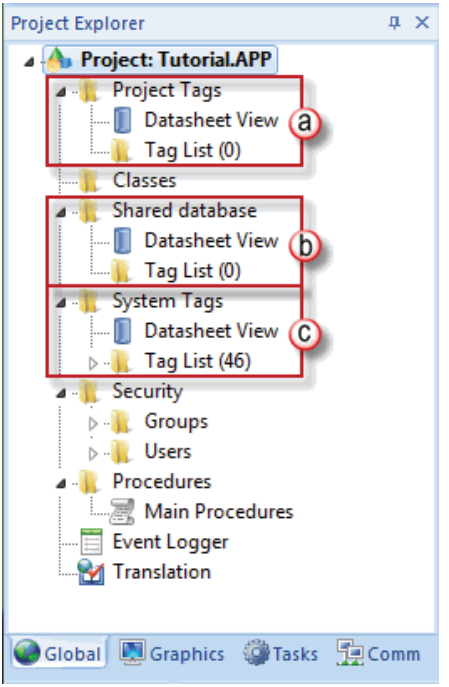
About Tags and the Project Database

ABOUT TAGS AND THE PROJECT DATABASE..... 3-2
Understanding the Tag Name Syntax.....3-4
Choosing the Tag Data Type.....3-5
Using Array Tags.....3-6
About Indirect Tags.....3-9

About Tags and the Project Database

Tags are a core component of any POV project. Simply put, tags are variables used by POV to receive and store data obtained from communication with plant floor devices, from the results of calculations and functions, and from user input. In turn, tags can be used to display information on screens (and Web pages), to manipulate screen objects, and to control runtime tasks.

But tags are more than simple variables. POV includes a real-time database manager that provides a number of sophisticated functions such as timestamping of any value change, checking tag values against runtime minimum and maximum values, comparing tag values to alarming limits, and so on. A POV tag has both a value and various properties that can be accessed, some at development and others only at runtime.



All tags are organized into one of the following categories, which are represented by folders on the Global tab of the Project Explorer:

a. Project Tags are tags that you create during project development. Places where project tags are used include:

- Screen tags
- Tags that read from/write to field equipment
- Control tags
- Auxiliary tags used to perform mathematical calculations

b. Shared Database tags are created in a PC-based control program and then imported into POV's

tags database.
For example, you might create tags in Productivity Suite and import them into POV so POV can read/write data from a P3000 PAC based control product.
You cannot modify shared tags within POV — you must modify the tags in the original PC-based control program, and then re-import them into the Tags database.

c. System Tags are predefined tags with predetermined functions that are used for POV supervisory tasks. For example,

   - Date tags hold the current date in string format
   - Time tags hold the current time in string format

   Most system tags are read-only, which means you cannot add, edit, or delete these tags from the database.
   To see a list of the system tags, select the Global tab in the Project Explorer, open the System Tags folder, and open the Tag List subfolder. The figure, shown on previous page, shows a partial list of system tags.

After creating a tag, you can use it anywhere within the project, and you can use the same tag for more than one object or attribute.

NOTE: *System tags do not count against your project tag count.*

Understanding the Tag Name Syntax

Observe the following guidelines when naming a tag:

- Your tag names must be unique — you cannot specify the same name for two different tags (or functions). If you type an existing tag name, POV recognizes that the name exists and will not create the new tag.
- You must begin each tag name with a letter. Otherwise, you can use letters, numbers, and the underscore character (_) in your tag name.
- You cannot use the following symbols in a tag name:
  ` ~ ! @ # $ % ^ & * ( ) - = \ + \ [ ] { } < > ?
- You can use a maximum of 255 characters for a tag name or a class member name.You can use uppercase and lowercase characters. Tag names are not case sensitive. Because POV does not differentiate between uppercase and lowercase characters, you can use both to make tag names more readable. (For example: TankLevel instead of tanklevel.)
- Tag names must be different from system tag names and math functions.

NOTE: *Use the @ character at the beginning of a tag name to indicate that the tag will be used as an indirect tag in the project.*

Some valid tag examples include:

- Temperature
- pressure1
- count
- x

Choosing the Tag Data Type

Another consideration when designing a tag is what type of data the tag will receive. POV recognizes the following, standard tag data types:

- **Boolean** (one bit): Simple boolean with the possible values of 0 (false) and 1 (true). Equivalent to the "bool" data type in C++. Typically used for turning objects off and on or for closing and opening objects.
- **Integer** (four bytes): Integer number (positive, negative, or zero) internally stored as a signed 32-bit. Equivalent to the "signed long int" data type in C++. Typically used for counting whole numbers or setting whole number values. Examples: 0, 5, –200.
- **Real** (floating point, eight bytes): Real number that is stored internally as a signed 64-bit. Equivalent to the "double" data type in C++. Typically used for measurements or for decimal or fractional values.
- **String** (alphanumeric data, up to 1024 characters): Character string up to 1024 characters that holds letters, numbers, or special characters. Supports both ASCII and UNICODE characters. Examples: Recipe product X123, 01/01/90, *** On ***.

You can also assign a new tag to a class that you have previously created.
You can find these tag types (and their respective icons) in the Global tab of the Project Explorer.

See also: Understanding Tag Properties and Parameters help file topic

Using Array Tags

POV tags can consist of a single value or an array of values.

NOTE: *The maximum array size is 16384 as long as it does not exceed the maximum number of tags supported by the license (Product Type) selected for the project. Each array position (including the position 0) counts as one tag for licensing restrictions, because each position has an independent value.*

An array tag is a set of tags with the same name, which is identified by indexes (a matrix of n lines and 1 column). The maximum array size depends on the product specification. You can use the following syntax to access an array tag: *ArrayTagName[ArrayIndex]*
For example: tank[0], tank[1], tank[2], and tank[500].

WARNING: You must specify a maximum index for each array tag in the size column of any datasheet. You can specify n to indicate the array tag has positions from 0 to n. For example, if the size of TagA is 3, the tag elements could be TagA[0], TagA[1], TagA[2], and TagA[3].

NOTE: *When using the PAC3K Driver, the index minimum is 1.*

Use the array tag whenever possible because it optimizes memory use and simplifies the configuration task. For example, if you want a display to monitor each tank, you could use array tags to configure a single display containing tags linked to any tank. For example (use the tk tag as an index containing the number of the tank): **pressure[tk]**, **temperature[tk]**, and **temperature[tk+1]**.
An array index can be a tag, a numeric value, or an expression with the arithmetic operator "+".

NOTE: *When you refer to an array with an index using the + arithmetic operation, you must use the following syntax: ArrayTagName[NumValue1+NumValue2]*
Where NumValue1 and NumValue2 can be an integer tag or a numerical constant. For example: temperature[tk+2] or temperature[tk+6].

Using array tags in any POV task can save a signification amount of project development time. For example, if you needed tag points related to the temperature of four tanks. The conventional configuration method is the following:

- **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, as follows:

- **temperature[j]:** high temperature on tank {j}

NOTE: *When you create a four-position array tag, the system creates five positions (from 0 to 4). For example: tag_example[15] //start position=0, end position=15*
Therefore, the tag_example[15] array has 16 elements.

When using another tag to reference the index of an array, if the value of the tag is outside the size of the array, then the following results are given:

- If IndexTag is greater than the size of the array, then MyArray[IndexTag] will point to the end position of the array; and
- If IndexTag is less than 0, then MyArray[IndexTag] will point to the start position of the array.

Array Tags

An array tag consists of a set of tags that all have the same name, but use unique array indexes (a matrix of n lines and one column) to differentiate between each tag. An array index can be a fixed value, another tag or an expression. Maximum array sizes are determined by product specifications.

You can use array tags to:

- Simplify configurations
- Enable multiplexing in screens, recipes, and communication interfaces
- Save development time during tag declaration

You specify array tags in one of two formats:

- For a simple array tag, type:

  ArrayTagName[ArrayIndex]

- For a complex array tag (where the array index is an expression consisting of a tag and an arithmetic operation), type:

  ArrayTagName[ArrayIndex+c]

Where:

- ArrayTagName is the tag name;
- [ArrayIndex] is the unique index (fixed value or another tag);
- "+" is an arithmetic operation; and
- "c" is a numerical constant.

NOTE: *You must specify a maximum index for each array tag by typing a value (n) in the Array Size column of an Project Tags datasheet or in the Array Size field on a New Tag dialog. (See "Creating project database Tags" in help file).When you create an n-position array tag, POV actually creates n+1 positions (from 0 to n). For example, if you specify ArrayTag[15], the array will have 16 elements, where 0 is the start position and 15 is the end position.*
You must not use spaces in an array tag.
When POV reads a tag it begins with the first character and continues until it finds the first space or null character. Consequently, the system does not recognize any characters following the space as part of the array tag. For example, if you type a[second + 1], POV regards a[second as the tag and considers it invalid because POV does not find (recognize) the closing bracket. However, if you type a[second +1], this is a valid array tag.

You can specify an array tag wherever you would use a variable name. Also, because array tags greatly simplify configuration tasks and can save development time, we suggest using them whenever possible.
For example, suppose you want to monitor 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

You can use array tags to simplify this task as follows (where [n] represents the tank number):

- **temperature[n]** — high temperature on tank [n]

The following table contains some additional examples of an array tag:

Array Tag Examples

Array Tag Example	Description
Tank[1], Tank[2], Tank[500]	Simple arrays, where the array indexes (1, 2, and 500) are numerical constants. For example, tank numbers.
Tank[tk]	A simple array, where the array index (tk) is a tag. For example, a tag representing the tank number.
Tank[tk+1]	A complex array, where the array index (tk+1) is an expression. For example, the value of tk (tank number) plus 1.

NOTE: *When using another tag to reference the index of an array, if the value of the tag is outside the size of the array, then the following results are given:*

- *If IndexTag is greater than the size of the array, then MyArray[IndexTag] will point to the end position of the array; and*
- *If IndexTag is less than 0, then MyArray[IndexTag] will point to the start position of the array (i.e., MyArray[0]).*

About Indirect Tags

POV 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 for an indirect tag is straightforward: @IndirectTagName.
For example, assume that a tag named X holds a "TEMP" string. Reading and/or writing to @X provides access to the value of the TEMP variable.

NOTE: *Any tag created as a string-type tag is potentially an indirect tag (pointer).*

To refer to a class-type tag, you can declare a string-type tag that points to a class tag. For example:

Class	TANK with members Level
Tag	TK of the class TANK
Tag	XCLASS of the String type

To access the TK.Level value, you must store the " TK.Level" value within the XCLASS tag and use the syntax, @XCLASS. You can also refer to a member of a class-type tag directly; identifying a class-type that points to a class member. For example:

Class	TANK with members Level
Tag	TK of the class TANK
Tag	XCLASS of the class TANK

To access the TK.Level value, you must store the "TK" value within the XCLASS tag and use the syntax, @XCLASS.Level.
When creating tags for indirect use, place an X in the tag column rather than creating them as strings. For the type, write the type of tag for which you are creating a reference. Follow the XCLASS example: @Z Integer, @X Class:TANK.

Indirect Tags

Indirect tags "point" to other database tags (including class-type tags). Using indirect tags can save development time because they keep you from having to create duplicate tags (and the logic built into them).
You create an indirect tag from any string-type tag simply by typing the @ symbol in front of the tag name @TagName.

- To reference a simple tag, assume the strX tag (a string tag) holds the value "Tank", which is the name of another tag, then reading from or writing to @strX provides access to the value of the Tank tag.
- To reference a class-type tag and member, you simply create a string tag that points to the class tag and the member. For example, if a tag strX (a string tag) holds the value "Tank.Level", which is the name of the class tag, then reading from or writing to @strX provides access to the value of the

Tank.Level member.

- You can also point directly to a class-type tag member; by identifying a class-type that points to a class member. For example: to access the Tank.Level member of the class, you must store the "Tank" value within the strX tag and use the syntax, @strX.Level.