# AN-POV-007 – General POV application guidelines for making effective projects

## Overview

This document describes the methodology used when creating applications with Point of View (POV). The suggestions and tips described in this document apply to most of the real-world applications. However, different rules may apply when necessary (e.g. to comply with different customers standards).

## Recommendations

- **Before beginning the development of the application:**
  - Specify the tags database (main classes and arrays). Even though tags can be created during the application development, it is recommended creating the main structures (classes) that encapsulate properties from the same equipment, area, or unit.
  - Specify the screen navigation (links among the screens). Also, study the feasibility to re-use the same screen when applicable, using indirect tags, dynamic array index, are screen mnemonics to display different information on the same screen (e.g.: Trend screen that displays different pens depending on the user selection).
  - Specify the application resolution. When the application is meant to run in different resolutions, develop it in the smallest resolution that must be supported by the application, for the target ratio (width/height). When this document was written, the most common resolution for legacy stations was 1024x768 and the most common resolution for new stations was 1280x800 (wide-screen).
  - Specify the platform where the application will run (e.g.: Windows 7, Windows XP, etc) as well as the license type required for the project.
  - Specify the standard language for the project, as well as the additional languages, if any, available via the translation tool.
  - Specify the objects (or group of objects) that could be used more than once on the same application (different instances) and create linked symbols for them. If you need to modify a common characteristic of this object, you can modify the master symbol once and all instances of this object are updated. It also ensures a higher level of consistency throughout the application.
  - Specify the tags that should be saved in the history (Trend history) and define the frequency required for each group of tags. It is also recommended that you analyze if the history should be saved in a proprietary or external database format. External databases supports flexible queries that allow faster retrieval of data, especially in applications that save a large number of tags with high frequency for a long period of time and allows the user to visualize the trend (e.g.: average) for a long period of time (days, months, years).
  - Configure all global project settings (Project tab).

- **After finishing the development and tests of the application:**
  - Clear all settings from the debugging tools (Database Spy and LogWin).
  - Make sure that the LogWin module is not configured to save logging information to disk.
  - Remove all tags not used on the application.
  - Execute the “Verify” command and check for errors in the Output (Log) window. Fix the errors, and run the Verify Application command again. Repeat this procedure until all errors are solved.

- Look for unused tags and delete them. Make sure not to delete tags reported as unused which may be actually used indirectly (either using indirect tags with the syntax @TagPointer or with the functions $GetTagValue() or $SetTagValue()).
- Delete the trend history files from the \HST sub-folder of the application, unless this data cannot be lost.
- Delete the alarm history files from the \Alarm sub-folder of the application, unless this data cannot be lost.
- Delete any file generated by the application, such as recipe files and database files that were created only to test the application.
- Create a backup of the application (compact the application folder, including all files and sub-folders), and store it in the appropriate place.
- Make sure that all tags that should hold independent values on each Thin Client station are configured with Scope "Local" instead of "Server". Scope is a tag property.
- Review the "Quality Feedback Service" settings section from the Project > Preferences dialog. Usually, it is recommended to enable the option to "Generate dump files when an unexpected error happens", so the cause of the error can be investigated after analyzing the dump files. However, it is important to make sure to monitor the \Web\Dump sub-folder of the application and make sure to send the dump files (if any) in this folder to your software vendor.
- Delete the dump files (if any) from the \Web\Dump sub-folder before deploying the application.

❑ **General:**

- Assess risks associated with the development, deployment, and operation of the application and take actions to prevent or mitigate foreseeable issues. For example, if the application saves data into history files/databases, create a method to manage the data before running out of resources (storage memory).
- The application's documentation (description and comments), as well as the tag names, must be written in English, unless requested otherwise for the end user.
- Write an objective description for each worksheet created in the project.
- Avoid creating file names with the space character (e.g. application name, screen name, recipe name or report name).
- Avoid using accentuation (e.g. ~, ^, `, and so forth) anywhere in the application (including file names).
- Although POV is not case sensitive, it is recommended to follow the same standard for tags and expressions throughout the whole application (e.g. UserName, Str2Asc(), and so forth).
- Avoid adding space characters at the beginning or at the end of labels, captions or descriptions. It may cause problems when translating the application to different languages.
- Keep in mind that the background tasks are executed on the Server only (not on the Thin Client stations).
- Avoid using synchronous methods that potentially takes a long time to be executed, such as the Math() function, Wait() function, or recursive loops. Keep in mind that the execution of each task (thread) is sequential. For example, when calling the Wait() function in a Math(), the whole cycle of execution for all Math sheets is "paused" until the Wait() function returns.
- Avoid calling synchronous functions that display dialogs (e.g. RDFileN() built-in function, MsgBox VBScript function) from background tasks (math sheets or scheduler sheets). The task that executed the function will be paused until the user closes the dialog. Keep in mind that scripts executed with the built-in function $RunGlobalProcedureOnServer() are executed on the Server (backgroup), even when called from the screens (Viewer).

- Never call synchronous functions that display dialogs (e.g. RDFileN() function) from scripts executed on the server, but called from the Thin Clients (e.g.: executing the RunGlobalProcedureOnServer() function). The Server will not communicate with all Thin Clients until the dialog is closed on the Server.
- Avoid using the Changed() function. Use the Scheduler module instead. The Changed() built-in function is kept in the product for legacy reasons only.
- Avoid hard-coding paths, unless when necessary. Files saved and stored in the application's directory can be accessed using the built-in function GetAppPath(), which returns the current path of the application. Using this approach, access to files will keep working even if you move the application to a different directory.
- Use Recipes (XML format) to store configurable settings that should be loaded in specific events (e.g.: when loading the application).
- Organize the procedures defined in the Global > Procedures interface using the '$region:<GroupName> syntax to group functions and sub-routines in the same sub-folder of the Project Explorer.
- Use the Output window (Log window) to monitor the communication with the external devices (Field Read Commands, and Field Write Commands) to make sure that the communication is flawless. Depending on the driver configuration (no simultaneous connections), communication errors (e.g.: timeout, invalid block size, and so forth) in one communication group will decrease the overall communication performance.
- Use the $Ext() built-in function to translate text that may be displayed in external dialogs, such as message boxes launched by the VBScript MsgBox function.
- Avoid repeating scripts. Instead, create procedures (functions or sub-routines) in the Global > Procedures or Graphic Scripts libraries.
- When writing scripts in the VBScript that might fail during the runtime (e.g.: initialize an external COM Server object), use the VBScript "On Error Resume Next" and "On Error Goto 0" statements to handle the errors gracefully. In this case, check for error and create a mechanism to notify the user if it occurs (e.g.: alarm message).
- Whenever using a built-in function to open a database connection (e.g.: DBCursorOpenSQL, DBCursorOpen), make sure that the built-in function to close the database connection is eventually called (DBCursorClose). Opening connections continuously and not closing them will cause "memory leak" and it will eventually crash the system for lack of resources.

❑ **Applications for Touch-screen devices:**

- Enable the Virtual Keyboard from the Project > Viewer dialog.
- Do not use the "Right-click" events and avoid using the Double-click event from the Command dynamic.
- Design objects that support interactions from the user (e.g.: buttons, scroll bars, sliders) with an appropriate size to be operated from a touch-screen device, which does not offer the same precision of a mouse pointer.

❑ **Tags Database:**

- Create Classes in the Tags Database to encapsulate properties from a specific device, object or area of equipment of the process.
- Create array tags when the same property is applied to many instances of the same device, object or area of equipment of the process.
- Add a description to all tags created in the tags database. The description must be objective and provide minimum information for future maintenance of the project.
- Avoid creating tags to hold information that is available in the tag fields (e.g. HiLimit, AlrStatus, TimeStamp, Quality, and so forth).

- The tag name should be as short as possible, but it must be long enough to be meaningful, indicating the value that the tag handles.
- Avoid using retentive values or retentive parameters for many tags. Doing so may decrease the performance of the system, because the tag values must be saved in disk whenever the tags change of value. If it is necessary to keep the value of many tags retentive, study the feasibility of using the Recipe task to save the values at a suitable rate.

❑ **Graphics:**

- Use groups of screens when applicable to avoid replication of objects and configuration. Information that should be available continuously to the operator should be configured in one or more screens that are always open and visible. For instance, use a Header screen, which displays the logo of the company, date, time, username, main menu and so forth.
- Do not duplicate screens that are identical or very similar. Use features such as indirect tags and indexed array tags to display different information from the same screen.
- If the application must be translated to different languages during runtime, make sure that the interface is designed to accommodate the text in all supported languages—and not only in the original language used to created the screens.
- When designing screens with many static graphics, study the feasibility of using a background graphic (instead of several graphic objects) to increase the performance during runtime. If more than one screen uses the same background, use the Shared Image option from the Screen Attributes dialog (with screen active, Graphics > Attributes).
- Consider using Linked Symbols to any interface that is used more than once in the application (e.g.: pump icons, PID faceplates, and so forth).

❑ **Performance and optimization:**

- Use the Execution field from math sheets or VBScript sheets to enable their execution only when necessary. Running a math sheet or VBScript sheet continuously (Execution=1) when it is not necessary will use resources unnecessarily and it may decrease the performance of the system.
- When configuring native driver worksheets, use sequential addresses as much as possible. Any communication protocol has a maximum block size (protocol-specific limitation), which restricts the maximum number of data that can be exchanged by each request. The more requests are necessary to read all addresses required by the application, the longer the overall communication cycle. Therefore, the more sequential the addresses configured in the driver worksheets, the lower the number of different communication blocks to read the data from the controllers (e.g.: PLCs) and the higher the overall performance.
- When communicating with external middleware components (e.g.: OPC Servers), keep in mind that on the application startup the link with the external system must be initialized (link local tags with external items). In some cases (e.g.: specific OPC Servers) the middleware may take a significant period of time to initialize a large number of items. Therefore, minimize to the minimum the items linked to external systems to optimize the startup performance. In other words, avoid configuring items that will not be used in the application.
- OPC communication is driven by exception events. In other words, after initializing the connection, any item that changes of value on the Server is sent to the Client, and vice-versa. This method potentially optimizes the communication since it avoids the need to pool data between the Client and the Server. However, depending on the nature of the process, it may overload the communication network, especially if many items are changing continuously. In order to avoid unnecessary communication traffic, it is strongly recommended to configure a “Dead Band”, so only material information is exchanged.
- The communication between the Server and Thin Clients is driven by exception events (tag changes). It optimizes the communication and avoids pooling between the Server and the Thin Client stations. However, if tags monitored on the Thin

Client stations change too frequently, it is recommended to increase the “Send Period” setting to minimize the traffic in the network.

- Point of View allows the user to save history data either in the proprietary history files (binary files) or in an external SQL Relational Database, such as Microsoft SQL Server, Oracle, MySQL and so forth. SQL Relational Databases support flexible queries that optimize the data retrieval from a large database. Therefore, for applications that require history for a large number of tags saved at a high frequency and kept for a long period of time, it is recommended to use an external SQL Relational Database to store the data.
- The Trend Control is able to retrieve data from the proprietary history files and also from external SQL Relational Databases. Depending on the architecture of the application and schema of the database tables, it will be more efficient to configure the Data Source “Database” instead of using the Data Source “Tags” to retrieve data from the external SQL relational database. Using the data source “Database”, the user can customize the queries to optimize the data retrieval according to the particular schema of the database tables and type of data required by the user.
- Avoid writing scripts that are executed continuously in the “Screen_WhileOpen” sub-routine of screens or in the “Graphics_WhileRunning”, unless when necessary. Scripts configured in these sub-routines are continuously executed by the Viewer module and, therefore, competes for the resources to uptime the screens with optimum performance.
- The scripts configured in the “Screen_OnOpen” sub-routine of screens are executed before displaying the screen. Therefore, scripts that potentially take a long time may delay the screen appearance.
- Pictures, such as bitmaps demand more resources than native objects and can decrease the performance when opening and updating screens with large pictures or with a large number of pictures