In this paper, we will discuss all the key concepts of tags, annotations and symbols, with special emphasis on customization to meet the needs of end users. Learn how to customize AutoCAD P&ID to create and use custom symbols, and define and utilize custom tag formats that inherit properties from different sources. Learn about property acquisition to propagate property values from one component to another and define new rules for acquisition. Learn about the Join Type functionality and how to set it for different symbols, including inline Equipment behavior to achieve desired workflow.
L. Special Behavior for Line Segment Tag Uniqueness ................................................................. 57
V. Annotation Styles .......................................................................................................................... 57
A. Simple Annotations ....................................................................................................................... 57
B. Compound Annotations .................................................................................................................. 58
C. Creating a New Annotation Style .................................................................................................. 61
D. Assigning Format Strings to Attribute Definitions ...................................................................... 64
   1. Using Expressions in Annotation Format Strings .................................................................. 72
   2. Using Annotations to Edit Underlying Data ............................................................................. 75
   3. Special Behaviors --Reducer Annotation Styles ..................................................................... 76
E. Principles of Dynamic Sizing ....................................................................................................... 77
   1. A brief overview of Dynamic Block Principles ................................................................. 77
   2. Linear Parameters in Annotation Block Definitions ........................................................... 78
   3. TotalX and TotalY .................................................................................................................. 83
VI. Custom Symbols .......................................................................................................................... 84
A. Creating Custom Symbols ............................................................................................................. 84
   1. Block Definition Management .................................................................................................. 84
   2. Creating a new Graphical Style with a new Symbol ........................................................... 85
   3. Graphical Style Properties ...................................................................................................... 90
   4. Manually switching between Graphical Styles ..................................................................... 91
   5. Associating Graphical Styles with a Property ........................................................................ 91
   6. Using Multiple Graphical Style for an Asset that Spans Drawings .................................... 94
   7. A note about Attachment Points .............................................................................................. 95
B. Dynamic Assets ............................................................................................................................. 96
   1. Special Behaviors of Off Page Connectors .......................................................................... 97
   2. Special Behaviors of Instruments ......................................................................................... 99
VII. Property Propagation/Acquisition .............................................................................................. 99
VIII. Appendix ................................................................................................................................... 102
A. Format string syntax .................................................................................................................... 102
B. APIs ............................................................................................................................................. 104
I. Introduction

AutoCAD® P&ID is a design software used to create, modify, and manage piping and instrumentation diagrams. Built on the latest AutoCAD® platform, AutoCAD P&ID is easy to use and familiar to designers and process plant engineers. Made specifically for P&ID designers and drafters, AutoCAD P&ID enables faster and more efficient creation, manipulation, and revisions to P&ID drawings.

This course will cover key concepts in P&ID diagrams — tags, annotations, and symbols — with special emphasis on customization to meet the needs of end users and their organizations. Topics covered include how to create and use custom symbols; how to define and utilize custom tag formats that utilize properties from different sources and use alpha-numeric expressions in formatting. There are examples of how annotations can be formatted and placed inside custom blocks which are dynamically resized when the size of the text changes. At the end of the course, the reader should feel confident about customizing P&ID for their own usage.

These things will be covered in this document:

- Tagging vs. Annotating: Key Concepts
- Creating custom Tag Formats: Properties and Expressions
- Creating custom Annotation Styles: Block Definitions and Evaluatable Strings
- Principles of Dynamic Sizing
- Creating custom symbols for Assets

II. Project: Default vs. Template

A. What is a Default Project?

- A preset project is used when a new project has not been specified by the user
- Default settings are set to what the majority of people would choose. There is often no reason to change them.

B. What is a Template Project?

- A customized project value used when a setting has been specified by the Administrator (user).
- A copy of the default, new, existing, Sample or finished Project
- A Template Project contains all defined settings, folders and required for a client(s).

For many companies, the default standard Project may not be sufficient. It is a good base for starting a project, but there are always specific company's palettes, symbols, environments and settings that require customization. A Project contains everything you need to create process and instrument diagrams. Items such as symbols, annotations, and templates as well as the location to store your settings and drawings are part of your project.
See default AutoCAD 2015 P&ID folder below.

Examples of the default Project (left) and the example Template Project (right)
Below is a Client specific Template folder structure

![Folder Structure Image]

When creating a new project using the template to create it, the following procedure will be crucial.
Start the Project creator Wizard, fill in the information and check “copy settings from existing project” box, then click on the button and find the template project to use.

See example below:

Left Click “Next”
Even though you chose a define template, you still have the option to change some settings during Project creation such as file paths.

Left Click “Next”

Make sure to specify the Directory where your P&ID drawings will be stored. Left click “Next” and specify the Plant 3D models DWG directory.

Left Click “Next”, and choose the SQLite local database.
Now you are at the end of the Project creation. You can now check mark on the “Edit additional project settings after creating project” (recommended) or click “Finish”.
When finished, open the Project Setup and look at the similarities. Remember to click on the “User defined reports directory” button to define appropriate location.
When finished, the new Project created using the client define template file creation should look like the example below:

![Example of a Project created using AutoCAD P&ID]

So the question remains.....

- Why is a Template better than the original?

Think about the time it would take to specify all the page size/layout properties, Toolbars, Blocks, Layers, Borders, apply fonts, ensure all is lined up and matching to your company's standards.

Project Templates can help ensure a consistent look and feel.

Default projects are a great start, but most developers and administrators will want to know how to create their own templates. AutoCAD 2010 P&ID makes it extremely easy to create your own templates. In addition, once they are created, you have the ability to modify them.

A Master template is the beginning of a new, updated, and better Client specific Master project Template.

### III. Tags vs. Annotations

The word “Tag” and the word “Annotation” have a variety of usages in ordinary English, in plant engineering and design domain, in various AutoCAD features, and in the AutoCAD P&ID product. The following discussion is specific to the Tagging and Annotation features as built for and shipped with the AutoCAD P&ID product, and is meant to prevent confusion in the remainder of this course. In particular, prior experience has shown that too many overloaded meanings for the word “tag” leads to confusion.
A. What is a Tag in AutoCAD P&ID?

- A Tag is a unique identifier for an asset or line segment
- A Tag is DATA
- A Tag lives in the database (tracked as a property)
- An asset or line segment can have only one Tag
- A Tag is built up of sub parts as defined by the Tag Format

B. What is an Annotation in AutoCAD P&ID?

- An Annotation is text (and optional geometry) in the drawing which presents information about the asset or line segment.
- An Annotation is graphical.
- An Annotation lives in the drawing, but is linked to the database.
- An Annotation need not be unique
- An asset or line segment may have multiple Annotations
- The text shown by an Annotation is determined by the Format String as defined in the Annotation Style’s block definition.
- The text shown by the Annotation may include the asset or line segment’s Tag property, but it is not “the” Tag. It is still an annotation.

C. What do we mean by “tagging” in AutoCAD P&ID?

- “Tagging” means assigning the values for the unique identifier of a given asset or line segment into the Data Cache
- “Tagging” means creating and storing data.
- “Tagging” is done via the Assign Tag dialog.
- Placing an annotation in the drawing that shows the tag property is NOT “Tagging”, it is “Annotating”.

IV. Tag Formats

Tag Formats are defined at a given level in the P&ID Class Definitions hierarchy, and once defined, are available to be used by all P&ID classes below that node in the Class Definitions tree. The subparts consist of Properties and/or Expressions, and may be Autogenerated. The subparts are concatenated together into a single value, and this completed tag value is validated for uniqueness and stored as a tag property in the Data Cache. We’ll examine each of these concepts, and discuss a few pitfalls that should be avoided.
A. Defining a Tag Format for a family of P&ID Class Definitions

Tag Formats are accessed from the Project Settings dialog, under the P&ID Class Definitions portion of the tree.

Pressing the “New…” button raises the Tag Format Setup dialog.

Subparts can be added and removed using the spin control next to “Number of Subparts.” A Tag Format consists of maximum 25 subparts.

- Each subpart consists of a Field and an optional Delimiter.
- The Field is built up from Properties and/or an Expression.

Note that this Tag Format is being defined for the class “HandValves”, because the Hand Valves node was selected in the P&ID Class Definitions tree when we pressed “New…”
This Tag Format will be available to all child P&ID classes defined at a more deeply nested level in the P&ID Class Definitions hierarchy; in other words, this Tag Format will be available for all Hand Valves.

**B. Properties**

A Tag Format may specify properties of the object to be tagged, properties of the Drawing, or properties of the Project. The first three buttons next to each sub part provide a means to select a property of each type for the subpart, and are mutually exclusive. We will examine each in turn.

*Drawing and Project properties are read from the Drawing and Project respectively, and will become part of the overall tag property, but assignments made to these properties via the tagging operation will NOT be stored back to the Drawing and/or Project. These properties are used for initialization only.*

**1. P&ID Class Definitions -Properties**

Pressing the leftmost button raises the Select Class Property dialog.

In most situations it is necessary to specify a property that actually exists on the class for which the Tag Format is being designed. Therefore, this example navigates to the Hand Valves node in the Class tree, and selects one of the properties, Valve Code, defined for Hand Valves.
The “Use Target Object’s Property” checkbox is also selected. Pressing OK yields the following result for the subpart in the Tag Format Setup.

![Tag Format Setup](image)

As can be seen, the Property has been specified as TargetObject.ValveCode. Also, notice that the graphics for the first button now appear raised. This indicates that a class property is in use.

If the “Use Target Object’s Property” checkbox were deselected, the result would have looked like this:

![Tag Format Setup](image)

Notice that the Property is now defined using the actual class name: HandValves.ValveCode.

TargetObject indicates to the program that the requested property is expected to be defined for the P&ID class for which this Tag Format is being defined. It is a shortcut in the nomenclature, and if you are careful in selecting the property, is generally appropriate to use.

Any property available at a given level in the hierarchy is always available at the child level. That is, since “ValveCode” is a property of HandValves, you can be certain that it is a property of Gate Valve, Ball Valve, etc.

There is an exception to the requirement to specify a property defined on the P&ID class being targeted.
In some special situations, you may want to include a property from a P&ID class directly related to the item being tagged, as part of the tag value. For example, the Tag Format for PipeLines specifies a property from the PipelineGroup as two of the subparts.

![Image of Tag Format Setup dialog box]

To utilize a class property from a class other than that for which the Tag Format is defined, the following must be true:

- The property must be from a class that shares a direct relationship to the target class.
- The actual class must be specified -- the TargetObject nomenclature cannot be used.
2. Drawing Properties

Pressing the second button beside the subpart raises the Select Drawing Property dialog.

A drawing property may be specified as one subpart of a Tag Format. Selecting the properties above yields the following in the subpart.

The second button now shows the raised graphics, indicating that a Drawing property is selected, and the Drawing property is specified in the Field by Category and Property.
Users may add custom Drawing categories and properties in the Project Setup dialog.

Any custom Drawing properties added here will be available to select from the Select Drawing Property dialog.
3. Project Properties

Pressing the third button next to the subpart raises the Select Project Property dialog.

Any project property may be selected as one subpart of the Tag Format. Selecting the properties above yields the following subpart:

The third button now shows the raised graphics, indicating that a Project property is selected, and the Project property is specified in the Field by Category and Property.
Users may add custom Project categories and properties in the Project Setup dialog.

![Project Setup dialog](image)

Any custom Project properties added here will be available to select from the Select Project Property dialog.

**C. Expressions**

1. **Expression Syntax**

Expressions define a simple syntax for character matching.

- The character ‘A’ specifies an upper case alphabetic character.
- The character ‘a’ specifies any alphabetic character.
- The character ‘N’ specifies a digit.
- The character ‘*’ specifies that any number of the previous type of character (A, a, or N) may follow.

When expressions are used in conjunction with a property, they enforce the type of input that is accepted when the property value is being assigned, via the Assign Tag dialog. When an expression is used independently, not in conjunction with a property, it is called a “pure” expression, and also enforces the type of input.
Enforcement is “weak”, meaning that mismatched input will result in user feedback (red text) in the Assign Tag dialog, and an attempt to correct the input automatically via padding, truncation, or uppercasing, if possible. However mismatched input that cannot be corrected is still accepted.

The upper two sections of the Define Expression dialog illustrate this syntax in action.

The Expression portion of the dialog provides a means to define the most common expressions.

The Result portion of the dialog shows what the expression looks like as it is constructed, and also allows direct modification of the expression, or disabling of the expression.

**a) Example 1: Any alphabetic characters, any length**

![Define Expression dialog](image)

In the example above, the Expression specifies Text Characters. It does not specify uppercase or a length. Thus the expression built in the Result box is: a*

- a --requires any upper or lower case character
- * after the a --means any number of upper or lower case characters.
b) Example 2: Upper case characters, any length

Here, checking the “Uppercase” box in the Expression area of the dialog has changed the Result to: A*
  •  A --requires an upper case character
  •  * after the A --means any number of upper case characters.
c) Example 3: Upper case characters, fixed length

In this example, selecting the “Fixed Length” checkbox and incrementing the spin control to 3 has changed the result to: AAA

- Each A requires an upper case character.
- The asterisk (*) is no longer shown.

This expression requires exactly 3 upper case character as input. If too few characters are input, the missing characters will be represented by a ‘?’ character. If too many characters are input, the extraneous characters will be truncated.
d) Example 4: Numbers, any length

In this example, selecting the Numbers radio button in the Expression has changed the result to:

\( N^* \)

- \( N \) - requires a digit
- \( * \) after the \( N \) -- allows as many digits as desired.
e) Example 5: Numbers, fixed length

In this example, selecting the “Fixed Length” checkbox and incrementing the spin control to 3 has changed the result to: NNN

- Each N requires a digit. If too few digits are entered, the value will be padded with leading zeros.
- The asterisk (*) is no longer shown.

Note that for digits, the Fixed Length is actually interpreted as a **minimum** length, rather than an exact length. This is special behavior in order to avoid truncating numeric values. In our syntax the following are equivalent:

NNN  NNN*
f) Example 6: No Expression

In this example, the “No Expression” checkbox has been selected in the “Result” section. This result box now shows “Free Style”. When this option of the Expression dialog is being used in conjunction with a Property, it indicates that the user should be allowed to enter any value they like for the property.

The same thing can be accomplished by not opting to use an expression at all (not pressing the 4th button) when creating a subpart involving a property. When this option is used as a pure expression, the user is allowed to enter anything they like for the subpart.
g) Example 7: Manually Defined Expression

In this example, the “Manually Define Expression” checkbox is selected, and the edit box showing the expression is no longer disabled.

This option is required to construct more complex expressions involving combinations of characters and numerals.

The expression above, AANNa*, would require the user to input:

- Two upper case characters followed by Two digits followed by Any amount (including zero) of text characters.

  AB10 AB10s MM12qwE

  would all be acceptable input.

The following input would fail:

- ab10 --initial characters should be upper case AB10s10 --final characters must be alphabetic, this ends with digits
- AB5ss --only one digit, exactly two are required AB345xy --three digits, exactly two are required.
2. Using Expressions

The first three buttons, specifying the property source, are mutually exclusive. The fourth button, which raises the Define Expression dialog, may be used either alone, or in conjunction with one of the Property buttons.

When an expression is used in conjunction with one of the Property buttons, it acts to enforce a specific type of input for that Property.

If the expression from Example 3, requiring three uppercase alphabetic characters, were used in conjunction with the ValveCode property, the resulting subpart would appear as follows:

Notice that both the Class Property and Expression buttons show the raised graphics, indicating that both are in use. In addition, notice that the expression is being shown in square brackets after the property.

When an expression is used alone, it acts to require the user to input a pure alphabetic or numeric value which is not associated with any property.
If the expression from Example 5, requiring 3 numbers, were used by itself, the resulting subpart would appear as follows: Notice that the Expression button shows the raised graphics, indicating an expression is in use, but none of the Property buttons show the raised graphics. In addition, the Field shows ONLY the expression, NNN, and does not show any property. This is a “pure” expression.

**D. Auto-Generation**

Some values can be Auto-generated. There are two approaches to AutoGeneration.

1. **Simple Auto-generation**

   Simple Auto-generation has the following requirements:
   - Every subpart but one is a property
   - Every property subpart has a known value
   - The only non-property subpart is a pure numeric expression

   If these requirements are met, it will calculate a numeric value that results in a unique Tag.

   The Tag Format for Nozzles provides an example of a Tag Format that has been authored to support simple Autogeneration:
2. Property based Auto-generation

Property based Auto-generated values can be per-Drawing or per-Project.

Example 1: Assume that the Tag for all pumps in a drawing must reflect which drawing contains the pump. For example, all pumps in drawing 1 should contain a numeric subpart starting from 100, all pumps in drawing 2 should contain a numeric subparts starting from 200, etc.

Example 2: Assume that all Tanks in an entire project should be numbered incrementally across the entire project.

Both of these examples can be supported with Property based Autogeneration. There are two fundamental requirements to use property-based autogeneration:

1) You must define and initialize the autogen properties
2) You must indicate in the tag format that you are using auto-generation, and which property is the auto-generation basis

a) Auto-generation based on Drawing Properties

You can access the drawing Autogen properties by right clicking on the drawing in the Project Workspace, and selecting “Drawing Autogen Properties.”
Selecting Drawing Autogen Properties will raise the Drawing Autogen Properties dialog.

In this example, the “Last Used Value” for Last Line Number is initialized to 190, and the Increment is initialized to 10. The auto-generation code looks at the “Last Used Value” and adds the increment amount, so the first auto-generated value will be 200, not 190. Each time an auto-generated value is used, the program will update the value stored in “Last Used Value.”

Note that you MUST enter values in “Last Used Value” and “Increment”, otherwise the code cannot calculate an auto-generated value. In this example, any attempt to use the “Last Valve Number” will not succeed. For Drawing Auto-generation Properties, you must do this for EACH drawing.
To use these values, you must select the Expression button on the Tag Format Setup dialog, to raise the Define Expression dialog.
In this example, we intend to auto-generate a numeric value based on a Drawing Field. To do so, we select the Drawing Field radio button, and press the Drawing Field icon. Pressing this icon raises the “Select Drawing Autogen Property” dialog. This dialog presents all the defined Drawing Autogen Properties already defined, and allows you to select one. Here we can select “Last Line Number”, which we initialized in the steps above.

In addition, new Drawing Autogen Properties can be added by pressing the “New...” button. This button allows the creation of a new Drawing Autogen Property, but it is necessary to return to the “Drawing Autogen Properties” dialog to assign their initial values.
After selecting “Last Line Number”, the Define Expression dialog shows the selection.
The default view of “Define Tag Format” dialog doesn’t provide much feedback to indicate that an autogen property is in use. It shows the expression N*, but it does not indicate that autogeneration is in effect or upon what value it is based. Pressing the “More>>” button does show the additional information.

b) Auto-generation based on Project Properties

Setting up Project Auto-generation Properties is similar to setting up Drawing Auto-generation Properties, except that there is no project-level UI giving direct access.
Selecting a Project Autogen property starts with the “Define Expression Dialog” raised from the “Tag Format Setup” dialog, just as above, except that the “Project Field” radio button must be selected.

Pressing the Project Field icon raises the “Select Project Autogeneration Property” dialog. This dialog differs slightly from the “Select Drawing Autogeneration Property” dialog in that the property AND the initial values are shown in the same dialog. You may also create new Project Autogeneration Properties from this dialog by pressing the “New...” button.
Since Project Autogen properties exist once for the entire project, initialization is done from this centralized location, in contrast to the way per-drawing initialization for each drawing in the project workspace.
As with the Drawing Autogen properties, selecting the “Last Pump Value” auto-generation property results in that property being shown in the “Define Expression” dialog, but not in the “Tag Format Setup” dialog, unless you press the “More>>” button.

![Define Expression dialog](image)

**E. Delimiters**

Each subpart of the “Tag Format Setup” dialog has space for a delimiter. Most commonly, a delimiter is a single character, such as a dash or colon, but there are no restrictions on the type or number of characters that may be used as a delimiter. The delimiter is placed in between each subpart when the tag is built up and stored.

It is also allowable to leave the delimiter blank, however there can be situations in which this is inadvisable. Several other features in AutoCAD P&ID may need to parse the completed tag value back down to its constituent parts, and it is possible to create Tag Formats where this is impossible. This will be covered further in the “Pitfalls and Workarounds” section.
F. Assigning the Tag to an Asset

To complete the exercise above, the following Tag Format is being defined at the HandValves level.

Notice it utilizes most of the features previously discussed.

- The first subpart is a property of the object, and requires upper case character input.
- The second subpart is a pure numeric expression based on a Drawing Autogen Property.
- The third subpart is a Drawing property.
- The fourth subpart is a Project Property
- The last subpart is a value from a PipeLine. Since HandValves are inline instruments that sit upon a PipeLine, it is admissible (although perhaps not advisable) to use a property this way.

Recall that the “Drawing Autogen Property” called “Last Line Number”, being used in the second subpart, was initialized to a Last Used Value of 190 and an increment value of 10 in the section describing Drawing Autogen Properties. This Tag Format also relies on a Drawing property and a Project property, which will be initialized as follows.
Right clicking on the drawing icon in the Project workspace, and selecting the “Properties...” menu item raises the Drawing Properties dialog. In the image below, the value 51 is being assigned to the Drawing’s Area property.
Project properties are initialized in the Project Settings dialog. Below, the value 18238 is being assigned to the Project Number.
After a Tag Format has been set up, it must be selected as the Tag Format to use for a given P&ID class.

Here we see that two Tag Formats are now defined at the Hand Valves level, and that the default Tag Format for Hand Valves is “Hand Valve Tag [Code-Number].”

Since “Big Example Tag Format” was defined at the Hand Valves level, it is available to be used by any Hand Valve. In the following example, the new Tag Format is assigned as the Tag Format for Ball Valves, only, by selecting it from the picklist for the TagFormatName property.
The following drawing contains two Ball Valves, one of which is on a PipeLine. “Assign Tag” is an option that is available on the right click menu of the Ball Valve.

Selecting “Assign Tag” raises the Assign Tag dialog.

As can be seen here, the second property has initialized based on the Drawing Autogen Property. The increment arrow only appears when focus is placed in that subpart. Each press of the increment arrow will increment the value by the increment amount specified in the autogen property.

In addition, the Project and Drawing based properties have been initialized from the values that are currently assigned to those properties.
We defined the ValveCode as requiring upper case characters, so entering lower case characters appears as an error; the text is presented in red.

Notice that the preview of the completed Tag in the top box shows the ValveCode capitalized. If possible, the Assign Tag dialog tries to correct mismatched input automatically, based on the expression. If correction is not possible because the mismatch is too egregious (e.g. entering digits when characters are specified) the mismatched input will still be accepted. This “weak” enforcement anticipates a need to sometimes enter temporary information that may not reflect the final data in its final format.

Automatic correction, depending on the expression, can include padding short numeric values with leading zeros, padding short alphabetic input with trailing ‘?’ characters, truncating alphabetic input that is too long, and uppercasing alphabetic input.
Retyping the Valve Code in upper case shows that the input is acceptable by displaying the input with black letters.
When we select “Assign Tag” on the second Ball Valve, the one which is not on a pipeline, again, some values are pre-populated in the dialog, as they are available from the Drawing and Project, respectively. Note that the autogenerated value has also already been populated with the next available value.
The image below shows assignments being made to the remaining subparts. Since this Ball Valve is not actually associated with any line, filling in the Paint Code value doesn't really have any meaning. We'll explore that more in the next section.

For one final example, let's examine a Tag Format that utilizes the simple variation of autogeneration.

The Tag Format shown below has one property subpart that will always contain a known value, and one subpart which is a pure numeric expression.
When assigning this tag, the numeric subpart shows up with the increment arrow.
Pressing the increment arrow produces the first numeric value that would result in a unique tag value.

This type of auto-generation is designed to find the first available unique tag. Repeatedly pressing the increment arrow will not change the value of the numeric expression. Every time you press the button, the result returned will be the first available unique value, which will be the same. It is helpful to be aware of this, otherwise it might seem that the increment button is not functioning.
G. Tagging Prompt Behaviors and the Graphical Style

Each P&ID Class Definition specifies one or more Graphical Styles. (These will be covered in more depth in the section on custom symbols.) In addition to the symbolic properties defined by the Graphical Style, several behavioral properties are also defined. One of these involves the prompting behavior for Tags for that specific P&ID class.

While most aspects of the Tag are defined using the Tag Format as described so far, if the Tagging Prompt setting in a P&ID class’s Graphical Style is set to “Not a Tagged Component”, then the “Assign Tag” option will not be available in the right click menu for assets of that type.

The value “Prompt for tag during component creation” will cause the “Assign Tag” dialog to be raised each time an asset of this type is placed in the drawing.
The value “Automatically assign an auto-generated tag” will not raise the “Assign Tag” dialog, but will attempt to fill in the Tag property by filling in all known and autogeneratable subparts. Depending upon how the Tag Format is defined, this may or may not produce a complete Tag.

H. How Tags are Stored

Once the Assign button on the Tag Assign dialog has been pressed, there are two types of storage which occur. For each sub part which refers to a class property, the individual value is assigned back to that property, as seen in the previous section.

In addition, all the subparts and delimiters are concatenated together and stored as a single string in the Tag PROPERTY of the asset. This is shown in the data grid image below.

![Data Grid Image]

Our second subpart was a pure expression, assigned 200 for the first Ball Valve and 210 for the second Ball Valve. Since these values are not based on any property, the ONLY place where they are stored is as part of this fully combined string, in the Tag property.

In order to keep the Tag property synchronized with the underlying properties, many activities will cause the Tag to be rebuilt from its constituent subparts. These include:

- Changing any class property of the tagged asset
- Changing any class property of a related item that is used by the tagged asset

Changes to Drawing and Project properties used in the Tag Format do not automatically rebuild the Tag. Drawing and Project values, once assigned, are retained in the completed tag, even if those properties are later changed at
the Drawing or Project level. In other words, when the tag property is rebuilt from subparts, only changes to class-property subparts are incorporated.

Examples:

In this example, I modify the second Ball Valve by assigning GE as the manufacturer. Notice that the PaintCode subpart value changes to a ‘?’. Since this Ball Valve is not situated on a PipeLine, and the Tag Format specified PipeLine.PaintCode, this value is actually blank. In point of fact, it shouldn’t have shown the LTX value even after the initial assignment, but the fully formed string from the Tag Assign dialog is always assigned at first.
In the next example, I change the PaintCode property of the PipeLine upon which the first Ball Valve sits to OIL. Notice that the Tag property of that Ball Valve has updated to reflect the change.

You may notice that the PaintCode property of the line doesn’t show up in the BallValve’s Tag immediately. An operation must occur that triggers the rebuilding of the Tag, such as changing the property in another cell. *(This delay will be addressed in the future.)*

**I. Notes on Uniqueness**

The tag property, in its entirety, is compared for uniqueness with all other tags in the project.

Duplicates are not allowed in the same drawing. Duplicates in different drawings of a single project are assumed to be different parts of the same item, for example, a line which spans drawings using an Off Page Connector could have the same tag in each drawing. It would be represented by a single row in the data grid.

Note that there is no allowance for items of different types to have the same tag. In other words, if a pump were tagged A-123, and a subsequent attempt was made to assign A-123 as the tag of a Tank, the Tank tag would be rejected as a duplicate.

For nozzles, the Tag value of the owning asset is considered along with the nozzle tag itself when determining uniqueness. In other words, two tanks, tagged TK-100 and TK-200 can each have a nozzle tagged N-1, and N-1 will not be treated as a duplicate. This is because the program considers all of TK-100N1 and TK-200N1 when comparing nozzle tag uniqueness.
J. Tag Behavior during Cloning

“Cloning” refers to any operation that places a copy of an AutoCAD entity into a drawing. Examples of cloning include the COPY, ARRAY, and MIRROR commands, Clipboard Copy/Paste (Ctrl-C/Ctrl-V), BLOCK, WBLOCK, INSERT and EXPLODE.

Since AutoCAD P&ID is a project-based product, copying a tagged asset within a single drawing or between drawings belonging to the same project would tend to create duplicate tags, which would be rejected.

To avoid this duplication and rejection, the product will blank out both the Tag property, and all the class property subparts which contribute to the tag. Blanking out the subparts is necessary because automatic tag regeneration would simply rebuild the duplicate tag if the subparts were left intact.

K. Pitfalls and Workarounds

There are a number of operations which will parse the tag back down into its constituent sub parts. One example is any operation that presents the sub parts to you for editing. For example, raising the Assign Tag dialog on an asset that is already tagged (for example to modify the Tag property) presents all the sub parts in each edit box. Another example is the operation which rebuilds the tag from its subparts whenever one of those subparts has been changed.

To do this, it has to parse the combined Tag property back down into its constituent parts. It has several ways of distinguishing between the subparts. One is to compare against the delimiters. Another is to re-retrieve the property from its original storage and compare the strings.

However, if pure expressions are placed without any delimiter, then it cannot determine where one sub part ended and the next begins.

If your Tag Format requires that there be subparts which are not delimited, you may need to add a new property to hold each subpart.

This is important as any refresh of the Tag could result in the loss of the pure expression portions of the Tag.
In the following example, the Tag Format is defined with a Property and a pure numeric expression, but no delimiter.

Here, a Tag based on this Tag Format is being assigned. The concatenated, un-delimited Tag can be seen in the Data Grid. “C3333”
Re-raising the Tag Assign dialog illustrates that the pure-expression subpart has not been successfully parsed.

In the example below, assigning a new property to one of the asset’s fields has caused the Tag to be recalculated. In doing so, the pure expression portion of the Tag has been lost.
If you must have un-delimited subparts, then those subparts should always be based on a property. Adding properties in AutoCAD P&ID is straightforward, and can be accomplished from the Project Settings dialog.

L. Special Behavior for Line Segment Tag Uniqueness

Line Segments are collected into Line Groups. This leads to special behavior with respect to how Line Segment tags are compared for uniqueness. In AutoCAD P&ID, it is actually the Line Group that is tested for uniqueness, not each individual Line Segment. In addition, it is only the “Line Number” property of the Line Group which is tested in the current release. A future release will expose a Tag property on the Line Group directly via the UI, so that users can add additional properties to the Line Group Tag.

V. Annotation Styles

Annotations are implemented as regular AutoCAD Block References and Text entities, with some additional behaviors that allow the actual textual values shown to be dynamically evaluated based on properties of the item being annotated. We call the item being annotated the “target”.

Annotations utilize the same mechanism as the Tag Formats, with a few minor differences, to specify the property that should be shown.

A. Simple Annotations

Simple Annotations are created by dragging and dropping a Data Grid cell onto the drawing pane. These annotations are not based on any annotation style, are created using AutoCAD text entities, and the text properties (font, size, etc.) are based on the current AutoCAD settings for text.
Behind the scenes, a simple annotation Text entity carries the Format String “TargetObject.Property,” where “Property” is the property specified by the Data Grid column from which the cell was selected.

B. Compound Annotations

Compound Annotations provide a great deal more flexibility and customizability than Simple Annotations. Creating a new Annotation Style has similarities to both Tag Format creation and creation of a Symbol for a P&ID object.

Annotation Styles are defined at a given class level in the P&ID Class Definition hierarchy. Once defined, that style can be used to annotate any target P&ID entity defined at or below that node in the hierarchy.

Annotation Styles are also associated with an AutoCAD Block Definition, which provides for their appearance and behavior.

Annotation Styles are manipulated from the Project Settings dialog. Pressing “Edit Annotation” raises the Symbol Settings dialog for Annotation Styles.
“Symbol Name” is the name of the Annotation Style. This is the name that appears in the submenu when the right-click/Annotate... context menu item is selected.

“Block” is the name of the block definition upon which this Annotation Style is based. The name is always the Symbol Name post-pended with “_block.” To select a different block definition, select the Block edit box, and press the more button.
Doing so will raise a three-step navigation dialog allowing you to choose a drawing, and a block within that drawing, and alter settings for the annotation to use for the Annotation Style. Once you have selected the block definition, the contents of that block definition will be cloned into your ProjSymbolStyle.dwg and renamed to SymbolName_block.

AutoCAD P&ID ships with a file called AnnotationTemplates.dwg. This drawing file contains a number of block definitions that may be useful as templates for a variety of standard annotation styles, and for which the more complex settings (the dynamic sizing settings) are already in place.

The “General Style Properties” are the standard AutoCAD entity properties, such as color, layer, and linetype that will be used when the annotation (an AutoCAD block reference) is created and inserted into the drawing.

The “Other Properties” area of the dialog offer settings that are specific to Annotations.

- “Symbol Scale” is a scale factor at which the annotation will be inserted into the drawing.
- “Use Target Properties” will cause the annotation to be inserted with the same entity properties (color, layer, linetype) as the target entity, rather than using the settings from “General Style Properties.”
- “Linked” controls whether the annotation moves along with the target entity.
- “Auto Insert” controls whether the annotation will be automatically inserted when the target entity is inserted into the drawing.

“Auto Insert?” has three possible states, worthy of further discussion.

“No Auto Insert” is self explanatory. Annotations based on this style will never be automatically inserted into the drawing.

The other two settings are primarily meaningful for the default Annotation Style for a given P&ID class, and take effect when placing an asset of that class into the drawing from the tool palette.

“Auto Insert With Prompt” --If an asset is placed in the drawing whose default Annotation Style specifies Auto Insert With Prompt, the user will be automatically prompted to select a position for the annotation.

“Auto Insert” --If an asset is placed in the drawing whose default Annotation Style specified Auto Insert, then the annotation will automatically be placed in the drawing at the positions specified by “Offset X” and “Offset Y” from the asset’s insertion point.

The Offset X and Offset Y settings are only enabled when the Auto Insert state is Auto Insert.

Note that the Auto Insert option, relying on the Offset X and Offset Y settings, does not work when the target is a pipe or signal line. The reason is that the software cannot determine a single insertion point from which to calculate the offset from a line, as it can do from an asset.

“Auto Insert” using an offset will also have an effect when placing an annotation via the right click “Annotate” menu. Normally, when using the Annotate... context menu, you would be prompted to select a position using the mouse. However, when annotating an asset using an annotation style which specifies “Auto Insert” with an X,Y offset, the annotation will be placed at that offset.
Use leader indicates whether a leader will be included with the annotation.

Orientation changes how the annotation is rotated.

**C. Creating a New Annotation Style**

In this example, a new Annotation Style at the Hand Valves level will be created by pressing the “Add Annotation” button, and then pressing the more button beside the Block property, as described above. In the navigation dialog, select “AnnotationTemplates.dwg” and press “Open.”
In the “Select Block” dialog, select “InfoTag2LineBox” as the block definition upon which we’ll base the new Annotation Style.

Note: See the section ‘Linear Parameters in Annotation Block Definitions’ for subtle improvements to the as-shipped “InfoTag2LineBox” block definition that you will need to make to get the best possible results when this annotation is placed.
Here, the annotation is given the name “Valve InfoTag” and the desired settings are made to the Style properties.

Press OK. This will copy the block definition into ProjectSymbolStyle.dwg and rename it to “ValveInfoTag_block”.

Every Annotation Style points to a unique block definition. Graphical Styles (“Symbols”) may share block definitions, but Annotation Styles do not, because the Attribute Definitions for each Annotation Style need to be set up so as to be unique to that Annotation Style.
D. Assigning Format Strings to Attribute Definitions

Pressing the “Edit Block“ button allows editing of the Block Definition upon which the Annotation Style is based. To set up for this section, select our new Annotation Style, “Valve InfoTag“ from the picklist and press “Edit Block...“
This operation will drop you into the AutoCAD block editor for the Annotation Style’s block definition.

The template contains two Attribute Definitions, named UNASSIGNED1 and UNASSIGNED2. It also contains some geometry; an underline beneath UNASSIGNED1, and a box around both Attribute Definitions. Finally, it contains 3 linear parameters, each associated with a pair of stretch actions. We’ll discuss the geometry and parameters in the next section. The block may have several duplicates directly on top of each other (4 duplicates should be erased for the line, each of the attributes and the rectangle).

To fix an action, right click on the action icon, select Action Selection Set, and then Modify Selection Set.

Notice that the action as a dotted window associated with it. Re-create the stretch window, and make sure the object that needs to stretch is selected.
Repeat the process for all of the actions.
These Attribute Definitions currently are not set up to display any particular properties of an Annotation’s target entity. Pressing the icon in the small toolbar will prompt you to select an Attribute Definition. Making that selection will then raise the Assign Annotation Format dialog. This dialog is extremely similar to the Assign Tag Format dialog, and lets you set up the Attribute Definition to display properties of the target entity, properties of the Drawing, or properties of the Project.

When this dialog is first raised for an UNASSIGNED Attribute Definition, the word “UNASSIGNED1” appears as the first delimiter. In fact, for Annotation Formats, the area between subparts needn’t be thought of as a delimiter at all. Any kind of free text is allowable. Remember to remove or change this delimiter value if you don’t want any additional text appearing when the annotation is placed in the drawing.

Using the same skills discussed in defining a Tag Format, we’ll assign the target object’s Tag property to the first Attribute Definition, and remove the word UNASSIGNED1 from the delimiter field.

After pressing OK, the value of the UNASSIGNED1 Attribute Definition has been changed to #(TargetObject.Tag). Notice that the linear parameter named UNASSIGNED1 has been edited in the same way. We’ll discuss why this is
important in the next section. You may have to execute BSAVE in order to see the linear parameter label change. This is an AutoCAD artifact.

Each Attribute Definition should be used to place a single line of text, but each line of text may present more than one property. We’ll define the second subpart to show two properties, one from the Valve, and one from the Line upon which the Valve sits. The following properties and delimiters yield the format string as shown below.
Just as with Tag Formats, it is allowable to specify a property from another P&ID class, so long as that class is directly related to the target object’s class. The second subpart illustrates using a property from a related class.

Notice that the properties to be retrieved are enclosed inside the field identifiers #(), while the delimiters appear as free text in between the fields.
It is also possible to have leading text. You can add leading text by one of two methods. Start off with an additional Field, and set the delimiter on the first blank item. The rest of the fields get filled out normally. You can also select the Attribute Definition in the AutoCAD properties palette, and pre-pend the free text you want. In this example, the free text “Model No.” is pre-pended.

Close and save the changes to the block, and OK out of the Project Settings dialog.
Now, let’s use this annotation. Create a Pipe Line and assign a Test Pressure of 250 using the Data Grid. Place a Gate Valve on that line and assign the tag value of GV-001, and a model number of 810.
Select the Gate Valve, right click and select Annotate... from the context menu. From the submenu, select the “Valve InfoTag” style that we have just defined.

![Gate Valve Diagram]

GV-001
Model No. 810 for line at 250 PSI

The Attribute Definition for #(TargetObject.Tag) collected the target object’s tag value. The second Attribute Definition has replaced #(TargetObject.ModelNumber) with 810 and #(PipeLines.TestPressure) with 250. The rest of the text, both the leading text and those placed using the delimiter boxes, appears unchanged.

1. Using Expressions in Annotation Format Strings

It is possible to use the Expressions portion of the Assign Annotation Format dialog to enforce formatting. Let’s define another, simple Annotation Style.

![Assign Annotation Format Dialog]

AutoCAD P&ID Customization
Using the same techniques as for create Tag Formats, expressions have been applied to both the subparts of this Annotation Format. The first subpart specifies 5 numerals, and the second subpart specifies that the property should be upper cased, any amount of characters.

The resulting format string appears as follows:

```plaintext
#(TargetObject.ModelNumber%NNNNN)#(TargetObject.Manufacturer%A*)
```

The main difference between using an expression in a Tag Format vs. an Annotation Format is that when the expression is used for Tagging (which is the act of creating and storing data), the expression is used to validate and enforce input. When the expression is used as part of Annotation (which reads and displays existing data) the expression is used to format output. The percent sign appears as a symbol after the property, but inside the format string field, to separate the property being read from the formatting being requested.

Our Gate Valve already has a model number of 810. We will add “Bosch” in mixed-case as the manufacturer, and place the annotation. Notice how the number has been pre-pended with zeros to make it 5 digits long, and the manufacturer has been upper cased.

It is possible to use the Autogeneration portion of the Define Expression dialog when creating an Annotation Format String so long as you do so in combination with a pure expression, and not with a property. Remember that Annotations READ data, they do not create data, so using an autogenerated expression as part of an Annotation will store the result ONLY in the annotation itself, and not in any underlying storage. AutoCAD P&ID will pre-evaluate any autogenerated expressions so that they do not continue to increment with every refresh of the Annotation. While possible, this usage is not recommended.
In the image above, notice the small dot above the Gate Valve.

The default Annotation Style for Hand Valves is “Valve Label”, which is defined as a single Attribute Definition showing #(TargetObject.Size). Because there is no size assigned to the Gate Valve, the annotation shows up as a dot. Whenever the annotation cannot retrieve a specific property, it shows that property as a dot. As soon as we assign a size value to the Gate Valve, the dot will update to show that size.

Any modification to a class property of the target or the class property of a related asset will cause the annotations to refresh automatically. The refresh button on the Data Grid will also refresh all the annotations in the drawing.

If an annotation becomes disconnected from its target entity, all the properties show the string “No Target”.

The most common way to cause such an occurrence is to copy an annotation by itself, and then paste it into a different drawing. To re-associate the annotation, select it, right-click and then choose Assign Annotation Target. Then choose the object the annotation should use to read properties.
The shortcut nomenclature TargetObject is particularly useful for Annotations, even more so than for Tag Formats. Many P&ID classes in the same family have the same properties. For example, all Hand Valves have the Size property.

AutoCAD P&ID provides a very simple method to substitute related P&ID objects; the substitution palette. Using this tool, one can easily convert a Gate Valve into a Ball Valve. Now consider if the Annotation for the Gate Valve was #(GateValve.Size). Upon substitution to a Ball Valve, your annotation would cease showing the size and would instead show the dot. The reason for this is that it is trying to retrieve a GateValve property, specifically, from a Ball Valve. It recognizes the class mismatch, and fails to return any data. However, by using #(TargetObject.Size), data will be retrieved for any target entity that has a Size property. Thus, the TargetObject nomenclature provides the best possible support for maintaining annotation integrity in conjunction with the substitution palette.

The main thing to remember about Annotations is that the Format Strings in an Annotation Style’s block definition are always used to retrieve and display existing data, never to assign data to storage.

2. Using Annotations to Edit Underlying Data

Once an annotation has been placed, double clicking on that annotation brings up a special editor.

Double clicking our Valve InfoTag annotation yields the following dialog.
Recall that the first line of this annotation style was based on the format string #(TargetObject.Tag), and that Tags are themselves built up of subparts as defined in the Tag Format.

The first line of this editor utilizes the Tag Format to break TargetObject.Tag back into its constituent parts and present those parts for editing.

The second line of this editor directly uses the Annotation Format String to present each part for modification.

Modifications made in this UI are stored in different locations. All PROPERTY changes to the target object’s properties are actually stored directly back to the underlying storage in the datacache. (Drawing and Project properties cannot be modified from this dialog.) The changes seen in the annotation are a result of the annotation refreshing after these values have been changed in their original location.

Modifications to the delimiters are stored in the Annotation itself, and affect ONLY the annotation being edited, not ALL annotations.

Although it would be odd to use pure expressions in an Annotation Format String, nothing prevents you from doing so. Be aware, however, that if you use pure expressions without any delimiters in an Annotation Format String, the program will have the same problem reparsing the annotation into its subparts as it has when parsing Tags, and you may get unexpected results in the Edit Annotation dialog.

3. Special Behaviors --Reducer Annotation Styles

AutoCAD P&ID ships with 4 standard annotation styles for Reducers:

- Fixed Reducer Style
- Large by Small Reducer Style
- Flow Based Reducer Style
- Adjacent Pipe Reducer Style

The first of these is based on the properties of the reducer itself. The Annotation Format String for Fixed Reducer Style is: #(TargetObject.Size1)x#(TargetObject.Size2).

The other three styles are “Pipe Based” reducer styles, and AutoCAD P&ID applies special rules when evaluating the format string, which is the same in all three styles: #(PipeLines.Size)x#(PipeLines.Size).

For these special Annotation Styles, even though the same property appears twice in the format string, the program will collect the sizes from the two separate pipelines attached on each side of the reducer. It will then calculate, based on the name of the reducer style, which size value should be shown first.

- Large by Small Reducer Style --Shows larger size value first always.
- Flow Based Reducer Style --Shows the size value of the incoming pipe first.
- Adjacent Pipe Reducer Style --Shows the size values on each side for the pipe on that side.

Users may create other pipe based reducer styles that show additional data. In order for it to recognize such styles and apply the special processing, the following must be true:

- The Annotation Style must be defined at the Reducers level in the class hierarchy
- The Annotation’s target must be a Reducer.
• The Annotation Style’s name must begin with:
  o Large By Small Reducer
  o Flow Based Reducer
  o Adjacent Pipe Reducer
• The Format string must contain exactly one delimiter
• The Format string must contain a property #(PipeLines.Property) on each side of the delimiter.
• The Format string may contain other properties, as long as the properties on each side of the delimiter are identical and in the same order.

It is not recommended to use the Edit Annotation dialog to modify the values of Pipe Based Reducer annotations, as this dialog cannot distinguish to which pipeline the new size value should be assigned.

E. Principles of Dynamic Sizing

In the Annotation examples above, several geometric items dynamically changed size to match the evaluated values of the attributes. When the “Valve InfoTag” Annotation Style was placed in the drawing, the underline under GV-100 matched the width of the text, and the box fit nicely around all of the evaluated text.

AutoCAD P&ID leverages the AutoCAD “Dynamic Block” feature in custom ways to enable this behavior. Dynamic block definitions are defined in such a way that each block reference (aka insert) of that block can be independently manipulated in ways that change its appearance, for specific sub portions of the block.

The AutoCAD Dynamic Block feature is extensive and will not be covered in depth here, but it is recommended that interested users spend some time studying the documentation and/or taking an AU course on this feature. Authoring a dynamic block has nearly as much power and potential complexity as writing computer code!

1. A brief overview of Dynamic Block Principles

Creating a Dynamic Block Definition in AutoCAD involves placing and defining several items in the block definition which are not geometry. These items are known as “parameters” and “actions”.

There are a variety of different types of parameters, each of which track some set of properties, for example:

• Linear parameters have a length
• Rotational parameters have a rotation.
• Point Parameters have a position.

Each parameter can be associated with one or several types of actions. Typical actions include:

• Stretch
• Move
• Rotate
• Flip

Each action is associated with an “action frame”, which indicates which geometry within the block definition participates in the action, and in what fashion. The available actions may be presented to the user as a special grip on the block insert, or the grips may be hidden from the user, depending upon whether the block author intends.
For each block insert, the properties tracked by the parameter may have different values. A single block definition with a single Linear Parameter may be inserted into an AutoCAD drawing multiple times, and the length of the linear parameter may be different in each block insert. Depending upon that length, actions associated with the linear parameter will yield different results.

The list above is by no means exhaustive. The AutoCAD P&ID Annotation feature extends the behavior of Linear Parameters, so the remainder of the discussion will focus on those.

2. Linear Parameters in Annotation Block Definitions

In truth, AutoCAD P&ID makes only a slight tweak to the behavior of Linear Parameters. It recognizes when an Attribute Definition carries an annotation Format String. If the block definition also carries a linear parameter whose label is identical to the Attribute Definition, then in each block reference, after the format string has been evaluated, it will set the linear parameter to be the same width as the evaluated value.

In other words:

In the block definition you may have a format string #(TargetObject.Tag), 19 characters long, but which becomes GV-001 in the block reference, 6 characters long. In the block reference, the linear parameter will be set to the length equal to the six characters.

Let’s look at a simple example, the “Oval Tag Style” Annotation Style. Using the skills discussed above, navigate to the Engineering Items node in the P&ID Objects Definitions class hierarchy, select the Oval Annotation Style, and press the Edit Block button.
The block editor shows an oval shaped piece of geometry consisting of line and arc segments, an Attribute Definition carrying the #(TargetObject.Tag) format string, and a Linear Parameter also carrying the #(TargetObject.Tag) format string. The Linear Parameter is associated with two stretch actions.

Selecting the stretch action on the left highlights the stretch action itself, the parameter with which it is associated, and the geometry which is participating in the action. A small rectangle shows the “action frame” for this action.

For stretch actions, the rules are as follows:

- Entities which are not selected by the action are not modified
- Entities which are selected by the action and are completely inside or completely outside the frame are moved.
- Entities which are selected by the action and which cross the frame are stretched.

In the example above, the arc on the left side of the oval is completely inside the frame, so when the left side of the linear parameter moves right or left (depending upon how the overall linear parameter is sized) the arc will move. The line segments at the top and bottom of the oval cross the linear parameter, so they will stretch.

For completeness, we highlight the other stretch action as well.

The right hand stretch action also selects the top and bottom lines, but selects the arc on the right hand side of the oval.
Selecting the linear parameter itself and raising the properties palette shows some special settings specific to the linear parameter.

Notice that the endpoints of the linear parameter are actually slightly inside the edges of the geometry, rather than directly on the edges. Doing this ensures that there will be some space around the text when the stretch actions are applied. If the linear parameter were exactly on the edges of the geometry, then when the parameter is sized to match the attribute’s width, the geometry will exactly enclose the attribute, which usually doesn’t look very good.

Note: The InfoTag2LineBox block definition in AnnotationTemplates.dwg used in part C., “Creating a New Annotation Style” actually shipped with this very problem. To get the best results, you may need to adjust the endpoints of the linear parameters to be inside the box, and assure that the Base Location property on each is “midpoint”, as discussed below.
The property called “Distance Label” is the property that must match the Attribute Definition’s format string. If these values ever fail to match, then the linear parameter will not be associated with the Attribute Definition. You can use the property palette to fix such errors, should they occur.

The linear parameter’s **Base Location** property can be start point or midpoint, and indicates how the linear parameter will be anchored as its size is changed. In most of our shipping annotation templates, we use a midpoint anchor for the linear parameter, and center-justify the Attribute Definition at the same X Value. This combination results in the geometry controlled by the linear parameter staying centered on the attribute.
The linear parameter’s **Show Properties** property controls whether the linear parameters properties will be shown in the properties palette of the block *reference*. Generally, the annotation block definitions we ship have this value set to No, because the block reference’s parameter’s properties are being controlled programmatically, and user modifications made via the property palette would not “stick”. However, setting this property to Yes can be extremely helpful when you are first authoring, and testing, your dynamic blocks.

The linear parameter’s **Chain Actions** property is an advanced property, and indicates whether the parameter can be controlled by another parameter in the same block definition.

The linear parameter’s **Number of Grips** property indicates how many grips should be shown for each block *reference* placed in the drawing. The number and type of grips differs depending upon the type of parameter and the number of associated actions. Because the linear parameter is going to be programmatically controlled, we ship our annotation block definitions with the number of grips set to zero for our linear parameters.

Here are several examples of this annotation style inserted into the drawing.

![Example Annotation](image)

The Gate Valve has a short tag value, and the Ball Valve has a very long tag value (based on our “Big Example Tag Format” above.) As can be seen, the oval has stretched to fit around the text, with a bit of padding.
For illustrative purposes, I enabled the “Show Properties” setting in the block definition, and show the properties for the two block references below.

Notice the property under the “Custom” heading called #(TargetObject.Tag). In this case, you are being shown the “Distance Label” of your linear parameter as the property name, and the actual width of the linear parameter as the value. When you first begin authoring dynamic blocks, you may find it helpful to have this enabled, and to make manual changes to the width of the linear parameter, in order to preview how your actions will behave at various widths.

Directly below, under the Attributes heading, you see the attribute for #(TargetObject.Tag), and its current text value. Selecting this item and clicking the ellipsis button will raise the Edit Annotation dialog for that attribute.

3. TotalX and TotalY

There are two special names that AutoCAD P&ID understands for linear parameters, in addition to those named to match Attribute Definition’s format strings. Let’s review the block definition for our Valve InfoTag annotation style.
Notice the two linear parameters named TotalX and TotalY.

In addition to using the width of an evaluated format string to set the width of an associated linear parameter, AutoCAD P&ID also calculates the combined extents of all the evaluated format strings in the entire block reference. If it encounters the TotalX linear parameter, it sets its size equal to the width of those combined extents. If it encounters the TotalY linear parameter, it sets its size equal to the height of those combined extents.

This can be useful, as in this example, for creating a box that surrounds several format strings.

Note that TotalX and TotalY will not collect the extents of other textual entities in the block; only those associated with format strings.

VI. Custom Symbols

A. Creating Custom Symbols

1. Block Definition Management

There is a difference between how Block Definitions are managed for Annotations vs. Graphical Styles. By default, each Annotation Style is always associated with a unique Block Definition. When creating a new Annotation Style, if you select a Block Definition for an Annotation Style that already exists, the product will automatically make a copy of that Block Definition for you to modify.

In contrast, Graphical Styles are designed to share Block Definitions. If you create a new Graphical Style and select the Block Definition for an existing Graphical Style, both the old and new Graphical Style will use that one Block Definition, and changes to that Block Definition will affect both Graphical Styles.

The reason that an Annotation Style automatically creates a new Block Definition is that usually, each Annotation Style will define different Attribute Definitions with different Format Strings, thus each Annotation Style must have its own Block Definition.

On the other hand, Graphical Styles were designed with the assumption that users would most often want to create several different graphical styles for the same P&ID class which differ only with respect to the other style
properties, such as color, layer, linetype, but which would otherwise want to share the same image. This is best accomplished by sharing the block definition.

2. Creating a new Graphical Style with a new Symbol

Given the Block Definition handling of Graphical Styles, it follows that to create an entirely new symbol, one must first create a new Block Definition.

Use the methodology of your choice to create the new Block Definition. In the following example, the Block Definition for the PIP Screw Pump has been copied and the copy has been renamed and modified. The new Block Definition has several lines added to the pump body. Both versions are shown.

Remember to save the drawing in which you have defined the new block definition.
The next step is to add a new Graphical Style for the Screw Pump class based on this Block Definition. Raise the Project Setup dialog and navigate to the SCREW PUMP node under P&ID Object Definitions.
Pressing the Add Symbol button raises the Browse dialog. Navigate to the current project’s projsymbolstyle.dwg.

Type in a name (Reversible Screw Pump Style).
The new style is has now been created in the Project Setup and is available.

For the final step, press the “Add To Tool Palette...” button to place a button on the Tool Palette which will create a screw pump using this new style.

The following dialog indicates that the tool has been successfully added, but be aware that it will be added to the bottom of the palette group currently active, and you may need to reposition in using drag/drop into the desired location on the palette.
The product will do its best to provide a reasonable bitmap on the tool palette for the new tool, but you may want to modify the image. Using the tools of your choice, you can design a bitmap to represent your new tool. Right clicking on the Tool and selecting “Properties...“ will raise the following dialog.

From here, you may right click on the image, and select “Specify Image...“ This will raise a navigation dialog in which you can select the image of your choice for the tool.
3. Graphical Style Properties

![Graphical Style Properties](image)

A brief description of the remaining Graphical Style properties follows:

- **Symbol Scale**: This is the default scale factor at which the symbol will be inserted.
- **Scale on Insert**: This controls whether the user will be prompted to scale the asset when it is dropped from the Tool Palette. If the user declines to manually scale the asset, the default scale factor specified in Symbol Scale will be used.
- **Scale Mode**: This is only enabled when “Scale on Insert” is set to Yes, and controls whether non-uniform scaling (scaling differently in X and Y) is allowed.
- **Rotate on Insert**: Controls whether the user will be prompted to rotate the asset when it is dropped from the Tool Palette.
- **Mirror on Insert**: Controls whether the user will be prompted to mirror the asset when it is dropped from the tool palette.
- **Tagging Prompt**: Controls whether the asset can be tagged, and whether the user will be automatically prompted to fill in tag values when the asset is placed from the Tool Palette.
- **Join Type**: Specifies how the asset is expected to join with the line. In recent release more emphases has been given to ‘Join Type’ property of any P&ID symbol to control the behavior of symbols especially the graphical representation with respect to line.
  - **Endline**: Asset remains stationary when line is moved. Moving the asset or the line causes the line to elbow. Erasing the line does not erase the asset.
  - **Inline**: Asset cleans the line without breaking it. Moving the line causes the asset to move as well. Erasing the line erases the asset.
  - **Segment Breaker**: Inserts into a line and breaks the line into two segments in the same group. Retains the line group and line number for both line segments. Example: Spec Break, etc.
  - **Segment Group Breaker**: Inserts into lines and breaks line into two segments and two line groups. Example: pumps, blowers.
4. **Manually switching between Graphical Styles**

Having created the additional symbol for Screw Pump, place a screw pump in the drawing. Select the screw pump and raise the properties palette. The properties palette will show both the original style “Screw Pump Style” and the newly created style, “Reversible Screw Pump Style”. You can use the Tool Palette to select either Style, and the selected Screw Pump will drawing using the selected Style.

5. **Associating Graphical Styles with a Property**

The ability to have multiple Styles available for a given P&ID object type is generally useful, but becomes more powerful when these various graphical representations are combined with a specific property. Each P&ID Object Definition can have a single Property defined that is of the “Symbol List” type.

Raise the Project Setup dialog, navigate to the Screw Pump, and add a new Property.
This raises the “Add Property” dialog. Select “Symbol List” as the type, and enter “Reversible” for the property name and its description. Then press OK.

The “Symbol List Property” dialog allows you to define the items in the list. This property is similar to a “Selection List”, except that each property VALUE allowed defined for the property is associated with a specific Graphical Style. Using the “Add Row...” button, and define two values, “Yes” and “No” for the “Reversible” property.

After adding the rows for “Yes” and “No”, the Symbol List Property dialog shows the rows that have been defined. For each row, all the Graphical Styles that have been defined for a Screw Pump are shown as options under the “Block Value” column. Select the new “Reversible Screw Pump Style” to be associated with the “Yes” value of the
new Reversible property, and select the original “Screw Pump Style” to be associated with the “No” value of the new Reversible property.

Ok out of this dialog. Back in the Project Settings, the new property is shown. Select No as the default value.

Place a Screw Pump in the drawing. Initially it will have the original representation. Open the data grid and scroll to the right. The new “Reversible” property is shown. Selecting the “Yes” option will cause the alternative style to be shown. *(You may have to click in the drawing pane to trigger the redraw of the Screw Pump.)*
There need not be a one to one correspondence between the property values and the Graphical Styles. This exact technique is used to implement the “Normally” property of Hand Valves in the product. The “Normally” values include 3 open and 3 closed variants, but only two Graphical Styles (one open and one filled) are used to represent each of the 3 open and closed values.

6. Using Multiple Graphical Style for an Asset that Spans Drawings

Another use of multiple Graphical Styles is to support a single asset that spans multiple drawings.

Using the techniques above, three additional graphical styles are defined here for the Vessel class. The default graphical style is a tall, vertical vessel. The three new styles are shown below.

In a project containing three drawings, place a vessel in each drawing. When prompted, assign each vessel the same Tag value, TK-100. When the second attempt is made to assign this tag value, which is a duplicate, it recognizes the fact that the Tag value is already in use, although in a different drawing, and queries you as to whether these assets are really meant to represent portions of the same asset, with the following dialog.
Select “Assign this tag to the selected component” for each vessel.

In the image above, a different Graphical Style has been assigned to the vessel in each drawing, so that each drawing shows the bottom, middle, and top of the Vessel respectively. However, note that in the Project View of the data grid shown, there is only one internal representation (PnPID) for the vessel. Thus, the single vessel spans 3 drawings, with an independent and appropriate representation in each drawing.

7. A note about Attachment Points
Attachment Points in the block definition of a graphical style have two purposes.

The first purpose is to provide snap points. When you begin drawing a PipeLine from a piece of equipment, any Attachment Point in the equipment’s block definition functions as a snap point.

Additionally, for assets that have flow direction, the naming of the Attachment Points is important. The validation feature, when completed, will interpret the flow direction of an asset as going from AttachmentPoint1 to AttachmentPoint2, and will use this to recognize situations in which an inline asset’s flow direction is in conflict with the flow direction of the line upon which it sits. Validation will also check for line connections at the attachment point to determine if a line is fully connected to an asset.
Finally, some assets, such as Hand Valves, support the “End Connections” property. This property allows for various end connection graphics to be drawn on the valve. These graphics also rely on the attachment points in order to determine where to draw. In addition, they rely on the label of the attachment point to determine at what angle to draw. The intention to support endcodes is indicated by adding a colon, the word EndCode, and an angle, in counterclockwise degrees, to the attachment point. Thus, an attachment point label could be: AttachmentPoint1:EndCode180.

The following example shows attachment points with endcodes defined for both a 2 port and a 4 port valve.

B. Dynamic Assets

Dynamic Assets combine the principles discussed in both Annotation Styles and Graphical Styles.

A Dynamic Asset is an asset which intrinsically includes textual elements based on the asset’s properties as part of the asset itself. As the values of the asset’s properties change, the textual representation of those values update as well. One of the main motivations for defining a Dynamic Asset, as opposed to using a regular asset combined with an Annotation is that the text is fully integrated with the asset graphics, and cannot be moved away from the asset.
As with an Annotation, the properties to retrieve are specified using a Format String, associated with an Attribute Definition, as part of the Block Definition for the symbol. All Format String syntax is supported. The target object used to evaluate the Format String is the Dynamic Asset itself.

Including a Format String in the Block Definition of a Graphical Style is sufficient to alert the program to create a Dynamic Asset when the asset is placed in the drawing. There are no other settings that need to be touched, no other support required.

Also, it is perfectly safe to modify the Block Definition of a Graphical Style to add or remove Format Strings. After the changes are made, the program will automatically convert all the existing assets in the drawing between being a Dynamic or a regular Asset, depending upon the current Block Definition.

1. Special Behaviors of Off Page Connectors

In AutoCAD P&ID, the Off Page Connectors are special types of Dynamic Assets.

First, the special “use mate” syntax is included in the Format String of the Block Definition, in order to show values from the “other” drawing in which the mated connector resides. This syntax is not supported by the “Assign Annotation Format” dialog, and must be inserted manually while editing the block definition.

For example:

```
#(Drawing.General.PnID)
#(=Drawing.General.PnID)
```

The first Format string would return the PnID (drawing number) of the drawing in which the target object resides.

The second Format String would return the PnID (drawing number) of the drawing in which the target object’s mate resides.

The equal sign just inside the opening parenthesis specifies “use mate’s property.” Naturally, if the target object is not a mate-able type, then this format string will not evaluate, and the DOT will be seen instead.
In the image below, the connector on the left, in drawing 1, automatically shows the value 2 inside the right side of the arrow, and the connector on the right, in drawing 2, automatically shows the value 1 inside the right side of the arrow, because these connectors have been mated.

Secondly, the implementation of the shipping connector types includes some special coding that is relevant to exactly the combination of graphics and Format Strings in our shipping OPC symbols. For example, the Closed Drain symbol ships with Format strings defined, from top to bottom, to show the connector number, the destination line service code, and the mate’s drawing number. In the image below, the default or “right side up” orientation is shown on the left, and the drain on the right is inverted due to its location on a downward flowing line. Notice, though, that the values read in the same top to bottom order on the second drain, in spite of it being inverted. This behavior is specific to our shipping symbols, and is unfortunately not user extensible at this time. For this reason, and also because the dynamic blocks for the Off Page Connector arrows are among the most complex shipped in the product, extensive modification of these blocks is discouraged. Minor modification such as font changes can be successfully accomplished by most users, but major shape changes should be undertaken only with a good understanding of Dynamic Block authoring, and after making a backup. Repositioning, changing, or erasing any of the specific properties displayed in our shipping OPC blocks may cause the automatic re-orientation of those properties to fail.
If the exact combination of properties, positions, and behaviors is too difficult to accomplish in the connector’s own Block Definition, a combination of the asset and annotations can often be used to accomplish the desired results.

2. Special Behaviors of Instruments

General Instruments, and the bubbles for Inline Instruments, exhibit special behaviors which are not user modifiable in this release.

While placing a general instrument, an additional prompt is available by which the instrument can be attached to a line segment or another asset via a Leader. In the case of General Instruments attached to a Pipe Line, the leader line is part of the General Instrument itself, and is drawn by the General Instrument. When a General Instrument is attached to another asset via a leader, the leader line will be inserted as a piece of Tubing.

When an Inline Instrument, such as a magnetometer is placed, the product prompts for a position for the default annotation, which is the instrument bubble. After placement, the product places a leader between the bubble and the annotative bubble. In this case, the leader is actually a separate AutoCAD Leader entity.

For both General Instruments and the annotative bubble of Inline Instruments, the product enforces a horizontal orientation of the contained text, regardless of the anchorage of the bubble. In other words, a bubble anchored to, or inline with, a vertical pipeline will still have text that reads horizontally.

This differs from the default behavior for annotations, in which a vertical target results in vertical (bottom to top) orientation of the text.

VII. Property Propagation/Acquisition

For some P&ID classes, their properties are closely associated with the properties of another P&ID class.

For example:
Both P&ID Hand Valves and Pipe Lines have a size property.

You may want the size property of the Hand Valve to automatically acquire the same value as the Pipe Line upon which you place the valve. Similarly, if you modify the size property of a Pipe Line, you may want the Pipe Line to automatically propagate that size value to all the hand valves on that line.

On the other hand, there could be situations in which you want the size property of the Pipe Line and the Valve to differ.

The fundamental rules are: Property propagation and property acquisition are the same concept, when component A propagates its value to component B; it means component B acquires its value from component A. The Acquisition type property allows the user to specify a RULE as the means by which a property of an instance gets its value.

To add Acquisition property, open Project Setup and select Add button in P&ID Class Definition. In Add Property dialog create a new property of the type ‘Acquisition’. This will take you to select data source where you can select a property to acquire from. You can select Class, Drawing or Project property to acquire from.

The acquisition rule defined can be seen in the Properties list from the defined property under ‘Acquisition’ column. This cell also allows removing or editing acquisition rule by clicking on the button in the cell which has two options; ‘Remove acquisition rule’ and ‘Edit acquisition rule’.

The Acquisition property type also supports the concept of “Initialization Only”, where the instance first acquires the value from its source and immediately goes to the Override mode.

![Image](image-url)

Acquisition properties can get their values from Project properties, drawing properties, or PnID Class.
Properties.
When an instance property is at the “Acquire” mode, changes to the source are picked up by the instance that is acquiring. In this mode the data is in Read-only format and cannot be entered manually.

When an instance property is at the Local mode called as “Override” mode, changes to the source are NOT picked up by the instance doing the acquiring. This mode let you enter new values manually, but remember changes made here do not get pushed back to source property.

Acquisition rule can be edited or remove via Project Setup.

Acquisition property is just another type of properties, if you setup a (Acquisition) property at the EngineeringItems level, all Class Definitions below that level will inherit the property.

VIII. Appendix

A. Format string syntax

The following format string syntax allows identification/tokenization of retrievable properties, formatting/input rules, auto-generation, and assign-back states. (This is the underlying syntax that is created by the Assign Tag Format/Assign Annotation Format dialogs.) Knowledge of these details may be useful for developers who wish to programatically create format strings for their own features.

While there is an overarching syntax, portions of the syntax are more appropriate to Tag Format strings. In particular, the assign-back token should be used only with Tag Format strings, never with Annotation Format strings. (This token is hidden in the various Tag Format UIs.)

The auto-generation token may be used with either, but has few if any use cases for Annotations.
The formatting token is used by several dialogs as a means to enforce input for Tag Formats, but is used to apply formatting to the output of Annotation Formats.

A field is delimited by the form #().

A class property is indicated by Class.Property. A drawing property is indicated by Drawing.Category.Property. A project property is indicated by Project.Category.Property.

Examples of simple retrievable properties are:


The user may combine an expression with a property, for the purposes of formatting, autogeneration, and assignment.

If the user wants to specify a format for a property, it must be included inside the field, specified with %.

For example, if the Flow Rate should always be shown as 3 numbers, then the field would be:

#(CentrifugalPump.FlowRate%NNN)

If the value needs to be autogenerated, the @ sign is used instead. #(PipeLine.LineNumber@NNN)

(The @NNN syntax immediately above was in support of an early concept for arbitrary autogeneration which was based on a dumb counter. It is not surfaced in the UI, but will produce a value if created and assigned via APIs.)

To specify the lookup location for auto-generation source and increment value (as opposed to arbitrary generation), a $ is added at the end of the @ portion of the field.

#(PipeLine.LineNumber@NNN$Project.LastLineNumber) If the autogenerated value should be assigned back, the ^ sign is added. #(PipeLine.LineNumber^@NNN$Project.LastLineNumber) If the user must input a value to be assigned back, the autogenerate symbol is not used. #(PipeLine.LineNumber^NNN)

For variants where user input is expected, lack of input will be indicated by the mask symbol ‘?’.

Note that “simple” auto-generation as described in section III.D.1 is implemented directly in the code, and is not represented by the Format String syntax in any way.

If the user wants to specify an optional field, the ~ symbol is used. For example, an optional field would be indicated by: #(HotWaterPipe.Insulation~A*)

Optional fields are never auto-generated, but may be assigned back. Optional fields will not cause an error condition if they cannot be assigned or retrieved. They will simply be skipped. Note that the optional syntax isn’t surfaced in any UI.

Expressions can also be specified independent of retrievable properties. For example, to simply generate a 3-digit-minimum numeric value, the syntax would be:

#(@NNN)

Again, the syntax above was in support of the early counter-based arbitrary generation, which is not surfaced in the UI.
To require user input of a 3-digit-minimum number, the syntax would be:

```plaintext
#(%NNN)
```

When being evaluated, any text that occurs between fields is passed through unchanged. This is referred to as “free text.” Delimiters fall into this category.

If the user specified AREA-LINENUMBER, the format string would be:

```plaintext
#(Drawing.Category.Area)-#(Primary.LineNumber)
```

Note the “dash” delimiter between the fields.

As is standard in various programming languages, the backslash character can be used to ‘escape’ special characters outside of the fields. If the user wants to use the # character as a delimiter, it should be escaped to prevent it from being interpreted as the beginning of a field.

Note that the output of evaluating the format is a single text item, not multi-line text, so the newline character is not valid.

To specify that the property should be retrieved from the target object’s mate (in the special case of CONNECTORS), the property should be pre-pended by the equal sign.

For example, to retrieve the PnID number of the Off Page Connector in the other drawing from the target Off Page Connector, the format string would be:

```plaintext
#(=Drawing.General.PNID)
```

Setting and retrieving the Format Strings of Tag Formats and Annotation Formats programmatically is done via the AcPpTagFormat and AcPpAnnotation class APIs respectively.

**B. APIs**

The following header files will be of interest to developers wishing to work directly with Annotations and Tag Formats.

The header file: AcPpAnnotation.h contains methods for creating P&ID Annotations, getting and setting annotation properties (including accessing the Format Strings of specific attributes within the block reference), finding the associated annotation style, etc.

The header file: AcPpAnnotationUtils.h contains methods for identifying whether a given text or block reference entity is a P&ID Annotation, and for refreshing P&ID Annotations.

The header file: AcPpTagFormat.h contains methods for working with Tag Formats, getting and setting of Tag Format properties (including getting and setting the Format String) and storing a Tag Format in the AcDbDatabase.

The header file AcPpTagUtils.h contains methods for validating that a Format String is properly formed, and for evaluating a Format String against a target AcDbObjectId.
NOTE THAT the AcPpTagUtils namespace was unfortunately misnamed in the API. This should have been AcPpFormatUtils. Do not let the API namespace confuse you into thinking that Format Strings are equivalent to or only for use by the Tag feature.

DISCLAIMER:

Autodesk may make statements regarding planned or future development efforts for our existing or new products and services. These statements are not intended to be a promise or guarantee of future delivery of products, services or features but merely reflect our current plans, which may change. Purchasing decisions should not be made based upon reliance on these statements.

Autodesk assumes no obligation to update these forward-looking statements to reflect events that occur or circumstances that exist or change after the date on which they were made.