-compatible pointing device such as a mouse or touch pad. You can run an application in the Viewer without a pointing device if you configure keypad or keyboard keys for all commands.

### Advantech StudioTasks




The Advantech Studio tasks is where you configure task-specific worksheets, each composed of a Header where the global information in the sheet is defined and a Body where the tags and expressions used in each task are configured.

**Alarm groups**

Here you define an alarm group, its characteristics, and messages that are reported in alarm conditions. The main purpose of the alarms is to inform the operators about any problem or change of state during the process so that corrective action can be taken.

To show alarm messages on the screen, you must create the alarm object on the screen.

**Trend groups**

Trend groups keep track of process variables behavior. You can store the samples in a history file and show both history and online samples in a screen trend graph.

**Recipe**

This module allows you to read and write ASCII files from and to the hard disk; it transfers values between files and real-time memory. Its typical use is to store process recipes, but these files can store any type of information such as operation logs, passwords, and so forth. It allows also you store data in XML format.

**Report**

Use this module to configure your own reports with system data, in both ASCII or RTF format. The main purpose of this module is to make report creation easier and more efficient.

**Math Worksheet**

This module allows you to implement additional routines to work with the basic functions of the Advantech Studio modules. A Math worksheet is a group of programming lines that are executed as one of the application Background Tasks. You can configure the mathematics in blocks in different worksheets.

This worksheet provides a free environment for logical routines and mathematical calculations that the project may need. For these purposes, the scripting language is simple and easy to use.

**Scheduler**

This module generates time bases used in the application and it is capable of triggering events.

**ODBC Configuration (not available for WinCE applications)**

The ODBC interface will allow Advantech Studio applications to access any database compatible with the ODBC protocol, like Access, Excel, Oracle, SQL Server and so on.

# 4. Creating a New Project

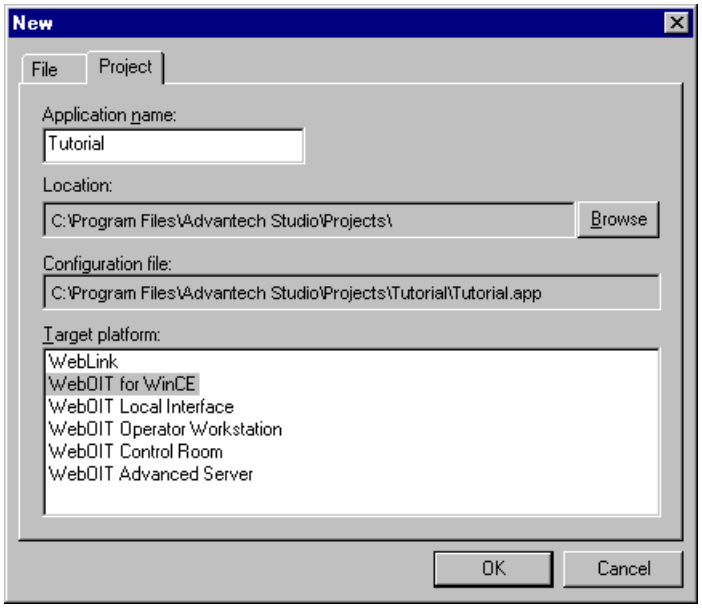
To create a new project, run Advantech Studio by double-clicking the icon, or by choosing **Start | Programs | Advantech Studio Tools | Advantech Studio**.


**Note:** You can also open Advantech Studio in the Windows Run window (accessed by **Start | Run...**) command line, with the following command:

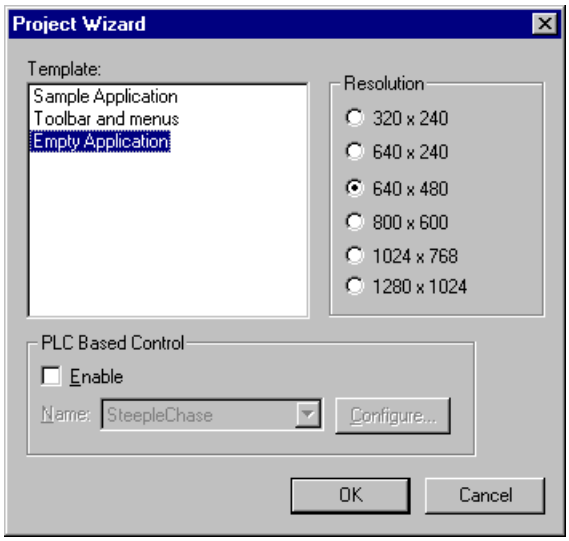
```
"C:\Program Files\Advantech Studio\Bin\RunStudio.exe"
```

This command will work only if you installed Advantech Studio in the default folder during installation. Otherwise you will need to enter the path that you used during installation. You can find **RunStudio.exe** using the Windows Find command (**Start | Find | Files or Folders...**, then type **RunStudio.exe** in the **Named** field.)

In the Advantech Studio environment, select **File | New...** (or the **New** button ) to open the New window. Select the **Project** tab and type the name of the project (**Tutorial**) in the **Application name** field. For this tutorial, the default selection (*WebOIT for WinCE*) should be highlighted in the **Target platform** pane. After you are done, press the **OK** button.



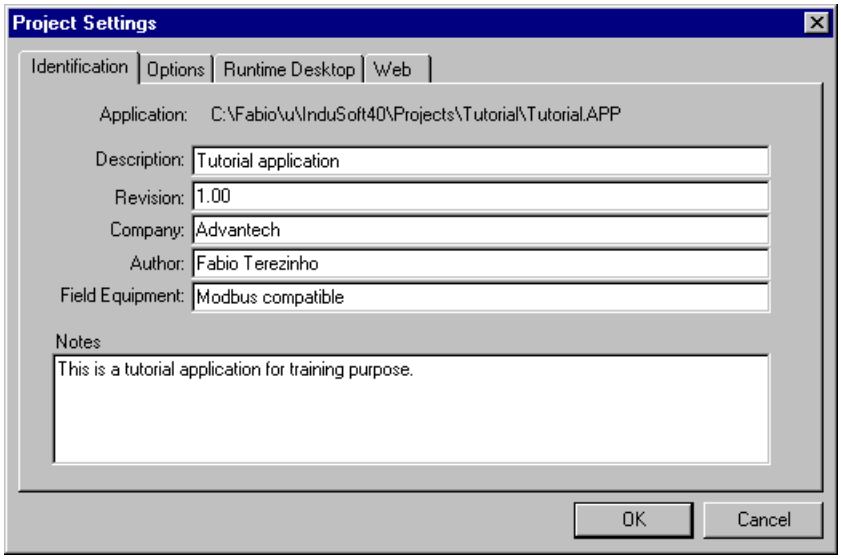

The Project Wizard window will open. Select the **Empty Application** option in the **Template** pane and the **640 x 480** radio button in the **Resolution** group box. After you are done, press the **OK** button.



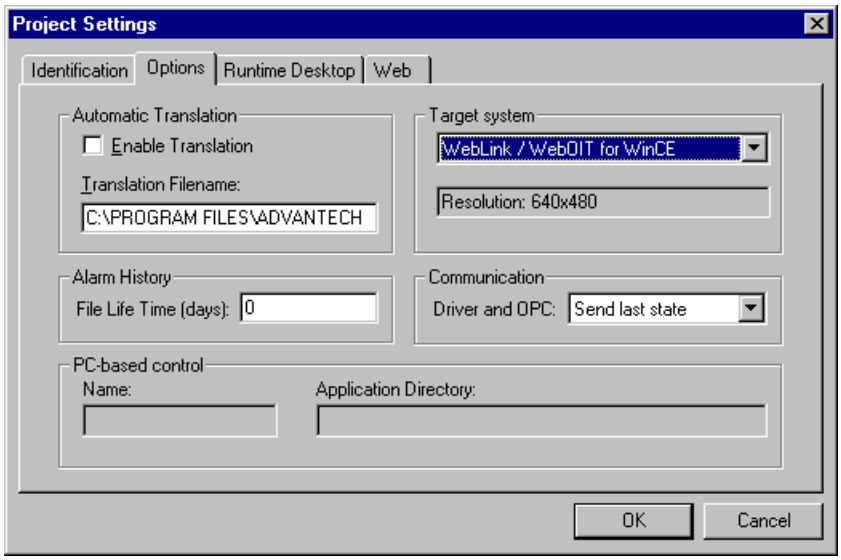

## Configuring Project Settings

In the main menu, select **Project | Settings...** to open the Project Settings window. The Project Settings window has four tabs.

The **Identification** tab is reserved for documentation information about our project. These fields are optional.

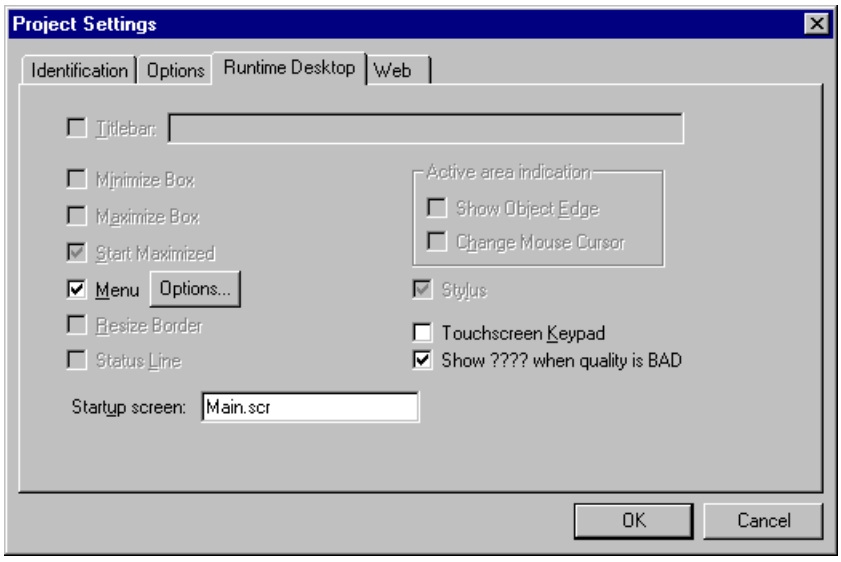



The **Options** tab contains settings for language translation, target system, PC-based control, driver write command buffering, and general information about the application.



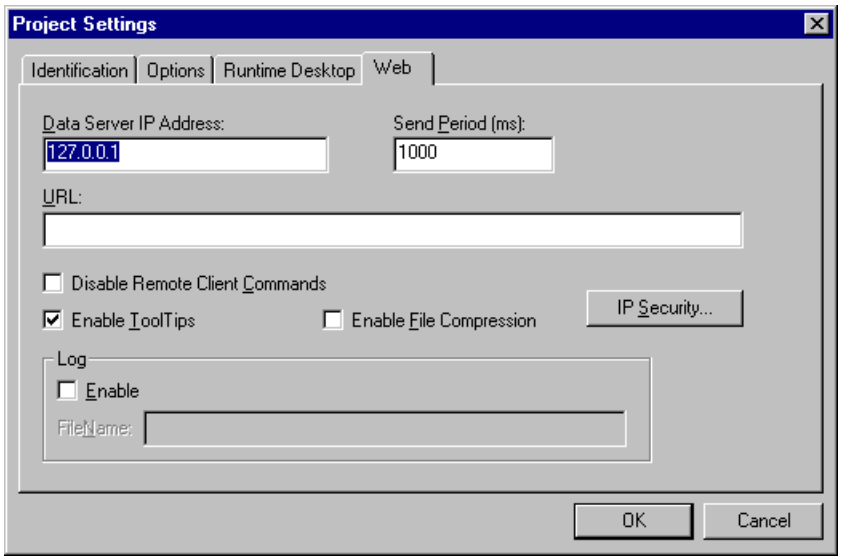

The **Runtime Desktop** tab contains global settings for the application that determine how it will run on the runtime workstation and what menu options will be available. You always need to specify the first screen that will open in the application when it runs in emulation mode or on the runtime workstation. For this tutorial, type Main.scr in the **Startup screen** field. (The file extension is optional.)

**Note:** There is a screen group feature that allows you to open a linked set of screens all at once. It's not supported for WinCE applications or for screens which should be exported to HTML format.




**Note:** All settings defined in the Project Settings window can be changed during application development, but we recommend that these settings be configured at the beginning. For example, the **Startup screen** field defines which screen will be opened when the application is started and there will be an error message generated if you try to emulate or run the application without a legitimate value in this field.

The **Web** tab contains global settings for the remote thin clients, which will access the application by a web browser (Internet Explorer). These setting can be configured at any time, however, after modify any parameter it's necessary to execute the command ***Verify Application*** from the ***Tools*** menu to update the HTML files with the new settings.

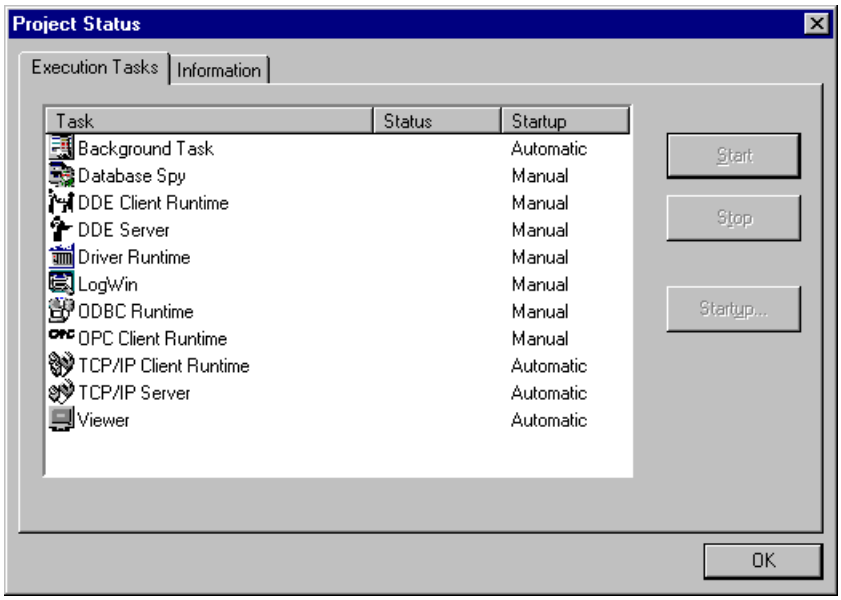



Press the **OK** button to exit the Project Settings window.

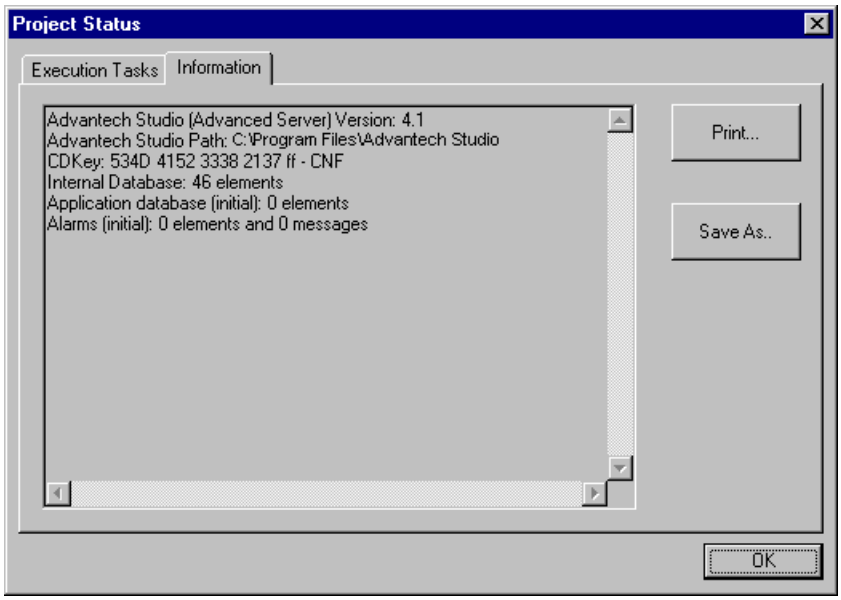
## Configuring Project Status

In the main menu, select **Project | Status...** to open the Project Status window. It has two tabs.

The **Runtime Tasks** tab allows you to monitor and control the execution of each runtime task by starting and stopping the tasks using the **Start** and **Stop** buttons. The **Startup...** button is used to configure whether a runtime task is started by *Automatic* or *Manual* methods. It is not used in applications for WinCE because all necessary runtime tasks are started automatically in the target system.



The **System Information** tab provides some general information about the development system and about the application.




Press the **OK** button to close the Project Status window.

# 5. Tag Database Configuration

*Tags* are variables used in screens and task worksheets. Tags can be communication points in the field equipment, results of calculations, alarm points, and so forth. User-created tags are called *application* tags; tags predefined by Advantech Studio are called *internal* tags. You can use either type of tag with any Advantech Studio module; the only difference is that internal tags have predetermined functions. Tags values are stored in the Application Database.

Tag syntax rules:

- It can be composed of letters, numbers, and the underscore character (_).
- It must begin with a letter.
- Its maximum length is 32 characters (for a tag), or 16 characters (for a class member).
- The tag name must be different from internal tag names and math functions.
- Tag names are NOT case sensitive.

*Examples: Temperature*, *pressure1*, *count*

Advantech Studio does not differentiate between uppercase and lowercase characters. However, you can use both uppercase and lowercase characters to make names more clear (Example: *TankLevel* instead of *tanklevel*).

## Types of Tags

The value of a tag can be one of the four standard types:

**Boolean** (4 bytes): Boolean or digital variable (**`0`** or **`1`**).

**Integer** (4 bytes): Integer number (positive, negative, or zero). Equivalent to *C* type long integer. (range of **`-2147483647`** to **`2147483647`**)

**Real** (floating point, 8 bytes): Real number internally stored as a double word. Equivalent to C type double.

**String** (ASCII, 256 bytes): Character string up to 255 characters (from 0 to 254) that holds letters, numbers, or special characters.

*Examples:* **`Recipe product X123`**, **`01/01/90`**, **`*** On ***`**

All tags are declared in the Application Database module of the Database tab. In addition to the four types previously listed, you can define new types called *classes*.

## Classes, Arrays and pointer tags

### Array tags

Advantech Studio tags can hold a single value or an array of values. An array tag is a set of tags with the same name; it is identified by indexes (a matrix of n lines and 1 column). The maximum array size is 512 (positions from 0 to 512).

*Examples:* **`tank[1]`**, **`tank[2]`**, **`tank[3]`**, **`tank[500]`**

Use array tags whenever possible because they simplify the configuration task. Suppose that you want to have a display to monitor each tank. Using array tags makes it possible to configure a single display that contains tags linked to any tank:

**`pressure[tk]`**, **`temperature[tk]`**, **`temperature[tk +1]`**

The tag *tk* is the index that contains the number of the desired tank. An array index may be a tag, a numeric value, or an expression with a value plus a tag.

To refer to an array that has an index with the arithmetic operation +, you must use the following syntax: **<*tag name*>`[`<*tag*>`+N]`**, where **`N`** is a numerical constant.

Examples: temperature[tk+2], temperature[tk+6]

Using array tags can save a lot of application development time. Suppose that you need tag points related to the temperature of four tanks. The conventional configuration method is:

**`temperature1 high temperature on tank 1`**

**`temperature2 high temperature on tank 2`**

**`temperature3 high temperature on tank 3`**

**`temperature4 high temperature on tank 4`**

Using array tags simplifies this task:

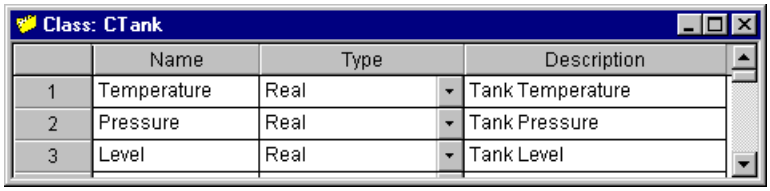
**`temperature[j] high temperature on tank {j}`**

When you create a four-position array tag, the system creates five positions (from 0 to 4). Therefore, the TagExample[15] array has 16 elements.

**Note:** When you try to reach an invalid index, such as 20 on a 15-element array, the 0th element is used, therefore you are advised not to use it for normal operations.

## Classes

In addition to the standard tag types, you can also define new types of tags called *classes*. Classes are structures that allow a high degree of encapsulation in the application database. When a class-type tag is created, it does not contain a single value, but a whole set of values. You can create class-type tags by grouping up to 32 simple tags, called *elements*. When you create a class tag, its tag does not have a single value, rather it has a group of values associated with the class, for example:




Members of a class can hold standard values, as previously described. If you create a new tag *Tank* of type *CTank*, you are actually creating a tag with all the properties of the class *CTank*. To access the members of a class tag, you should use the period (.) separator.

*Examples: Tank.Level*, *Tank.Temperature*

If tag *Tank* is an array, the syntax would be: *Tank[1].Level*, *Tank[n].Temperature*

## Indirect tags – references / pointers

Advantech Studio supports indirect access to tags in the database. For example, consider a tag X of the string type. This tag can hold the name of any other tag in the database (that is, it can provide a pointer to any other type of tag, including a class type). The syntax is:

**@**<*name of indirect tag*>.

For example, assume that a tag named *X* holds the string **`Temp`**. Reading and/or writing to **`@X`** provides access to the value of the tag *Temp*. To refer to a class-type tag, you must also use the Database Manager to define a tag of type string that points to this tag. You can define the tag directly by simply declaring, for example *@XClass* in the **Tag Name** column as a member of the class. By doing so, you are making *XClass* a reference to another tag. To access a member of an indirect class tag, use the following syntax:

**@**<*name of indirect tag*>**`.`**<*member*>

*Example:* **`@XClass.Level`**

In this case you are accessing the member *Level* of the tag that *XClass* points to. When you create tags for indirect use, place an *@<name>* in the tag column rather than creating them as strings. For the type, write the type of tag for which a reference is being created. Follow the *XClass* example:

| | Name | Array Size | Type | Description |
|---|---|---|---|---|
| 1 | @Z | 0 | Integer | |
| 2 | @X | 0 | Boolean | |

Any string tag is a potential indirect tag (pointer).

## Fields - Tag Parameters

*Fields* are a set of parameters related to each tag in the database. The application can access these fields during runtime (or during application development) using the following syntax:

**`tagname->`**<*field name*>

*Examples:* **`level->Max`**, **`Temp->Unit`**, **`pv101->HiHiLimit`**

You can access the following fields at runtime:

*Max* – the maximum tag value.

*Min* – the minimum tag value.

**Note:** If you try to write a value outside of the range specified by *Max* and *Min*, it will not be accepted and a warning message will be generated in the LogWin file. If you do not wish to use these properties, enter **`0`** in both fields.

*Unit* – an up to 8 character string used as reference for engineering units.

**HiHiLimit* – the numeric threshold for a High High alarm.

*Example:* **`TP->HiHiLimit=70`**

**HiLimit* – the numeric threshold for a High alarm.

**LoLimit* – the numeric threshold for a Low alarm.

**LoLoLimit* – the numeric threshold for a Low Low alarm.

**DevLimit* – the numeric threshold for a Deviation alarm.

******RateLimit* – the numeric threshold for a Rate alarm.

******DevSetpoint* – the reference setpoint for a Deviation alarm.

*Description* – the documentation-only description field.

******AlrDisable* – disables alarm checking as follows: **`1`** disables alarm and **`0`** enables alarm

*Example:* **`TMP->AlrDisable=1`**

*Size* – the size of an array tag; the default is size 0 for a non-array tag.

*HiHi* – if nonzero, a High High alarm is present.

*Hi* – if nonzero, a High alarm is present.

*Lo* – if nonzero, a Low alarm is present.

*LoLo* – if nonzero, a Low Low alarm is present.

*Rate* – if nonzero, a Rate alarm is present.

*Dev* – if nonzero, a Deviation alarm is present.

*TimeStamp* – the date and time of the last tag value change.

*b0* through *b31* – allows access to each bit from an integer tag.

**Note:** During runtime, you can modify fields related to alarm limits, indicated by the ***** in the description above.

## Internal, Application, and Shared Tags

### Internal Tags

Internal tags are predefined and perform specific functions within Advantech Studio supervisory tasks. Most internal tags are read-only. To change the time, for instance, use the proper math function rather than writing to the internal time tag.

*Examples: Date* holds the current date in string format and *Time* holds the current time in string format.

### Application Tags

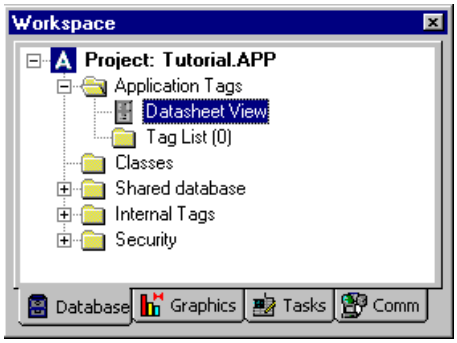
Application tags are created by the user during application development (for example: displays, tags that read from and write to field equipment, tags used for control, auxiliary tags to perform mathematical calculations, and so forth).

### Shared Tags

These are tags read from another software package (generally a PC-based control software package). These tags cannot be edited.

## Database Exercise

In the Workspace window, select the **Database** tab. Click the **Application Tags** folder to expand it and double-click the **Datasheet View** line to open the database worksheet.




The database may be updated during the development so that new tags may appear as they are created. We can also define tags that we already know we will need at the beginning of our application. The first tags are contain values for the state of the valves that fill or empty the tanks (**`Valve_Fill_State`** and **`Valve_Empty_State`**). Each valve has only two possible states (opened or closed), so a tag associated with these valves should be a Boolean tag. There will be three tanks in the application, each configured similarly, and there are two types of valves: one to fill and one to empty each tank. We can use arrays to quickly configure the tags associated with all three tanks. We need to configure the **Application Datasheet** as follows (in order to fill in the field, click on the field and type the data; you can also tab to the next field):

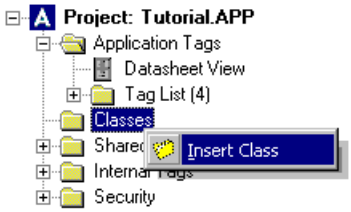
| | Name | Array Size | Type | Description | Web Data |
|---|---|---|---|---|---|
| 1 | Valve_Fill_State | 3 | Boolean | Fill valve state (openclosed) | Server |
| 2 | Valve_Empty_State | 3 | Boolean | Empty valve state (openclosed) | Server |

**Note:** We are only using two lines in the database to configure six tags. Arrays reduce the time spent configuring the database. Also, arrays allow us to configure functions and scripts that can optimize the overall application.

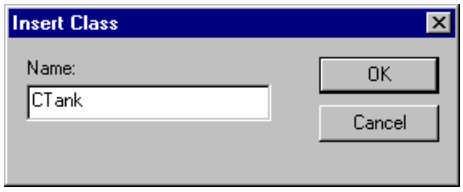
We have configured the tags that will receive the state of each valve. Now, we need to configure the tags that will be used to send commands to the host controller. These tags have the same number of states and other characteristics of the previously configured tags:

| | Name | Array Size | Type | Description | Web Data |
|---|---|---|---|---|---|
| 1 | Valve_Fill_State | 3 | Boolean | Fill valve state (openclosed) | Server |
| 2 | Valve_Empty_State | 3 | Boolean | Empty valve state (openclosed) | Server |
| 3 | Valve_Fill_Command | 3 | Boolean | Fill Valve command (open/close) | Server |
| 4 | Valve_Empty_Command | 3 | Boolean | Empty Valve command (open/close) | Server |

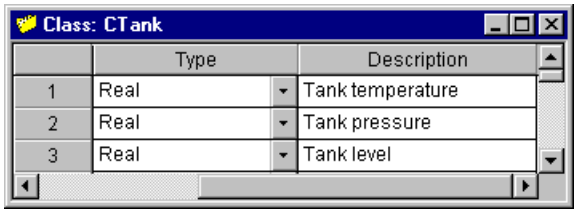
Finally, we need to create tags to store the properties associated with the tanks: temperature, level and pressure. These attributes are associated with one piece of equipment, a tank. We will use a tag class to tie all these properties together and associate them to a tank. To create a new class, select the folder **Classes** (in the **Database** tab), right-click it, and select **Insert Class**.




In the dialog box, type in the new class name, **`CTank`** and click the **OK** button.




In the **Application Tags** database worksheet, we will reference the properties of the tank by using tags of **Type** *Class: CTank*. Each tank property is defined as a member of the class *CTank* and each is defined similarly to a tag with a **Type** parameter: *Boolean*, *Integer*, *Real*, or *String*. All of the tank property class members are analog, so declare them to be real:

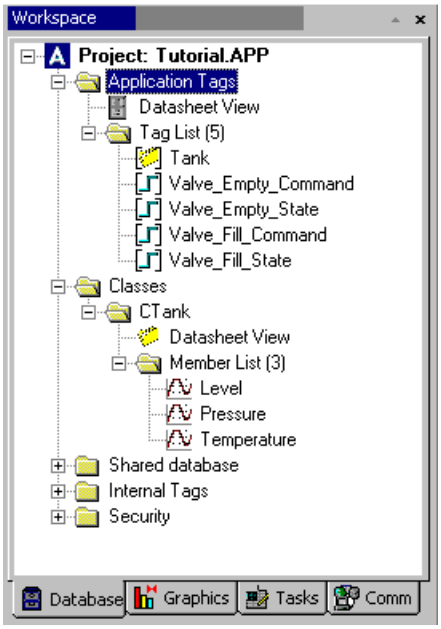



**Note:** You can expand the **Classes** folder and its subfolders to see the data structure.

Close the **Class: CTank** worksheet and create a tag associated with this class. To create the tag, open the **Application Datasheet** and insert a tag (**`Tank`**). In the **Type** column choose the option *Class: CTank*. Since we have three tanks, configure the **Array Size** to be **`3`**.

| | Name | Array Size | Type | Description | Web Data |
|---|---|---|---|---|---|
| 1 | Valve_Fill_State | 3 | Boolean | Fill valve state (openclosed) | Server |
| 2 | Valve_Empty_State | 3 | Boolean | Empty valve state (openclosed) | Server |
| 3 | Valve_Fill_Command | 3 | Boolean | Fill Valve command (open/close) | Server |
| 4 | Valve_Empty_Comman | 3 | Boolean | Empty Valve command (open/close | Server |
| 5 | Tank | 3 | Class: CTank | Tank Data | Server |

You have completed creating the initial tag database for our application; now expand the **Application Tag** and **Classes** folders in the **Database** tab to see the database you have created:




**Note:** You can create new Application Tags and new Classes, right-clicking on their respective folders and chosing the ***Insert*** option. In addition, you can also modify a tag properties, right clicking on its icon and chosing the ***Properties*** option. During the application development, when you type an inexistent tag, Advantech Studio will prompt you if you desire to create this new tag. If you accept, it'll create a wizard which allows you create the tag by a easy and fast way.

# 6. Creating New Screens

Before creating any screens, you should think about the structure of the application screens. It is possible to open more than one screen at the same time in applications for Win9x/NT/2000, but in the Windows CE version you can create a default screen with a header and footer (template) and insert objects in this window, saving it under different names to make the different screens. Usually, there are three basic areas on a screen (or types of screens):

Header: Objects on the top of the screen that provide standard information (date, time, etc).

Footer: Objects at the bottom of the screen, often an alarm object showing the last alarm.

Regular: This is the space between the header and the footer. It can show information about the process, an alarm screen, trend, etc. …

The advantages to using this structure to develop our screens in our applications are that it:

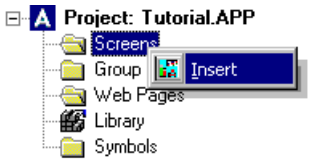
- Ties screens together according to their utility in the application.
- Allows you to configure the links and dynamics that are common for all screens just once.
- Gives the application a default format.
- Allows you to build modular screens that can be used in other projects.

With this recommended structure, we can begin to create screens for the application.

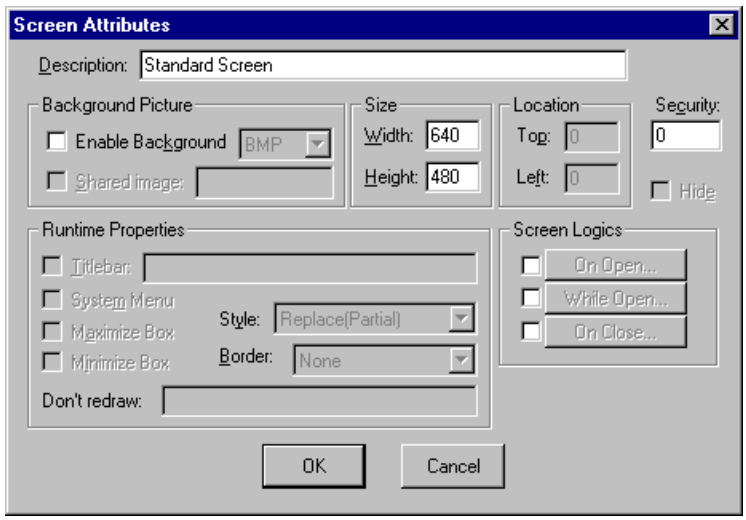
## Creating a Standard Screen

The standard screen is going to be used as a template for all of the other screens. The rest of the screens will be created using this screen and saving it with other name. This way we always will have the same screen attributes and basic objects across all of the screens.

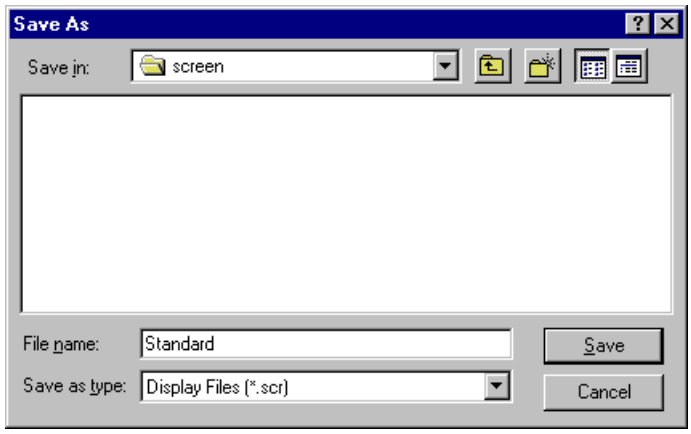
Select the Workspace window **Graphics** tab, right-click the **Screens** folder, and select **Insert** (or select **Insert | Screen**).



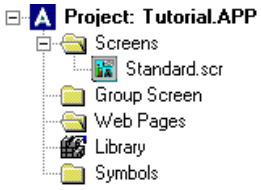

In the Screen Attributes window, you must configure the general information about the screen. The first screen we are going to create is the standard screen.



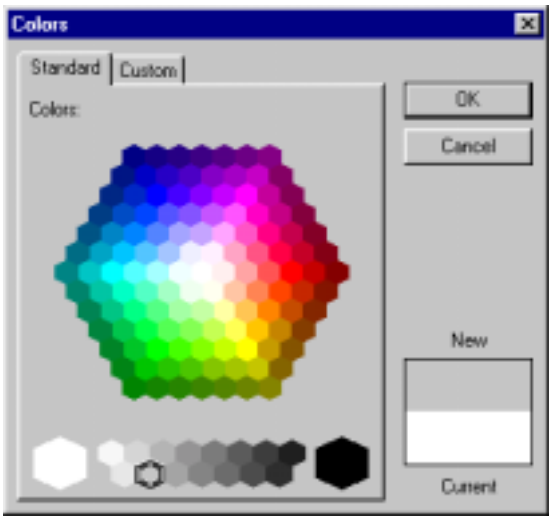

Press the **OK** button and an empty screen with the selected attributes will appear. Save the screen to preserve the settings by clicking the **Save** icon in the Main toolbar (or **File | Save** or **File | Save As...**) and type **Standard.scr** (or **Standard**) in the **File name** field.




Expanding the folders in the **Graphics** tab, you can see the saved screen.




After creating the screen, change the background color. Select the gray background color using the **Background Color** button on the Tools toolbar to open the Color window, or just right-click on the screen and chose the **Background Color** option:



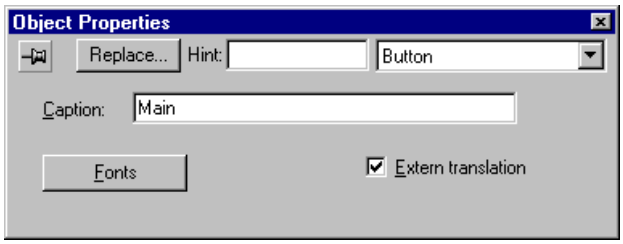

Select the light gray color and select the **OK** button or double-click the light gray color.

## Drawing the Header Objects

### *Drawing the Buttons*

Draw one button on the top of the screen. This button will be used to navigate between application screens. (Sugestion: draw a button with Width=120 and Height=30 pixels. Use the status bar to monitor the object size).

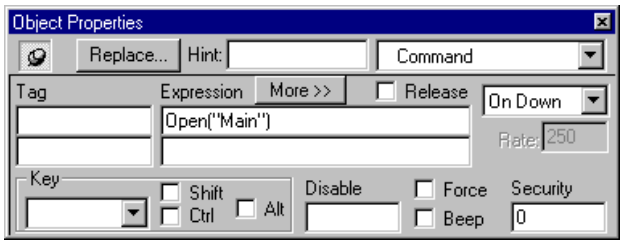

After drawing a button, either right-click it and select **Properties** from the menu, use **Alt** + **Enter** while it is selected, or double-click it to open the Object Properties window for the button. In the **Caption** field, type **`Main`** to change the caption text on the button.




### *Creating the Button Links*

All of the buttons except for **`Exit`** are used for opening screens. The function **`Open("<`***screen name***`>")`** is used for this. Although the screens are not created yet, you can still configure the buttons.

To add a dynamic property to an object, select the object, select the **Command** button on the Tools toolbar, and open the Object Properties window, which should have *Command* selected in the drop-down menu in the upper right corner. (If not, select *Command* on the menu.) If the Object Properties window is already opened, it will switch to the *Command* configuration. Select the **`Main`** button, add the **Command** property and enter the **Open()** function in the first **Expression** field as shown below:




After this, create eleven more buttons and label them **`Trend Online`**, **`Trend History`**, **`Recipe`**, **`Report`**, **`Alarm Online`**, **`Alarm History`**, **`LogOn`**, **`Exit`**, **`Translation`**, **`PLC`** and **`OPC`**.

**Note:** You can use Copy (**Ctrl** + **C**) and Paste (**Ctrl** + **V**) commands to quickly create buttons that are all the same size. You can also hold the **Ctrl** key and click-and-drag copies off of the original button with the mouse. Using the alignment and spacing buttons on the Tools toolbar, you can quickly symmetrically align and space the buttons. To change the font, press the **Fonts** button and select the new settings (Sugestion: Use font size=10 and font type=Arial).

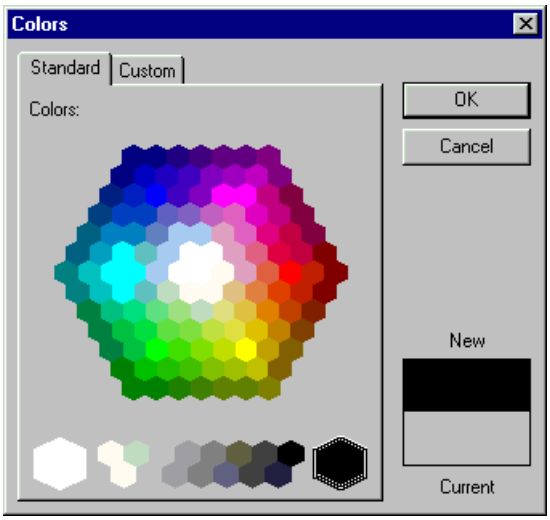
When you are done, the top of your screen should look like this:


After this, select another button, add the **Command** property, and configure the **Open()** function in the Object Properties window with the correct screen name. The screen names should be: **`TrendHistory`**, **`TrendOnline`**, **`Recipe`**, **`AlarmHistory`**, **`AlarmOnline`**, **`Report`**, **`Translation`**, **`PLC`** and **`OPC`**. Configure the expression **`LogOn()`** for the **`LogOn`** button. Configure the expression **`ShutDown()`** for the **`Exit`** button.

### *Drawing the Legend Objects – Rectangles*

Draw three rectangles with the **Rectangle** button, they will look like this:

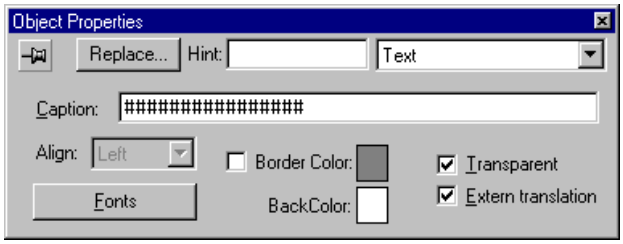

Select all rectangles and select the **Fill** group box, select the **Fill Color** button on the Tools toolbar. Select the black fill color shown below. (You can also use the **Fill Color** propertie in each Object Properties dialog window.)



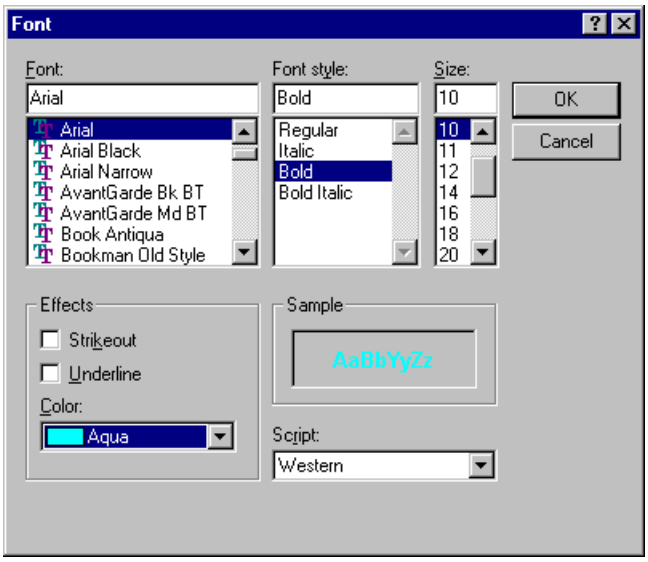

**Note:** Depending on the palette configured on the system, graphic objects that are imported into the Advantech Studio environment may have color distortion. If this happens, change the pallete configured in your system.

### *Drawing the Legend Objects – Rectangle Text*

Using the Tools toolbar **Text** button, add a text object in each rectangle consisting of ten number signs in the upper left rectangle, eight number signs in the upper right rectangle, and at least ten number signs in the lower rectangle.



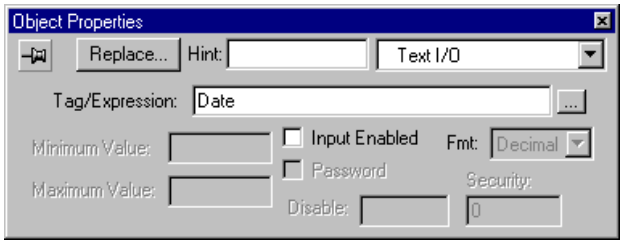

In the Objects Properties window for each, select the **Transparent** check box and then click the **Fonts** button and set the text to the following parameters. (The Tools toolbar **Fonts** button may also be used to change text properties.)




**Note:** We suggest to set Arial font with wize=10 and Aqua color.

### *Adding Dynamic Properties to the Legend*

To associate tags or expressions to text objects, use the **Text I/O** button on the Tools toolbar. Select any text with a number sign character (**#**), then select the now-active **Text I/O** button. The upper text object (with ten number signs) will be used to display the system date. Open its Object Properties window. The **Text I/O** Object Properties window should be shown; if not, select the *Text I/O* option in the upper right drop-down menu. Type **`Date`**, the name of the internal tag that holds the system date, into the **Tag/Expression** field. When running, the application will display the system date in place of the number signs.




Copy the object twice and move them on the other two rectangles previously created.

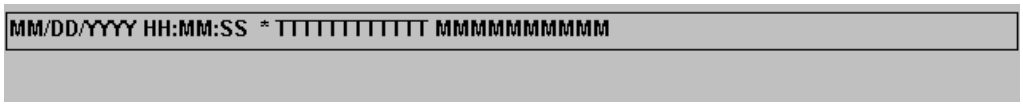
Select the middle text object and repeat the above process, entering **`Time`**, the name of the internal tag that holds the system time, into the **Tag/Expression** field of the **Text I/O** Object Properties window for that object. Repeat again for the lower text object, using **`UserName`**, the internal tag that holds that string identifying the current user of the application.

At this time, we can quickly test these functions by selecting the **Test Display** button of the Execution Control toolbar. This is what you should see:

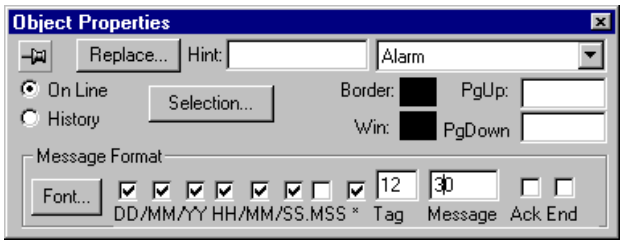
## Creating the Footer Object

This footer area will include a single line alarm object that will display the latest active alarm.

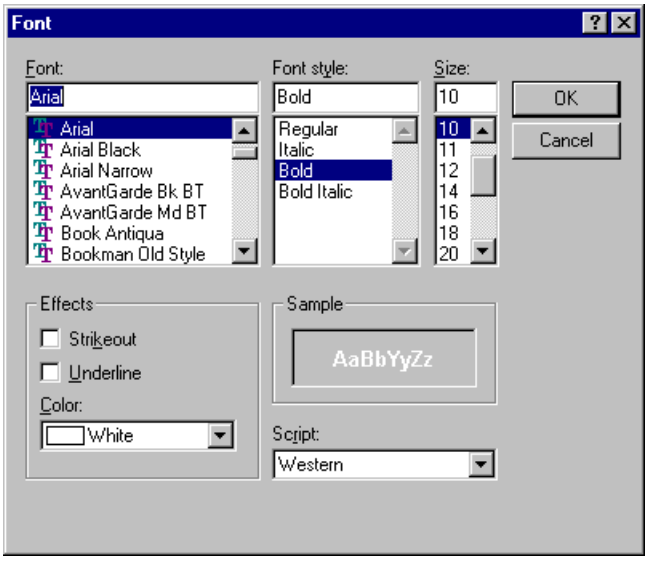
To create the alarm object, select the **Alarm** button from the Tools toolbar and click-and-drag to define a rectangle that shows on line of masking letters. Suggestion: Position the Alarm object higher than 40 pixels from the bottom of the screen (Y=420) so that it will not be cut off when the application Menu Bar is displayed.




Open the Object Properties for the alarm object and configure it as shown below. Changing the **Border** and **Win** colors to black will change the alarm object to a solid black rectangle. Configure up to 30 characters in the *Message* text box.

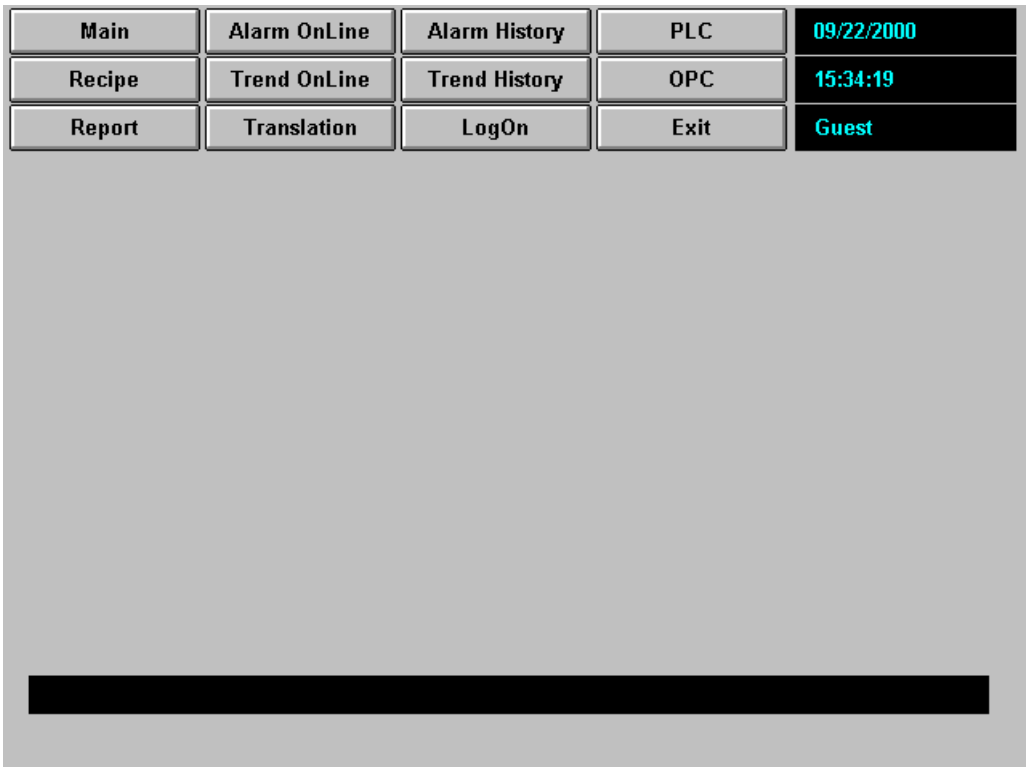



Press the **Font...** button to change the **Font** to *Arial*, **Font style** *Bold*, **Size** *10*, **Color** *White*. (You may also use the **Font** button from the Tools toolbar.) The font color will be set later in an **Alarm** worksheet.




You will need to make the alarm object rectangle about 600 pixels wide (as shown in the Status Bar) to accommodate the masking letters at a 10-point Arial typeface.

When you are done, make sure that you save the Standard screen.




## Creating the Main Screen

This screen will always be opened when the application is started. Save the Standard screen as **`Main.scr`** using **File | Save As...**. The Standard screen still exists and will be used as the template for all of the other screens as well.

In this main screen now you will:

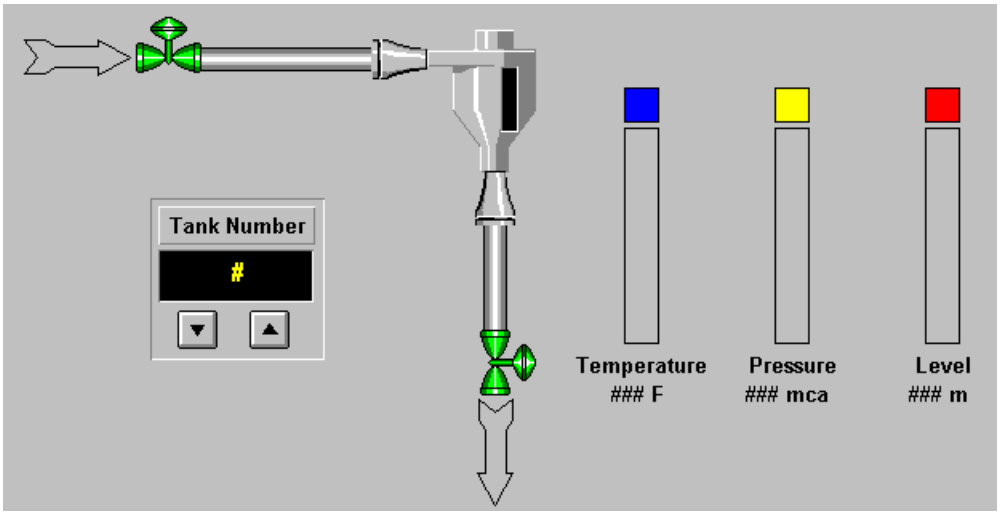
- Display the properties of the tanks (temperature, pressure and level) graphically and numerically.
- Display the state of the valves using color (red = closed, green = open).
- Give commands to open or close each valve.

**Note:** Since the three tanks have the exact same characteristics, we will build just one screen that is generic for any tank in our application. To switch to a different tank, we will simple change the index of the tags in the array used in our configuration.

The individual objects that make up the tank and plumbing graphics (the arrows, valves, pipes, and tank) are taken from the preconfigured objects in the Advantech Studio Library. To access the symbol Library, right-click the **Library** folder in the **Graphics** tab of the Workspace window and select **Open**, select the  **Open Library** button on the Standard toolbar, or select **View | Library**.

The two large arrows can be found on the *arrows* screen (selected in the right pane of the Library window), the two pipes are from the *pipes* screen, the tank is from the *tanks* screen,

and the arrow buttons are from the *buttons* screen, the bargraphs are from the *Bragraphs* screen and the indicator is from the *frames* screen. To move the objects from the Library to your Main screen, keep the Library window at its default size so that you can see the Main screen behind it, select the correct page, scroll to the desired object, and click-and-drag it from the Library to the screen. When you are done, close the Library window. Then rearrange the Library objects (you may need to resize the length of the pipes) on the screen so that their arrangement resembles that of the final screen:



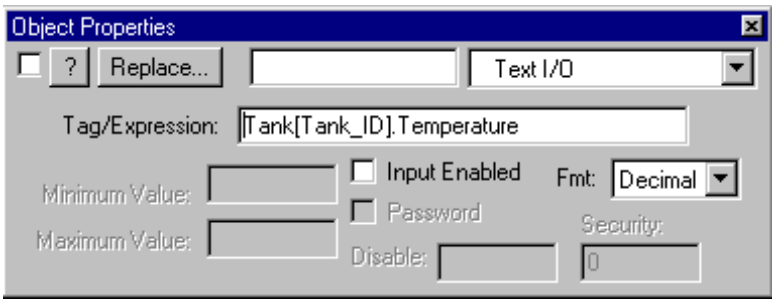

You can also draw similar objects using the tools available on the Tools toolbar, but these preconfigured objects simplify and speed the process of screen creation.

After importing the symbols from the library, we can configure the links on the screen. First, create the text objects as follows:

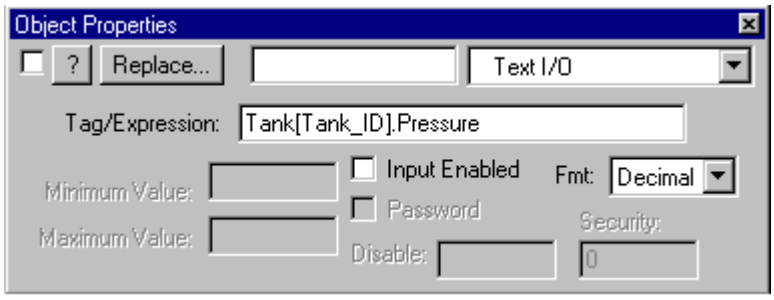

**Note:** Before beginning to configure the links, create another tag in the Application Database (TankID, Type Integer, Array Size 0, Web=Local) that will be the index for the tag array.

- For each text objects, associate one previously created tag in the **Tag Database.** To apply this association, let us use the "**Text input/output**" link as we did before.

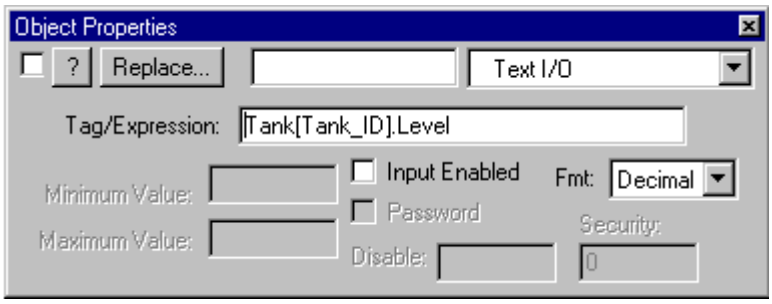
Select the text " ### F" and click on the "**Text input/output**" icon. Configure it as follows:




- Select the text " ### mca" and click on the "Text input/output" icon. Configure it as follows:



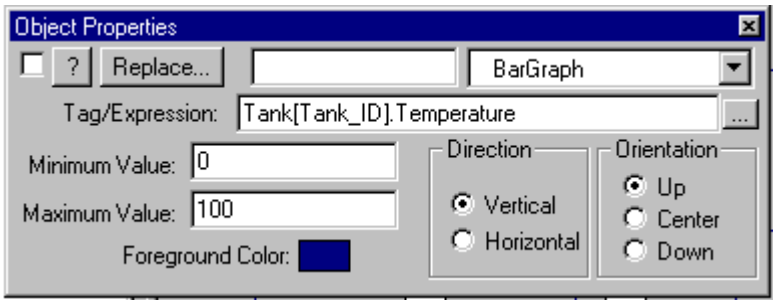

- Finally, select the text " ### m" and click on the "Text input/output" icon. Configure as follows:




- To show the values of temperature, pressure, and level in a graphical format, we use the **bargraph** link.

- First, we need to create three rectangles. For each one of them, associate one tag, which we previously created in the Tag Database. We will associate these tags using the "**Bargraph**".

- Select the rectangle above the text " **Temperature ### F**" and click on the "**Bargraph**" icon. Configure the rectangle as follows:



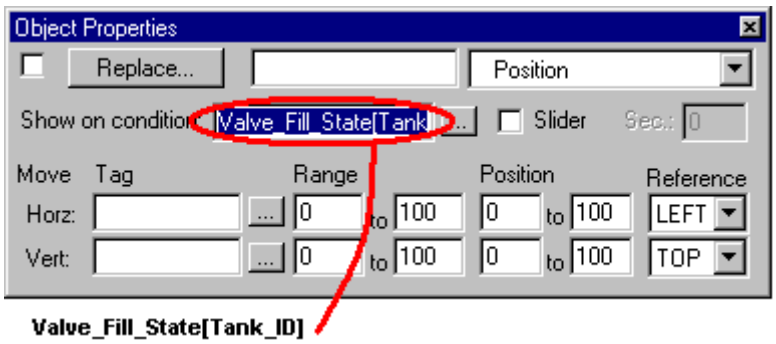

Select the rectangle above the text "**Pressure ### mca**" and click on the "**Bargraph**" icon. Configure it as follows:


Select the rectangle above the text "**Level ### m**" and click on the "**Bargraph**" icon. Configure it as follows:

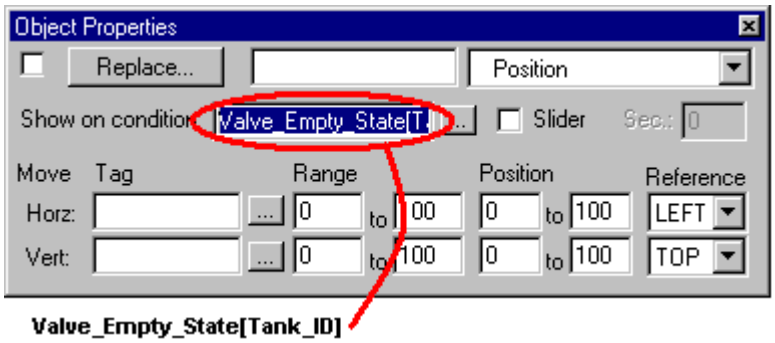

- For the valves "**Valve_Fill**" and "**Valve_Empty**", configure two links: "**Color**" (to show the valve's status) and " **Command**" (to enable that authorized user can give commands to the valves).

- Select "**Valve_Fill**" and select the “position” property. Configure it as follows:

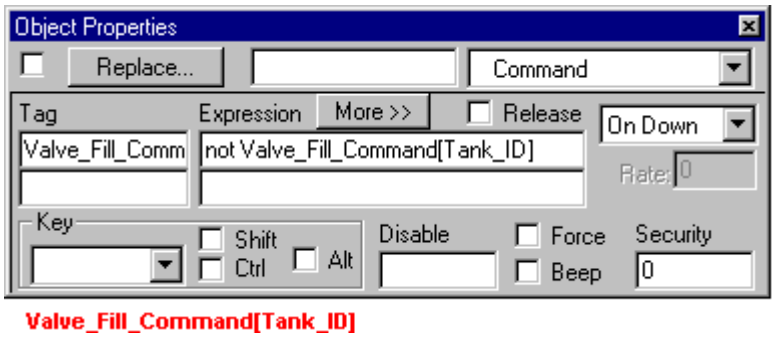
- Select "**Valve_Fill**" and select the “position” property. Configure it as follows:




- Select "**Valve_Fill**", select the "Position" property. Configure it as follows:




- Select "**Valve_Fill**" and click on the "**Command**" icon. Configure it as follows:

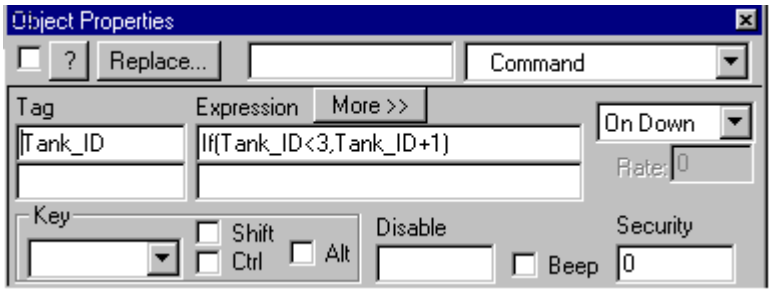



- Select "**Valve_Empty**" and click on the "Command" icon.

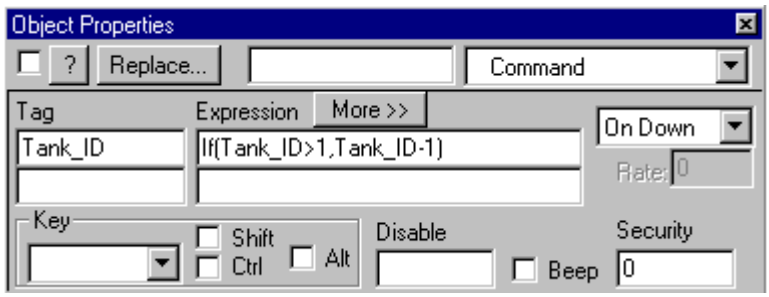

**Note:** The "Security" associated with these valves block the action (or command) from any users who do not have the right privileges.

- Finally, configure the buttons. In order to do this, we need to change the index of the Array Tags used on the screen (i.e.: switch the tank number).

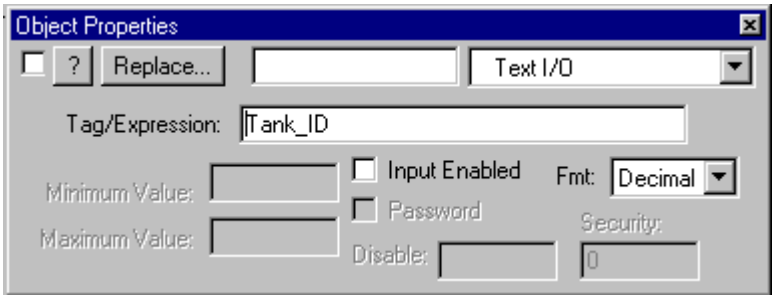
- Select the *increase* button  on the screen and click on the  "**Command**" icon. Configure it as follows:



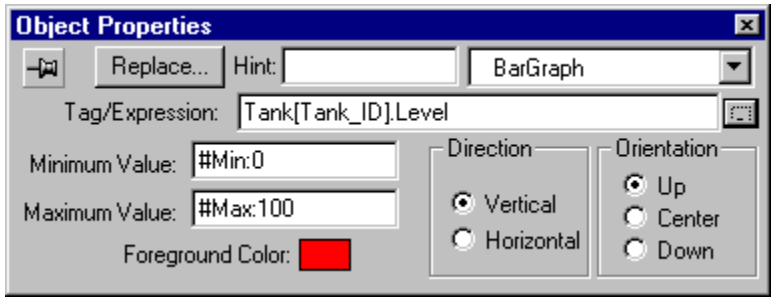

- Select the *decrease* button  on the screen and click on the  "**Command**" icon. Configure as follows:




- To see the number of the tanks that are being supervised, let us configure a text string ("**Tank #**") and associate the follow "**Text input/output**" function to it.




- For the main tank, configure one link: "**Bargraph**" (to show the level animation in the tank)
- Select the tank object and select its "**BarGraph**" dynamic. Configure it aws shown below:



We need to save the screen and execute the application to test the configuration. Execute the command ***Project - Run Application*** from the menu or click on the *Run Application*  icon from the toolbar.You can use the DBSpy and Output Window to debug your application.

Before we keep on creating the other screens, we will learn about Advantech Studio language translation and create a process simulation.

# 7. Expressions, Functions and Script Language

This chapter describes the script language syntax and functions. This script language is used in many places, such as:

- Dynamic object properties in Application Builder
- Screen logic in Application Builder
- Scheduler Worksheet
- Math Worksheet

The math expressions have two columns: **Tag** and **Expression**.

- ***Tag***: Tag name that receives the result of the expression in the Expression column.
- ***Expression***: Any mathematical expression defined by Advantech Studio.

***Example:***

| | Tag Name | Expression |
|---|---|---|
| 1 | a | 10 * c - 5 |

The ***a*** variable receives the result of expression ***(10c)-5***.

**IMPORTANT**: No attributions are done on the Expression column. If you write A=2 in this column, it means that you are comparing A with the number 2. The integer result of this expression (the Boolean value 0 if false or 1 if true) will be written to the tag in the Tag Name column.

**NOTE**: The system is not case sensitive.

**NOTE**: To add comments to an expression line, use the "//"characters.

### Data types

- **Integer numbers**: 1 23 45 -123
- **Floating point**: 1.234 -775.344
- **Hexadecimal integer numbers**: 0x5 0xA0 0xBC4
- **Strings**: "demo" "new demo"

Integer numbers are 32 bits. Floating numbers are 8 bytes and the strings are up to 255 characters.

## Examples to access the application database

To read a value in the database, use the tag name directly.

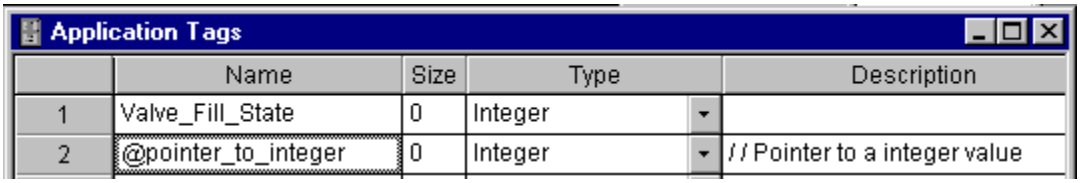
**Example 1:**

In the following script line, the X tag will receive the sum of two tags, level and temp:

| | Tag Name | Expression |
|---|---|---|
| 1 | X | Level + temp |

**Example 2:**

Advantech Studio allows you to read and write in the tags using references or pointers. A tag used as pointer to another tag can be declared in two ways: as a string (a pointer to an undefined type) or as a pointer of a specific kind (recommended).




In Figure above, the pointer_default is a variable of the string type that is a pointer. The variable @pointer_to_integer is a pointer to integer values.

**NOTE**: The syntax @tag allows a tag to access another by reference.

**NOTE**: Any tag declared as a string can be used as an indirect tag (pointer).

Operators

Advantech Studio supports all the following operators.

## Arithmetic operators

| | |
|---|---|
| + | addition |
| - | subtraction |
| * | multiplication |
| / | division |
| > | greater than |
| < | less than |
| = | equal |
| >= | greater than or equal to |
| <= | less than or equal to |
| <> | different than (unequal to) |

## Logic operators

| | |
|---|---|
| AND | AND, logic. |
| NOT | NOT, logic. |
| OR | OR, logic. |
| XOR | exclusive or, logic. |
| & | AND, bit. |

| | |
|---|---|
| \| | OR, bit. |
| ~ | NOT, bit. |
| ^ | XOR, bit. |
| >> | rotate right - Rotate n bits to right. |
| << | rotate left - Rotate n bits to left. |

## Functions list

Advantech Studio has more than one hundred functions ready for use:

### SEND MESSAGES TO THE LOGWIN

TRACE( strOutputMessage )

### ARITMETIC FUNCTIONS

ABS(numValue)

DIV(numDivisor, numDividend)

FORMAT(strFormatFlag, numValue)

GETBIT(strTagName, strBitNumber)

MOD(numDivisor, numDividend)

POW(numBase, numExponent)

RESETBIT(strTagName, strBitNumber)

ROUND(numValue)

SETBIT(strTagName, strBitNumber)

SQRT(numValue)

SWAP16(strTagName)

SWAP32(strTagName)

TRUNC(numValue)

### STATISTIC FUNCTIONS

AVG(numValue1, numValue2, &ldots; , numValueN)

MAX(numValue1, numValue2, &ldots; , numValueN)

MIN(numValue1, numValue2, &ldots; , numValueN)

RAND()

### LOGARITIMIC FUNCTIONS

EXP(numExponent)

LOG(numLogArg)

LOG10(numLogArg)

## LOGIC FUNCTIONS

IF(numCondition, numThen, numElse)

TRUE(numExpression)

FALSE(numExpression)

## FUNCTIONS FOR STRINGS MANIPULATION

ASC2STR(strChar1, strChar2, &ldots; , strCharN)

CHAR2ASC

CHARTOVALUE("strTagName", "numArray")

CHARTOVALUEW("strTagName", "numArray")

NCOPY(strSource, numStartChar, numQtdChar)

NUM(strValue)

STR(numValue)

STR2ASC(strChar)

STRLEFT(strSource , numQtdChars)

STRLEN(strSource)

STRLOWER(strSource)

STRRCHR(strSource, strCharSequence)

STRRIGTH(strSource, numQdeChars)

STRSTR(strSource, strSequence)

STRSTRPOS(strSource, strCharSequence)

STRTRIM(strReference, numOptionalFlag)

STRUPPER(strValue)

VALUETOCHAR("numArray", numQdeChar)

VALUEWTOCHAR("numArray", numQdeChar)

## DATE AND TIME MANIPULATION

CLOCKGETDATE(numSeconds)

CLOCKGETDAYOFWEEK(numSeconds)

CLOCKGETTIME(numSeconds )

DATETIME2CLOCK(strDate, strTime)

GETCLOCK()

HOUR2CLOCK(strTime)

SETSYSTEMDATE(strDate)

SETSYSTEMTIME(strTime)

GETTICKCOUNT ()

## TRIGONOMETRIC FUNCTIONS

ACOS(numValue)

ASIN(numValue)

ATAN(numValue)

COS(numAngle)

COT(numAngle)

PI()

SIN(numAngle)

TAN(numAngle)

## FUNCTIONS FOR OPENING AND CLOSING WINDOWS

OPEN(strScrFile, numOptionalX1, numOptionalY1, numOptionalX2, numOptionalY2)

CLOSE(strScrFile)

## SECURITY SYSTEM

CREATEUSER(strUserName, strGroup, strPassw)

REMOVEUSER(strUserName )

## MODULE ACTIVATION FUNCTIONS

SHUTDOWN()

APPACTIVATE(strAppTitle, numOptionalActiv)

APPISRUNNING(strAppTitle )

APPPOSTMESSAGE(strAppTitle, numwParam, numlParam)

APPSENDKEYS(strKeys1, strKeys2, &ldots; , strKeysN)

CLEANREADQUEUE()

CLOSESPLASHWINDOW()

DISABLEMATH()

ENABLEMATH()

EXITWINDOWS(numExitCode)

ISSCREENOPEN(strScrName)

ISVIEWERINFOCUS()

LOGOFF()

LOGON(strOptionalUser, numOptionalPassw)

MATH(numMathWorksheet)

NOINPUTTIME()

RECIPE(strOperation&File)

REPORT(strOperation&File)

SETAPPPATH(strDirPath)

SETVIEWERINFOCUS()

VIEWERPOSTMESSAGE(strScrTitle, numwParam, numlParam)

WAIT(numMilliseconds)

WINEXEC(StrFilePath, numOptionalState)

## FILE MANIPULATION FUNCTIONS

FILECOPY(strSourceFile , strTargetFile)

FILEDELETE(strFilePath)

FILELENGTH(strFileName)

FILERENAME(strOldName , strNewName)

FINDFILE(strFileMask)

PRINT(strFilePath)

RDFILEN(strSelectedFile, strSearchPath, strMask, numChangeDir)

## FUNCTIONS FOR GRAPHICS SCREENS PRINTING

PRINTWINDOW(strScrName)

## FUNCTIONS FOR TEXT TRANSLATIONS

EXT(strText)

SETTRANSLATIONFILE(strFileName)

## MULTIMEDIA FUNCTIONS

PLAY(strWavFile)

## SYSTEM INFORMATIONS

DBVERSION()

GETAPPHORIZONTALRESOLUTION()

GETAPPVERTICALRESOLUTION()

GETCOMPUTERNAME()

GETHARDKEYMODEL()

GETHARDKEYSN()

GETPRODUCTPATH()

GETOS()

GETPRIVATEPROFILESTRING(str_Section, str_Name, str_Default, str_FileName)

GETTICKCOUNT()

INFOAPPALRDIR()

INFOAPPDIR()

INFOAPPHSTDIR()

INFODISKFREE(strDiskUnit)

INFORESOURCES(numResourceCode)

NOINPUTTIME()

PRODUCTVERSION()

SETAPPALARMPATH (strPath)

SETAPPHSTPATH(strPath)

SETDATEFORMAT(strSeparator, strDateFormat)

## DATABASE ACCESS FUNCTIONS

CHANGED(strTagName)

FORCETAGCHANGE(strTagName, numValue)

## LOOPS

FOR(numInitialValue, numFinalValue, numStep)

NEXT

## INTERNAL TAGS

GOTO

LABEL

## MAIL

CNFEMAIL( strSMTP , strFrom )

SENDEMAIL( strSubject, strMessage, strTO )

## SPECIAL FUNCTIONS

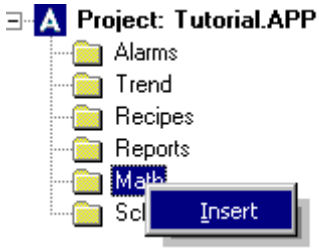
**IMPORTANT**: You can use the Database Spy module to execute any math *expression*; write the *expression* in the Tag Name field, and click the mouse over the Toggle button. The return *value* of the *expression* will display in the *Value* field.

# 8. Configuring Worksheets

## Configuring a Math Worksheet (simulate the field process)

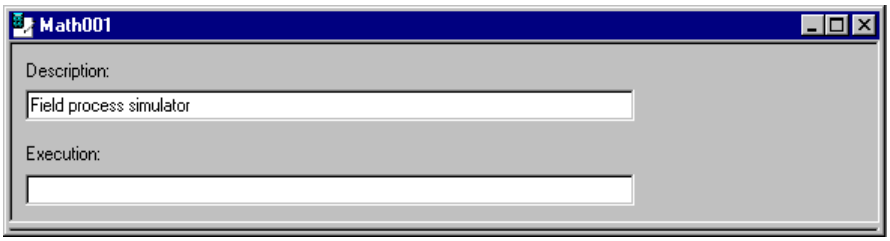
Before continuing our application development, we need to create a script to simulate some variables. In a real world application, these variables would be coming from field equipment such as a PLC or a Soft Control. This simulation will be done in a math worksheet, which needs to be constantly executed

- Right click on the "**Math**" folder located in the **Tasks** tab. Select the option "**Insert**" to create a new math worksheet.




The ***Execution*** field is where it is possible controlling the Math Execution. You can type here a full expression, a simple condition, a tag name, a value, and, when this condition is TRUE, the math is executed.

So, in the header, configure the field Execution with the value 1. This enables the continuous execution of this math worksheet just because 1 is always a TRUE condition.




In the math worksheet's body, configure the functions to simulate:

The valves status, according with the command given.

The level, temperature and pressure of each one of these three tanks.

- To simulate the status of each valve, just transfer the value from the command tags to the status tags.

- To simulate the properties for temperature and pressure for each tank, let us consider these properties using trigonometric functions (sine and cosine).

- To simulate the properties for level of each tank, remember that both valves (fill and empty) allow for the same flow.

- So, with the above understanding, let us fill the body of the math worksheet in as follows:

Description:
Field process simulator

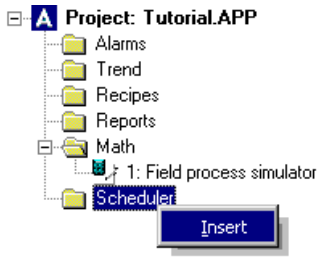
Execution:
1

| | Tag Name | Expression |
|---|---|---|
| 1 | Valve_Empty_State[1] | Valve_Empty_Command[1] |
| 2 | Valve_Empty_State[2] | Valve_Empty_Command[2] |
| 3 | Valve_Empty_State[3] | Valve_Empty_Command[3] |
| 4 | Valve_Fill_State[1] | Valve_Fill_Command[1] |
| 5 | Valve_Fill_State[2] | Valve_Fill_Command[2] |
| 6 | Valve_Fill_State[3] | Valve_Fill_Command[3] |
| 7 | Tank[1].Temperature | (Sin((Second/30)*PI())+1)*50 |
| 8 | Tank[2].Temperature | (Sin((Second/20)*PI())+1)*50 |
| 9 | Tank[3].Temperature | (Sin((Second/10)*PI())+1)*50 |
| 10 | Tank[1].Pressure | (Cos((Second/30)*PI())+1)*50 |
| 11 | Tank[2].Pressure | (Cos((Second/20)*PI())+1)*50 |
| 12 | Tank[3].Pressure | (Cos((Second/10)*PI())+1)*50 |
| 13 | J | For(1,3,1) |
| 14 | Tank[J].Level | If(Valve_Empty_State[J]>Valve_Fill_State[J] and Tank[J].Level>0,Tank[J].Level-1) |
| 15 | Tank[J].Level | If(Valve_Empty_State[J]<Valve_Fill_State[J] and Tank[J].Level<100,Tank[J].Level+1) |
| 16 | Next | |

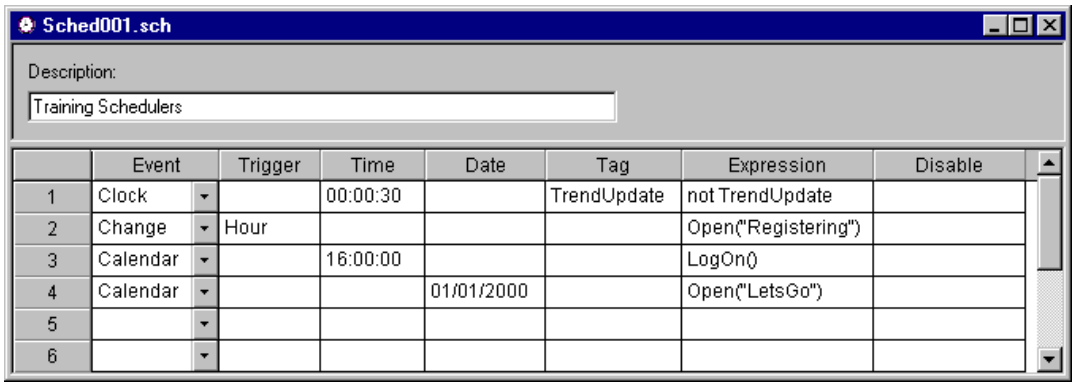
☞ Note: In our example, worksheet 001 is executed continuously. In a real world application, it is strongly recommended that the execution of each math worksheet be carefully controlled to improve the system's performance.

## Configuring a Scheduler worksheet using the events Clock, Calendar and Change

To create a new scheduler worksheet, right click on the **Scheduler** folder in the **Task** tab from the **Workspace** window. Select the option **Insert** to create a new scheduler worksheet.




Configure the scheduler worksheet as follows:




| | Event | Trigger | Time | Date | Tag | Expression | Disable |
|---|---|---|---|---|---|---|---|
| 1 | Clock | | 00:00:30 | | TrendUpdate | not TrendUpdate | |
| 2 | Change | Hour | | | | Open("Registering") | |
| 3 | Calendar | | 16:00:00 | | | LogOn() | |
| 4 | Calendar | | | 01/01/2000 | | Open("LetsGo") | |
| 5 | | | | | | | |
| 6 | | | | | | | |

**Notes**:

The **Clock** event is used to trigger actions based on regular time intervals such as timers and counters. In the **Time** column, we can configure the base time (minimum of 100ms). In the **Tag** column, we must configure the tag that will receive the result from the expression configured in the **Expression** column. Finally, the **Disable** field can be used to prevent an expression in the line from being executed. The results of the expression in the **Disable** field will always be TRUE.

The **Calendar** event is used to trigger actions on a scheduled time. Also, it is possible to specify a fixed date for the event in the **Date** column. The **Tag**, **Expression** and **Disable** columns are used the same in all three-scheduled event functions.

The **Change** event is used to trigger an action upon a change in a tag value. In the **Trigger** column, we must configure a tag that will be used to trigger the event when a change in value has occurred. The **Tag**, **Expression** and **Disable** columns are used of the same in all three-scheduled event functions.

# 9. Recipes and Reports

## Creating Recipes

The Advantech Studio Recipes module allows the user creating, loading and deleting recipes. Recipes, in this case, mean a group of tags that are going to have their values saved and retrieved like a database.

For preparing Recipes you need to create a Recipe worksheet. This worksheet will tell the system what tags you want to store in the disk for later retrieving, and where you are going to do that. When you save a recipe, it is created an ASCII file with the Tag values and the Recipe file name. For retrieving these tag values, the system will find them in this ASCII file in standard format or in XML format.

### Creating the Recipe Worksheet

First, create the CLASS:**CCake** with the INTEGER type members **Suger**, **Fruit**, **Milk, Flour** and **Yeast (Baked Powder):**

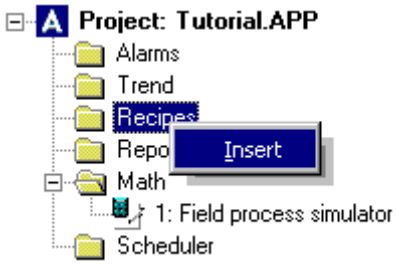
| | Name | Type |
|---|---|---|
| 1 | Sugar | Integer |
| 2 | Fruit | Integer |
| 3 | Milk | Integer |
| 4 | Flour | Integer |
| 5 | yeast | Integer |
| 6 | | |

Next, create the TAG **Cake** of the type **class:CCake**.

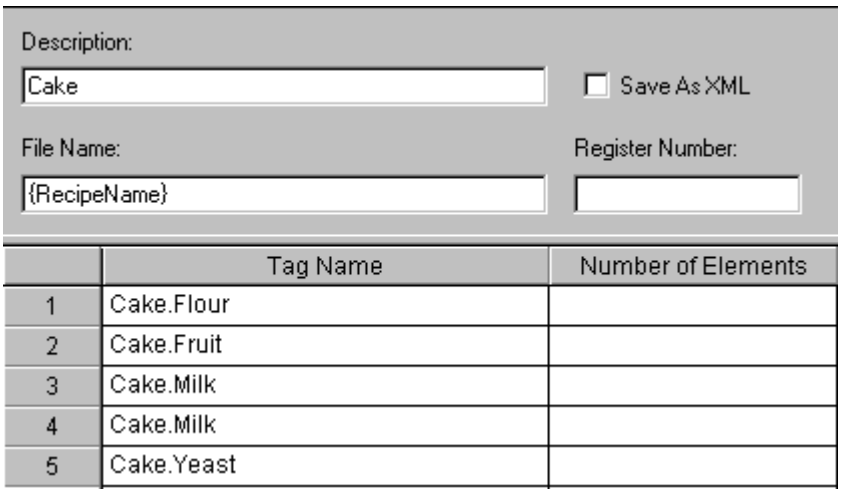
Note: Remember that the syntax to access the value from a class tag is: <tag_name>**.**<member_name>. (e.g. **Cake.Sugar, Cake.Fruit, etc…**).

So, create the type STRING tag **RecipeName** (it is not a CLASS type) that will be used to store the input file name used in the recipe task.

Now, select the **Tasks** tab in the **Workspace** window and create a new recipe sheet.




Fill in the fields as shown below and save it with the default name **Recipe1.rcp**.



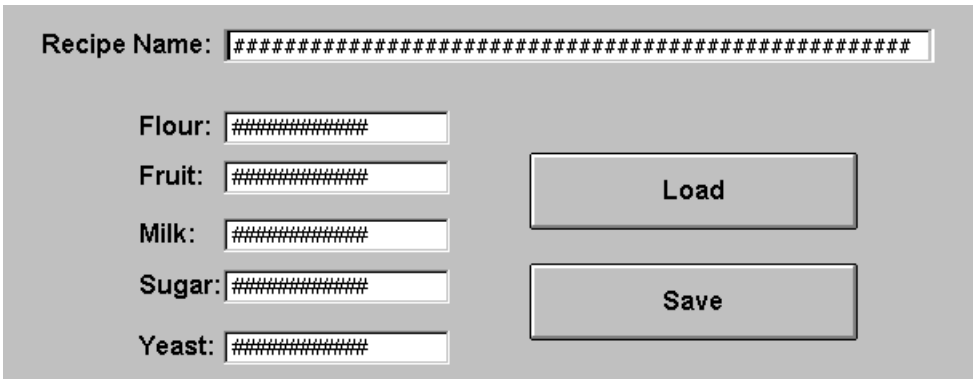

The "File Name" field is where you are going to save your recipe tags values. If you type a tag name between curly brackets (like the example), the file will use the tag value to compose the File Name. For example, you can have a "File Name" like: *c:\AppName\Recipe\{RecipeName}.* In this case, the value inside of the tag *RecipeName* will give file the name, at the *c:\AppName\Recipe* directory.

**Register Number Field -** Tag that defines the register number to be read or written in a DBF file. Not used anymore.

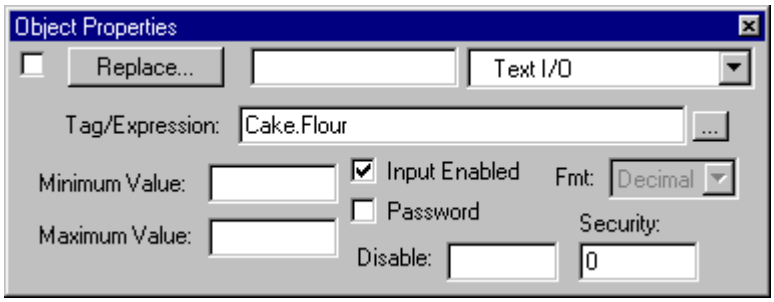
**Number Column:** Sets how many positions of the array tag are in use. So, if you want to have an array tag size 120 in a Recipe, you do not need to type all the 120 positions, I mean, Tag[0], Tag[1], Tag[2],… All you need to do is to type the *Tag name* and in the *number* column just type how many positions.

## Creating the Recipe Screen

Open the Standard Screen, draw the objects below and save it as "Recipe.scr"

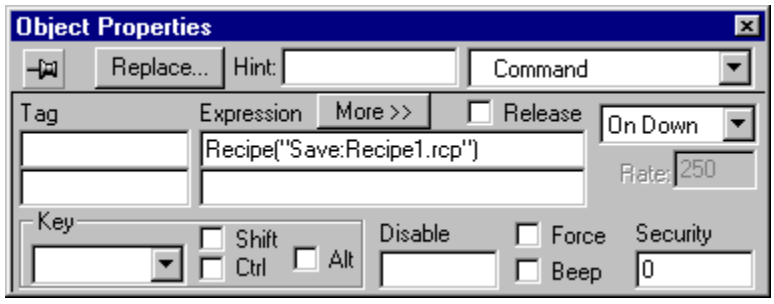



Now, associate it to the **Text I/O** function to the ############ texts and fill each one of their **Tag/Expression** field with the tag names and members of **Cake** tag. Leave the "Input Enabled" check box checked. For example: to the ############ text in front of the "Flour" text, type *Cake.Flour* in the "Tag/Expression" field of the Tex I/O property.

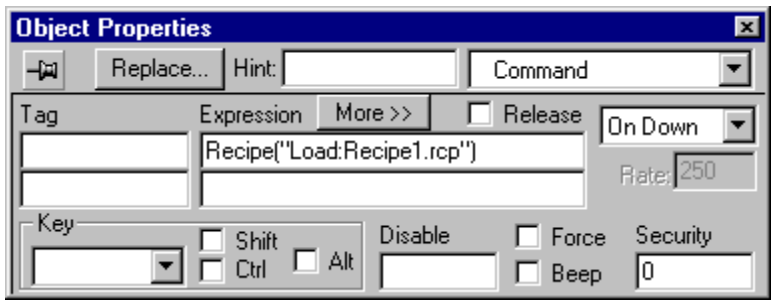



For the Recipe Name field associate the string tag **RecipeName**.

Insert the "Command" property in the "Save" and "Load" buttons. In the "Save" button, type the following command:




In the "Load" button:




## Recipe laboratory

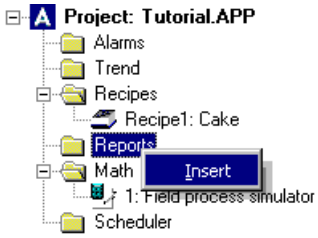
Type a name to the Recipe and some values to its ingredients. Save it. Then type another name with other values and save it as well. Now type the name of the first recipe and "Load" it. Play a little bit with that.

# Creating Reports

The Advantech Studio Report tool allows you creating reports very easily, without needing any other programming tool, like VB, etc… All you need to do is to prepare your report mask in a ASCII format or using our Report Writer tool (that creates RTF files) putting the Tags that you want to have the values inside of curly brackets. Let us go to the tutorial:

## Creating ASCII Reports

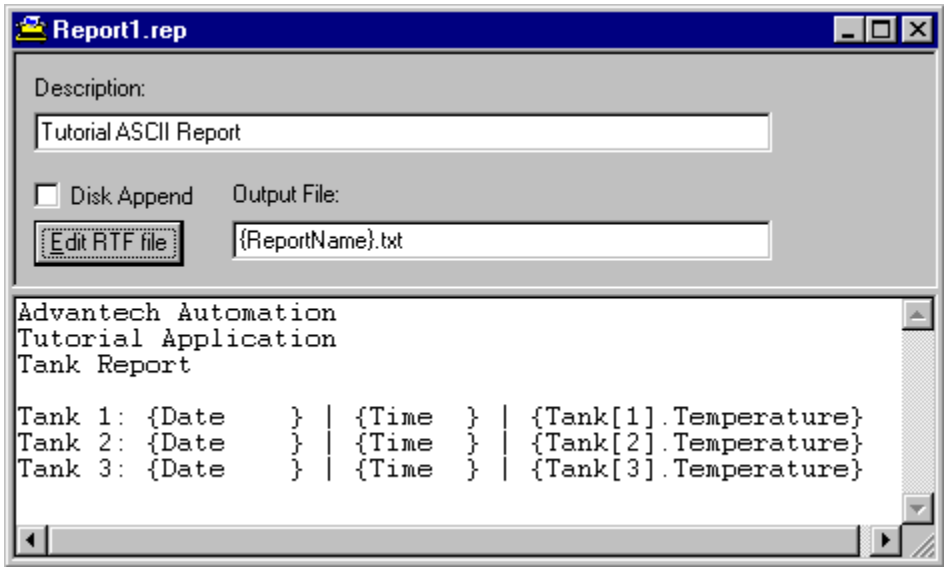
First, let us create a report worksheet that will be used to create the body of our report file. Right click on the **Reports** folder in the **Task** tab in the Workspace window. Select the option **Insert** to create a new report worksheet.




Configure the report worksheet as follows:

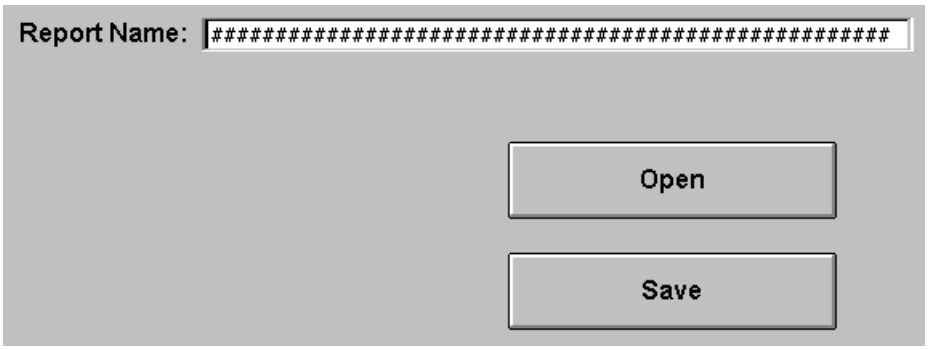
Just like in the Recipe case, the "Output File" is where the report is going to be created including its name. A Tag between curly brackets can be used here for given report a name.

The "Disk Append" check box will be explained in the next example.

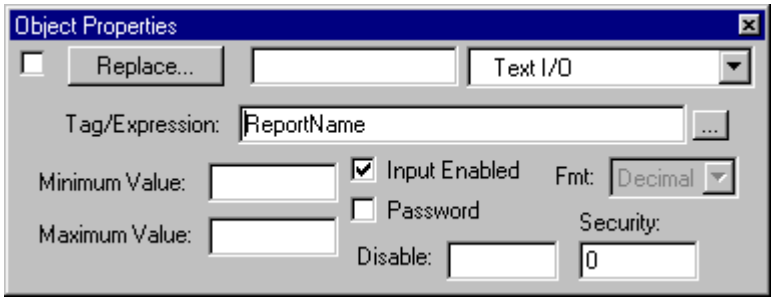



Save the report worksheet with the default name **Report1.rep**.

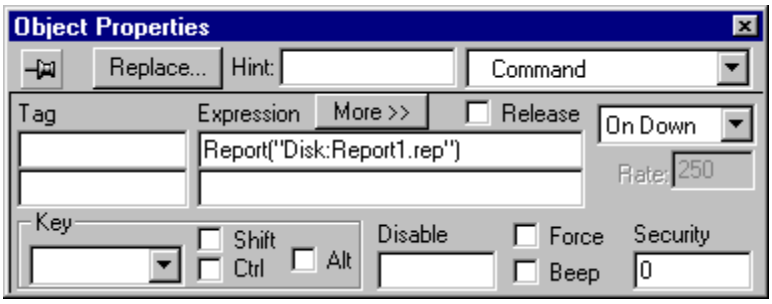
Create the “Report” Screen as follows.



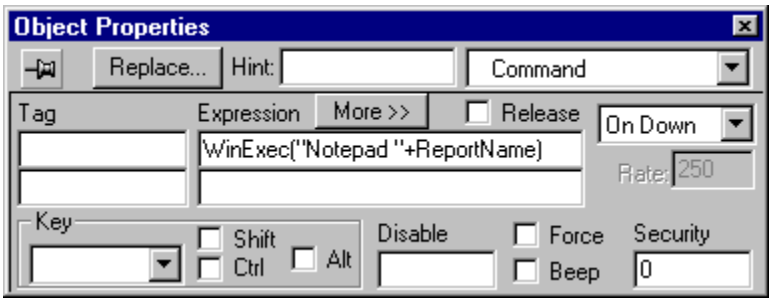

To the ##################### text associate the "Text I/O" property and configure it like shown below:




To the "Save" button, add the following command:




To the "Open" button, add the following command:




This command will use the Windows NotePad program to see the ASCII Report just created.

The next step is to give the report a name and play with the commands Save and Open.

# 10. Language translation

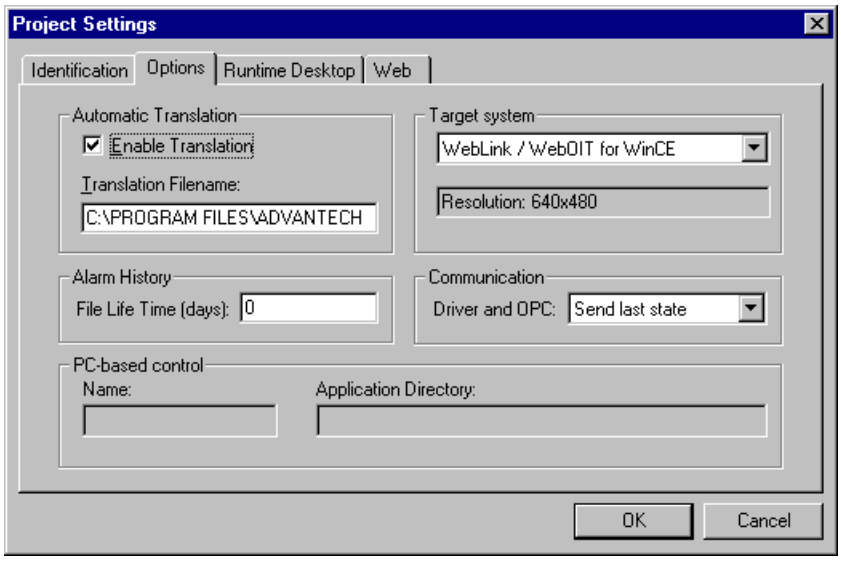
The translation tool enables you to change the texts from the buttons, or any other text field. All you need to do is to create the translation worksheets and use the translation functions.

When you use a translation function it looks for all the texts typed at the "Original" column and changes it to the text typed at the "Translation" column. If our application has other different texts and they are not typed in the Translation worksheet, these texts maintain the original one.

So, after translating and wanting to have the original text back, we must create an empty worksheet and use the translation function to this worksheet. Now, let us go to the tutorial:

## Enabling external translation

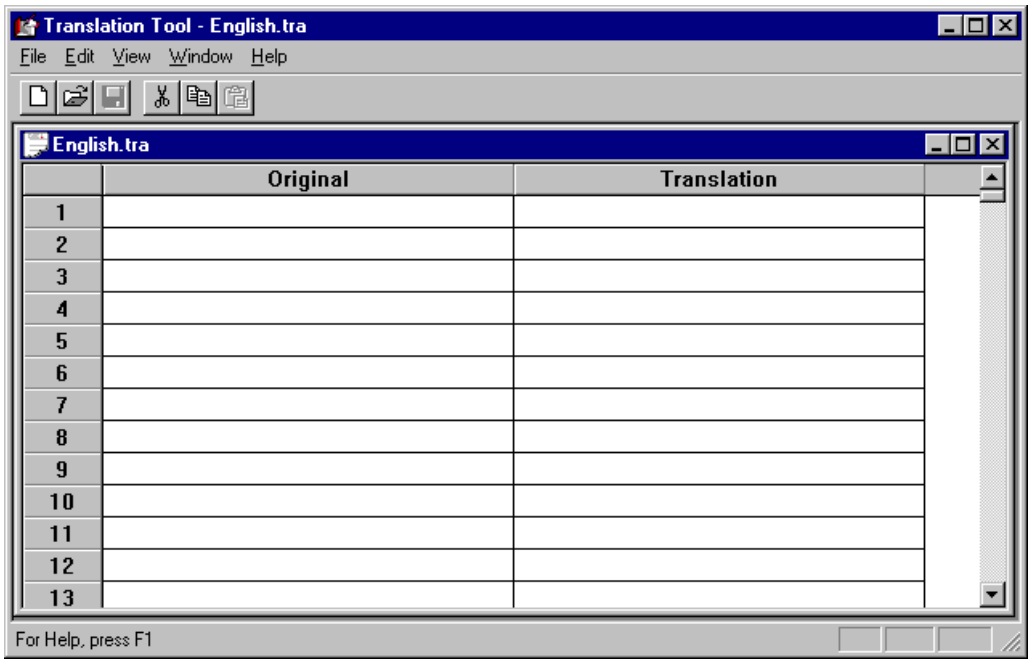
To allow the user having success in the translation operation the first step is to open the "Project Settings" window, in the "Options" tab. To do that, go to the menu "Project", then "Settings" and choose the tab "Options". You will find the "Enable Translation" check box. Check it.




## Creating the Translation worksheets

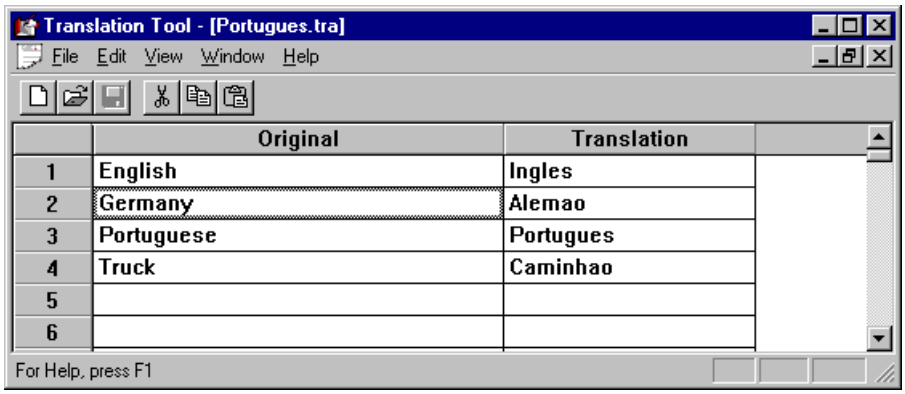
We need to create a translation sheet for each language that we want to configure in our application including our original language (e.g. English).

Choose the option **Tools** + **Translation Editor** in the main menu and save a blank sheet with the name **English.tra**.



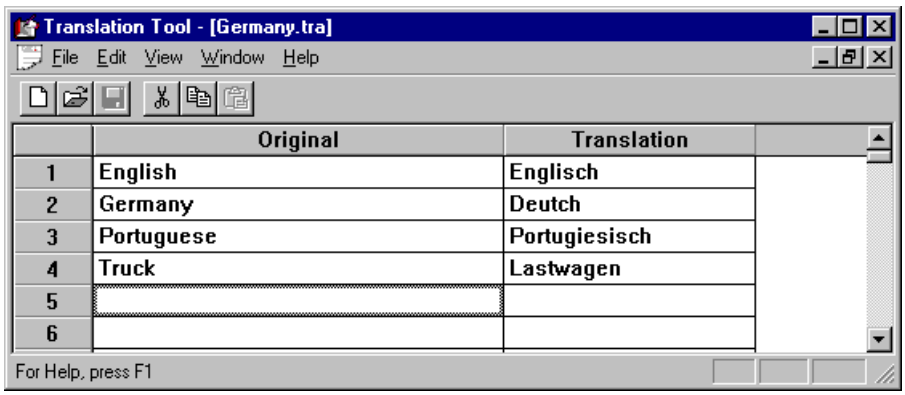

Note: It is necessary to create a blank sheet for the default language. In this tutorial our default language will be the English language as we explained before.

Create a new translation sheet and fill it in as follows:




Save it with the name **Portuguese.tra**;.

Create a new translation sheet and fill it in as follows:

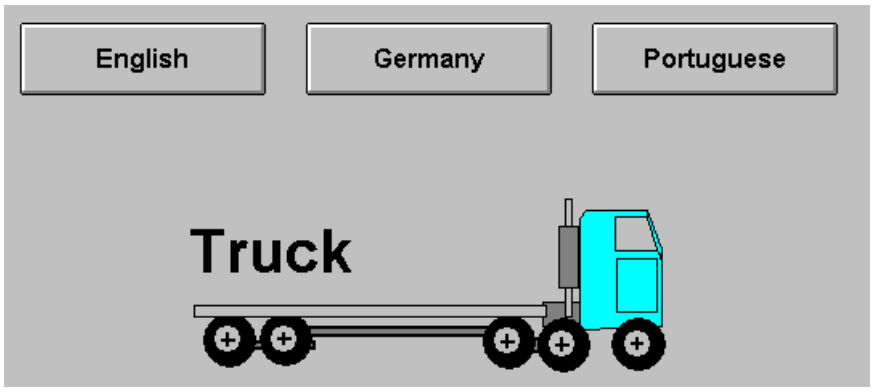



Save it with the name **Germany.tra**.

Note: It is possible to build an application with as many languages as necessary. To do that, we simple need to create a translation worksheet for each language used.

## Creating the Translation screen.

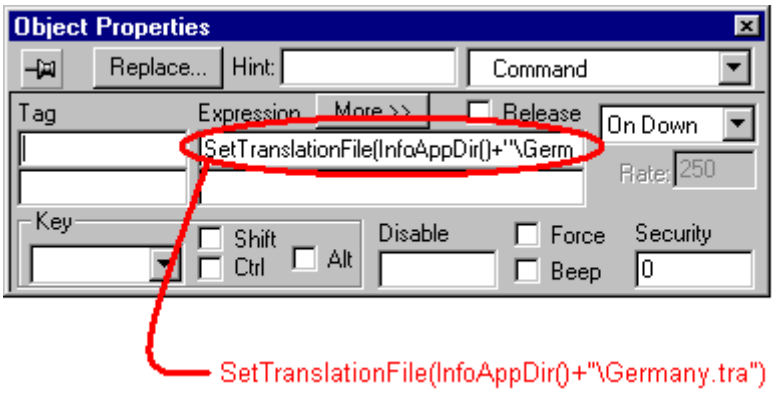
Create a new screen like this one shown below



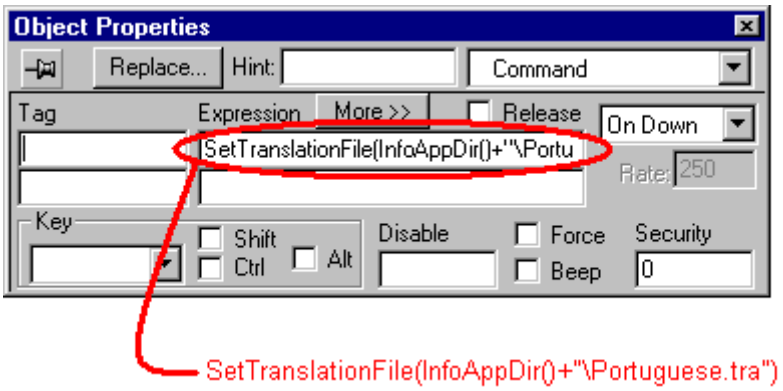

In the **English** button previously created, associate it to the **Command** function. Fill in the fields for the **Object Properties** window as follows:


Note: The function **SetTranslationFile** defines the translation worksheet that will be used in the application.

In the **Portuguese** button previously created, associate it to the **Command** function. Fill in the fields for the **Object Properties** window as follows:




In the **Germany** button previously created, associate it to the **Command** function. Fill in the fields for the **Object Properties** window as follows:

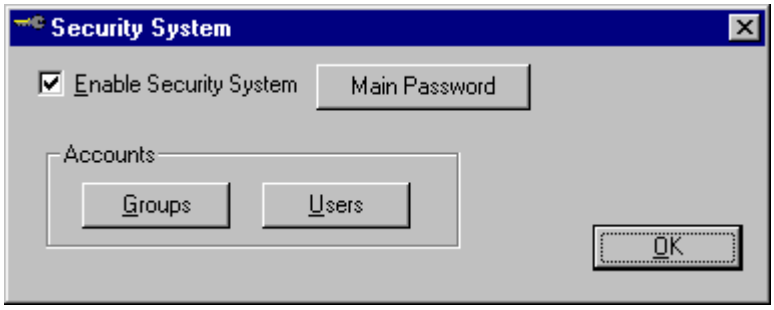



Finally, save the screen and execute the application to test the configurations.

# 11. Configuring the Security System

## Security System Window

The **Security** folder allows you to define groups and users as well as their access privileges to Advantech Studio tools and to the application. Through the **Database** tab, you can select or create new groups and users. To access the **Security System** window, right-click on the **Security** folder and choose "Settings".




**Enable Security System Check-box** - Enables the Advantech Studio Security System.

**Main Password Button** - Opens the **Security System Main Password** window (see below).

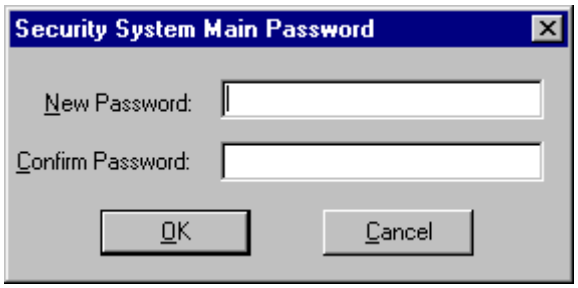
**Accounts Group Box**

**Groups Button -** Opens a **Groups** window (see below).

**Users Button** - Opens a **Users** window (see below).

### Password

The **Main Password** button of the **Security System** window opens the **Password** window where you define a password for accessing the Advantech Studio Security System.



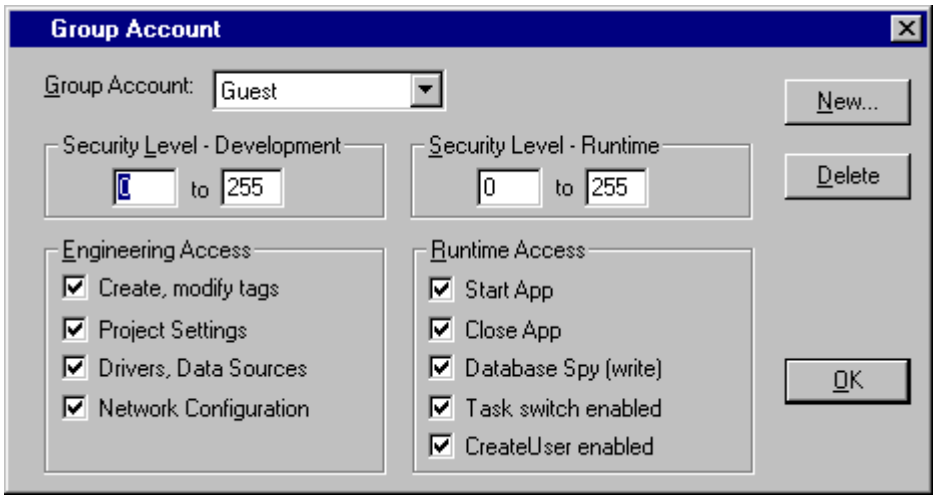

**New Password Field –** Type a new password here to define it.

**Confirm Password Field -** Confirm the password you typed in the **New Password** Field by typing it again and clicking on **OK**. If the password is different, the system asks you to type it again.

**IMPORTANT:** After you define your password, you will need to use it each time you access the Security System, so it is mandatory that you remember it.

## Groups

The **Groups Account Button** of the **Security System** window opens the Group Account window in which you can create and maintain ***user groups***. In this window, you enable/disable operations and set the range level. Groups can also be accessed by opening the **Groups** folder within the **Security** folder or by selecting the **Security Group** option under **Insert** on the Main Menu Bar. Select a specific group to view.




**Group Account Drop-list** - Select the group to which the user belongs from the drop-list.

**Security Level Development Group Box** - Defines the security level of each group (0 to 255). Any object for data input in the Display Screen (such as input commands, sliders, or screens) has a **Security Level** field. If the object level is not in the group security scale logged in at the moment, then the object is disabled. A level **0** (zero) means that the object is always enabled.

**Security Level Runtime Group Box** - Defines the security level of each group (0 to 255). Any object for data input in the Display Screen (such as input commands, sliders, or screens) has a **Security Level** field. If the object level is not in the group security scale logged in at the moment, then the object is disabled. A level **0** (zero) means that the object is always enabled.

**Engineering Access Group Box** - Lists Engineering (development) tasks that can be accessed when a user in this group is logged on. Includes check-boxes for **Create, modify tags; Project Settings; Drivers, Data Sources; Network Configuration**.

**IMPORTANT:** The security level can also be set to each document (worksheets and displays) to protect them in the development environment. This refers to the Engineering Access box.

**Runtime Access Group Box** - Runtime modules that this user group can access. Includes check-boxes for **Start App, Close App, Database Spy (white), Task switch enabled, CreateUser enabled**.

**NOTE:** You cannot delete the Guest group (the default logged group).

**New Button** - Opens the **New Group Account** window, in which you can create a new group.

**Delete Button** - Deletes the currently selected user group.

# SECURITY ACCESS LEVEL

In the **Group Account** window, it is possible to set a range of access values in the **Security Level- Development** group box. Each group can be assigned its own range of values. When any Advantech Studio worksheet is opened (Alarm, Math, Recipe, Report, Scheduler, TCP Client, Trend, and those not available on CE: DDE Client, OPC Client, and ODBC), it is possible to set an access range to THAT worksheet. Click on any part of the worksheet body to activate the **Access Level** option under **Edit** on the Main Menu Bar. When **Access Level** is selected, a window opens in which an Access Level number can be assigned. This means that to edit the worksheet again, it would be necessary that the worksheet have an **Access Level** within the **Security Level – Development** group box range of the user logged onto the system.

For example, UserA of GroupA has a **Security Access Level** range of 0-10, UserB of GroupB has a **Security Access Leve**l range of 5-15. To continue the example:

Math Worksheet 001 has Access Level = 1

Math Worksheet 002 has Access Level = 7

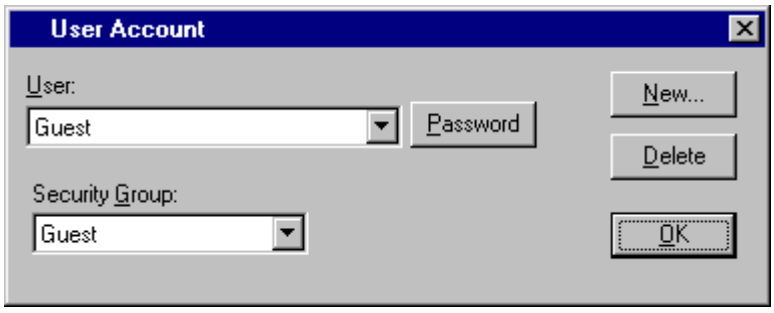
Math Worksheet 003 has Access Level = 12

Math Worksheet 004 has Access Level = 20.

In this situation, only UserA can access Math Worksheet 001, both UserA and UserB can access Math Worksheet 002, only UserB can access Math Worksheet 003, and neither UserA nor UserB can access Math Worksheet 004.

## Users

The **User Account Button** of the **Security System** window opens the **User Account** window in which you create and maintain accounts for application users. Define the application users that will be in each group in the **Group Account** list. Users can also be accessed by opening the **Users** folder within the **Security** folder or by selecting the **User** option under **Insert** on the Main Menu Bar. Select a specific user to view.



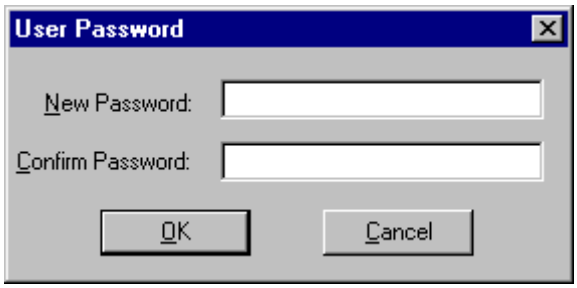

**User Drop-list -** Lists application users in a drop-list.

**Security Group Drop-list -** Lists application groups.

**New Button -** Opens the **New User Account** window to create a new user.

**Delete Button -** Deletes the selected user.

**Password Button -** Opens the **User Password** window, in which you can define a password for the user.




**New Password –** Enter a password to define it.

**Confirm Password -** Confirm the password you typed in the **New Password** field by typing it again and clicking on **OK**. If the password is different, the system asks you to retype it.

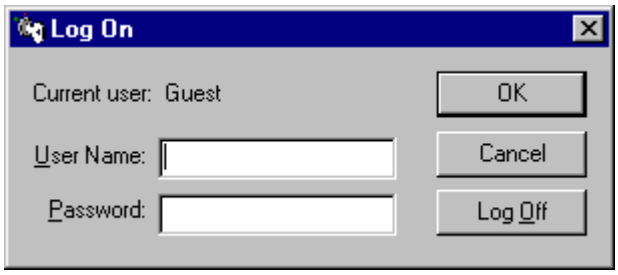
## GUEST USER

After you initialize Advantech Studio, a default user is logged on the **Guest** user. If no user is logged on or the current user has logged off, **Guest** user is automatically logged on.

The **Guest** group has default privileges. Since the installation parameters of the **Guest** group leave all tasks enabled, you should change it and set as few privileges as you want for a start up procedure.

## Log On/Log Off

This utility is used to log users on and off. The user names and passwords are defined through the **Security** folder on the **Database** tab. You can also log on or off by using the Advantech Studio Scripting Language module activation functions **LOGON( )** and **LOGOFF( )** or by selecting **Logon** under **Project** on the Main Menu Bar.




**User Name -** Name of the user to be logged in.

**Password -** User password.

**Log Off -** Logs off the current user.

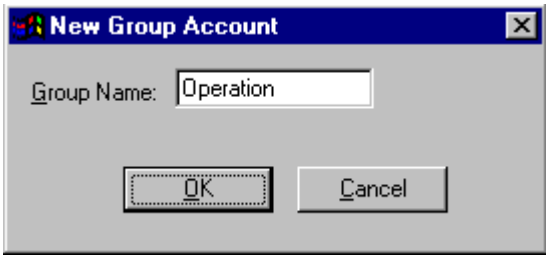
**NOTE**: When a **Logoff** is executed, the **Guest** user is automatically logged on.

So now let us start the tutorial

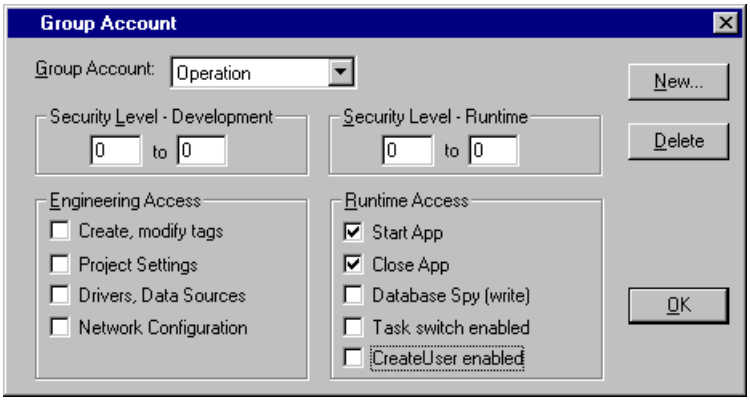
Before beginning the security system, it is important to have the groups and users that we want to configure in mind. We need to define the rights that each group has in our environment. In our example, we will create three groups: **Operation**, **Maintenance** and **Development**.

- In the Workspace, select the "**Database**" tab. Click on the "**Security**" folder and select the subfolder "**Group**." Use the right click button to open the pop-up menu and select the "**Insert group**" option.

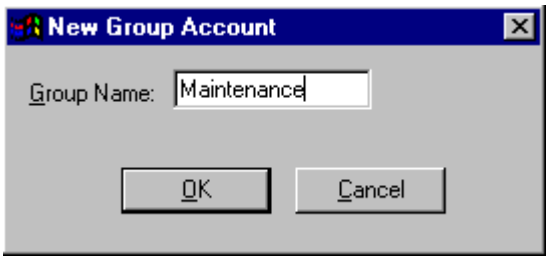

- There is a default group called Guest that can be deleted. Let us create a new group by clicking on the button ***New*** and typing in the new group name (For example: **Operation**).



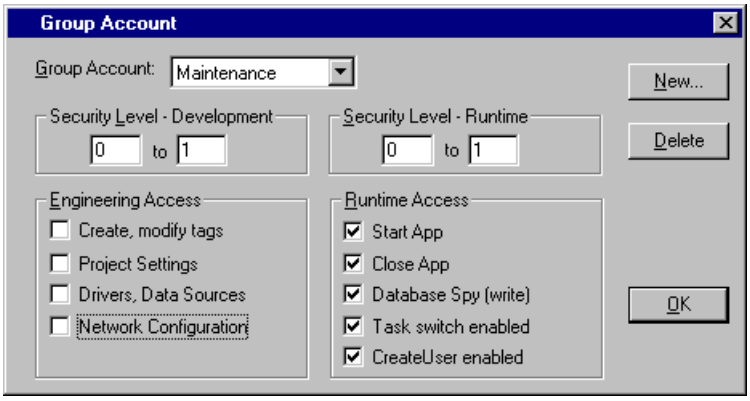

- Press ***OK***, to open the Group Account menu. We can configure the access rights for that group account as follows:



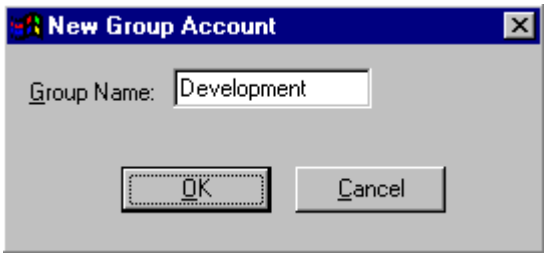

- Press the ***New*** button again and create a new group account (For example: **Maintenance**).



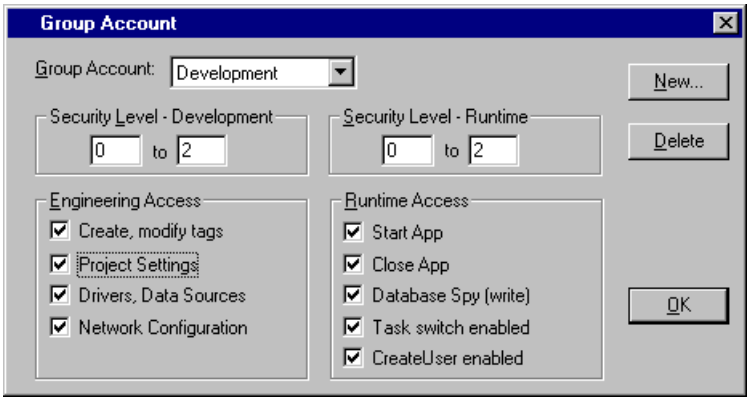

- Press ***OK*** to once again open the **Group Account** window. Configure the access rights for this group as follows:




- Finally, press the *New* button again to create the last group account (For example: **Development**).




- Press ***OK*** to open the **Group Account** window and configure the access rights for that group account as follows:

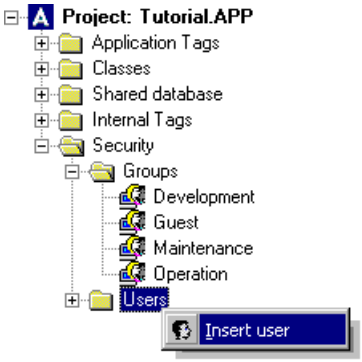



- Press the ***OK*** button to save the configuration.

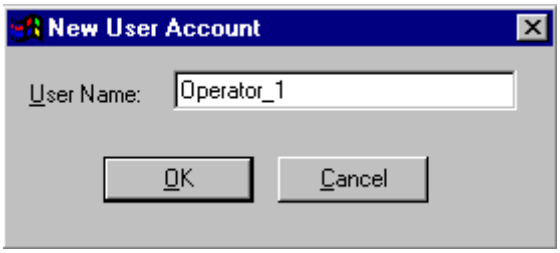
☞ Note: Each group has a range for the level in Development and Runtime. In some sheets (for example, the math sheet) we can set an access level to provide them with access to configure that sheet. The user logged in to the system must be associated to a group in that range (development) that has access to configure the math worksheet. Buttons can also be configured with access levels so that the commands (scripts) configured in these buttons will be executed only for users that belong to a particular group for the runtime environment.

Now we must create new users associated to the group accounts just created. In the Workspace, select the "**Database**" tab and expand the "**Security**" folder. Select the subfolder

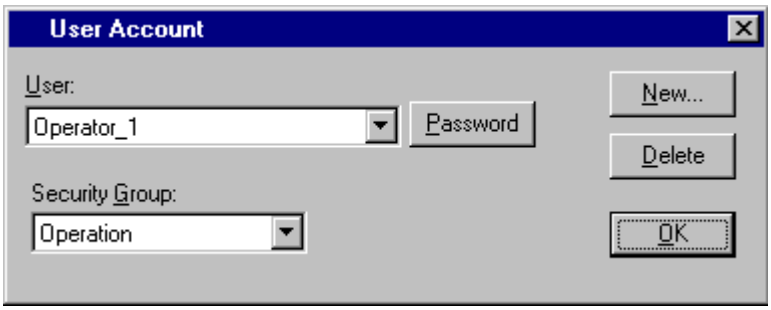
"**Users**" and click right so that the pop-up menu with the option for "**Insert user**" appears. Select **Insert user**.



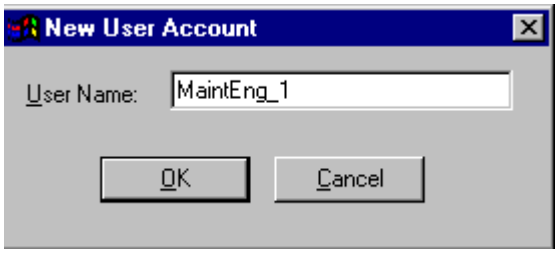

There is a default user account called Guest that cannot be deleted. To create a new user, click on the ***New*** button and type in the new user name (For example: **Operator_1**).



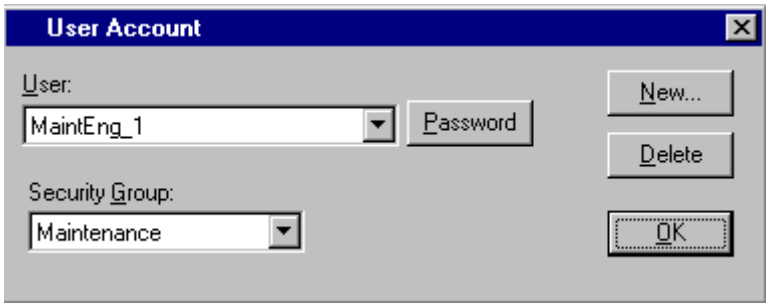

- Press ***OK***, now we can associate a group accounts to the new user. For example, selecting the group "**Operation**" in combination with the "**Security Group**". In addition, we can also configure the password by clicking on the button ***Password.*** For example, the password for **Operator_1** is **oper_1**.



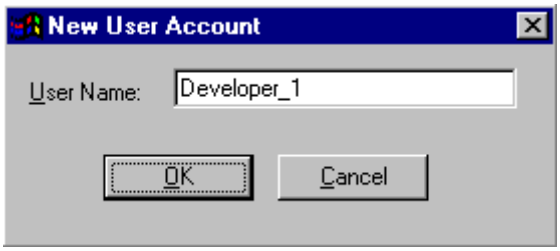

- Press the ***New*** button again to create another user. For example: **MaintEng_1**.



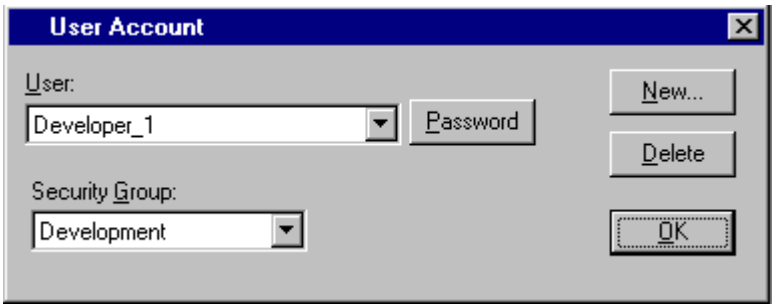
- Press ***OK***. Again, we can associate a group account to the new user. (For example, selecting the group "**Maintenance**" in combination with "**Security Group**"). In addition, we can also configure the password again by clicking on the ***Password.*** button. For example, the password is **main_1**.



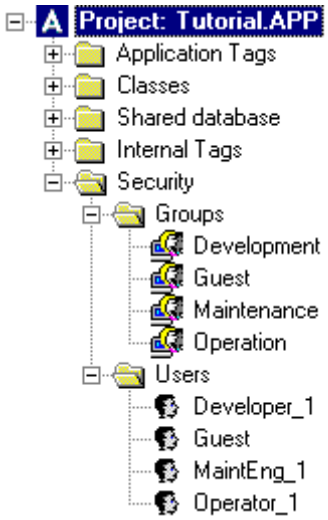

- Press the ***New*** button again to create the last user. For example: **Developer_1**.




- Finally, press ***OK***. Again, we can associate a group account to the new user. (For example, selecting the "**Development**" group in combination with the "**Security"** group). In addition, we need to configure the password by clicking on the *Password* button. For example, the password is **deve_1**.




- Press the ***OK*** button to save the configuration. Now we can expand the subfolders of the folder "**Security**" in order to visualize all the groups and users that we have created.

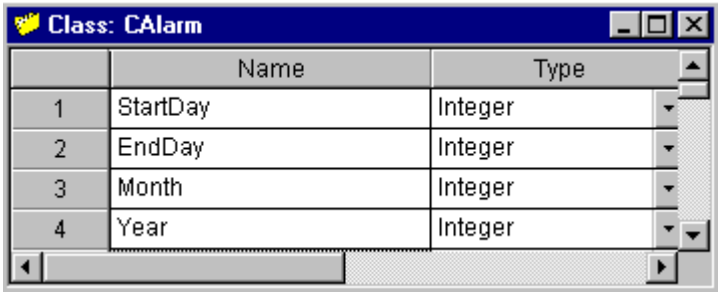



☞ Note: In the Tutorial, we created only one user for each group. However, it is possible to create a lot of users for each group. It is also possible to create a new user for an application by using the function "**CreateUser**".

# 12. Creating Alarms Group

## Creating an Alarm Group

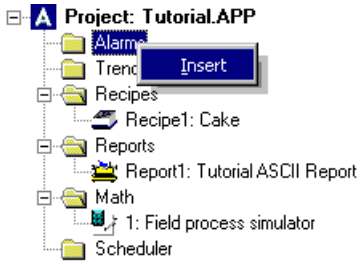
Before creating screens with alarm objects, we should create alarm groups. Create the Alarm_Settings tag with the type "Class: CAlarm " as shown below:



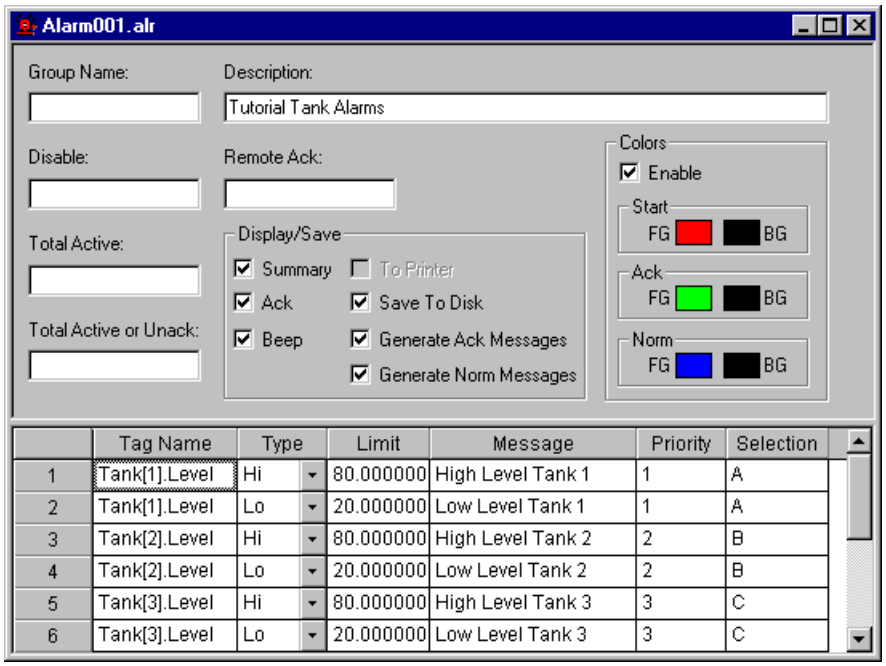

After creating the Class, let us create the Tags. The first one is the Tag called ***Alarm_Settings***, type "CAlarm".

Let us also create a String TAG to be used as Alarm Filter. This tag tag name should be ***Alarm_Sel***, type string and a integer tag called ***View*** as such as two boolean tags ***PGUp*** and ***PGDown***i. All of them with array size = 2.

To create a new alarm group, click on the right button in the "**Alarm**" folder (in the workspace "**Tasks**" tab) and select the option **Insert**.




- Configure the alarm group worksheet as follows (Don't forget to press the diskette icon to save it):




| | Tag Name | Type | Limit | Message | Priority | Selection |
|---|---|---|---|---|---|---|
| 1 | Tank[1].Level | Hi | 80.000000 | High Level Tank 1 | 1 | A |
| 2 | Tank[1].Level | Lo | 20.000000 | Low Level Tank 1 | 1 | A |
| 3 | Tank[2].Level | Hi | 80.000000 | High Level Tank 2 | 2 | B |
| 4 | Tank[2].Level | Lo | 20.000000 | Low Level Tank 2 | 2 | B |
| 5 | Tank[3].Level | Hi | 80.000000 | High Level Tank 3 | 3 | C |
| 6 | Tank[3].Level | Lo | 20.000000 | Low Level Tank 3 | 3 | C |

When you create an alarm worksheet you are telling the systems what tag you want to have as alarm and what kind of alarm, limits, message, priority and filter selection., as well as the message colors in the alarm object.

When you check the ***Save To Disk*** box, the system will record all the alarm occurrences in an ASCII file in the *alarm* subfolder of the application folder, with the *hst* extension.

The manual explanation regarding the Alarm Worksheet Header is the following:

## ALARM WORKSHEET HEADER

This defines a group of common characteristics for all alarms of the group.

**Group Name Field -** Name used to distinguish the alarm groups.

**IMPORTANT:** Before changing the **Group Name** field, save the alarm worksheet, because alarm settings in an unsaved worksheet can be lost.

**Description Field -** Enter remarks here for documentation purposes.

**Disable Field -** Disables all alarms in the group. You must fill this field with a tag. If the value of this tag is greater than zero, the group is disabled, and alarm messages are not generated. If the field is left blank, the group will be always enabled.

**Remote Ack Field -** Tag for alarm acknowledgment. The acknowledgment occurs when there is a value change for this tag.

**Total Active Field** – Holds the total number of active alarms in the group. The system always updates this value when one of the tags changes its alarm condition.

**Total Active or Unack Field -** Holds the total number of active or unacknowledged alarms in the group. The system always updates this value when one of the tags changes its alarm condition.

**Group Box**

**Summary Check-box -** When selected, sends alarm messages to an alarm object on the screen. **IMPORTANT:** If you did not select the **Summary** option, the alarms of this group will not appear in the alarm objects in the screens and printer, during execution.

**Ack Check-box -** Demands the acknowledgment of the alarm messages. Only available if the **Summary** field is enabled.

**Beep Check-box -** Sounds the beep until the alarm is acknowledged. Only available if the **Ack** and **Summary** fields are enabled.

**Printer Check-box -** Sends the each alarm messages of this group to the printer. This option can only be used with a dot matrix printer (or any other which prints line by line).

**Disk Check-box -** Sends the alarm messages of this group to a file on the hard disk. You must select this option if you want to have history alarm objects.

**Generate Ack Messages Check-box -** Generates messages whenever the alarms of this group are acknowledged. Only available if the **Disk** or **Printer** fields are enabled.

**Generate Norm Message Check-box -** Generates messages whenever the alarms of this group return to their normal state. Only available if the **Disk** or **Printer** fields are enabled.

**Colors Group Box**

When the *Enable* check-box is checked, the user can select the alarm messages colors. Otherwise, the default colors will be used.

Click on a Color rectangle to display a **Color Selection** window. Double-click on the desired color or click the color and then the **OK** button.

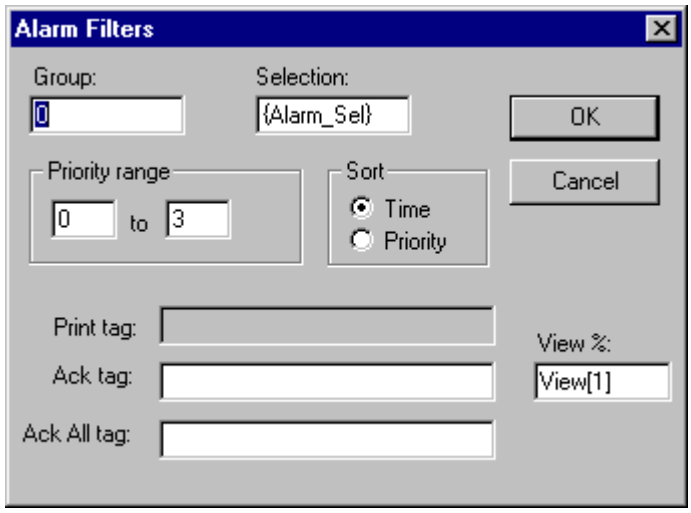
## Creating on-line Alarm screens

- Open the Standard display and save it as “AlarmOnLine”

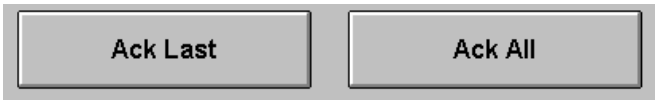
- Create an  "Alarm" object on the screen by clicking on the "Alarm List" icon.

- Now, double click on the "Alarm" object to bring up the "Object Alarm Properties" window. Follow the example below, but be sure that the "On-line" option is selected. (Configure the font as Arial ; 8 ; White)

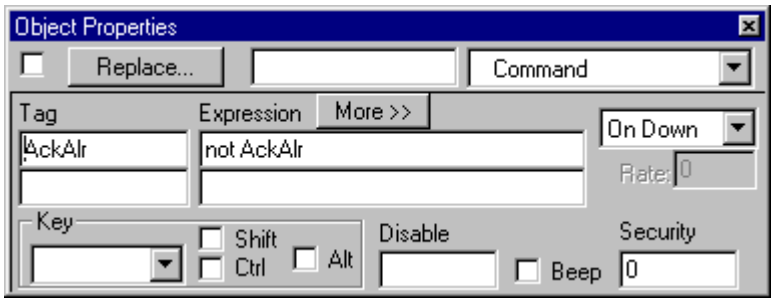

- Press the ***Selection*** button to configure the filter shown below:



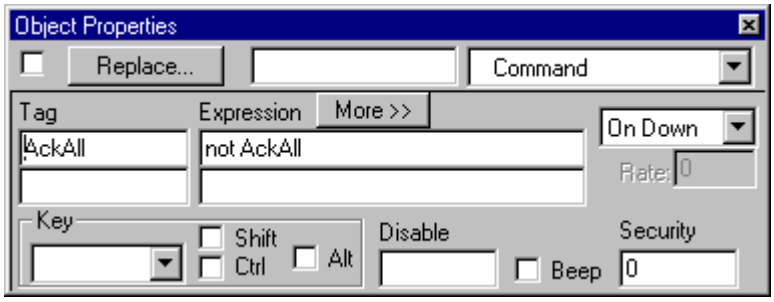

- Now, create two buttons on the screen to acknowledge these alarms.




- The first button is configured with the caption "**Ack Last**" using the internal tag **AckAlr**, which must change by value to acknowledge the last alarm that occurred. Configure the first button as shown below:

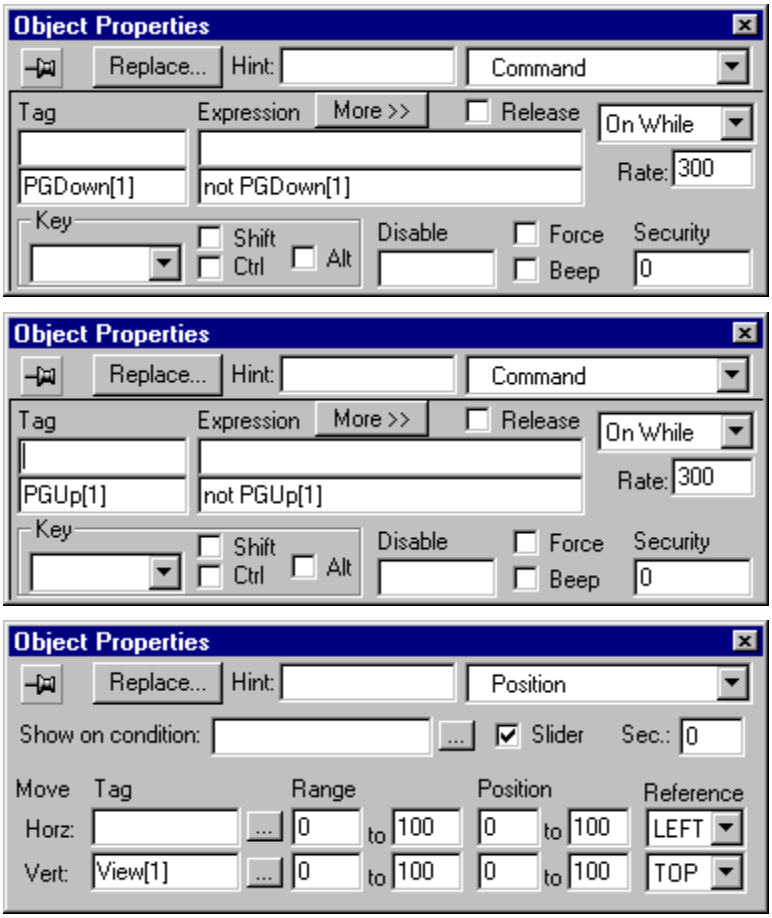



- The second button should be configured with the caption "**Ack All**" and configured with the properties below. Once again, use internal tag **AckAll,** which must change value to acknowledge all the alarms that have occurred.

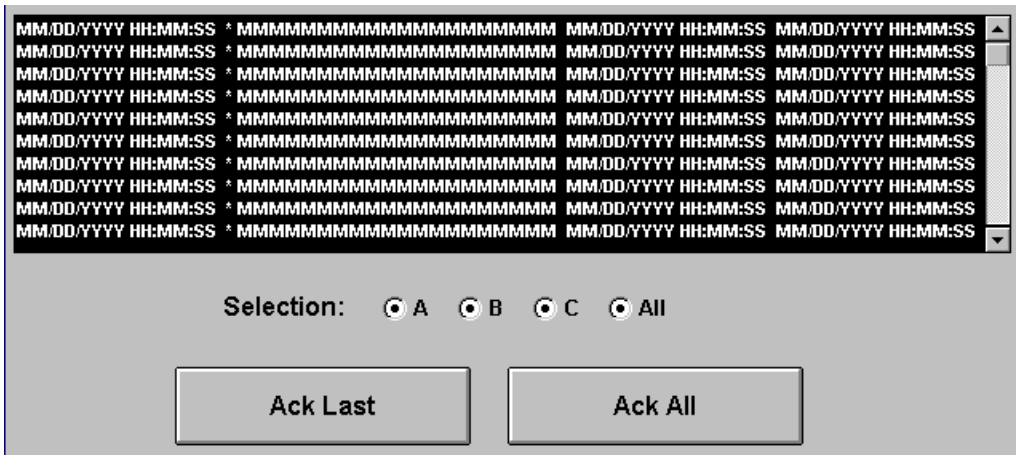



- Draw the text and objects as shown below and configure them with the ***Alarm_Sel*** tag to sort the alarm messages shown in the alarm object.


- Finally, import a vertical slider from the library and configure it as shown below:



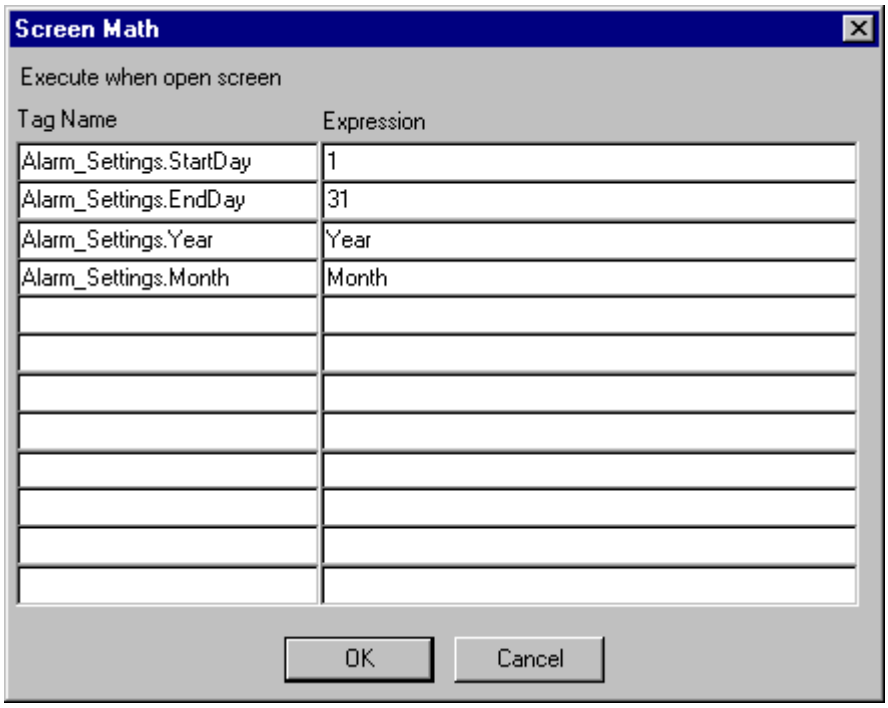

- Save the screen. It should looks like the next picture:



## Creating Historical Alarm screen

- Open the "**AlarmOnLine**" display and save it as "**AlarmHistory**"

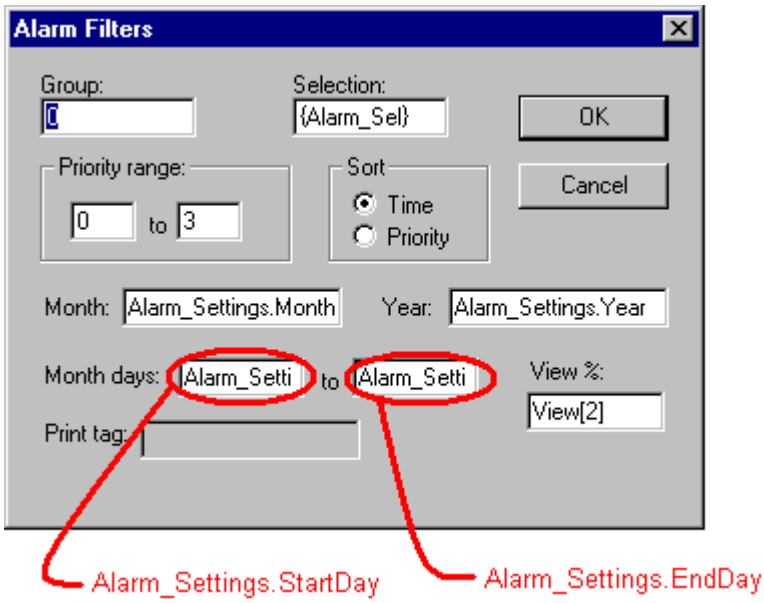
- Press the *On Open* button and fill in the script window using the example below:




- Create an "**Alarm**" object on the screen by clicking on the "Alarm List" icon and selecting an area on the screen to display the alarm.

- Double click on the object to bring up the "**Object Alarm Properties**" window and configure the object following the example below, but be sure that the "History" option is selected.

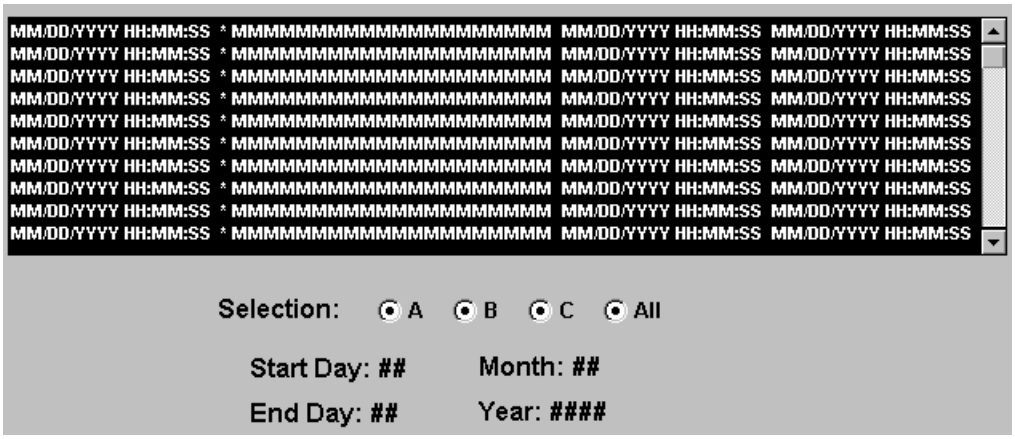

- Press the ***Selection*** button to configure the filter like the example below:




- Replace the Tags ***PgUp[1]***, ***PGDown[1]*** and ***View[1]*** for the tags ***PgUp[2]***, ***PGDown[2]*** and ***View[2]*** in the vertical slider as such as in the Alarm object.

- To select the month , year and range of days in the alarm object, create a Text I/O objects and associate the tags ***Alarm_Settings.StartDay***, ***Alarm_Settings.EndDay***, ***Alarm_Settings.Month*** and ***Alarm_Settings.Year*** to them.


The screen should look like the picture below:




Save the screen and Run the runtime!

Check the color differences according to the alarm states

## The Alarm Tag Fields

Some of the Tag fields (TagName->Field) are direct related to the alarm behavior. The alarm limits, for example, can be changed dynamically writing a new value to the limit field. See below these fields behavior: (The "*****" means that the field can be dynamically changed. )

***HiHiLimit** When creating Very High alarms to tags, this field holds the limits. You can also use it for runtime modifications.

***HiLimit** When creating high alarms to tags, this field holds the limits. It can also be used for runtime modifications.

***LoLimit** When creating low alarms to tags, this field holds the limits. It can also be used for runtime modifications.

***LoLoLimit** When creating very low alarms to tags, this field holds the limits. It can also be used for runtime modifications.

***DevLimit** Limit deviation in tag value that triggers an alarm.

***RateLimit** Limit of rate variation in tag value that triggers an alarm.

***DevSetpoint** Reference point for a tag value deviation that triggers an Alarm.

***Example:*** **Configuration used at runtime: TP->HiHiLimit=70**

*AlrDisable Disables alarm checking, as follows:

1 Disables alarm

0 Enables alarm

*Example:* Configuration used at runtime:

TMP->AlrDisable=1

**HiHi** If greater than zero, a very high alarm is present.

**Hi** If greater than zero, a high alarm is present.

**Lo** If greater than zero, a low alarm is present.

**LoLo** If greater than zero, a very low alarm is present.

**Rate** If greater than zero, an alarm is present.

**Dev** If greater than zero, an alarm is present.

# 13. Trend

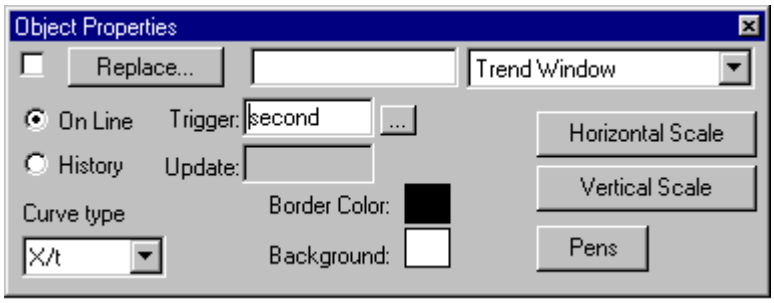
The Trend task keeps track of process variables behavior. You can store the samples in a history file and show both history and online samples in a screen trend graph. To show a trend graph on the screen, we must create a trend object with the Trend icon on the Object Editing Toolbar and to store the historical variable behavior we must create a trend worksheet.

## Trend On line

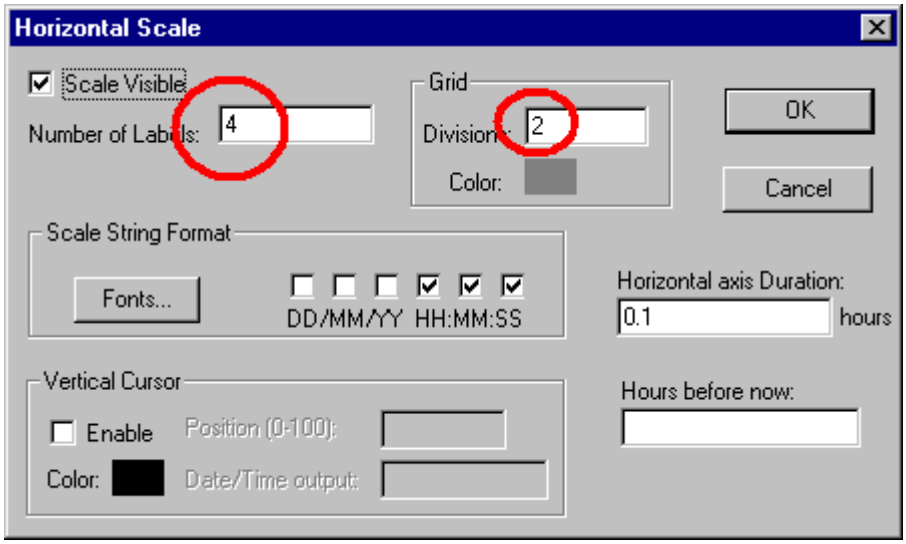
Our on-line trend will show the Temperatures behavior in real time, updated every second.

Open the Standard screen and save it as "TrendOnLine".

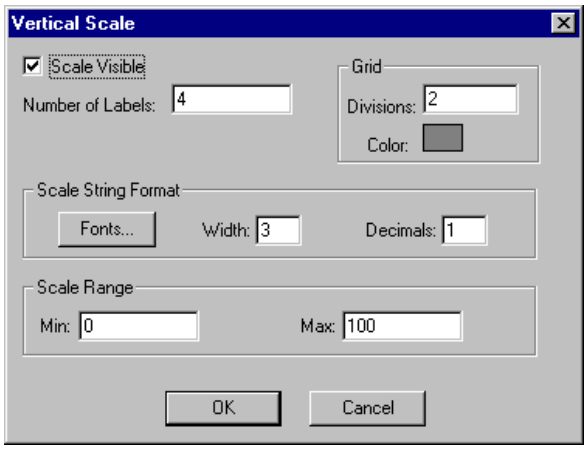
Insert the Trend object and configure it as follows:



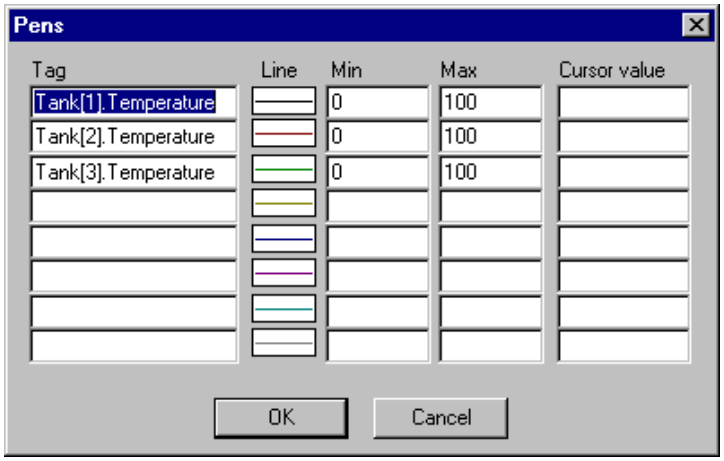

Click "Horizontal Scale" and edit it as follows:



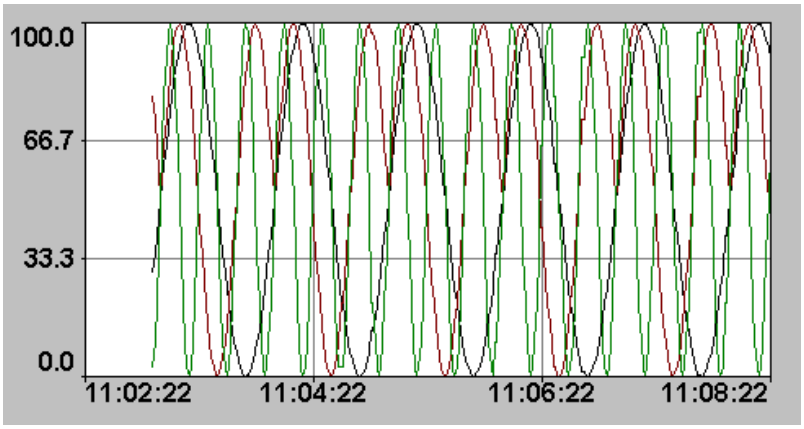

Click "Vertical Scale" and configure it as follows:




Hit "Pens" and edit it like the following:




Save the screen and run it! After a couple of minutes you might have a chart like this on:



# Creating a Historical trend

Now in this part of the tutorial we will show a lot of resources that can be used in the trend charts. Most of these resources are also available to the On-line trend but we are going to show them in the Historical one.

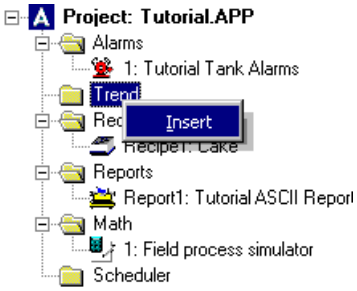
## Creating a Trend group

Before creating screens with trend objects, let us create a group of tags that will be stored on disk to display trend history in the future. One tag will be called **"Trend"** and belong to the class **CTrend**, as shown below.

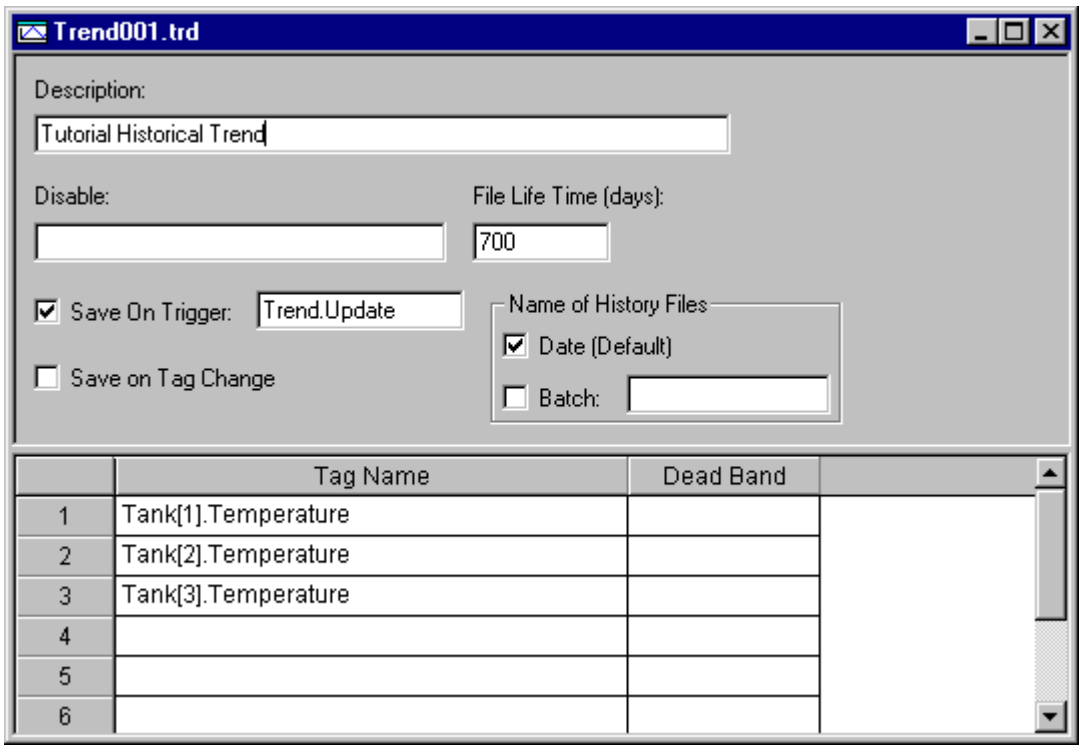
**Class: CTrend**

| | Name | Type | Description |
|---|---|---|---|
| 1 | HiLim | Integer | Trend Hi Limit |
| 2 | LowLim | Integer | Trend Low Limit |
| 3 | Duration | Real | Trend Duration |
| 4 | StartDate | String | Trend Start Date |
| 5 | StartTime | String | Trend Start time |
| 6 | CursorOutput | String | Trend Cursor Date/Time Output |
| 7 | CursorPosition | Real | Trend Cursor Position |
| 8 | CursorPen1 | String | Trend Intersection cursor/Pen 1 output |
| 9 | CursorPen2 | String | Trend Intersection cursor/Pen 2 output |
| 10 | CursorPen3 | String | Trend Intersection cursor/Pen 3 output |
| 11 | Update | Boolean | Trend update trigger (Scheduler) |

Create a new trend group by right clicking on the "**Trend**" folder (in the workspace under the "**Tasks**" tab) and selecting the option **Insert**.

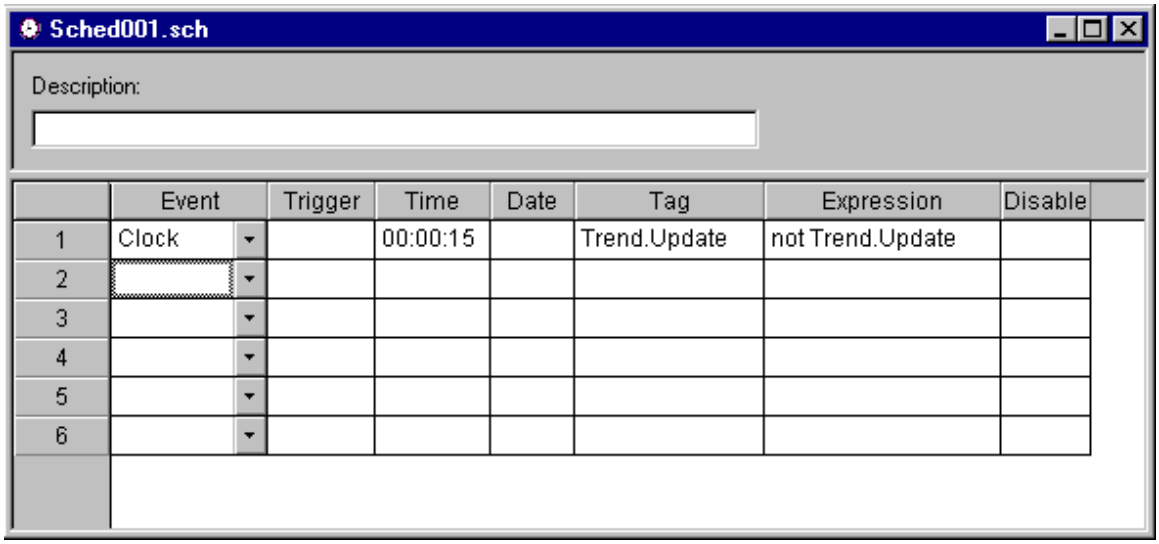



- Configure the trend group worksheet as shown:




With this worksheet we are configuring the system to store the values from the tags *Tank[1].Temperature*, *Tank[2].Temperature* and *Tank[3].Temperature*. The check box "Save on Trigger" and its field beside defines the store rate.

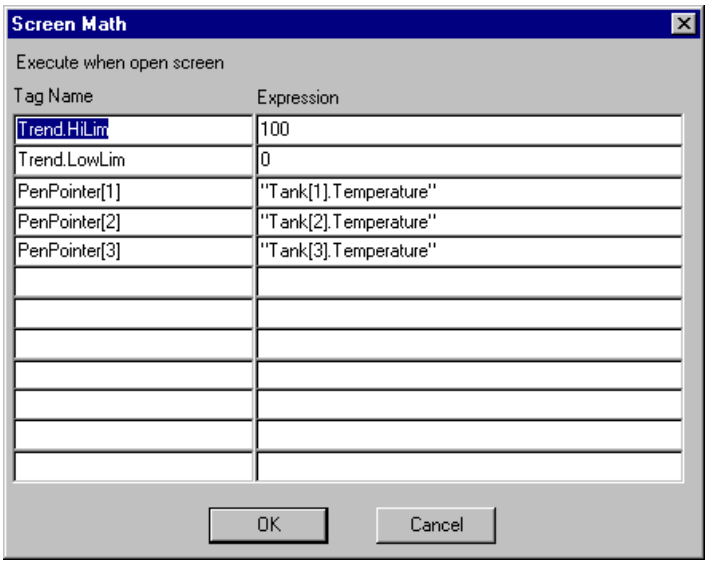
To define the store rate let us create a scheduler line toggling the *Trend.update* every 15 seconds.




## Creating a Trend History screen

- Let us open the "TrendOnLine" screen and save it as "TrendHistory". The trend object must have the following dimension: W:538 H:228.

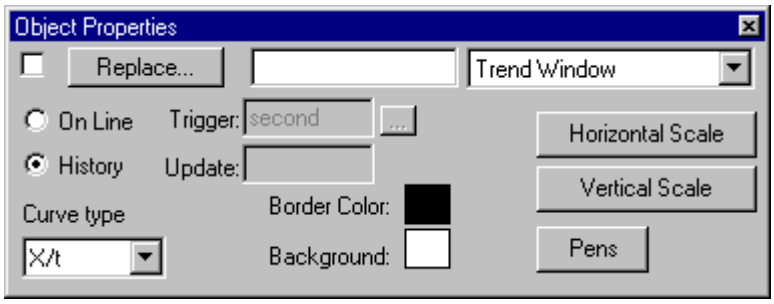
- As we are going to use a lot of variables let us do their initialization on the "Screen Open" logic. Press the button *On Open* to configure the script as shown below:



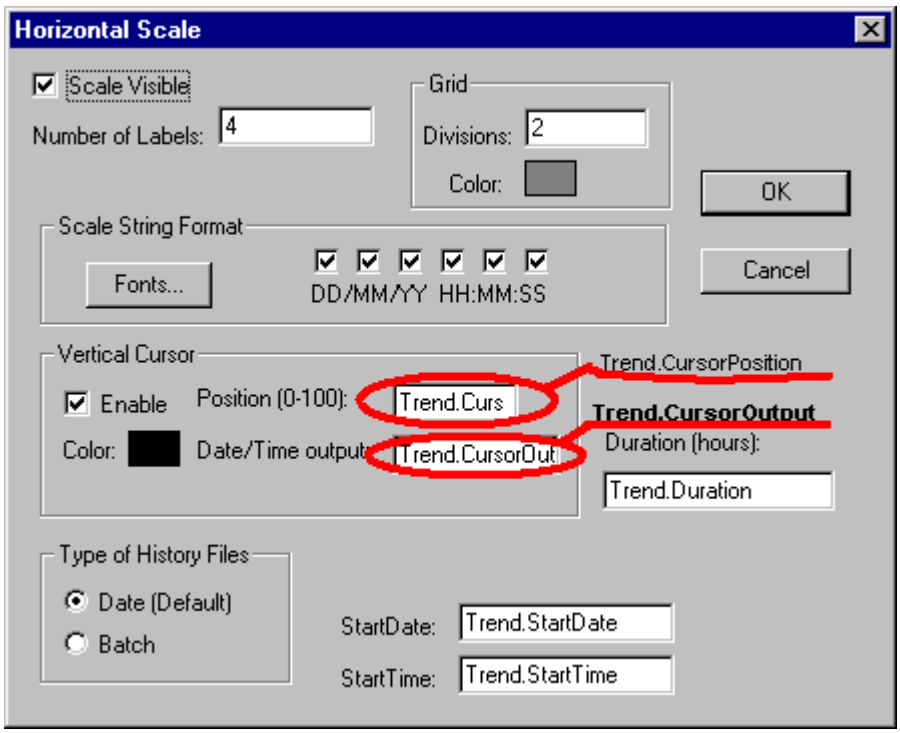

Sometimes it is necessary to adjust the Date format. The Advantech Studio default is MM/DD/YYYY. If in your case it is different, run the function *SetDateFormat("/","MDY")*. This function defines the Date separator (in our example is regular slash "/", but it can be period ".", dash "-", etc…) and the order ("DMY", "MDY", "YDM",…).

This script loads the Pen names into the pointer tags and sets the trend limits. After configuring the Pens and the Vertical Scale you will be able to understand it best.

- Double-click on the trend object and fill in the "Object Trend Properties" window as follows. Be sure that the "History" option is selected



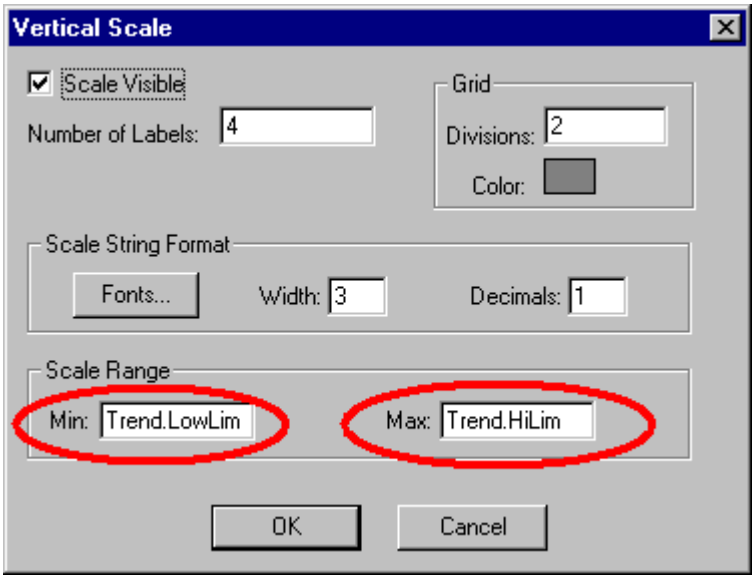

- Press the *Horizontal Scale* button and configure it as follows:




The cursor enables us to know the variable values in different chart positions. The "Position" field will be used in a slide object and the tag inside of the "Date/Time output" field receives Date and/or time of the cursor position.

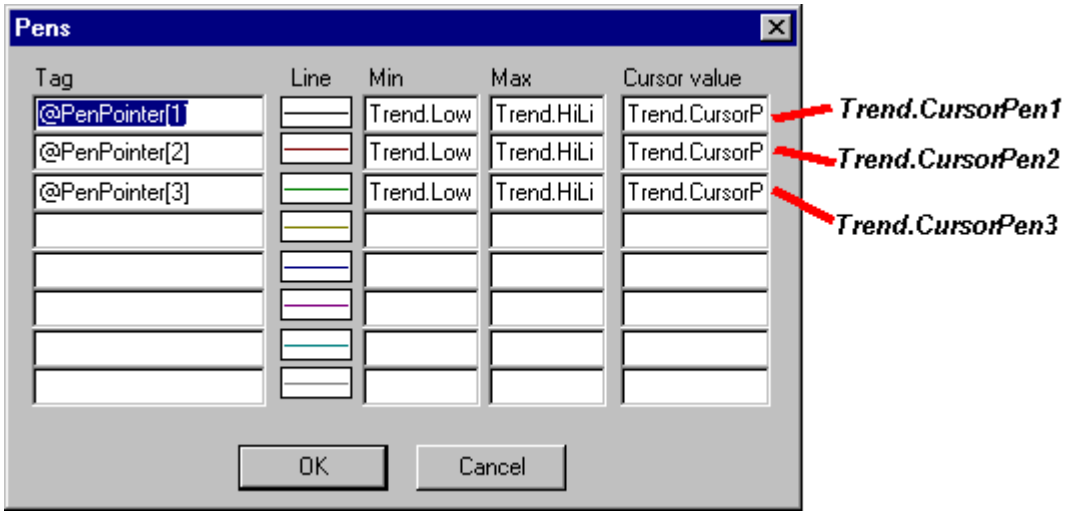
Note also the "Duration" field as well as the StartDate and StartTime fields are filled in with Tags. We are going to create texts with "Text I/O" and input enabled to allow dynamic data inputs.

- Press the *Vertical Scale* button and configure it as follows:




Now the fields Min and Max from the *Scale Range* are filled in with tags. These tags will have texts I/O with inputs to let it be configured by on the Runtime.

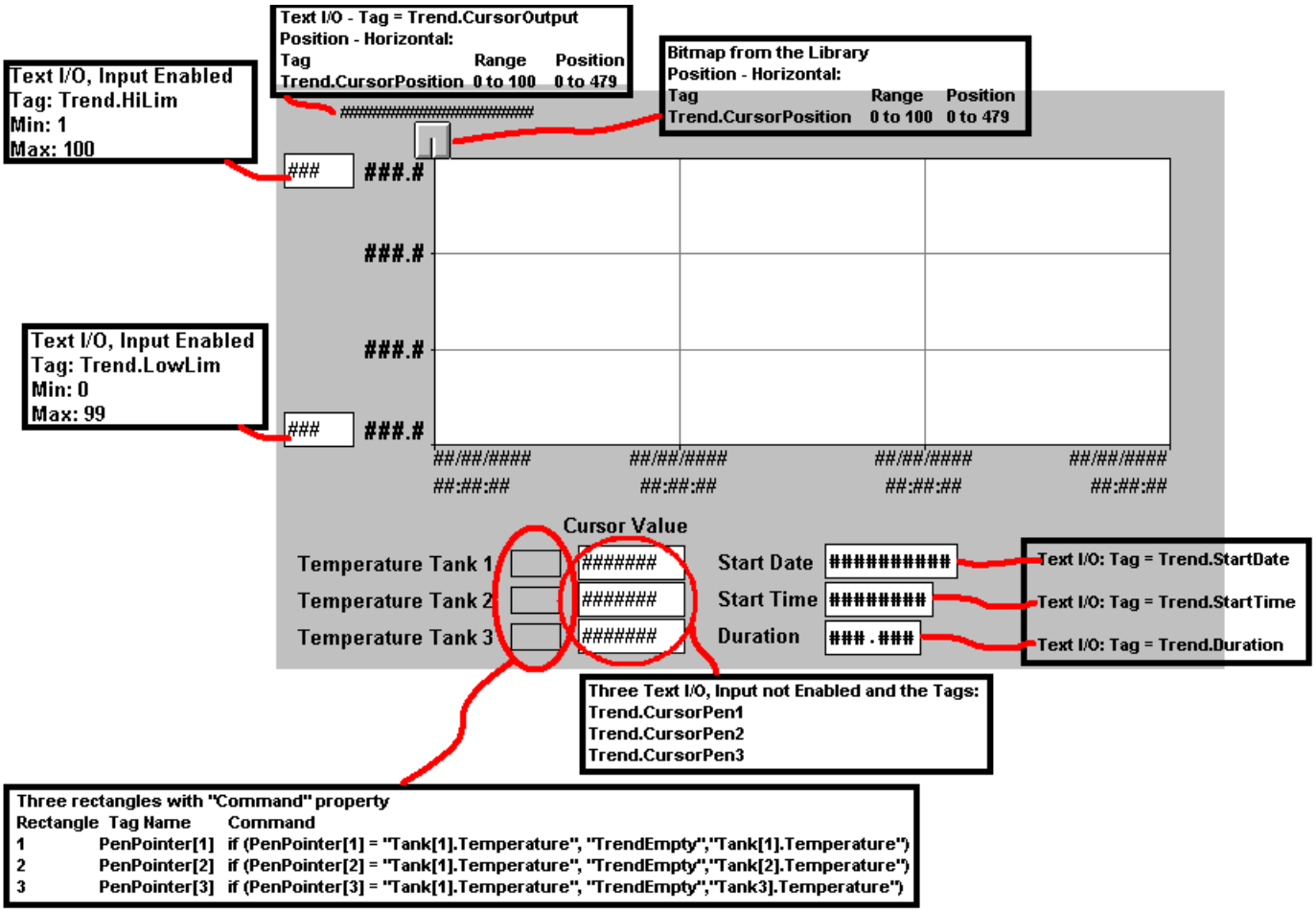
- Press the *Pens* button and configure it as follows:




With this configuration we are able to change the Chart limits *(Trend.Low* and *Trend.High)* and the tags *Trend.CursorPen1, 2* and *3* will receive the intersection between the pen curve and the cursor trace.

The Pointer tags inside of the *Tag* field will point to the *Tank.Temperature* tags or to an empty auxiliar tag, that we are going to call *EmptyTag*. This resource let us choose to hide the pens, as we will see on the Runtime.

The next step is to draw the remaining objects of the screen. Draw it just like picture below and configure the objects like indicated:




Now it is time to laboratory. Start the runtime, wait for 30 seconds (2 records), fill in the StartDate, StartTime and duration with existing values.

# 14. Communication

## PLC Drivers

The driver is a part of the Advantech Studio Software, and its function is to establish the communication between Devices and the Advantech Studio software.

Communication, in this case, means to read values from the Devices' memory to applications variables called "Tags", or write values from the application tags, to the Devices' memory.

We have the following drivers available to the Win9x, NT and CE Operating Systems:

Allen Bradley – DF1

Siemens – S5 – AS511 PG Port

Profibus DP Master and Slave (Hilscher)

Allen Bradley - ControlNet Slave

OMRON – Host Link

GE FANUC – SNP, 90-30 90-70 Series

Modbus – Schneider 984 series

Profibus DP Master

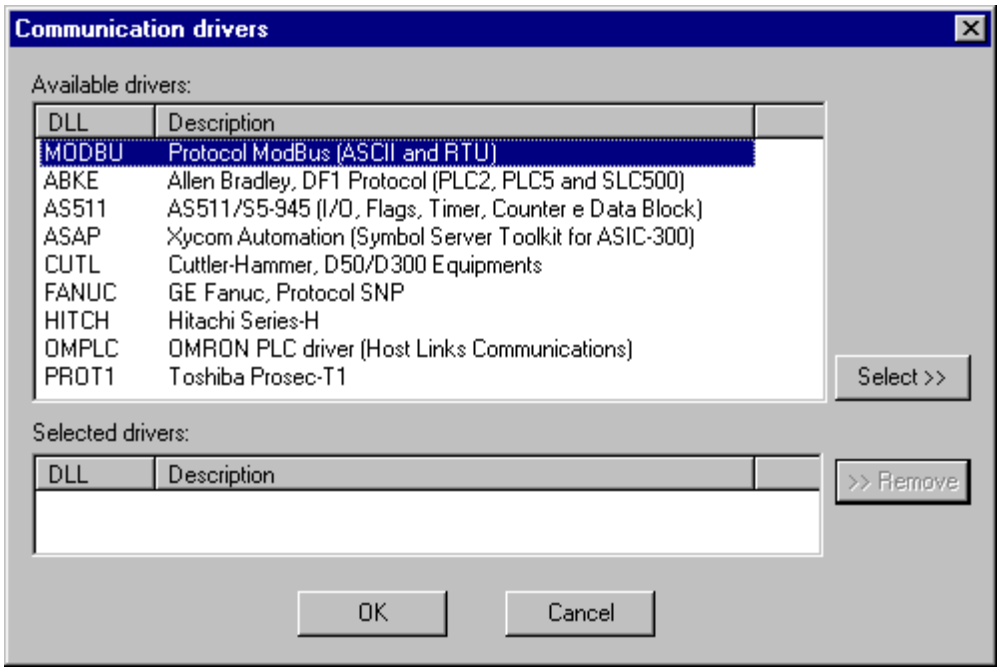
Cultler Hammer – D50 – D300 Series

Hitachi – H series

Toshiba – Prosec T1/T2

### Selecting a Driver

When the Advantech Studio is installed, all the drivers are also installed with it. To start configuring any driver, initialize the Advantech Studio at the Advantech Studio's icon. In the workspace, at the Comm's table, right click on the folder "Drivers", and then on "Add/Remove". On the dialog box opened, choose the desired driver, or its description, then hit "Select". To our examples, let's choose the ModBus Driver.

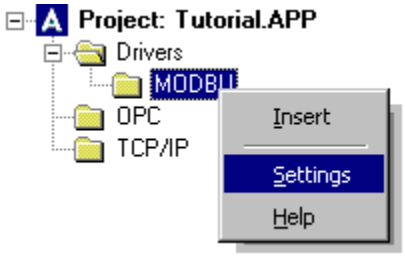



## Configuring the Communication Parameters

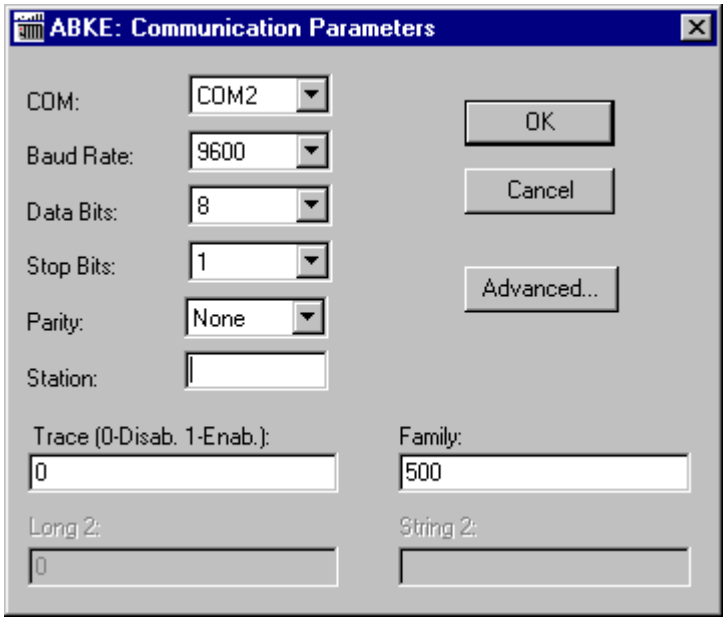
This module configures the serial channel and/or specific parameters of the driver. The values here configured are immediately stored on the system related file. There's no way to save it or cancel the operation.

Changes that have been made here will take effect only when the driver is initialized. Therefore, if the driver is running and the user change these settings, only if the driver get closed and be reopened so the changes will take effect.

To specify the communication parameters right clicking on the MODBU subfolder of the Driver Folder, at the Workspace's Comm's table, the option: **Settings**



## Communication Parameters




| Parameter | Default Value | Valid values | Description |
|---|---|---|---|
| COM | COM1 | COM1 to COM8 | Serial port of the PC used to communication with the device, if it is a serial driver. |
| Baud Rate | 19200 | 110 to 57600bps | Communication data's rate |
| Data Bits | 8 | 5 to 8 | Number of Data bits used in the protocol |
| Stop Bits | 1 | 1or 2 | Number of stop bits used in the protocol |
| Parity | Odd | even, odd, none, space or mark | Protocol's Parity |
| Station | 0 | 0 | Number or Name of the Computer, or Unit in a Network, if the protocol need it. |

Note: The Device MUST be configured with the SAME values defined in the **Communication Parameters** window.

## Long1, Long2, String1 and String2 Fields

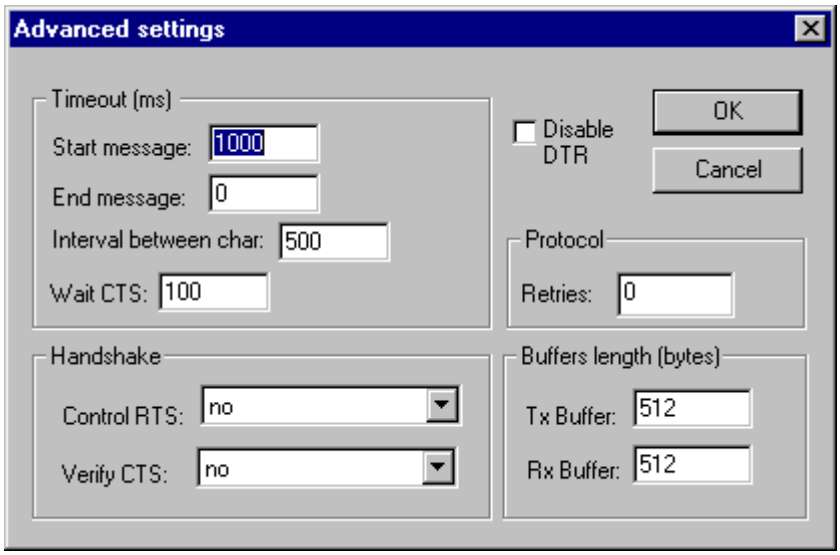
These fields are configured with different functions for each driver. For example, to the Driver ModBus, the String1 field is called "Protocol", and you type here "ASCII" or "RTU".

To the other drivers, these fields are different.

If you type an Invalid Entry in these fields, they will accept it, but, when you try to close the Comm's Parameter dialog, an error message will be shown, avoiding close the dialog.

## Advanced Settings

By clicking in the button **Advanced…** in the windows **Communication Parameters** will be possible to configure other parameters of the serial communication:




| Parameter | Default Value | Valid values | Description |
|---|---|---|---|
| Start message (ms) | 1000 | 0 to 10000 | Maximum time to receive the beginning of the answer from the device (time-out time) |
| End message (ms) | 0 | 0 to 10000 | Maximum time to receive the end of the answer from the device since the beginning of the answer. (Note: the value zero mean that the driver won't check these time) |
| Interval between char | 500 | 0 to 10000 | Maximum time between the characters sent from the device |
| Wait CTS (ms) | 100 | 0 to 10000 | Maximum time to receive the CTS signal after the set of the RTS signal (Note: valid only if the parameter **Verify CTS** has the value **yes**) |
| Control RTS | No | no, yes or yes+echo | Define if the handshake signal of RTS (Request to Send) must be set before a communication and if will have echo in the communication |
| Verify CTS | No | no or yes | Define if the driver must wait for the handshaking |

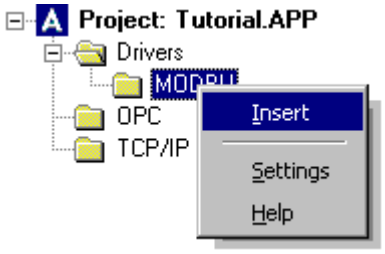
| | | | |
|---|---|---|---|
| | | | signal of CTS (Clear to Send) before send a message |
| Disable DTR | Not checked | Not checked or checked | If checked, the driver won't set the DTR signal before starting the communication |
| Retries | 0 | 0 to 5 | Communication's retries number by each tag configured in the driver sheets, in failure case. |
| Tx Buffer (bytes) | 512 | 0 to 512 | Maximum size of the buffer of information to be sent from the driver |
| Rx Buffer (bytes) | 512 | 0 to 512 | Maximum size of the buffer of information to be received from the host |

*Note: At most of times, these parameters must be changed only when is used a DCE (Data Communication Equipment) - converter (232/485, for example), modem etc - between the PC where is running the driver and the Host. It's necessary to know the features of the DCE before adjusting these parameters.*

NOTE: The settings of the Communication Parameters are the same for all the Driver Configuration sheets.

## Adding a new Driver's worksheet

To add a new driver sheet, right click on the folder with the Driver's name, and hit "Insert". You can also insert a new Driver's sheet by the Menu-> Insert ->Document->XXXX Driver Worksheet, where XXXX is the name of the driver.

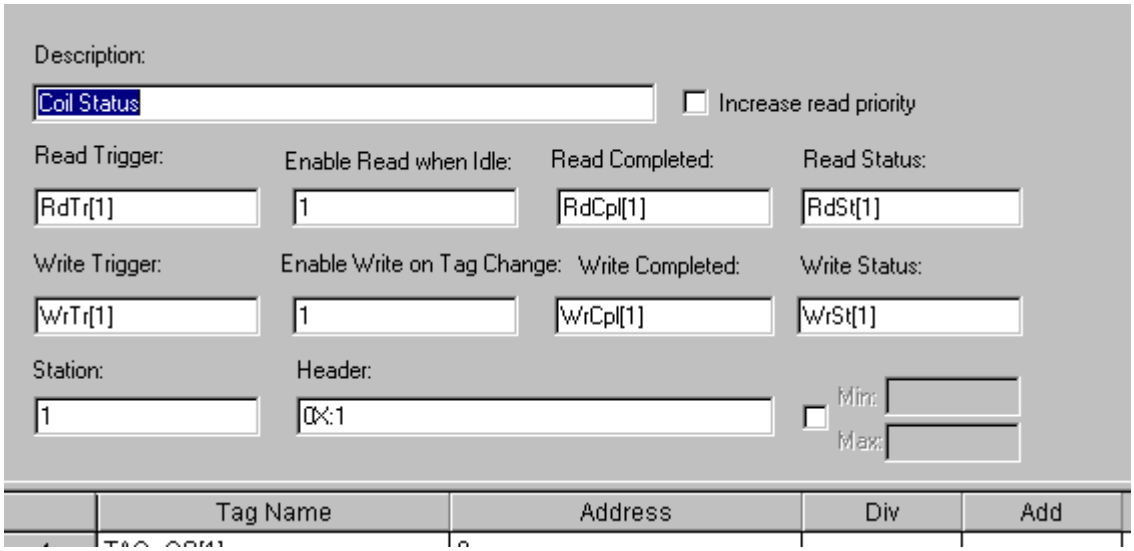



To our example, let's insert a new Modbu Driver Worksheet

Two parts compose the driver: The "Header" and the "Body"

The Header contains all the information about the reads and writes commands and the body, the operator's addresses.

## Header




The Driver Configuration's header contains the required information to the driver's functions, for that sheet. As initial information, for each area that you want to communicate, you must create a new Driver Configuration sheet.

### *Description*

Fill in this field with the description of the spreadsheet, like witch types of areas, their range, and if it's a reading, writing or both sheets. This description will be shown on the workspace, into the driver's folder

### *Increase read priority*

If it's a reading sheet, and there's more reading sheets with the same read trigger or enabled when idle, and a write event happens, the sheet with the priority increased will be the first one on the next reading called by the read trigger or the "read when idle" event.

### *Read Trigger*

This field might contain a tag that will generate a read event always when changed its value.

### *Enable Read When Idle*

This field might contain a tag or a value, that will enable the continuous reading, always when its value be greater than zero.

### *Read Completed*

This field might contain a tag that will have its value toggled always when a reading event has been finished.

### *Read Status*

This field might contain a tag that will have its value filled with an integer value, always when a reading event has been finished. If this value is equal to zero, the event has been Ok. Otherwise, the event has finished in error. The errors message can be viewed at the Logwin module (when in NT), or checked at the MODBU.MSG file, in the DRV Advantech Studio's directory.

### *Write Trigger*

This field might contain a tag that will generate a write event of the whole spreadsheet, always when changed its value. Here we have an extremely important point: when using this feature, the driver will write the TAGs value in the PLCs memory. This operation writes using blocks, from the first sheet operator up to the last. If there's an operator that has not been declared in such spreadsheet, and its address is between the first and the last, it will receive the value zero. Therefore, be sure about what do you want to write when using this trigger, and check out whether there's any kind of hole in the spreadsheet, that could bring any problem to the system or to the PLC's program.

### *Enable Write on Tag Change*

This field might contain a tag that, always when its value gets greater than zero, enables the writing of only the changed tag of the body's spreadsheet, different then the write trigger.

### *Write Complete*

This field might contain a tag that will have its value triggered always when a writing event has been finished.

### *Write Status:*

This field might contain a tag that will have its value filled with an integer value, always when a reading event has been finished. If this value is equal to zero, the event has been Ok. Otherwise, the event has finished in error. The errors message can be viewed at the Logwin module (when in NT), or checked at the OMPLC.MSG file, in the DRV Advantech Studio's directory.

### *Station*

This field must contain the CPU's ID, or Unit Number, or PLC Address, related to this specific worksheet. Each driver has a different syntax to this field. For example, the GE Fanuc SNP driver let you identify the PLC with all ASCII characters, but the OMRON Host Link Protocol, allows only 1 to 31 addresses. They call it Unit Number. So, for each driver this syntax can vary.

Most of times, this is the address of the PLC in a device network.

It can also be filled with a tag, between curly brackets (For example: {tag})

**NOTE**: The TAG placed between the curly brackets, cannot be test on its existence. It means, if is typed a TAG that has not been created yet on the Application Database, in a different form then the other fields, it won't be opened the dialog window with the question regarding its creation. In other words, typing an uncreated tag, the system won't work properly.

**NOTE**: This is a string field, and must be filled correctly, otherwise, the driver won't work properly.

### *Header field*

This is one of the most important field and must contain the worksheet header. Each driver has a different syntax to this field. Usually, you must type here something like the operator's type, followed by the initial address.

Check some examples below:

| Driver | Header | Meaning |
|---|---|---|
| MODBUS | 4X:100 | 4X indicates that this worksheet will communicate with the Holding Registers, |

| | | |
|---|---|---|
| | | from the address 100 on. In the AEG 984 case, from the address 400100 on. |
| OMPLC (Host Link) | IR:0 | IR indicates that this worksheet will communicate with the I/O and Internal Relays, from the address 0 on. In the C200H case, from the address IR00000 on. |
| FANUC (SNP) | %M | %M indicates that this worksheet will communicate with the %M discrete internal operator. There's no initial address to this driver. |
| ABKE (DF1) | N7:0 | N7 indicates that this worksheet will communicate with the N7 file, from the address 0 on. In the PLC-5/40 case, from the address N7:0. |
| AS511 (Siemens PG Port) | DB5:10 | DB5 indicates that this worksheet will communicate with the Data Block number 5, from the Data word 10 on. |

So, for each driver this syntax can vary.

Most of times, this is the address of the PLC in a device network.

In our example case, let's use the MODBUS Systax.

The syntax here is: <reference>:<initial address>. It means, you first type the reference you wish to communicate on this sheet, type the colon (" : "), and then the initial address.

For example, if the header is 4X:1, the sheet will read from the 4000001 until the highest offset configured on the Address column.

The references allowed are:

0X: Coil Status

1X: Input Status (read only)

3X: Input Register (read only)

4X: Holding Register

ID: Report Slave (read only)

There's no limit to the initial address.

Take care with the limits of the addresses. For example, if on the PLC there's no 30500. The "Header" field accepts the syntax "3X:500", but the runtime won't find this register.

Where is indicated "Read Only", the write functions won't work. It's not safe let write on the Input Status, Input Registers and the Report Slave functions.

This field can also be filled with a tag between curly brackets (For example: {tag}).

NOTE: Such as the *Station* field, the TAG placed between the curly brackets, cannot be test on its existence. It means, if is typed a TAG that has not been created yet on the Application Database, in a different form then the other fields, it won't be opened the dialog window with the question regarding its creation. In other words, typing an uncreated tag, the system won't work properly.

When creating a new driver sheet, this field starts blank. Once you place the mouse's cursor on it, even you try to let it blank again, the standard string "0X:1" will take place on it. From then on, it won't be possible let it blank again. But it's possible changing its value for some of above.

#### *Min / Max*

These fields are only enabled if the check box to the left is selected. When selected, it enables a range of values that can be converted into an engineering format. These fields determine the minimum and maximum range of values. Ex.: memory holds values from 0 to 4095 meaning 0% to 100% in the user interface. This setting takes effect for all tags in the worksheet. In this example, **the tag parameters min** and **max** must be set 0 to 100

### 12.1.4.2 Body

The Driver Configuration's body assigns the PLC's memory address to the declared tags, and also handles with the engineering units.

It has 4 columns: *Tag Name*, *Address*, *Add* and *Div*.

#### *Tag Name*

Tag name to be used by the communication driver.

#### *Address*

Address to read and write the tag value in the equipment. Again, as the Header field, this cell is different for each driver. Usually, here you type the offset related to the initial address configured on the Header field. And also, in some cases, you type which bit do you want to this specific Address.

In our driver's example case, you put the offset regarding the initial address configured at the "Header" field. It is not allowed typing a negative offset. The value "0" will overwrite it.

#### *Div / Add / Max / Min*

| Column | Range of Values | Mean |
|---|---|---|
| Div | Any Integer or Real | In read commands: Tag = (Host value) / **DIV**<br>In write commands: Host value = Tag *** DIV** |
| Add | Any Integer or Real | In read commands: Tag = (Host value) + **ADD**<br>In write commands: Host value = Tag - **ADD** |
| Min | Any Integer or Real | Defines the minimum value assigned for the tag, when the corresponding host's value is equal to the value defined in the field Min of the Driver Sheet's Header. |
| Max | Any Integer or Real | Defines the maximum value assigned for the tag, when the corresponding host's value is equal to the value defined in the field Max of the Driver Sheet's Header. |

NOTE: For read operations: <tag> =( <value in the equipment>) / Div + Add

For write operations: <value in the equipment> = (<tag> - Add) * Div

*Note: If the columns of the table above don't be configured (null), the tags of the drive sheet will receive the same value of the address configured.*

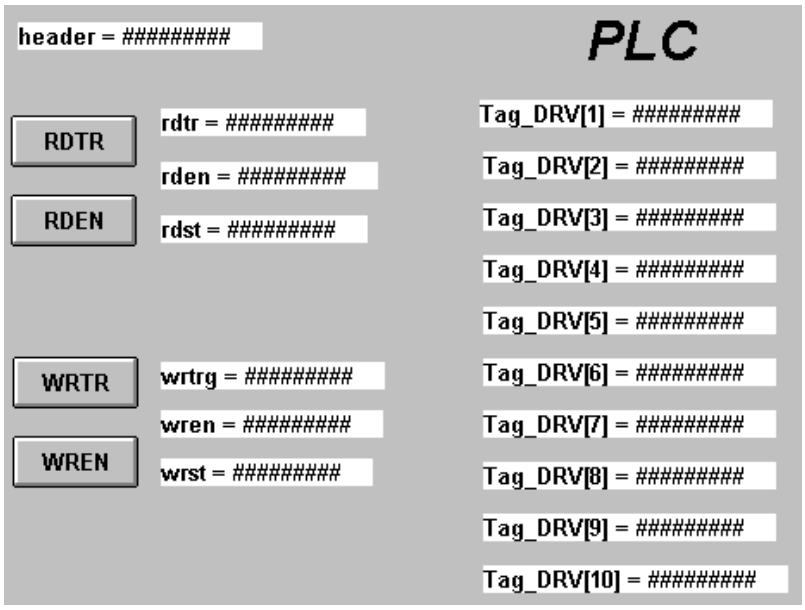
## Preparing the Application to the Driver Runtime Example

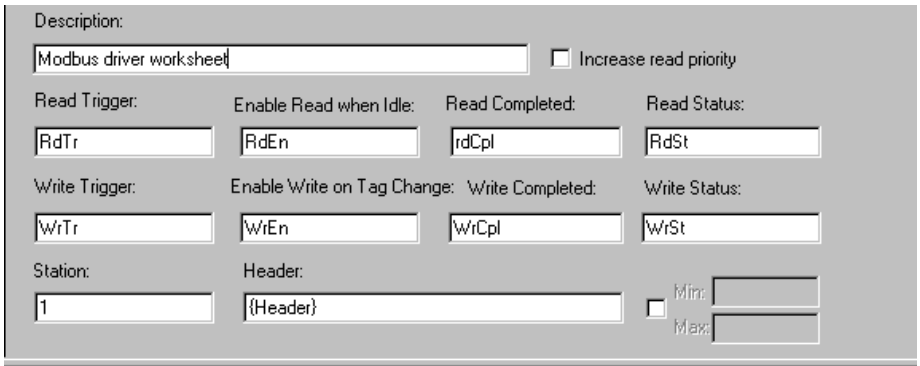
### *Header Tags*

The following tags will be on the Driver Configuration's Headers fields. All of them will be Arrays, and each element of it will be typed on each sheet. For example, RdTr[1] in the field "Read Trigger", to the "ABKE001.DRV"sheet, RdTr[5], to the "ABKE005.DRV", and so forth.

| Tag Name | Size | | |
|---|---|---|---|
| RdTr | 0 | Bool | Boolean tag that will be on the "Read Trigger" fields |
| RdEn fields | 0 | Bool | Boolean tag that will be on the "Enable Read when Idle" |
| RdCpl | 0 | Bool | Boolean tag that will be on the "Read Complete" fields |
| RdSt | 0 | Integer | Boolean tag that will be on the "Read Status" fields |
| WrTr | 0 | Bool | Boolean tag that will be on the "Write Trigger" fields |
| WrEn fields | 0 | Bool | Boolean tag that will be on the "Enable Write when Idle" |
| WrCpl | 0 | Bool | Boolean tag that will be on the "Write Complete" fields |
| WrSt | 0 | Integer | Boolean tag that will be on the "Write Status" fields |
| Station "Header" | 0 | String | String tag that will be, not in the test's beginning, on the field |
| Header "Station" | 0 | String | String tag that will be, not in the test's beginning, on the field |

The communication tags will be TAG_DRV, size 10. Develop a Driver worksheet and a screen like shown below:



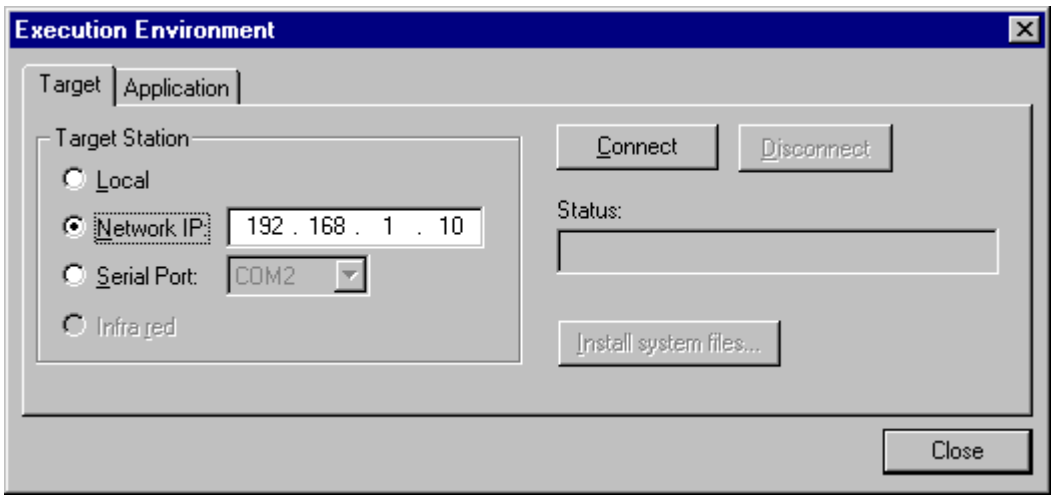




| | Tag Name | Address | Div | Add |
|---|---|---|---|---|
| 1 | Tag_DRV[1] | 1 | | |
| 2 | Tag_DRV[2] | 2 | | |
| 3 | Tag_DRV[3] | 3 | | |
| 4 | Tag_DRV[4] | 4 | | |
| 5 | Tag_DRV[5] | 5 | | |
| 6 | Tag_DRV[6] | 6 | | |
| 7 | Tag_DRV[7] | 7 | | |
| 8 | Tag_DRV[8] | 8 | | |
| 9 | Tag_DRV[9] | 9 | | |
| 10 | Tag_DRV[10] | 10 | | |
| 11 | | | | |

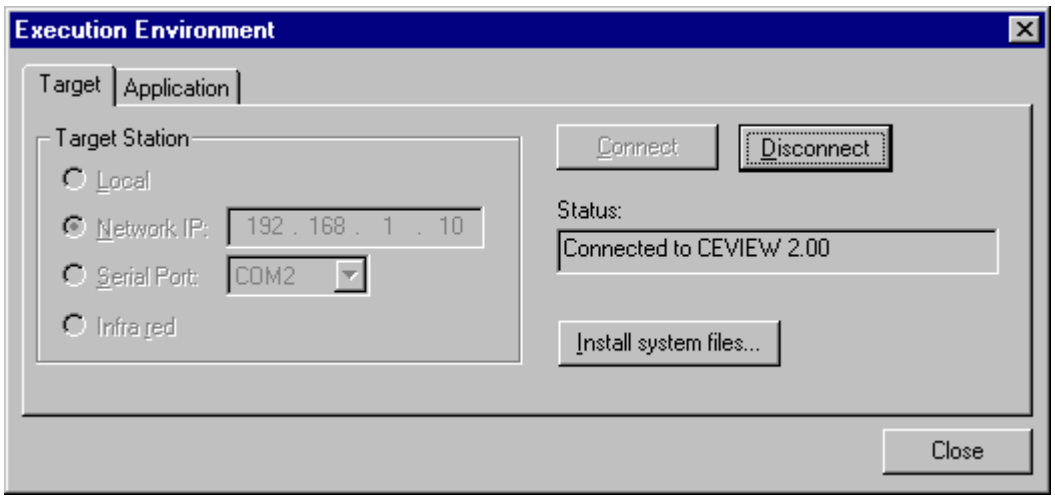
### *First CEView Application download*

The unit must have the executable software CEServer.EXE

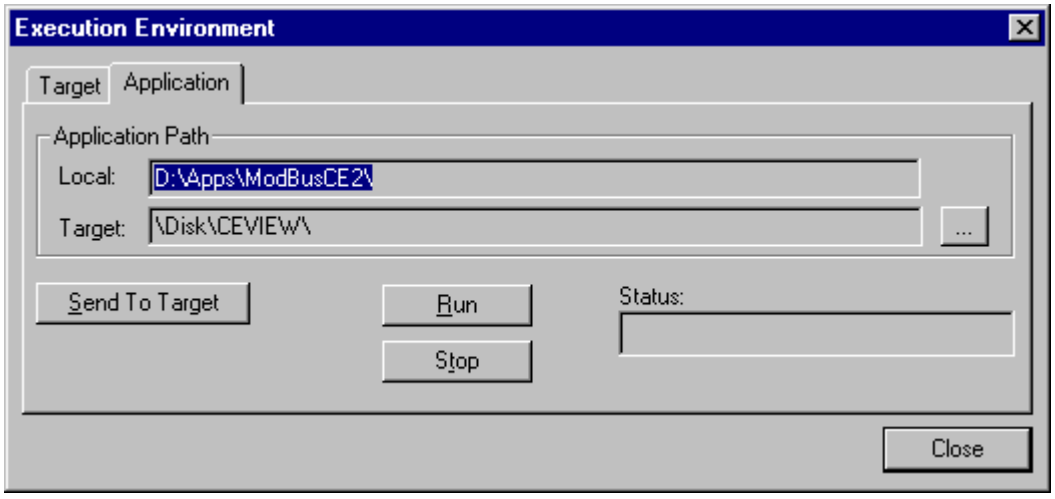
- On the develop station, hit the REMOTE button 



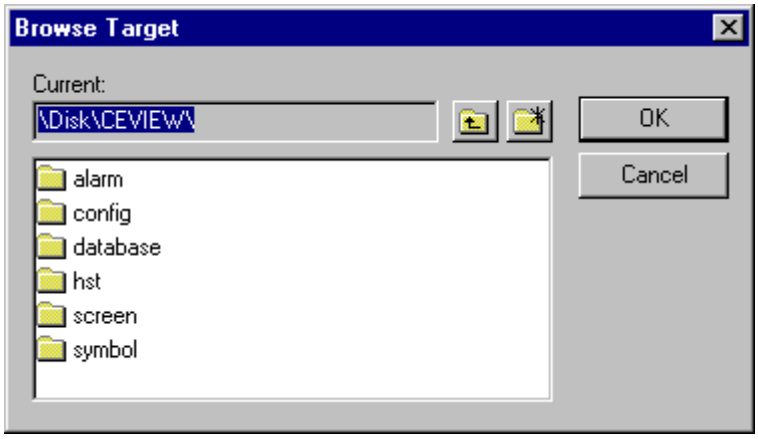

- Type the IP address, and then hit “Connect”




- Hit “Install system files” to install the CEView on the Remote Station.
- After installing the CEView, hit “Application”



- Beside the Field “Target”, choose the remote’s directory where the Application will be downloaded, by using the button ... .




- Download the Application by using the button “Send to Target”. The Unit must have the CESever.exe running.

- After downloading, the application can be started by the button “Run”, and stoped by the button “Stop”

- The field “Status” shows the last operation status.

## Running the application and monitoring the Driver

### *Open Appl_1.scr*

**Test 1: Using the Read Trigger**

Change the value of the Tag *RdTr[1]*. Verify if there's a read event checking if *RdCpl[1]* value has toggled. If so, check whether the values are the same as the PLC, to all the areas. Check also with the Tag *RdCpl[1]* change its value each times a read event is finished, and if the Tag *RdSt[1]* keeps its null value.

**Test 2: Using Enable Read when Idle**

Set the value of the Tag *RdEn[1]*. Verify if there is reading events continuously. If so, check whether the values are as the PLC to all the areas. Check also with the Tag *RdCpl[1]* change its value each time a read event is finished, and if the Tag *RdSt[1]* maintains its null value.

**Test 3: Write Trigger**

Change the value of the Tag *WrTr[1]*. Verify if there's a write event. If so, check whether the values are the same as the PLC, to all the areas, using the SSS programming software.

Check also with the Tag *WrCpl[1]* change its value each time a write event is finished, and if the Tag *WrSt[1]* keeps its null value.

**Test 4: Write on Tag Change**

Set the value of the Tag *WrEn[1]*. Change the value of the *TAG_IR[1].W* and the *TAG_IR[1].b0* tag. Verify if there's write event for each change. If so, check whether the value is as the PLC and if the other operators in the same sheet have not been written on the PLC. Check also with the Tag *WrCpl[1]* change its value each time a write event is finished, and if the Tag *WrSt[1]* keeps its null value.

To see the write on Tag change working well, hit F10 “Shift Coils”, and check what happens.

**Test 4: Write on Tag Change**

Set the value of the Tag *WrEn[1]*. Change the value of the *TAG_IR[1].W* and the *TAG_IR[1].b0* tag. Verify if there's write event for each change. If so, check whether the value is as the PLC and if the other operators in the same sheet have not been written on the PLC. Check also with the Tag *WrCpl[1]* change its value each time a write event is finished, and if the Tag *WrSt[1]* keeps its null value.

# TCP/IP

## Introduction

The Advantech Studio TCP/IP Client/Server modules enable two or more applications to keep their databases synchronized. These modules use TCP/IP protocol to make the communication between the applications.

Before using the Advantech Studio TCP/IP Client/Server modules, you need to install and configure the TCP/IP protocol in the machines you will run these modules.

## Server Configuration

In the server machine, you don't need to configure anything. You just need to run the module Advantech Studio TCP/IP Server. After running this program, a small icon will appear in your system tray.

To close the Advantech Studio TCP/IP Server module, right-click its icon in the system tray, and select

Exit.

## Client Configuration

In the client machine, you need to use the TCP/IP Client Configuration program to configure the

Server IP address and the tags you want to share with the server.

## How to make a TCP/IP Client Configuration

On the Advantech Studio Workspace, choose the “COMM” table, and then insert a new TCP worksheet, right-clicking the TCP folder.

*Description*: this field is used for documentation only. The TCP/IP Client module ignores it.

*Connection Status*: this field should contain a tag name. The TCP/IP Client Configuration module will update this tag according to the connection status. If the tag value is 0 (zero), then the connection is OK. Otherwise, it's the error code returned by the Windows Socket library.

*Server IP Address*: this field should contain the IP Address of the server. It may be a string, or you may use a tag enclosed by brackets. For example, if you fill this field with {tag_name}, the TCP/IP Client module will try to connect to the server indicated by the tag tag_name.

*Tag Name*: these fields should contain the tags you want to share with the server. If the tag is an array or a class (or both), every element and member is shared. You should only put the tag name in this field, without specifying the index or class member. If you specify an index or a class, the TCP/IP Client module will ignore it.

*Remote Tag*: these fields should contain the name of the tag that will be linked with the tag specified in the field Tag Name. This field is optional. If you leave it in blank, the same tag name will be used in the client and in the server.

**Warning**: if you need to share an array, then the tag in the server should contain the same number of elements of the tag in the client. If the tag is a class, then the class definition should be the same in both server and client applications. If you don't follow these rules, unpredictable results may happen.

### Running the TCP/IP Client Module

You can choose to run it auto or manually on the Menu->Status, Runtime Tasks Table, item TCP/IP Client.

After running this program, a small icon will appear in your system tray.

### Custom Parameters

There are three parameters you can configure in the Application Configuration (.app) file.

[TCP]

Port=< TCP/IP port number. Default = 1234 >

SendPeriod=< Time in milliseconds the client/server module will update the tag values of the other machine. Default = 250 >

ConnectRetryTimeout=< Time in seconds the client should retry to connect to the server. Default = 30 >

The Port parameter should be the same in both the client and server machines.

Only the client module uses the ConnectionRetryTimeout.

## OPC (OLE for Process Control)

### Introduction

The Advantech Studio OPC Client module enables the Advantech Studio system to communicate with any device that implements an OPC Server. This module implements the OPC standard as described in the document "OLE for Process Control Data Access Standard Version 1.0A", available at the site http://www.opcfoundation.com.

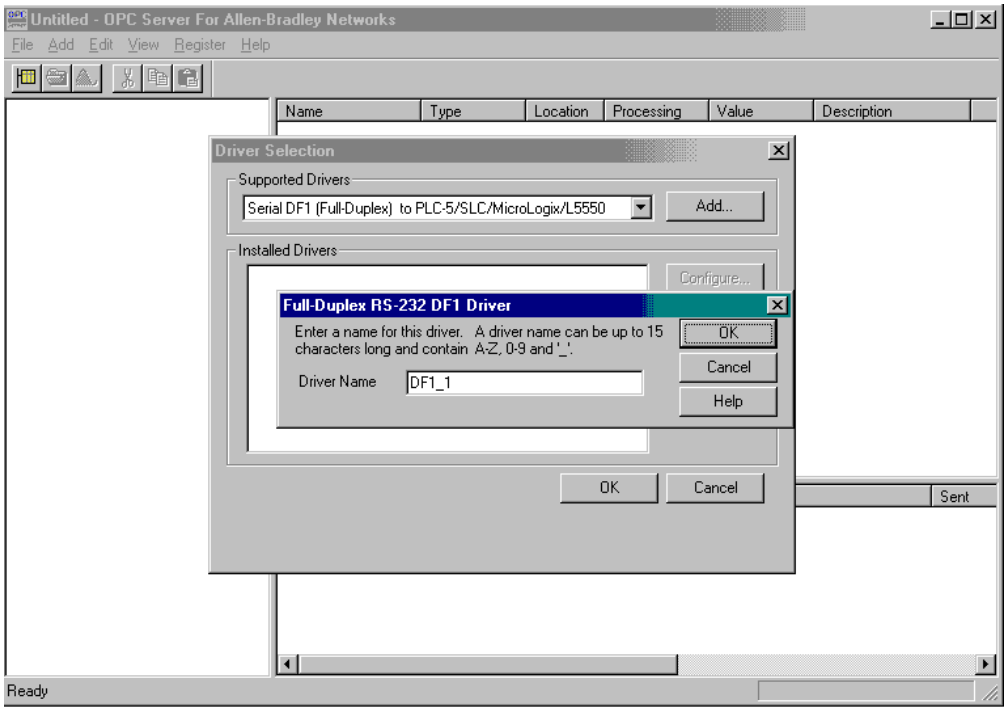
### Preparing an OPC Server Database

Before start using the OPC Client Configurator, you must install the OPC Server. In our example, let us install the ***INGEAR Allen Bradley OPC Server***.

After installing it, let us configure an OPC Server Database. Call the ***INGEAR AB OPC Server***.

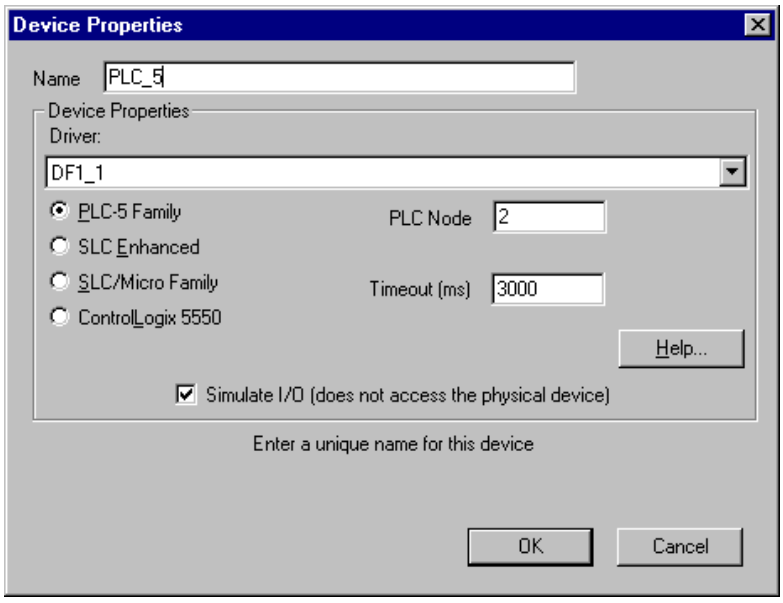
Menu *Add + New Device*. You will be asked for creating a new driver. Accept it.

The first option is the DF1 Serial. As it does not matter the communication (we are going to simulate a communication), the DF1 can be chosen.



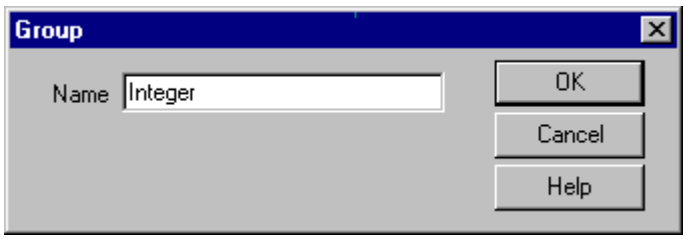

Accept the next steps, like, driver name and communication settings.

In the "Device Properties" window, configure it as shown below. Note the "Simulate I/O" check box checked:

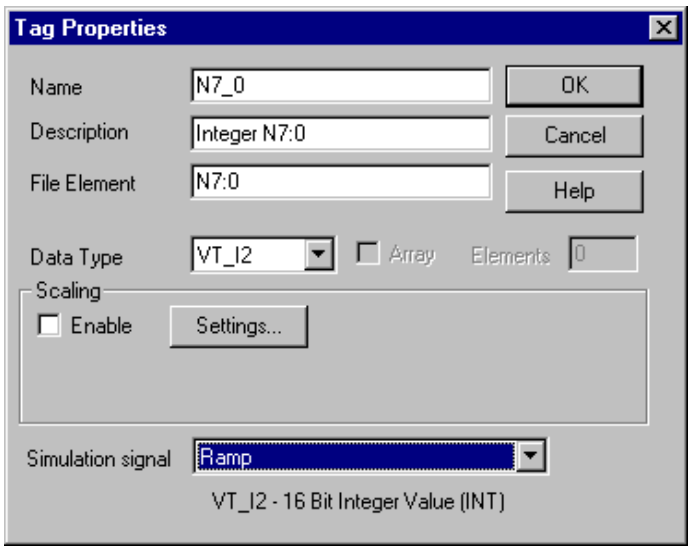



Now you are ready for start adding new OPC Server items. It means that you are going to tell to the OPC Server what PLC address you want to communicate with. You organize it in groups, sub-groups, etc… To our example, let us just create two groups.

Menu *Add* + *New Group*. Give it the *Integer* name.



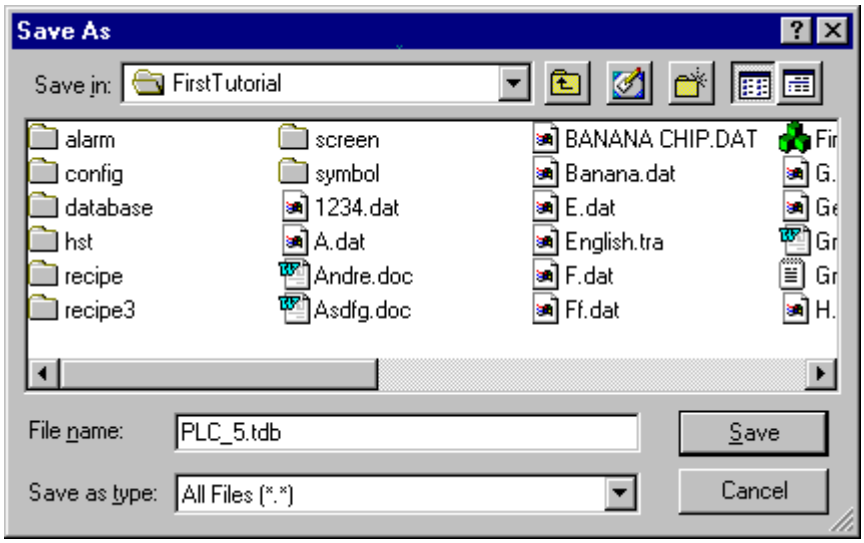

Now, *Add* + *New Tag*. (Note: this new tag means an OPC Server item; it has nothing about the Advantech Studio Tags). Configure the New tag as the following




Create now at least two more tags inside of the *Integer* group, with other elements, like T4:0.ACC, C5:0.PRE, etc…

After that, create a new group, called *Boolean* and create new Tags like B3:0/200, I:0/5, N7:100/4, etc…

Save it as shown bellow (In the FirstTutorial folder):




Check in the Menu *View* the option *Monitor*.

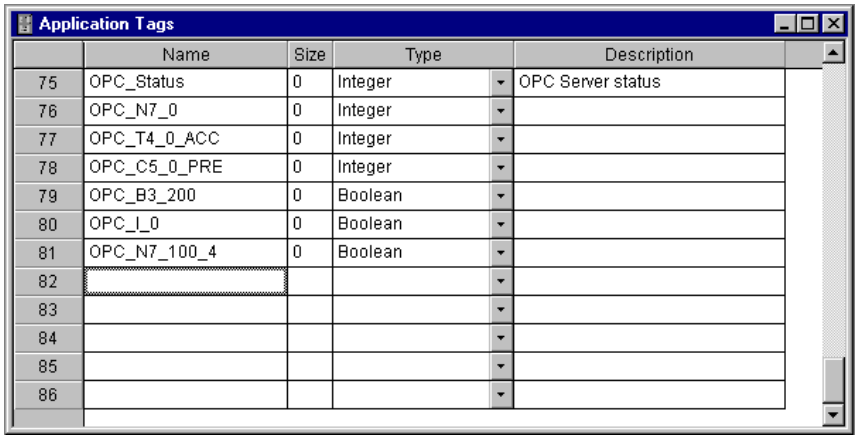
Now you already have an OPC Server database. We are ready to start configuring the Advantech Studio OPC Client worksheets.

## Configuring the Advantech Studio OPC Client Worksheet

Go to the Advantech Studio *Comm* table, right click at the *OPC* Folder and insert a new OPC Client Worksheet.

In the *Server Identifier* combo box you will be able to choose one among the registered OPC Servers. Choose the ***CimQuestInc.IGOPCAB.*** This is how the InGear OPC AB is registered.

Create a new set of tags which will communicate with the OPC Server, like shown below:

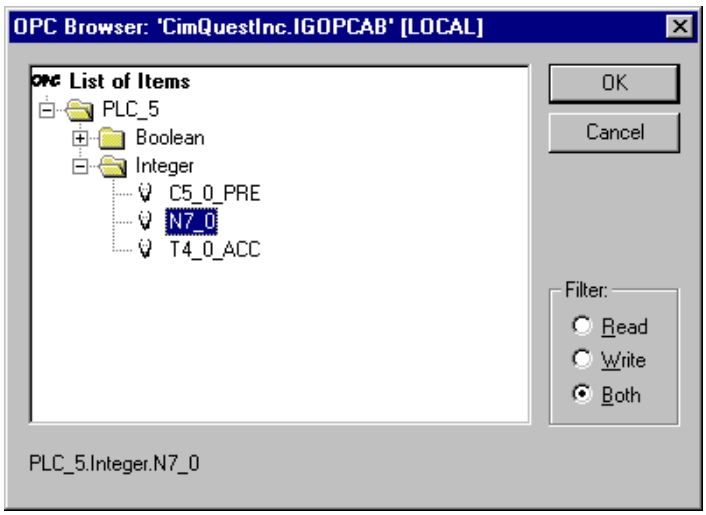



| | Name | Size | Type | Description |
|---|---|---|---|---|
| 75 | OPC_Status | 0 | Integer | OPC Server status |
| 76 | OPC_N7_0 | 0 | Integer | |
| 77 | OPC_T4_0_ACC | 0 | Integer | |
| 78 | OPC_C5_0_PRE | 0 | Integer | |
| 79 | OPC_B3_200 | 0 | Boolean | |
| 80 | OPC_I_0 | 0 | Boolean | |
| 81 | OPC_N7_100_4 | 0 | Boolean | |
| 82 | | | | |
| 83 | | | | |
| 84 | | | | |
| 85 | | | | |
| 86 | | | | |

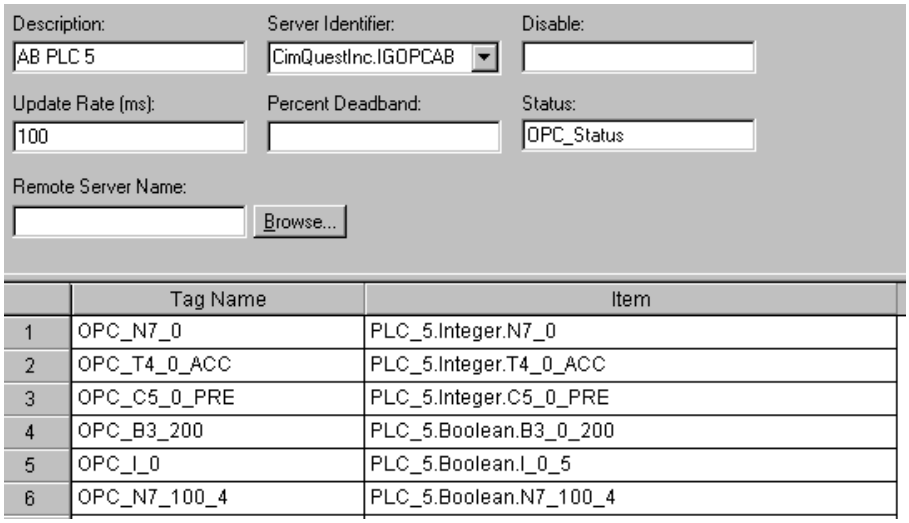
In the OPC Client worksheet, type the Tag name *OPC_Status* in the *OPC Status* Field.

In the first *Tag Name* column row, type the tag name *OPC_N7_0*.

To associate it to the OPC Server item, right click on the *Item* column and hit *OPC Browser*. This will enable you browsing all the OPC Server configured items. In this case, choose the *N7_0* item.

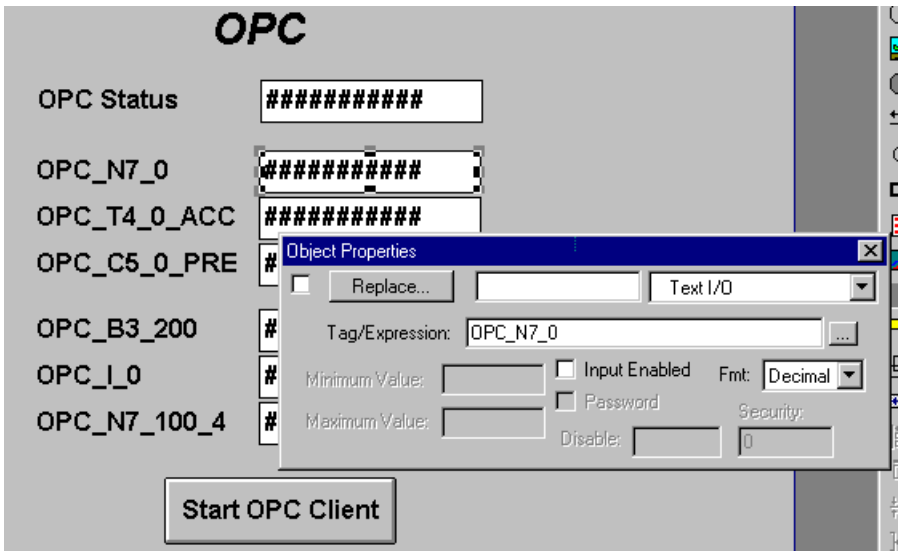



Then you are going to have an OPC Client worksheet as follows:




| | Tag Name | Item |
|---|---|---|
| 1 | OPC_N7_0 | PLC_5.Integer.N7_0 |
| 2 | OPC_T4_0_ACC | PLC_5.Integer.T4_0_ACC |
| 3 | OPC_C5_0_PRE | PLC_5.Integer.C5_0_PRE |
| 4 | OPC_B3_200 | PLC_5.Boolean.B3_0_200 |
| 5 | OPC_I_0 | PLC_5.Boolean.I_0_5 |
| 6 | OPC_N7_100_4 | PLC_5.Boolean.N7_100_4 |

The OPC Client Screen should look similar to this one below:




All the OPC Clients start their OPC Server when starting up. As this OPC Server is just a demo and it has a Dialog Box when opening, it can cause some errors during the Start up. That is why we have a push button to start the OPC Client Runtime task.

So, run the Application and check the OPC screen behavior with the OPC Server values.

# WEB

## Introduction

Advantech Studio allows you to save your application screens in HTML format and export them to Internet Browsers (Internet Explorer).

Once the application has been developed, you just need to set the paramenters in the Web tab from the ***Program Settings*** dialog window and save each screen as HTML (menu ***File - Save As HTML***).

The computer wher the HTML files are sotrage (Pages Server) must be a WEB Server (HTTP Server driver) and the computer where the application is running (Data Server) should have a fix IP address.