P2 Verify 4.0
Administrator's Guide

Contents

Preface

This guide provides administrators with information on how to configure P2 Verify.

This guide outlines how to use P2 Verify to design data area layouts, as well as configure collectors and submitters for use with entities.

Who Should Read This Guide

This guide is intended for administrators of P2 Verify who need to keep the data capture interface up to date, in line with changing business requirements and system configuration.

This guide assumes working knowledge of:

- Microsoft® Internet Explorer®
- Microsoft® Windows® operating systems

Related Documentation

Documents in the P2 Verify technical documentation suite are:

Title	Description
P2 Verify Release Notes	Release Notes for the latest version of P2 Verify.
P2 Verify Installation Guide	Installing P2 Verify components and configuring P2 Verify for first time use.
P2 Verify Administrator's Guide	How to administer data area layouts in P2 Verify.
P2 Verify User's Guide	How to use P2 Verify to collect, enter, and validate operational data and submit it to the production reporting system.

These documents are available from P2 Customer Support.

Support

P2 Customer Support provides a central point of contact for software assistance and the resolution of software issues. As part of this, P2 offers a variety of professional services, online resources, and access to experienced product specialists who are able to assist with your service requests. For support and information regarding our products, the following resources are provided:

ONLINE SUPPORT PORTAL

The P2 Support Portal (http://p2energysolutions.com/support) provides access to online support, where you can raise service requests for P2 software, track defects, get product information, and communicate with P2 Customer Support.

CUSTOMER COMMUNITIES

P2's customer communities offer a networking environment for you and other P2 users. Our boards and user groups offer an informal setting to exchange information and discuss issues relevant to

today's oil and gas companies. P2 is confident that together, we can create an interactive venue that will provide value by allowing our customers to communicate, collaborate and connect at multiple levels. For details, see www.p2energysolutions.com/services/customer-communities.

TRAINING

P2 offers a variety of standard and customised training courses (ranging from introductory courses through to administrator courses) to help you learn how to use P2 products.

CONTACT DETAILS

You can contact P2 Customer Support via phone or the Support Portal for technical support on any aspect of P2's products. Please also contact P2 Customer Support for further information on the Customer Communities, access to the online support portal, and information on available training courses. If you do not have a user account for the Support Portal, contact your internal support team or call the number below.

Phone:	1300 739 969 (Australia only) or +61 8 9241 0314 (outside Australia)
	USA: 1-844-REACHP2 (1-844-732-2472)
Support Portal:	http://support.p2energysolutions.com
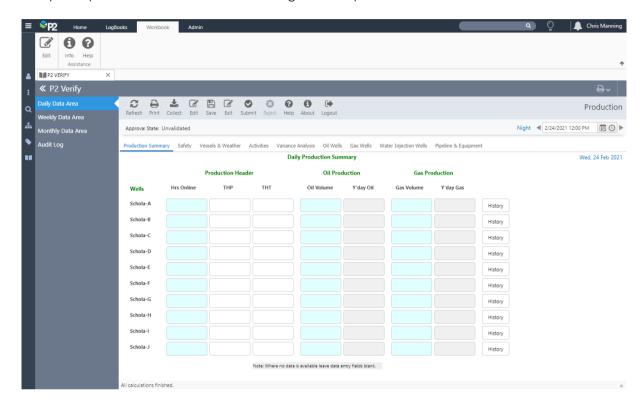
Layout Mode

P2 Verify collects data from facilities and systems that are expected to change over time. As such, P2 Verify needs to be updated to reflect these changes. Layout Mode allows administrators to easily manipulate the interfaces when changes are required.

Layout Mode allows P2 Verify administrators to add, move, and remove fields as necessary. When in layout mode, entities and tabs can be added or edited by right-clicking on the page or the tab strip to open the edit menu. The functions available depend on what is being edited. You can:

- Add, edit, or remove a tab.
- Add, edit, or remove an entity.
- Reset the tab order for tabbing between entities' controls on a P2 Verify page.

Layout Mode is similar to Input mode, with the following differences:

Layout version

Layout Mode displays the version of the Layout that is currently active. Use the Date/Time Picker to change between layout versions.

Grid lines

P2 Verify displays grid lines on the report page in Layout Mode, to assist with aligning your entities on the screen.

Entity Editor

In Layout Mode, you can use the Entity Editor to modify any details of the displayed entities.

A number of keyboard shortcuts are available to you in Layout Mode, to make it easier to edit and arrange your layout.

Shortcut	What it does
Double-click	Double-click on an item to open the Entity Editor.
CTRL + C	Copy the object to the clipboard, along with its properties.
CTRL + V	Paste the copied object into the layout. The location is slightly off to the lower right hand side of the copied object.

Restrictions and Performance Recommendations

Performance of P2 Verify is highly dependent on several factors, such as hardware, network configuration, data sources, page complexity, and data load. Essentially, the more objects on a page, the higher the potential for performance degradation.

While designing a page in Layout Mode, administrations should be mindful of the performance impact of a designed layout. Some basic recommendations for page construction can be made so that P2 Verify performs as expected.

Recommended page configuration limits are as follows:

Indicator	Metric
Number of entities per tab (peak)	200 (300) per tab
Number of calculation entities	200 per layout
Number of tabs	5 per layout
Number of embedded VFYIFrames	2 per layout
Number of custom scripts that iterate through Verify entities	2 per layout
Number of custom scripts that act like a calculation	Count each script as a calculation entity
Number of custom scripts that call an external web service or other data function	1 per layout

RESTRICTED OR UNSUPPORTED CHARACTERS

- The quotation mark character (") should not be used in any controls in the Entity Editor.

- The following characters should not be used in free-text inputs:

  = [ ] { } < > | ; ' " ?

- The following characters cannot be used in entity names:

  ~ ! @ # $ % ^ * ( ) + { } | : < > ? ` = [ ] \ ; , /

- The following characters cannot be used for entities that participate in calculations:

  ' & :

Entering Layout Mode

▸ To enter or exit Layout Mode, click the **Edit** 📝 button on the toolbar.

When you enter Layout Mode, snap-to-grid lines are displayed to help you line objects up on the screen.

When you exit Layout Mode, you return to Read-Only Mode. From here you can click the **Enter** button to view your layout in Input Mode.
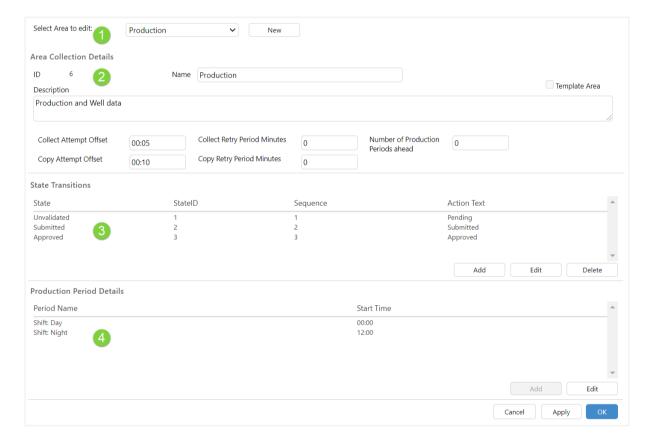
> **Note:** Before making any changes to your layout, make sure you are looking at the correct Layout Version.

Defining Data Areas

The Area Editor allows you to define data areas and specify state transitions, production periods, and data collection settings for the area. The Area Editor is only accessible in Layout Mode and is therefore restricted to administrators.

▶ To open the Area Editor, click the **Area Editor** ⚙ button in Layout Mode.

Here is an overview of the interface for the Area Editor.

1   Area Selector
Displays the areas available for editing, and allows you to create new areas and delete existing areas.

2   Area Collection Details
Configure data collection times for the area.

ID
The unique identifier in the database for this area.

Name
The name of the area. The name must contain only alphanumeric characters and spaces.

Template Area
Template areas cannot be edited in Input Mode, and cannot even be accessed in Read-Only Mode. They can only be designed using Layout Mode, and are intended to be accessed using additional tools available from P2 Customer Support.

 Note: After an area has been saved, it cannot be changed between a Standard and a Template Area. Therefore when you create a new area, make sure the Template Area check box is set correctly before you save.

Description
A description of the purpose of the area. You can be verbose with your description.

Collect Attempt Offset
The offset for the first time (hh:mm) the collection for the previous production period will be attempted. For example, if the Collection Period End Time was set to 08:00 and the attempt time was set to 0:30, the first collection attempt will occur 30 minutes after the Collection Period End Time. That is, at 08:30.

Collect Retry Period Minutes
The length of time, in minutes, that the system will retry collection. After this time, collection is aborted. For example, if this was set to 30, the system will retry collection every minute for 30 minutes.

Copy Attempt Offset
The offset for the first time (hh:mm) the copy from previous period will be attempted. For example, if the period finishes at 08:00 and the offset is set to 00:30, the copy will be attempted at 08:30.

Copy Retry Period Minutes
The length of time, in minutes, that the system will attempt to copy the newly collected data to the database. After this time, copy is aborted. For example, if this was set to 30, the system will attempt to copy the data to the database every minute for 30 minutes.

Number of Production Periods Ahead
Specify the number of production periods ahead of the current date for which users can navigate using the Date Picker. For example, specifying 5 means that a user can navigate forward five production periods ahead of the current date. Data collection occurs as normal.
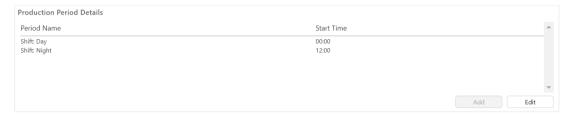
3   State Transitions

You can add, edit, and delete state transitions here. All state transitions that you create are stored in a central database and you can apply them to any area as required.

State Transitions			
State	StateID	Sequence	Action Text
Unvalidated	1	1	Pending
Submitted	2	2	Submitted
Approved	3	3	Approved

Add   Edit   Delete

4   Production Period Details

The list of configured production periods assigned to the Data Area. For detailed instructions, refer to *Setting Production Periods* (see page 10).

Production Period Details	
Period Name	Start Time
Shift: Day	00:00
Shift: Night	12:00

Add   Edit

▶   Click the **OK** or **Apply** button to save any changes. Click the **Cancel** button to cancel any changes.

Setting State Transitions

State transitions are completely configurable. You can choose the transitions that are meaningful to your business workflow.

State Transitions			
State	StateID	Sequence	Action Text
Unvalidated	1	1	Pending
Submitted	2	2	Submitted
Approved	3	3	Approved

Add   Edit   Delete

Note: For each state transition you add, you must also define an Object in the Security Manager.
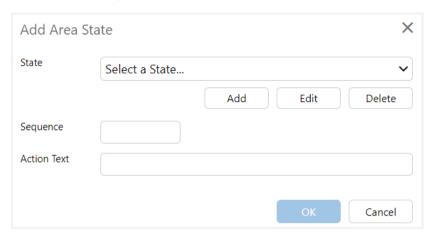
For example, you may have a workflow sequence that consists of five people, with some needing to validate data in all operational areas, and others needing to validate data in selected areas only.

As you define state transitions, they are all added to the database, but you can select which ones to apply in which areas.

To define state transitions:

1.   In the Area editor, click the **Add** button in the State Transitions section.
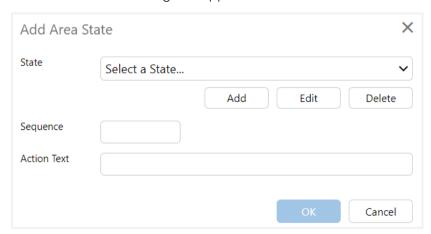
2.   In the **Add Area State** dialog box:

    a.   From the **State** drop-down list, select the state you want to add to the area.

        If the state you want to use is not in the list, you should add a new one (see page 8) or modify an existing one (see page 9).

    b.   In the **Sequence** box, specify the numerical sequence in which this state occurs. The lowest sequence (1) corresponds to the initial raw data state (typically called *Unsubmitted*). The highest sequence number corresponds to the final approved state, during which the data is sent to the destination tags in P2 Server or to a data store repository.

    c.   In the **Action Text** box, type the text that is to appear in the Action Text column of the Area Editor.

3.   Click **OK** to save your changes.

4.   Click **Apply** or **OK** at the bottom of the **Area Editor** window to save your states.

Add a State

To add or edit a transition state:

1.   In the Area editor, click the **Add** button in the State Transitions section.
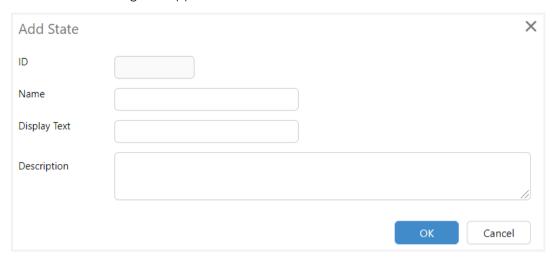
    The **Add Area State** dialog box appears.

2.   In the **Add Area State** dialog box, click **Add**.

The **Add State** dialog box appears.

3.   Complete the following fields:

ID
The unique identifier for this state. This is system-generated.

Name
The name of the state that appears in the header in Read Only mode.

Display Text
The text to display for this state in the Area Editor.

Description
A description of what this state transition means.

4.   Click **OK** to save your changes.

5.   When you return to the Add Area State dialog box, complete any remaining actions and then click **OK**.

Edit a State

To edit a transition state:
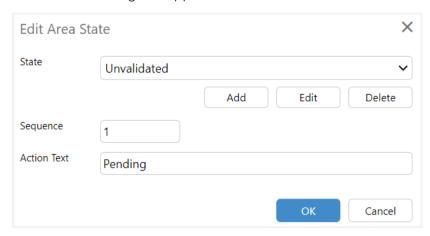
1.   In the Area editor, select the state you want to edit in the State Transitions section, and then click the **Edit** button.

      If the state you want to edit is not listed, click the **Add** button instead.

      The **Add Area State** or **Edit Area State** dialog box appears.

2.   Make sure the state you want to edit is selected in the drop-down list.

3.   Click **Edit**.

The Edit State dialog box appears.

4.   Complete the following details:

ID
The unique identifier for this state. This is system-generated.

Name
The name of the state that appears in the header in Read Only mode.

Display Text
The text to display for this state in the Area Editor.
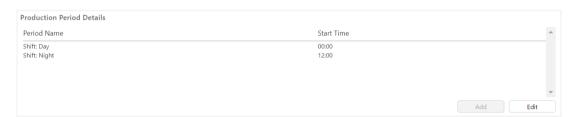
Description
A description of what this state transition means.

5.   Click **OK** to save your changes.

6.   When you return to the Add Area State dialog box, complete any remaining actions and then click **OK**.

Assigning Production Periods

Each operational area will most likely have people working during different production periods (shifts, daily, and so on) and you can define the production period for each operational area.

Production periods only apply to the areas in which they are scheduled, so if you have several areas, you need to assign the production periods for each area separately.

Defining production periods is similar to defining state transitions. However, you can only add one production period to each area. A production period, however, may consist of several shifts or periods.
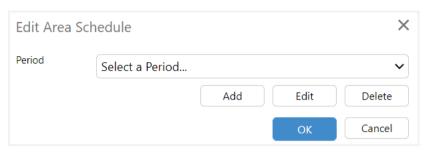
To assign a production period to an area:

1.   In the Area Editor, first select an area.

2.   If you do not have any Production Periods assigned to this area, click **Add** in the **Production Period Details** section.

3.   Otherwise, click **Edit**.

The **Add Area Production Period** or **Edit Area Production Period** window appears.

4.   Select the production period you want to assign to the area.

You can choose a previously defined period from the drop-down list, or click **Add** to add a new period (see page 11).

5.   Click **OK** to save the Production Period and close the window.
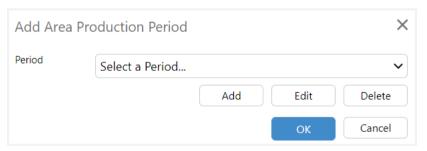
Note: It is possible to delete or modify a shift, or modify a production period that is currently in use. If there are reports created for the shift or production period, these reports may no longer be accessible. The report date has to match the exact date and time of a shift or a production period for it to be accessible.

Add a New Period

To add a new production period:

1.   In the Area editor, click the **Add** or **Edit** button in the Production Period Details section.
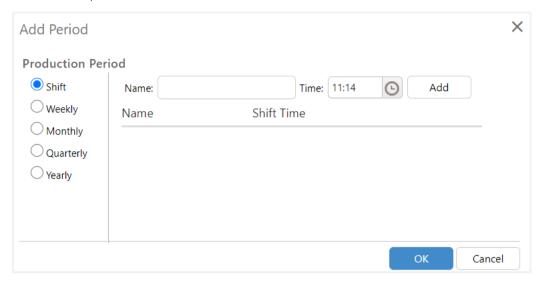
The **Add Area Schedule** or **Edit Area Schedule** window appears.

2.   Click **Add** to add a new period, or click **Edit** to modify an existing one.

The **Add Period** window appears.

If you selected Edit earlier, the dialog box will be pre-populated, and you can update the details as required.

When you first add a period, the **Shift** production period will be selected by default.
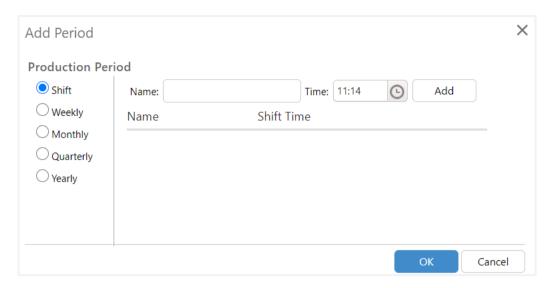
3.   Use the buttons on the left to select one of the following production periods:

    –   Shift (see page 12)
    –   Weekly (see page 14)
    –   Monthly (see page 15)
    –   Quarterly (see page 16)
    –   Yearly (see page 17)

4.   Specify the options accordingly.

5.   Click **OK** to save the Period and close the window.

6.   Click **OK** in the Area Schedule window.

Shift

A shift production period is one that starts at a specified time of day, and runs until another shift is specified. If no other shift is specified, it is treated as a daily production period and the next period starts at the same time the following day.

To specify a Shift production period:

1.   In the **Add Period** window, select the **Shift** radio button.

2.   In the **Name** box, type the name of the shift.

      Shifts must have a unique name. For example, *Shift A*.

3.   In the **Time** box, select the time of day at which the shift is to start. Click the ellipsis button to open the Time Picker.

      Shifts must have a unique start time.

4.   Click the **Add** button to add the shift to the list of shifts for this production period.

      To create a **daily** shift, just add a single shift production period.

5.   Repeat steps 2-4 to add further shifts as required.

6.   Click **OK**.

The name of the production period, as displayed in the Area Editor, will be a combination of all the shifts specified in the period. For example, if you have added a day shift beginning at midnight, and a night shift beginning at midday, the name of the production period will be something like *Shift: Day 00:00,Night 12:00*.
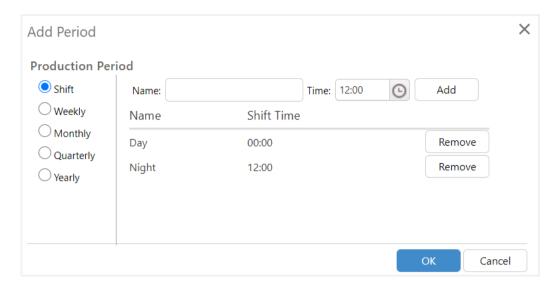
EXAMPLE

If you want to add a day shift beginning at midnight, and a night shift beginning at midday:

1.   In the **Name** box, type *Day*.

2.   In the **Time** box, click the ellipsis button and move the hands of the clock to midnight. Ensure AM is selected.

3.   Click **Add**.

4.   In the **Name** box, type *Night*.

5.   In the **Time** box, click the ellipsis button and move the hands of the clock to midnight. Ensure PM is selected.

6.   Click **Add**.
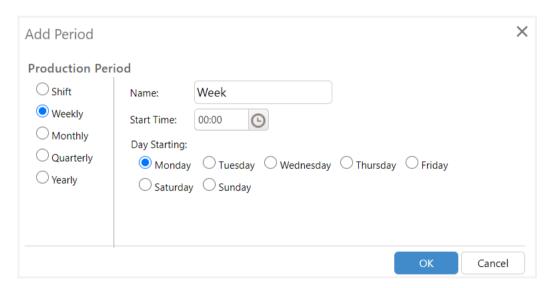
7.   Click **OK**.

Weekly

A weekly production period is one that starts at a specified day and time, and runs for a full week before starting again at the same day and time the following week.

To specify a weekly production period:

1.   In the **Add Period** window, select the **Weekly** radio button.

2.   In the **Name** box, type the name of the weekly production period.

3.   In the **Start Time** box, select the time of day at which the weekly production period starts. Click the ellipsis button to open the Time Picker.

4.   In the **Day Starting** area, select the day of the week that the production period starts on.

5.   Click **OK**.

Monthly

A monthly production period is one that starts at a specified day and time, and runs for a full month before starting again at the same day and time the following month.

To specify a monthly production period:

1. In the **Add Period** window, select the **Monthly** radio button.

2. In the **Name** box, type the name of the monthly production period.

3. In the **Start Time** box, select the time of day at which the monthly production period starts. Click the ellipsis button to open the Time Picker.

4. To specify the day of the month on which to start the period, select either:

    The (*date)* **of the calendar month**
    This will allow you to select which day of the month that the production period will start on. If you select a monthly production period day such as 30, and the user is entering data for a month with fewer days (such as February), the system will default to the end day of the month (for example, the 28th February).

    The *1st (day)* **of the calendar month**
    This will allow you to define a monthly production period that will start on the first Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, or Sunday of the month.
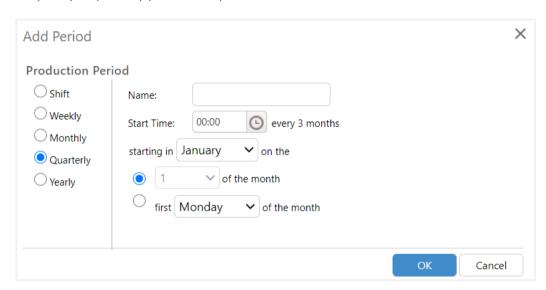
5. Click **OK**.

Quarterly

A quarterly production period is one that starts at a specified day and time, and runs for a full quarter (being three months) before starting again at the same day and time the following quarter.

To specify a quarterly production period:

1.   In the **Add Period** window, select the **Quarterly** radio button.

2.   In the **Name** box, type the name of the quarterly production period.

3.   In the **Start Time** box, select the time of day at which the quarterly production period starts. Click the ellipsis button to open the Time Picker.

4.   To specify the month in which the quarterly production period starts, select the month from the drop-down list labelled ***starting in (month) on the.***

        Subsequent quarters start every 3 months from the selected month. For example, if January is selected as the starting month, production periods will start in January, April, July and October every year.

5.   To specify the day of the month on which to start the period, select either:

        (*date***) of the month**
        This will allow you to select which day of the month, at the beginning of the quarter, that the production period will start on. If you select a day which is larger than the number of days a month has, the system will default to the last day of the month. For example, if 30 is selected, and February has only 28 days, the start of the production period will fall on the 28th of February.

        first (day) **of the month**
        This is the first occurrence of a particular day in the month at the beginning of the quarter. This will allow you to define a quarterly production period that will start on the first Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, or Sunday of the month at the beginning of each quarter.
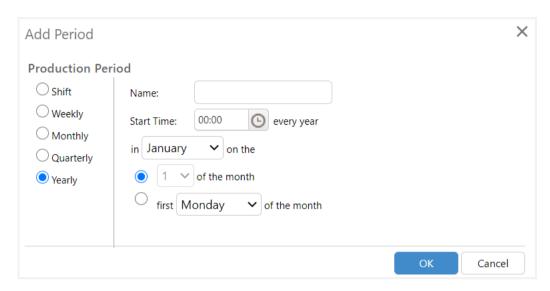
6.   Click **OK**.

Yearly

A yearly production period is one that starts at a specified day and time, and runs for a full year (being 12 months) before starting again at the same date and time the following year. To specify a yearly production period:
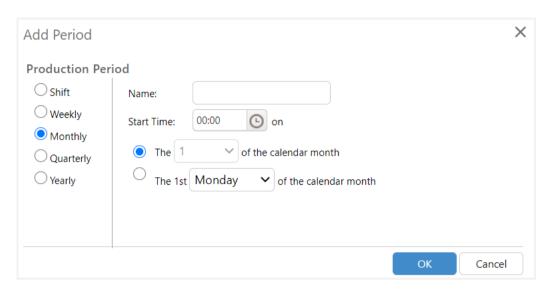

1. In the **Add Period** window, select the **Yearly** radio button.

2. In the **Name** box, type the name of the yearly production period.

3. In the **Start Time** box, select the time of day at which the yearly production period starts. Click the ellipsis button to open the Time Picker.

4. To specify the month in which to start the period, select the month from the drop-down list that is labelled *in (month) on the*.

5. To specify the day of the month on which to start the period, select either:

   (*date)* **of the month**
   This will allow you to select which day of the month the production period will start on. The production period will begin on the same selected date every year, with the exception of 29th of February, which falls on the 28th on common years.

   first (day) **of the month**
   This will allow you to define a production period that will start on the first Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, or Sunday of the selected month every year.

6. Click **OK**.

Defining Tabs

Tabs are unique to the operational area in which they are defined.

Initially, P2 Verify will display a single tab. As you configure the layout, you will probably want to add other tabs to group related objects.

Note: For each tab you add, you must also define a separate object in Security.

You probably already have an idea of what tabs you need. For example, you may want a tab that presents you with a broad overview of production, and you may want a separate more detailed tab on production downtime.

Note: Tabs can be added and removed from different Layout Versions in a similar manner to entities. Before making any changes to your layout, make sure you are looking at the correct Layout Version.
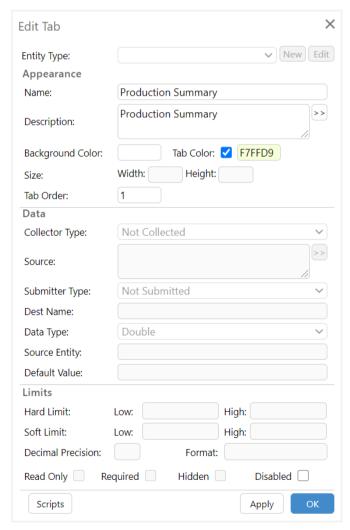
Adding and Editing Tabs

To add a new tab or edit an existing tab:

1.   Right-click on the tab bar and select **Add Tab** or **Edit Tab**.

The **Tab Editor** appears. This is essentially the same as the **Entity Editor**, however it has been modified so that only the properties that apply to tabs are active.

2.   Complete the properties as required. The relevant properties are:

Name

The unique name that identifies the tab. This is how the tab is referenced in the system.

Description

A name for the tab, which will be displayed on the tab itself in P2 Verify.

Background Color

The background colour of the page. Often, this will be white, but you may wish to change it to another colour.

Tab Color

Select the check box to assign a background colour to the tab itself. This does not affect the colour of the text.

Tab Order

The numerical order, from the left, in which the tab appears. The leftmost tab has an order number of **0**. The next tab on the right has an order number of **1**, and so on.

Disabled

Select this check box if you want to exclude the entities on this tab from collection.

Scripts

Click this button to add scripts to the tab (see page 22).

3.   Click **OK**.

4.   Click the **Save** 💾 button on the toolbar to save your changes.

Renaming Tabs

To rename a tab:

1.   Right-click on the tab bar and click **Edit Tab**.

2.   In the Tab Editor, change the name in the **Description** field.

3.   Click **OK**.

Note: Remember to click the **Save** 💾 button to save your changes.

Removing Tabs

To remove a tab:

1.   Right-click on the tab bar and click **Remove Tab**.

2.   At the prompt, click **OK**.
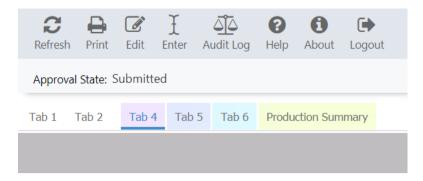
Note: Remember to click the **Save** 💾 button to save your changes.

Tab and Background Colours

P2 Verify allows you to define tab colours and background colours for your tabs:

- The *tab colour* refers to the colour of the tab selector in the header.
- The *background colour* refers to the colour of the space surrounding the entities' controls.

The image above demonstrates some tabs using custom tab colours, with the underlined active tab that uses a custom grey background colour.

Defining Tab and Background Colours

To specify the colours for the tab and the page background:

1.   Right-click on the tab bar and select **Add Tab** or **Edit Tab**.

Background Color
The background colour of the page. Often, this will be white, but you may wish to change it to another colour.
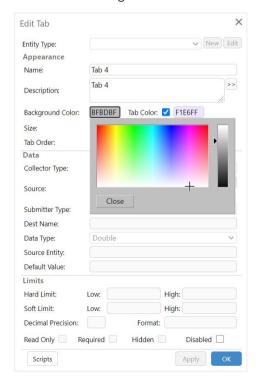
Tab Color
Select the check box to assign a background colour to the tab itself. This does not affect the colour of the text.

2.   In the tab editor, use the colour picker to select the colours.

Background Color
In the box, type the hexadecimal code for the desired colour, or click the box to select the colour from the colour picker.

The default background colour is FFFFFF (white).
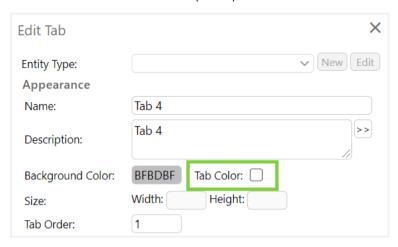
Tab Color

Select the Tab Color check box if you want to specify a colour for the tab. In the box, type the hexadecimal code for the desired colour, or click the box to select the colour from the colour picker.

To revert to the default colour (white), clear the **Tab Color** check box.

3.   Click **OK**.

4.   Click **Save** on the Verify toolbar.

Managing Custom Scripts

Additional JavaScript files can be included in P2 Verify to perform any desired custom functionality. Reusable custom scripts can be defined to perform complex calculations or any other special functions within the scope of JavaScript.

The function is usually set during installation and is placed into a JavaScript file that is accessible from the Expression Editor in P2 Verify. By default, the available functions are located in the folder:

`C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web\Scripts\`

To add a custom JavaScript file:

1.   Save the JavaScript functions in a JavaScript file (*.js), and copy the file to the following directory:

     `C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web\Scripts\`

     Note: This is the default directory; if you have installed Verify to a different location, this path will also be different.
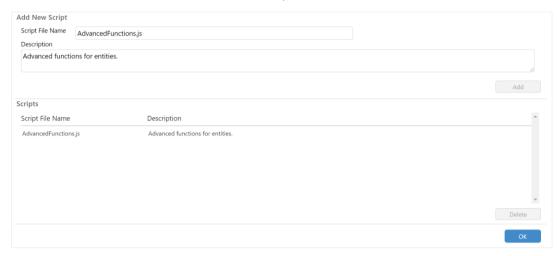
2.   In Layout Mode, click **Manage Scripts** on the toolbar.

     The **Script Manager** appears.

3.   In the **Add New Script** section:

     a.   In the **Script File Name** box, type the file name of the JavaScript file containing the calculation.

     b.   In the **Description** box, type a short description of the scripts in the file.

c.   Click the **Add** button to add it to the P2 Verify database.
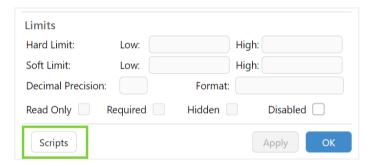
4.   You may now reference the newly added JavaScript file in a tab (see page 22).
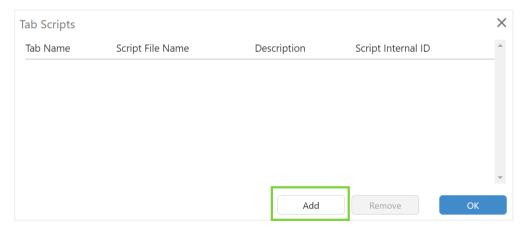
Using Custom Scripts in a Tab

To use a custom script, it needs to be assigned or added to a tab. A script file added to a tab will be included in the HTML that is rendered at runtime.

1.   In Layout Mode, right-click the tab in which you want to include the script file, and select **Edit Tab**.

2.   In the **Tab Editor**, click **Scripts** in the lower left corner.
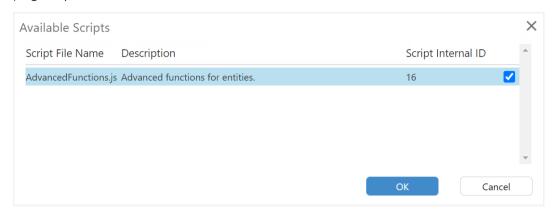
The **P2 Verify - Tab Scripts** editor appears, showing all the scripts that have been assigned to this tab.

3.   To assign the script to the tab, click **Add**.

The **P2 Verify - Available Scripts** dialog box appears, listing all the scripts that are available in the P2 Verify database. These have been previously added using the Script Manager (see page 21).

Available Scripts			×
Script File Name	Description	Script Internal ID	
AdvancedFunctions.js	Advanced functions for entities.	16	☑

OK   Cancel

4. Click the check box belonging to the script that you want to use.

5. Click **OK**. The script is now included in the tab.

6. Click **OK**.

7. Click **OK** in the Tab Editor.

8. Click **Save** in the Verify toolbar to save the changes in your layout.

REMOVING A SCRIPT FROM A TAB

1. In Layout Mode, right-click the tab from which you want to remove the script file, and select **Edit Tab**.

2. In the Tab Editor, click **Scripts** in the lower left corner.

   The **P2 Verify - Tab Scripts** editor appears, showing all the scripts that have been assigned to this tab.

3. To remove the script from the tab, select the script, and then click **Remove**.

4. Click **OK**.

5. Click **OK** in the Tab Editor.

6. Click **Save** in the Verify toolbar to save the changes in your layout.

Editing Tabs

Layout Mode gives administrators access to the fully-featured visual designer. Intuitive navigation and rich selection of visual controls for entities help you build the data capture interface you need.

Saving

Click the **Save** button to save your changes back to the database before you switch to Read-Only Mode to view your interface. Otherwise you will lose your latest changes.

This will only save your data to the Verify database, it will not submit any data for validation.

Selecting
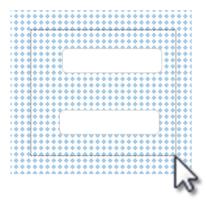
▸   To select a single object, click it.

As soon as you select the control, a black rectangular border appears around it.

▸   To select multiple objects, hold the **CTRL** key down and click all the objects you want to select.

Alternatively, while not having any objects selected, drag your mouse on the grid to create a box that covers all the objects. This is called a lasso.

Note: When objects are grouped into a group box, you cannot use the lasso tool to select them. Where possible, add your group boxes last.

Moving

You can only move objects within a selected tab. You cannot move an object from one tab to another tab.

▸   To move an object, first click on it to select it, and then either drag it to the desired location or use the arrow keys to move it to the next gridline in the desired direction.

Click and drag

When you click and drag an object, it is automatically snapped to the layout grid.

Arrow keys

Press an arrow key to move an object to the next grid line (snap-to-grid).

If you don't want to use snap-to-grid, hold the **CTRL** key down and press the arrow key to move an object by one pixel.

Copying and Cloning

You can clone objects by copying existing ones and pasting them on the page. Copying and pasting makes it faster and easier to create new objects of the same type, which you can then edit. When you copy an object to the clipboard, all its properties are copied along with it.

▸ To copy an object, first click on the object to select it, press **CTRL + C** to copy the object, and then press **CTRL + V** to paste a duplicate of the object on a page.

The location of the duplicated object is slightly off to the lower right of the original object.

Resizing

You can resize a single object or several objects.
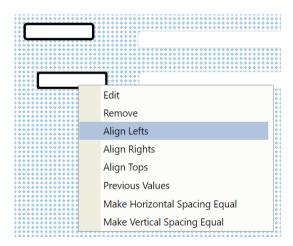
1. Select your objects.

2. Hover the mouse pointer over the edges or corner of the object.

   The pointer will change to display the resize direction.

3. Hold down the left mouse button and drag the mouse in the required direction.

You can also resize an object by changing the Width and Height fields in the Entity Editor to the desired size.

Aligning

▸ To align multiple objects, first select the objects and then right-click to display the shortcut menu.

Select from the following commands:

Edit

Open the Entity Editor. Any changes you make will apply to all selected entities.

Remove

Remove the selected entities from the page.

Align Lefts

Align the left edges of the selected entities to the left edge of the entity that is furthest to the left.

Align Rights

Align the right edges of the selected entities to the right edge of the entity that is furthest to the right.

Align Tops

Align the upper edges of the selected entities with the upper edge of the entity that is furthest up the screen.

Align Bottoms

Align the lower edges of the selected entities with the lower edge of the entity that is furthest down the screen.

Make Horizontal Spacing Equal

Make the spacing equal horizontally between the entity furthest to the left and the entity furthest to the right.

Make Vertical Spacing Equal

Make the spacing equal vertically between the entity furthest up the screen and the entity furthest down the screen.

Removing

You can select and remove multiple entities.

To remove one or more controls from the page:

1.   Select the controls you want to remove.

2.   Right-click the control.

3.   Select **Remove** from the shortcut menu.

Alternatively, you can select the fields and then press the **Delete** key.

Remember to **Save** your changes back to the database.

> **Note:** There is no **Undo** feature. If you accidentally delete something that you want to keep, then don't click the **Save** button. Simply refresh the screen and the object will reappear. You will however, lose any other unsaved changes.

Tabbing Order

In Input Mode, operators can navigate through entities by using the Tab key, following a sequence based on the **Tab Order** property of each entity, as determined by the Verify administrator.
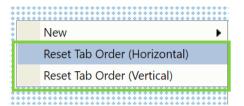
Alternatively, an administrator can allow P2 Verify to automatically set the tabbing order:

1. Right-click on the background in Layout mode, while no entity is selected.

   The tab reindexing actions appear in the shortcut menu.

2. Click the **Reset Tab Order** option that you want to implement.

The tabbing order can be reset either horizontally or vertically.

Horizontal: The tabbing order will start at the upper leftmost entity on the page, and then proceed horizontally before moving down the page.

For example:

```
1 2 3
4 5 6
7 8 9
```

Vertical: The tabbing order will start at the upper leftmost entity on the page, and then proceed down the page before moving across the page.

For example:

```
1 4 7
2 5 8
3 6 9
```

Note: The tab order will be reset for all entities, all tabs and all layout versions of the current data area.

Creating a New Layout Version

The Layout Version number is displayed in the upper left of the screen when in Layout Mode.

To create a new Layout Version:

1. Click the **New Layout** ⊕ button on the Verify toolbar.

2. When prompted, click **OK**.

The system copies the entire layout into the new version, and you can now make changes as required.

Note: You need to be careful with your versioning so that you don't inadvertently lose some of your controls. We suggest that you use the Layout Version Tracker to record your changes.

For more information, refer to *Layout Versions*.
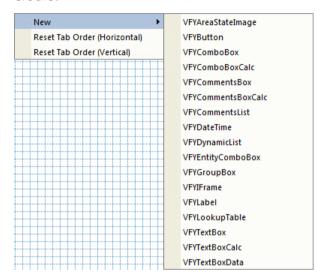
Defining Entities

In P2 Verify entities are represented as various visual controls. By specifying the properties of a control, you can define a data type, function and constraints of an entity, as well as change the visual appearance and behaviour of its associated visual component.
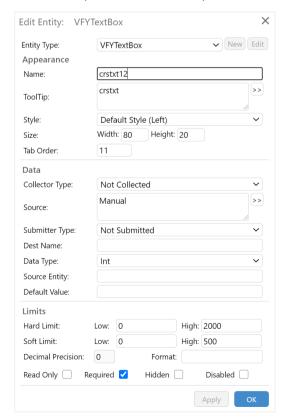
To add a new entity to your page layout:

1.   In a blank spot on the layout grid, right-click on the background.

2.   In the shortcut menu, mouse-over the **New** command to display the entities that you can create.

3.   Click the type of entity you want to create.

A new entity is added to the layout, and the **Entity Editor** appears.

4. In the **Entity Editor**, enter the properties as appropriate; these may differ for each entity. For a description of the properties, refer to *Editing Entities* (see page 48).

5. When you have finished, click **OK**.

6. Click **Save** 🖫 on the toolbar to save the changes to your layout.

Area State Images

The **VFYAreaStateImage** entity provides a visual indication of the data state of a Data Area. By default, a PNG image (*.png) with the same name as the state will be used to represent the state of the data area, but this can be overridden by manually defining the mapping.
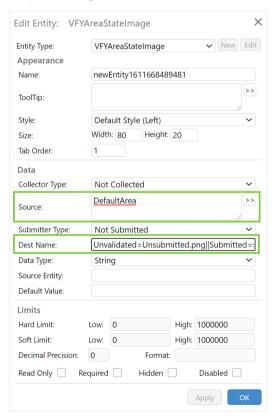
When an operator clicks the image, they are redirected to that data area, in Input mode.

To specify the data area, so that the Area State Image can look up the data state:

1. Open the Entity Editor.

2. In the **Source** property, type the name of the Data Area. For example, *Unvalidated*.

    Note: The Data Area must have the same schedule (that is, the same report start and end date) as the Data Area of the Area State Image, otherwise the state may not be retrieved correctly.

3. Map the images to each of the transition states that have been specified in the Data Area.

Set the **Dest Name** property in the following pattern:

```
State1=ImageFile1||State2=ImageFile2||State3=ImageFile3...
```

where `ImageFile` is a filename + extension, and `State` refers to the Action Text defined in the transition state.

For example:

```
Unvalidated=unvalidated.png||Validated=validated.png||Submitted=submitted.png
||Authorised=authorised.png||Approved=approved.png
```

4. Click **OK** in the Entity Editor.

5. Click **Save** on the Verify toolbar to save your changes.

6. Copy the image files corresponding to each state, to the `[Installation Directory]\Web\Images` directory.

    By default, this is:
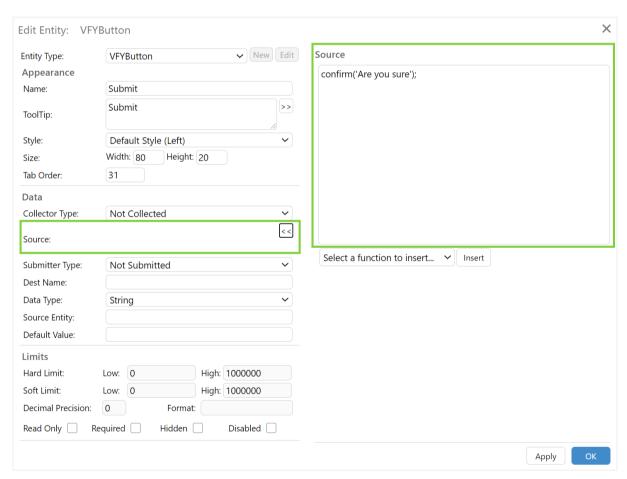
    `C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web\Images\`

Buttons

A button (VFYButton) allows an operator to perform a predefined action. For example, you may want to send an email to another person when a value reaches a critical level, such as an alert for pollution thresholds.

To make a button meaningful, you need to configure it with an action to perform when it is clicked. This is done by writing some JavaScript in the **Source** field of the Entity Editor.

When an operator clicks on the button, the JavaScript specified as the data source will be executed.

To configure an action for a button:

1.   Open the Entity Editor.

2.   In the **Source** box, click the **>>** button.

3.   In the **Expression Editor**:

   a.   Type the JavaScript that you want to execute.

   b.   When you have finished, click **OK** to return to the Entity Editor.

4.   Click **OK** again to apply the changes.

5.   Click **Save** 💾 on the toolbar to save your changes.

Calculated Combo Box

The **VFYComboBoxCalc** entity differs from a standard VFYComboBox in that it allows the value of the combo box to be set using the results of a calculation, if its value is blank or set to blank. This is useful for setting a default value when no input is made in that field, or to set the default value based on the value in another field.

The calculated combo box is editable if the Read Only check box is not selected. In this case, the user may override the calculated value by manually selecting a different item in the list. When this happens, the background colour of the calculated combo box is set to a pale yellow after the user tabs out of the entity. The selected value will not be changed, even if a source value in the calculation is changed (i.e. the value will not be recalculated once the value is overwritten).

▶   To force the calculated combo box to recalculate again, select the blank item in the calculated combo box.

For example, enter the following calculation in the **Calculation** field in Entity Editor, to set the default value of the calculated combo box entity based on the value in another text box:

```
if (Value('txtChoke') > 0) {return 'Open';} else {return 'Closed';}
```

In this example, the value of the calculated combo box is set to **Open** if the value in the **Choke** text box is greater than 0, otherwise the value is set to **Closed**. If the user overrides the value by selecting a different value, the background colour changes to a pale yellow, and the value will no longer update.

▶   To recalculate the value again, select the blank item in the calculated combo box.

Calculated Comments Box

The **VFYCommentsBoxCalc** entity differs from a standard VFYCommentsBox in that it allows the value of the comments box to be set using the results of a calculation, if its value is blank or set to blank. This is useful for setting a default value when no input is made in that field, or to set the default value based on the value in another field.

The calculated comments box is editable if the Read Only check box is not selected. In this case, the user may override the calculated value by manually changing the text in the entity. When this happens, the background colour of the calculated comments box is set to a pale yellow once the user tabs out of the entity. The overwritten value will not be changed, even if a source value in the calculation is changed (i.e. the value will not be recalculated once the value is overwritten).

▸ To force the calculated comments box to recalculate again, delete all the text in the calculated comments box.

For example, enter the following calculation in the **Calculation** field in Entity Editor, to set the default value of the calculated comments box entity based on the value in another text box:

```
if (Value('txtChoke') > 0) {return 'Well is Online';} else {return 'Well is Shut-in';}
```

In this example, the value in the calculated comments box is set to **Well is Online** if the value in the **Choke** text box is > 0, otherwise the value is set to **Well is Shut-in**. If the user overwrites the value, the background colour changes to a pale yellow, and the value will no longer update.

▸ To recalculate the value again, delete all the existing text in the calculated comments box.

Combo Box

Combo Boxes offer fast and easy selection of a wide range of information. They are a standard component in most web applications.

The VFYComboBox entity can derive the fields in its drop-down list from either manual sources or from a database.

For a manual combo box (see page 32), you add the values that will appear in the list.

A data source combo box (see page 34) will retrieve the values from the database using a specified SQL statement.

Manual Source
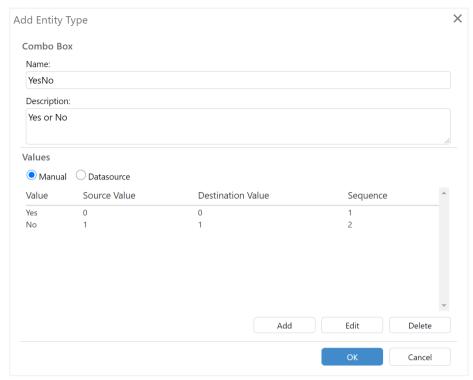
To populate a combo box manually:

1. Open the Entity Editor.

2. Next to the Entity Type property, click the **New** button.

This allows you to create a new Combo Box list.

Alternatively, click **Edit** if you want to edit an existing list.

3.   In the **Combo Box Values Editor**, enter the following parameters:
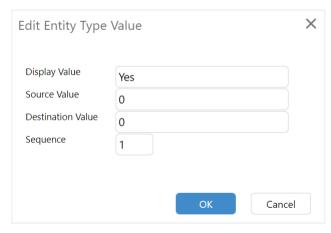
Name
Unique name for the combo box (not available if you are editing an existing combo box).

Description
A more verbose description of the purpose of the combo box and the data it contains.

4.   In the **Values** group, select the **Manual** radio button.

5.   Click **Add** to add a new value to be displayed in the list.

The **Add Entity Type Value** dialog box appears.

6.   Enter the values you want to display in the combo box. All fields are required.

Display Value
The value that is displayed in the combo box. For example, you may want to display a value bracket, such as 90-95%.

Source Value

An alternate value that may be used for other purposes, such as functions and calculations. For example, you may want to display 90-95% to the user, but you may want to use the minimum value in the range (90) as an input to another field.

Destination Value

The value that is submitted to the destination tag in P2 Server or a data store repository. For example, you may want to only submit the highest value in a range (95).

Sequence

The order in which this value appears in the combo box.

Note: Usually, the Display Value, Source Value, and Destination Value will be the same.

7.   When you have finished, click **OK**.

8.   Keep adding more values to the combo box as required by clicking **Add**, or change any values already entered by clicking **Edit**.

9.   When you have finished, click **OK**.

10.   Click **OK** in the Entity Editor.

11.   Click **Save** 💾 on the toolbar to save your changes.

Data Source
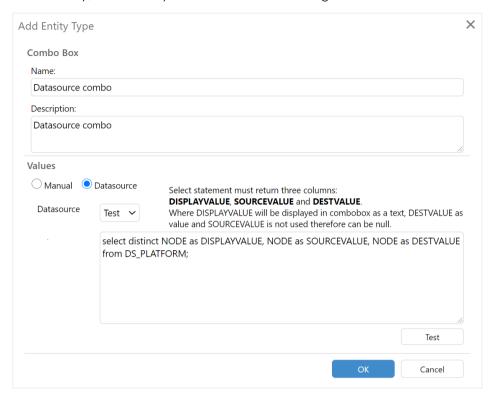
To populate a combo box from a data source:

1.   Open the **Entity Editor**.

2.   Next to the **Entity Type** property, click the **New** button.

This allows you to create a new Combo Box list.

Alternatively, click **Edit** if you want to edit an existing list.

Add Entity Type   ✕

Combo Box

Name:

Datasource combo

Description:

Datasource combo

Values

○ Manual ● Datasource

Datasource [Test ▾]

Select statement must return three columns:
DISPLAYVALUE, **SOURCEVALUE** and **DESTVALUE**.
Where DISPLAYVALUE will be displayed in combobox as a text, DESTVALUE as value and SOURCEVALUE is not used therefore can be null.

select distinct NODE as DISPLAYVALUE, NODE as SOURCEVALUE, NODE as DESTVALUE from DS_PLATFORM;

[Test]

[OK] [Cancel]

---

3.   In the **Combo Box Values Editor**, enter the following parameters:

Name
Unique name for the combo box (not available if you are editing an existing combo box).

Description
A more verbose description of the purpose of the combo box and the data it contains.

4.   In the **Values** group, select the **Datasource** radio button.

The **Datasource** dropdown list contains a list of data sources that are configured in the **web.config** file, located in:
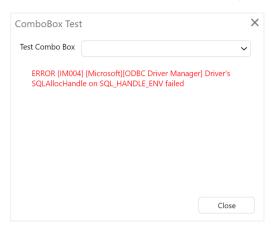
`C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web`

The list is contained within the `< verify><common><dataSources>` elements.

5.   From the drop-down list, select the appropriate data source from which you want to populate the combo box.

6.   In the **SQL Statement** box, type the SQL statement that you will use to query the database and derive the data with which to populate the combo box.

7.   Click the **Test** button to check that the SQL statement is correct and to preview the resulting combo box.

If the SQL statement contains an error, details of the error appear below the Test Combo Box.

ComboBox Test   ×

Test Combo Box [⌄]

ERROR [IM004] [Microsoft][ODBC Driver Manager] Driver's
SQLAllocHandle on SQL_HANDLE_ENV failed

[Close]
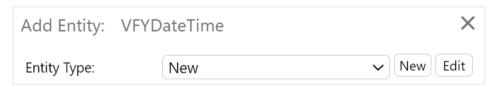
8.   Click **Close**, modifying the SQL statement if needed.

9.   When you have finished, click **OK**.

10.   Click **OK** in the Entity Editor.

11.   Click **Save** 💾 on the toolbar to save your changes.

Date/Time

The **VFYDateTime** entity can be modified to use a customised Date/Time format string, which is available from the Entity Type drop-down list in the Entity Editor.

To configure a Date/Time entity:

1.   Add a **VFYDateTime** entity to your layout.

2.   Near the **Entity Type** field, click the **New** button.

Add Entity: VFYDateTime   ×

Entity Type: [New   ⌄] [New] [Edit]

This allows you to create a new Date/Time format.

Alternatively, click **Edit** if you want to edit an existing format.

3.   In the **Add Entity Type** dialog box, enter the following parameters:

Name
Unique name for the Date/Time format.
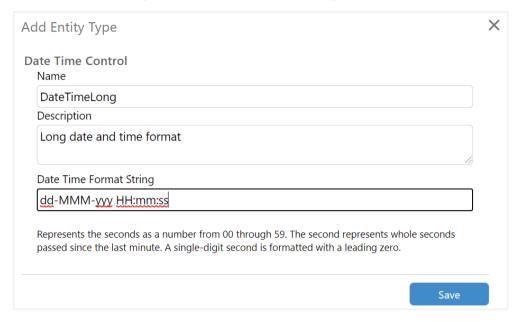
Description
A more verbose description of the purpose of the Date/Time format and the data it contains.

Date Time Format String

The format string specifying how you want the date/time entity to be formatted. For the available format strings, refer to *Appendix A* (see page 105).

Add Entity Type   ✕

Date Time Control
Name

DateTimeLong

Description

Long date and time format

Date Time Format String

dd-MMM-yyy HH:mm:ss

Represents the seconds as a number from 00 through 59. The second represents whole seconds passed since the last minute. A single-digit second is formatted with a leading zero.

Save

4.   When you have finished entering the information, click **Save**.

5.   In the Entity Editor, enter the remaining information as required.

6.   Click **OK**.

7.   Click **Save** on the Verify toolbar to save your changes.

If the **VFYDateTime** entity is not set as Read Only, the user may also manually type in the date/time without using the Date/Time picker. The entered value must have the expected format, which will be validated when the user tabs out of the entity.

Dynamic Lists

The **VFYDynamicList** entity can use the following controls:
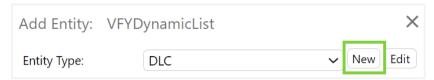
- VFYActiveComboBox
- VFYComboBox
- VFYComboBoxCalc
- VFYCommentsBox
- VFYCommentsBoxCalc
- VFYDateTime
- VFYEntityComboBox
- VFYLabel
- VFYTextBox

VFYTextBoxCalc To configure a Dynamic List:

1.   Add a **VFYDynamicList** entity to your layout.

2.   Near the **Entity Type** field, click the **New** button.

This allows you to create a new Dynamic List.

Alternatively, click **Edit** if you want to edit an existing dynamic list.
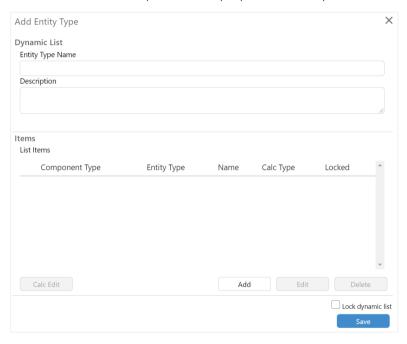
3.   In the **Add Entity Type** dialog box, enter the following parameters:

Entity Type Name
Unique name for the Dynamic List.
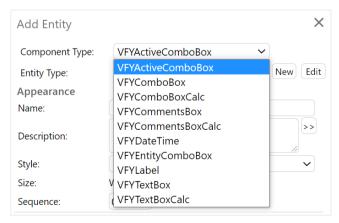
Description
A more verbose description of the purpose of the Dynamic List and the data it contains.

4.   Click **Add**.

The Add Entity dialog box appears. This is similar to the Entity Editor, but has the added option of **Component Type**.

At this stage, you can add any number of component entities to the Dynamic List. To add an entity:

a.   Select an entity from the Component Type drop-down list.

The dialog box refreshes to display the edit options for that entity .

b.   Configure the entity as required.

c.   Click **OK**.

You are returned to the **Add Entity Type** dialog box, and the entity you just configured appears in the item list.

5.   Continue adding entities as required.

6.   Make any other changes you require.

Calc Edit: Adds a row at the bottom of the list that calculates the sum or average of the values in the column, as specified. This option is only available for text box or text box calc entities.

Lock dynamic list: Locks the structure of the dynamic list. This will editors from deleting or making structural changes to an item in the list. It does however allow editors to add, edit, and delete new items in the list.

Use with caution! Once locked, the dynamic list cannot be unlocked.

7.   When you have finished adding entities, click **Save**.

8.   In the Entity Editor, enter the remaining information as required.

9.   Click **OK**.

10.   Click **Save** on the Verify toolbar to save your changes.

Linked Combo Box

The **VFYLinkedComboBox** entity obtains its item list from a hierarchy repository, such as P2 Server. It can only be used within a Dynamic List, and must be used with an Ad Hoc Combo Box. A Linked Combo Box contains a list of parent entities, while the associated Ad Hoc Combo Box contains a list of children entities.

A Linked Combo Box interacts with an Ad Hoc Combo Box in the following way:

•   Items of a Linked Combo Box are retrieved from a hierarchy repository indicated with a Hierarchy Source Pattern (see page 53) provided in the **Source** property of a Linked Combo Box.

•   The selected item of a Linked Combo Box is set automatically by a dynamic list collector during collection if a Linked Combo Box has been added in the first column, based on a Dynamic List's **Source** property and the value corresponding to the current row of the dynamic list. Otherwise, the selection is preformed manually by a user.

•   Items of an Ad Hoc Combo Box are initially collected based on its **Source** property. When the selection of a Linked Combo Box changes, the items of an Ad Hoc Combo Box indicated in its **Dest Name** property are reloaded.

•   The selected item of an Ad Hoc Combo Box is set automatically by a dynamic list collector during collection if an Ad Hoc Combo Box has been added in the first column, based on a

Dynamic List's **Source** property and the value corresponding to the current row of the dynamic list. Otherwise, the selection is preformed manually by a user.
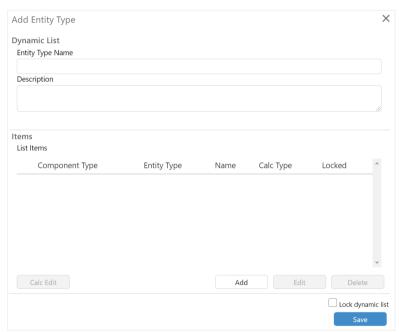
- Other values of entities in the same row of a dynamic list are initially collected based on their **Source** properties, provided the Source property value does not contain the @ character followed by a number. When the selection of an Ad Hoc Combo Box changes, the values of entities that contain the aforementioned sequence of characters in their **Source** property will be reloaded.

To add a Linked Combo Box to a Dynamic List:

1. In a blank spot on the layout grid, right-click on the background, and select **VFYDynamicList**.

    Alternatively, double-click an existing dynamic list entity to open it.

2. In the **Entity Editor**, click **New**.

    Alternatively, if you are editing an existing dynamic list, select the entity type from the drop-down list and click **Edit**.

3. In the Dynamic List Editor, click **Add** to add a new entity to the **Dynamic List**.

Add Entity Type	✕
Dynamic List	
Entity Type Name	
[ ]	
Description	
[ ]	

Items				
List Items				
Component Type	Entity Type	Name	Calc Type	Locked

[Calc Edit]   [Add]   [Edit]   [Delete]

☐ Lock dynamic list

[Save]

The Entity Editor opens.

4. In the **Component Type** drop-down list, select **VFYActiveComboBox**.

5.   In the **Entity Type** drop-down list, select **VFYLinkedComboBox**.

6.   Specify the **Sequence** as **0**.

     If used, a Linked Combo Box must always be the first column in the dynamic list.

7.   To specify the location in the hierarchy where the item list is retrieved from, set the **Source Name** property according to the Source Pattern (see page 53).

8.   Enter the **Dest Name** property according to the following format following a *Hierarchy Source Pattern* (see page 53), where the whole expression is additionally prefixed with the name of a **target combo box name** followed by a double pipe (||) character sequence. The target combo box name refers to the accompanying Ad Hoc Combo Box entity. The item list in the Ad Hoc Combo Box will be driven from the selection in the Linked Combo Box. In the value of the **Dest Name** property of a Linked Combo Box you can use a placeholder @parent, which will be replaced with the value of a selected item, as well as a placeholder @hierarchy, which will be replaced with the name of a hierarchy collector of the combo box.
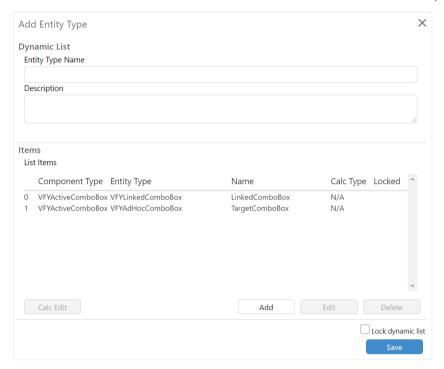
     For example, a Linked Combo Box may contain a list of platforms, while the Ad Hoc Combo Box contains a list of wells. Selecting a platform in the Linked Combo Box may then filter the list of wells in the Ad Hoc Combo Box, so that only wells belonging to the selected platform will appear in the Ad Hoc Combo Box.

9.   Click **OK** to add the Linked Combo Box to the Dynamic List.

10. Add an Ad Hoc Combo Box entity next to the Linked Combo Box, specified as follows:

     –   **Component Type**: **VFYActiveComboBox**

     –   **Entity Type**: **VFYAdHocComboBox**

     –   **Name**: Set this to the *Target Combo Box Name* as defined in the *Linked Combo Box*.

–   **Sequence**: The target Ad Hoc Combo Box must appear immediately after the Linked Combo Box. For example, if the linked combo box is set to *0*, the ad hoc combo box must be *1*.

–   **Source Name**: Configure this according to the Source Pattern (see page 53), such that the resulting list of entities include all possible child entities returned from the Linked Combo Box. This ensures that the value can be loaded correctly after a page refresh.

11.   Click **Save** to close the Dynamic List Editor.

12.   In the Entity Editor, make any other required changes, and then click **OK**.

13.   Click **Save** in the Verify toolbar.

14.   Add **LinkedComboBox.js** and **AdHocComboBox.js** to the list of scripts for the current tab.

These scripts are provided as part of the installation process, and by default are located in *C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web\Scripts*.

For more information, refer to *Using Custom Scripts in a Tab* (see page 22).

Ad Hoc Combo Box

The **VFYAdHocComboBox** entity, like the Entity Combo Box, obtains its item list from a hierarchy repository, such as P2 Server.

In addition, it can populate the values in other columns in the row by fetching the templated attribute or property of the selected entity.

This combo box can only be used within a Dynamic List, and if often used in conjunction with a Linked Combo Box (see page 39).

To add an Ad Hoc Combo Box to a Dynamic List:
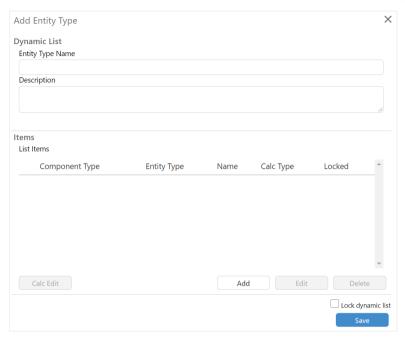
1.   In a blank spot on the layout grid, right-click on the background, and select **VFYDynamicList** .

Alternatively, double-click an existing dynamic list entity to open it.

2.   In the **Entity Editor**, click **New**.

Alternatively, if you are editing an existing dynamic list, select the entity type from the drop-down list and click **Edit**.
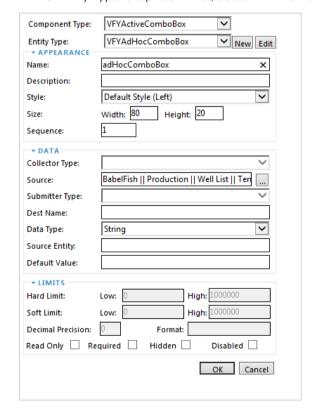
3.   In the Dynamic List Editor, click **Add** to add a new entity to the **Dynamic List**.

The Entity Editor opens.

4.   In the **Component Type** drop-down list, select **VFYActiveComboBox.**

5.   In the Entity Type drop-down list, select **VFYAdHocComboBox**.

6. Specify the **Sequence** as **1** if this entity is being used with a Linked Combo Box, otherwise specify it as **0**.

    An Active Combo Box must always be the first column in the dynamic list.

7. To specify where the item list is to be retrieved from, set the **Source** property for a Hierarchy Repository collector, prefixing it with the name of the collector followed by a double pipe (||) character sequence.

8. Click **OK** to add the Ad Hoc Combo Box to the Dynamic List.

9. Add the desired number of entities after the Ad Hoc Combo Box in the dynamic list, with a **Sequence** of 1 or greater, and with their **Source** 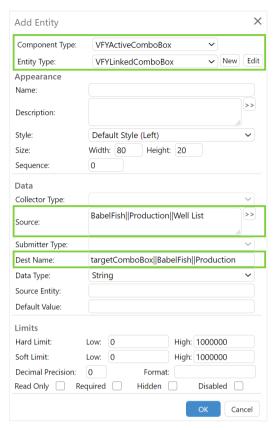property set for a Hierarchy Repository collector, prefixing it with the name of the collector followed by a double pipe (||) character sequence. See the section Source and Destination Name (see page 53) for more details regarding Ad Hoc Combo Box.

    Note: Only those entities that need to be collected dynamically when the entity is selected from the VFYAdhocCombobox, need to use the Value Source Pattern.

10. Click **Save** to close the Dynamic List Editor.

11. In the Entity Editor, make any other required changes, and then click **OK**.

12. Click **Save** on the Verify toolbar.

13. Add **AdHocComboBox.js** to the list of scripts for the current tab.

    This script is provided as part of the installation process, and by default is located at the following location:
    C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web\Scripts

    For more information, refer to *Using Custom Scripts in a Tab* (see page 22).

Editable Calculation Text Box

The **VFYTextBoxCalc** entity is editable if the Read Only check box is not selected.

In this case, the user may override the calculated value by manually changing the value in the entity. When this happens, the background colour of the calculation text box is set to a pale yellow when the user tabs out of the entity. The overwritten value will not be changed, even if a source value in the calculation is changed (i.e. the value will not be recalculated once the value is overwritten).

▶   To force the calculation text box to recalculate again, delete the value in the calculation text box.

Entity Combo Box

The **VFYEntityComboBox** entity is used to dynamically display a list of child entities from a hierarchy repository, such as P2 Server. It can also be used within a Dynamic List.

To configure the Entity Combo Box:

1. Open the **Entity Editor.**

2.   Set the **Source** property according to the Hierarchy Source Pattern (see page 53).
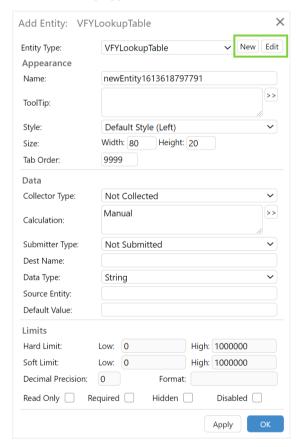
Lookup Tables

Lookup Tables (VFYLookupTable) work in much the same way as they would in a spreadsheet application, allowing you to use one or more values to derive another.

To configure a Lookup Table:

1.   Add a **VFYLookupTable** entity to your layout.

2.   Near the **Entity Type** field, click the **New** button.

This allows you to create a new Lookup Table.

Alternatively, click **Edit** if you want to edit an existing Lookup Table.

3.   In the **Add Entity Type** dialog box, enter the following parameters:

Name
Unique name for the Lookup Table.

Description
A more verbose description of the purpose of the Lookup Table and the data it contains.

Rows

The number of rows for the Lookup Table, including the header row.
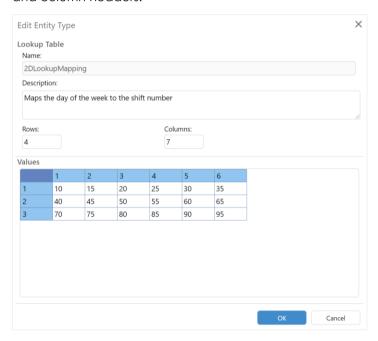
Columns

The number of columns for the Lookup Table, including the header column.
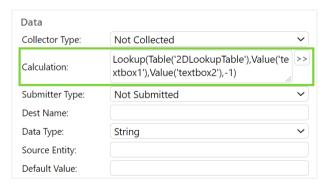
Values

The values you want to store in the cells of the Lookup Table. Each value must be numeric-only.

Click in each cell and type in the value. The cells that have a blue background are the row and column headers.

4.   When you have finished entering the information, click **OK** to return to the **Entity Editor** for the VFYLookupTable.

5.   In the **Calculation** field, enter the calculation details to do the lookup on the VFYLookupTable itself, or leave it as *Manual* to use a separate calculated text box (VFYTextBoxCalc) to look up the value.

To make entering the calculation easier, use the **Expression Editor** by clicking the **>>** button at the end of the **Calculation** field, and select the **Lookup** or **LookupStringBased** functions from the drop-down list.

For information on how to use the Lookup Table, refer to *Referencing Lookup Tables* (see page 62).

6.   When you have finished entering the calculation, click **OK** to return to the Entity Editor.

7.   Enter the remaining information as required.

8.   Click **OK**.

9.   Click **Save** on the Verify toolbar to save your changes.

Text Box Data
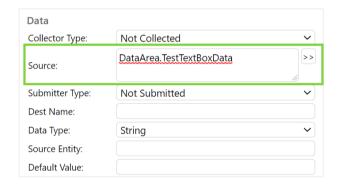
The **VFYTextBoxData** entity is a read-only text box that displays the value of an entity in a different Data Area.

For example, you may want to compare total gas production from different areas.

As the report date of the Data Area containing the Text Box Data is used to fetch the entity's value, the Data Area from which the value is to be fetched from must have the same schedule (that is, the same report start and end date).

To specify the entity to retrieve the value from, configure the **Source** property in the Entity Editor in the following format:

```
Data Area Name.Entity Name
```
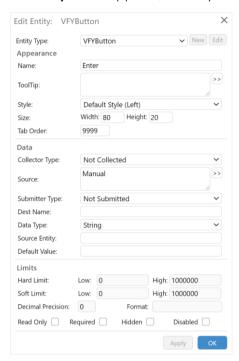
Editing Entities

To modify any property of an entity:

1.   In Layout Mode, double-click the entity or right-click it and select **Edit**.

       The **Entity Editor** appears, where you can change any of the properties for the entity.

Edit Entity:	VFYButton		✕

Tip: To edit properties for several entities at once, select all the entities first, then double-click or right-click and choose **Edit**. The changes you make will apply to all selected entities.

2.   In the Entity Editor, edit the properties as required (see page 48).

       Note: Details for a field may differ slightly between entity types. For example, a **Label** entity type will display a **Text** property where you enter the text you want to appear on the label, but most other entities will display a **Tooltip** property for text to display in that entity's tooltip.

3.   When you have finished, click **OK**.

4.   Click **Save** 💾 on the toolbar to save changes to your layout.

Entity Properties

The Entity Editor contains a number of fields in which you can define the properties for that entity. Details for a field may differ slightly between entity types. For example, a **Label** entity type will display a **Text** property where you enter the text you want to appear on the label, but most other entities will display a **Tooltip** property for text to display in that entity's tooltip.

The properties are described below.

ENTITY TYPE

The Entity Type indicates the type of a visual control that will be associated with an entity.

The **New** and **Edit** buttons are also available for some controls, such as Combo boxes (VFYComboBox, VFYComboBoxCalc), date/times (VFYDateTime), lookup tables (VFYLookupTable), and dynamic lists (VFYDynamicList).

Note: When you are editing most entities, you cannot change this value. For example, you cannot change a label for an input cell. To do this, you need to first remove the old entity and then create a new entity of the type you require. For dynamic lists, combo boxes, date/times, and lookup tables, the Entity Type can be changed, and is used to pick from the available types that are configured in the system.

APPEARANCE

The following properties affect how the entity is presented in the interface.

Name

The unique name that identifies the entity. This is how the entity is referenced in the system. For example, calculated entities use this property to specify which entities provide input for the calculation.

For details on what characters are permitted in entity names, refer to *Restrictions and Performance Recommendations* (see page 4).

ToolTip

A brief description of the entity, to be used for the tooltip in Input mode. This will also be used in prompts to the user about that particular control (such as if it fails to validate before submission).

In some circumstances, you may find that you want a different reference name (*friendly name*) for the control, rather than the text string used as a tooltip. In this circumstance, you can insert a double-colon *token* (**::**) to separate both a friendly name and a tooltip for each control:
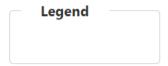
`FriendlyName::Tooltip`

For example, you may set the text string to be "*T9 Overflow Tank::This is the Tank used as overflow for T2 and T3.*" When the page loads, the user will see the following tooltip: "*This is the Tank used as overflow for T2 and T3.*" If they then get a validation error message or similar, the entity referenced will be "*T9 Overflow Tank*".

Legend

This property only applies to group boxes (VFYGroupBox), and determines the text that is displayed as the group box heading.

Text

This property only applies to labels (VFYLabel), and determines the text that is used as the visible label.

Style

The predefined format of the entity. This only affects the appearance of the entity in the system, such as the font size and colour. Several styles have been provided, however not all of them are applicable to every entity. Use the name of the style as a guide to indicate what entities they apply to. Initially, the entity will have *Default Style (Left)* applied.

Size

The width and height dimensions, in pixels, of the entity.

Tab Order
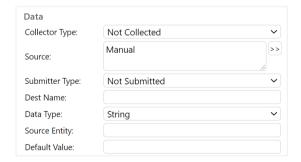
The order in which the cursor moves when the Tab key is pressed. This tab order applies only to entities within the selected tab. For example, you could have 20 entities on the Production Overview tab, and 10 entities on the Well Head Data tab. Tab Order 1 indicates where the cursor is initially positioned on a selected report. Tab Order 2 indicates where the cursor will move when the Tab key is first pressed, Tab Order 3 will be the next move, and so on.

For tabs, this defines the left-to-right order in which they appear on the layout, with 0 being the leftmost tab.

Tip: P2 Verify can create the tab sequence automatically. You can do this by right-clicking on the layout with no entity selected. For more information, refer to *Tabbing Order* (see page 26).

DATA

The following properties affect the type of data represented by the entity. These properties do not apply to tabs.

Collector Type

Specifies which Collector is used to retrieve the value of the entity. The list is populated by the Collectors configured during installation. A value of *Not Collected* means that the value of the entity must be entered manually by an operator.

Source

How the entity's value is obtained. This can be one of three options:

– **Manual:** An operator must manually enter the value for this entity.

– **Tagname:** If you want the value to be collected automatically, specify the tag name for the entity in the historian.

– **Calculated:** If you want this entity to be a calculation derived from values entered into other entities, enter the calculation in the **Expression Editor** dialog box. For more information, refer to *Calculations* (see page 59).

Calculation

This property only applies to calculated text boxes (VFYTextBoxCalc), lookup tables (VFYLookupTable), calculated combo boxes (VFYComboBoxCalc) and calculated comments box (VFYCommentsBoxCalc), and determines the entity's value. This can be one of two options:

– **Manual:** An operator must manually enter the value for this entity.

– **Calculated:** If you want this entity to be a calculation derived from values entered into other entities, enter the calculation in the **Expression Editor** dialog box. For more information, refer to *Calculations* (see page 59).

Submitter Type

The Submitter used to store the approved value of the entity. The list is populated by the Submitters configured during installation. A value of *Not Submitted* means the value of the entity is not distributed to any destination data source.

Dest Name

This should be either the tag name of the destination data source, or blank to indicate the value should not be distributed. Does not apply to labels, tabs, and group boxes.

Data Type

The type of value that is expected to be entered into this entity. This affects the validation of the entity. For example, if the data type is double and a string is entered, this will be picked up during validation. The following data types are allowed:

– **Int:** An integer with no decimal places.
– **Double:** A number with decimal places.
– **String:** An alphanumeric text.

Source Entity

The name of the entity from which you want to copy the contents for the previous production day.

Default Value

Optional. A default value for this entity if it is a manual entity. This default value will be assigned to the entity when the report is first created by the system.

LIMITS AND FORMATTING

The following properties determine the valid ranges of data that can be entered into an entity, and formatting options. With the exception of *Disabled*, these properties do not apply to tabs.

Limits		
Hard Limit:	Low: 0	High: 1000000
Soft Limit:	Low: 0	High: 1000000
Decimal Precision:	0	Format: $#,###.00
Read Only ☐	Required ☐	Hidden ☐ Disabled ☐

Hard Limit

The lowest acceptable value and the highest acceptable value for this entity. For example, a volume cannot be less than zero, or more than the container can hold.

Max Chars

When editing a VFYTextBox or VFYCommentsBox where the Data Type is set to *String*, the Hard Limit changes to **Max Chars**. You will be able to set a Maximum Length value to be applied to that control. Users will only be able to enter a certain number of characters based on this limit. Otherwise, when the Data Type of any other entity is set to String hard limits do not apply.

Soft Limit

Values that you want to monitor for. For example, you may want the volume of a container to always be between a quarter and three quarters full. Values outside this range may be valid, but you may want to double check them if this condition occurs.

When the Data Type of an entity is set to *String*, soft limits do not apply.

Decimal Precision

If the entity is a numerical entity, the number of decimal places to which you want to round this value. When the value is saved, distributed, or referenced in a calculation, this is the full decimal precision of the value that is used. This field is only available to entities that are set to the Double Data Type.

Format

Text boxes that display a real or integer value may be configured to display the number in a specific format. For example, adding a **$** prefix, thousands separator (10,000), and display decimal precision. This allows the user to use a different decimal precision for saved and displayed values (e.g. save the value with 8 decimal places, but only display with 2 decimal places). This field is disabled if the Data Type is set to *String*. For more information, refer to *Formatting Options* (see page 54).

Read Only

Select this check box if you do not want this entity to be editable. This is often used when an automatically collected value is not intended to be modified.

Required

Select this check box if you want a value to always be entered for this entity before a report can be saved. The value cannot be blank.

Hidden

Select this check box if you do not want this entity to be visible on the screen.

Disabled

> Select this check box if you want to disable this entity. Disabled entities cannot be edited and data for these entities will not be collected, regardless of configuration.

Source and Destination Name

The Entity Editor dialog window exposes two entity properties: **Source** and **Destination Name**, which are dependent on types of a collector and a submitter selected for an entity. Most entity types expect an adaptor supporting *data collector* requests selected as **Collector Type**, and an adaptor supporting *data submitter* requests selected as **Submitter Type**.

The notable exceptions include Dynamic List entities, Entity Combo Boxes, Linked Combo Boxes, and Ad Hoc Combo Boxes.

Dynamic List entities expect an adaptor that supports *dynamic list collector* requests selected as **Collector Type**, and an adaptor that supports *dynamic list submitter* requests selected as **Submitter Type**.

Entity Combo Boxes, Linked Combo Boxes, and Ad Hoc Combo Boxes use an adaptor supporting *hierarchy repository* requests to return collections of items. In case of Linked Combo Boxes and Ad Hoc Combo Boxes, which can only be used in dynamic list columns, and therefore they do not allow for explicit **Collector Type** configuration, the name of an adaptor must be used as a prefix of the **Source** parameter, for example:

```
{collector} || {collector specific source}
```

As entity combo boxes can be used outside of dynamic lists, their **Source** parameter must not be prefixed with a collector name. The **Collector Type** property indicates a hierarchy repository collector to be used for an entity. The **Source** has the following format:

```
{collector specific source}
```

DYNAMIC LIST COLUMNS

Individual columns of a dynamic list do not allow for setting **Collector Type** and **Submitter Type** properties. The values in a dynamic list depend on the **Collector Type** and **Submitter Type** used by an entire dynamic list.

Hierarchy Source Pattern

Certain types of adaptors that support *hierarchy repository* requests, use a standardized **Source** syntax for performing such requests. Such values allow for the following possible formats:

```
hierarchies: {hierarchies} || parents: {parents} || templates: {templates} || criteria:
{criteria} || pre: {prepended items} || post: {appended items} || cache: {cache life
duration} || delimiter: {delimiter character}
```

or

```
{hierarchies} || {parents} || {templates} || {criteria} || {prepended items} ||
{appended items} || {cache life duration}
```

When all the other parameters are prefixed with their type, as shown in the first example, they can be specified in any order and some parameters can be omitted.

When parameters are not prefixed, they are interpreted in the following order:

- Hierarchies
- Parents
- Templates

- Criteria
- Prepended items
- Appended Items
- Cache life duration

All parameters that accept multiple values should use a comma (**,**) character as a value separator.

The criteria (`{criteria}`) parameter lets the collector decide, based on a provided expression, whether a collection item should be returned as a part of the collection. For example, Dynamic List Collector (see page 86) uses JavaScript expressions, where a number and its surrounding curly braces (`{N}`) are replaced with the value of a column with that index from a row being evaluated. If the expression returns `true`, a row will be included in a dynamic list.

Formatting Options

The following syntax is used in the **Format** field in the Entity Editor:

Format syntax	Description
0	Digit, insignificant zeroes (0) are displayed
#	Digit, insignificant zeroes (0) are omitted
.	Decimal separator
,	Grouping separator (For example, thousands separator)
%	Percent (value is multiplied by 100)

There is a subtle difference between using **0** and **#** in the format. For example, the value **0.12345** with format **0,000.00** will be displayed as **0,000.12**. But with the format **#,###.##**, it will be displayed as **.12**.

For general usage, we recommend using a format similar to **#,###.00**, where insignificant zeroes in the decimal digits (Internet Explorer *before* the decimal separator) are not displayed, whereas insignificant zeroes *after* the decimal point are displayed.

If the Format field is empty, no formatting will be applied when the value is displayed.

Note: When the user is editing the field, the full decimal precision of the value is displayed, without any formatting. The formatting is only applied when the user leaves the field (such as clicking on another text box).

While the Format field is enabled in the Entity Editor for non-text box entity types, such as the combo boxes or comments boxes, certain formats may cause problems if not used wisely. For example, if the combo box is configured with a list of integer values, it does not make sense to apply a format **#.0** or **#%** because these formats will cause the item list to be blank, as the formatted display values do not match the item list.

Styles and Formats

The styles and formats of text displayed in P2 Verify are determined by a style sheet, which is created in the following directory during installation:

`C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web\Styles`
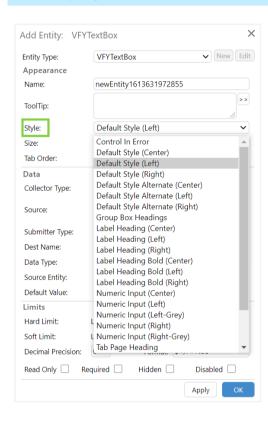
You can edit the style sheet to make new styles available, or to delete obsolete styles.

Note: If you are creating new styles, you also need to add additional rows in the STYLE table in the P2 Verify database.
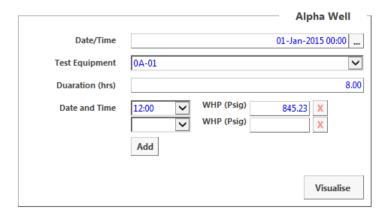
For instructions on how to apply a style to an entity, refer to *Editing Entities* (see page 48).

The built-in **Default Style** and **Default Style Alternate** style variants can be used for all types of entities (that is, simultaneously applied to text boxes, buttons, combo boxes, etc).

Tip: To apply the new style to more than one object on the page, select all the objects, right-click any object, and then select **Edit**. In the Entity Editor, change the **Style** as appropriate.

In the following image, all the controls have one of the Default Styles applied. Use Default Style Alternate (left, centre or right) for a different look and feel.

> **Note:** P2 Verify makes use of an external library to cache JavaScript and CSS resources. If you want to modify and test your own CSS classes, you may want to turn off the CSS caching feature of P2 Verify.

Configuring Colours

You can configure the colours of the various states of data (Ignored value, Value Out-Of-Limits, and so on) by editing the **web.config** file for the P2 Verify web application.

This is stored in the following folder:

C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web

It has a configuration section as follows:

```
<colours>
    <colour name="DISABLED_COLOUR">White</colour>
    <colour name="DISABLED_BORDER_COLOUR">#ACA899</colour>
    <colour name="DISABLED_TEXT_COLOUR">#ABABAB</colour>
    <colour name="CALCTEXTBOX_COLOUR">buttonface</colour>
    <colour name="REQUIRED_COLOUR">lightcyan</colour>
    <colour name="READONLY_COLOUR">EEEEEE</colour>
    <colour name="RAW_COLOUR">Black</colour>
    <colour name="MODIFIED_COLOUR">Blue</colour>
    <colour name="INVALID_COLOUR">Orange</colour>
    <colour name="ACCEPTED_COLOUR">Green</colour>
    <colour name="OUTOFLIMITS_COLOUR">Violet</colour>
    <colour name="IGNORED_COLOUR">GoldenRod</colour>
    <colour name="OUTOFHARDLIMITS_COLOUR">Red</colour>
    <colour name="DEFAULT_COLOUR">Black</colour>
    <colour name="ERROR_COLOUR">Red</colour>
    <colour name="FAILEDTOSUBMIT_COLOUR">Black</colour>
    <colour name="FAILEDTOCOLLECT_COLOUR">Red</colour>
</colours>
```

For each colour name, the value may be changed to any valid HTML Colour preset, or a hexadecimal RGB colour code.

Configuring the Date Format

You can configure the format of the date displayed in the header of a report by editing the **web.config** file for the P2 Verify web application.

This is stored in the following folder:

C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web

It has a configuration section as follows:

```
<!-- The following four nodes expect a valid System.DateTime format that will be
applied to the text printing out the current report date in each Verify tab top-right
corner.
For example, to change the weekly date format that is displayed in the tab corner from
23 Sep 2009 to
Mon, 22 Sep 09 enter the text "ddd, dd MMM yy" minus the quotes in the
weeklyTabDateTextFormat node-->

  <shiftTabDateTextFormat>ddd, dd MMM yyyy</shiftTabDateTextFormat>
  <weeklyTabDateTextFormat>ddd, dd MMM yy</weeklyTabDateTextFormat>
```

```
<monthlyTabDateTextFormat>MMMM yyyy</monthlyTabDateTextFormat>
<firstDayOfTheWeekTabDateTextFormat>ddd, dd MMM
yy</firstDayOfTheWeekTabDateTextFormat>
<quarterlyTabDateTextFormat>MMMM yyyy</quarterlyTabDateTextFormat>
<firstDayOfTheQuarterTabDateTextFormat>ddd, dd MMM
yy</firstDayOfTheQuarterTabDateTextFormat>
<yearlyTabDateTextFormat>yyyy</yearlyTabDateTextFormat>
<firstDayOfTheYearTabDateTextFormat>ddd, dd MMM
yy</firstDayOfTheYearTabDateTextFormat>
```

You can change the highlighted sections to rearrange the day, month, or year, as follows:

Property	Description
dd	Display two digits to indicate the day of the month.
ddd	Display three letters that indicate the day of the month.
MMM	Display three letters that indicate the month.
MMMM	Display the full name of the month.
yy	Display only the last two digits in the year.
yyyy	Display the full year.
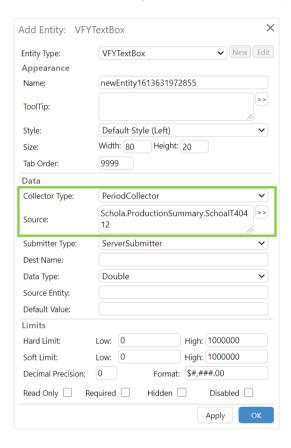
Collecting Data

Data capture can happen in three ways:

- Data is automated collected by P2 Verify at the end of reporting period (e.g. end of shift).
- Data is calculated by P2 Verify.
- Data is manually entered by operators.

To collect data, collectors and submitters (see page 64) need to be set up.

In the P2 Verify Layout Editor, the **Collector Type** and **Source** properties in the Entity Editor specify the source for the original value for the entity.

If the Collector Type is set to **Not Collected**, the entity contains a manual data entry field and the operator is able to set the value at any time.

If the Collector Type is specified and the Source is set to a tag name corresponding to an entity in the historian, the data value will be collected automatically and the entry field cannot be edited until that time.

Note: It is important to correctly configure the data source item, otherwise incorrect or bad values will be collected. Faulty metering may also lead to bad values.

Calculations

P2 Verify can perform calculations on the fly as operators enter data values. For the most part, the calculations are simple additions and subtractions of certain fields to create totals. But more complex calculations are also achievable using JavaScript, including such things as if-statement logic.
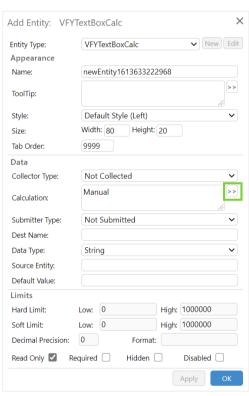
> **Note:** Entity names that contain a single quotation mark (**'**), ampersand (**&**), or colon (**:**) should not be used in calculations.

You can only set calculations for the **VFYTextBoxCalc**, **VFYLookupTable**, **VFYComboBoxCalc**, and the **VFYCommentsBoxCalc** entities. This is done by specifying the calculation in the Entity Editor.
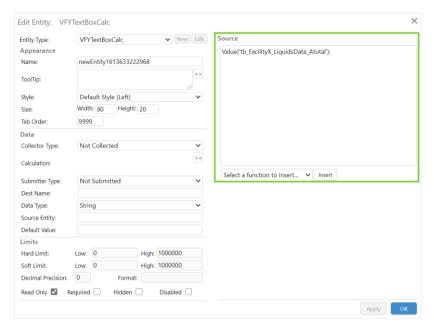
To configure an entity to display a calculated value:

1.   Double-click the entity or right-click and select **Edit**.

     The Entity Editor appears.

2.   Click the **>>** button at the end of the **Calculation** field, to expand the calculation editor.

3.   In the **Source** box:

a.   Type the equation you want to be calculated and displayed in the entity. For details on available calculations, refer to *Syntax* (see page 60).

> **Note:** As a general rule, it's a good idea to set the calculation up on the page in a way that makes the formula of the calculation immediately obvious. Sometimes this isn't possible and the fields involved in the calculation will need to be spread out.

Other than the standard mathematical functions, there is a set of standard functions that you can use, available in a drop-down list. When you select a function from the drop-down list, information on how to use that function is displayed in the text area below the function list.

b.   Click **Insert** to insert that function into the **Expression Editor**.

4.   Click **OK**.

5.   The **Read Only** check box is selected by default for **VFYTextBoxCalc**. To allow the user to override the calculated value, you need to clear the **Read Only** check box.

6.   In the Entity Editor, click **OK**.

7.   Click **Save** on the Verify toolbar to save your changes.

Syntax

Calculation syntax follows standard mathematical syntax, with the following caveats:

- Refer to entities by the value in their **Name** field.

- Use a semicolon (**;**) to indicate the end of the calculation, or at the end of each line for complicated functions.

- Where the syntax is more complicated than a single calculation, use one or more **return** statements to issue the correct result into the value of the field. You may also declare variables by using the **var** keyword.

```
You can include if-statement logic if required. For example: if
(Value('tb_FacilityX_LiquidsData_Atotal') < 10) return 0; else return 100;
```

- Use the keywords **Value** and **EnsureValue** to indicate the return value (that is, what P2 Verify will place into the cell when the calculation is completed). These keywords are defined as follows:

Value
Always returns the currently displayed value of the item requested. For example:

`Value('tb_FacilityX_LiquidsData_Atotal');`

EnsureValue
Retrieves different types of values. For example:

`EnsureValue('tb_FacilityX_LiquidsData_Atotal' , 'current', 0);`

The first parameter references the entity from which to extract the value.

The second parameter can either be **current**, **previous** or **original**:

Current
The value currently displayed by the item requested.

Previous
The approved value from the previous day's report.

Original
The original value as it was collected from the Historian.

The third parameter is the default value to return in case the item requested has no value.

Cross-Tab Calculations

To reference entities in different tabs (in the same data area), use the **Value** function. You should specify as a parameter the tab name followed by a colon, before the name of the entity you want to refer to.

For example:

```
Value('FacilityXLiquidsData:tb_FacilityX_LiquidsData_Atotal')
```

Note: When creating cross-tab calculations, you must use full tab notation to refer to all entities referenced in the calculation. This includes entities located in the current tab.

Important: Referencing an entity on another tab will introduce a tab dependency into the system (if tab dependencies are enabled in the *web.config* file). When a tab dependency exists, a user accessing either tab will also lock the other, whether or not that user can normally access the tab. This stops one user from overwriting the values of another user when both are editing the same report at the same time, but can make it unclear as to why a report is not able to be submitted.

To reference entities in a different data area, use the **VFYTextBoxData** object instead.

Predefined Calculations

For complex calculations that are often reused, you can predefine the calculation as a function that you can invoke from the **Expression Editor**.

The function is usually placed into a JavaScript file that is accessible to P2 Verify, and you can then call the function to perform your calculation. Unfortunately, the calculation itself cannot be easily viewed from the interface.

The available functions are located in the following folder:

`[Installation Directory]\Web\Scripts`

By default, this is:

`C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Web\Scripts\`

For instructions on adding a JavaScript file, refer to *Managing Custom Scripts* (see page 21).

Referencing Lookup Tables

To reference a Lookup Table in a calculation, you need to use one of the following functions in the Expression Editor:

Lookup
> Use this function when the row and column headers are numerical.

LookupStringBased
> Use this function when the row and column headers are text.

> **Note:** You can also use the Lookup functions inside other entities, such as VFYTextBoxCalc. In this way, you can look up two or more separate values from a single table, using different inputs.

The functions take four arguments:

`Lookup(TABLE, XVALUE, YVALUE, DEFAULTVALUE)`

`LookupStringBased(TABLE, XVALUE, YVALUE, DEFAULTVALUE)`

Table
> A **table** from which to reference the data, usually set up via a VFYLookupTable entity. You can specify a table two ways:
>
> – Use the `Table` function to return a Lookup Table. The `Table` function references a VFYLookupTable entity by name, in much the same way as the Value function returns the value inside a VFYTextBox. In this way, the `Table` function will most likely be used in the first argument of the `Lookup` function.
>
> – Specify a **static table** as a string, instead of using the `Table` function. If used, it should be a string representation of the array in the following format:
>
>> `'array = [[x0y0, x1y0, x2y0], [x0y1,x1y1,x2y1], [x0y2, x1y2, x2y2]];'`

For the `Lookup` function, all values in the header rows and columns should be **numeric values in ascending order**. These values will be used by the `Lookup` function to locate the resulting cell, the ascending order is necessary for the calculation engine to process in between values.

For the `LookupStringBased` function, all values in the header rows and columns should be **text values**. These values will be used by the `LookupStringBased` function.

XVALUE

The value to look up in the row header. For `LookupStringBased` functions, the value must be enclosed in single quotation marks.

YVALUE

The value to look up in the column header. For `LookupStringBased` functions, the value must be enclosed in single quotation marks.

DEFAULTVALUE

An alternate value to return if the `Lookup` function does not return anything with the specified X and Y values. For example, *-1* or '*Not Found*'.

For `LookupStringBased` functions, the value must be enclosed in single quotation marks.

LOOKUP EXAMPLE

Given the following VFYLookupTable, *testLookupTable*:

	0	5	10
0	1	2	3
5	4	5	6
10	7	8	9

- `Lookup(Table('testLookupTable'),5,10,-1)` will return `8`.

- `Lookup(Table('testLookupTable'),5,12,-1)` will return `-1`. The `YValue` is out of range.

- `Lookup(Table('testLookupTable'),7,10,-1)` will return `8`. The `Lookup` function does not require the `XValue` or `YValue` to be equal to the header values. In this case `XValue` is equal `7` and it is in between `5` and `10` (second and third row header values) so the calculation engine will use the second column. The result will be the same for all `XValues` superior or equal to `5` and inferior to `10`.

> **Note:** When you want to use the X and Y values to reference other entities on the same tab in your layout, you need to place the entity names in brackets, preceded by the keywords `Value` or `EnsureValue`. For example,
> `Lookup(Table('testLookupTable'),Value('textbox1'),Value('textbox2'),-1)` will use the values of the textboxes 'textbox1' and 'textbox2' to process the result.

LOOKUPSTRINGBASED EXAMPLE

Given the following VFYLookupTable, *testLookupTable*:

	col1	col2	col3
row1	1	2	3
row2	4	5	6
row3	7	8	9

- `LookupStringBased(Table('testLookupTable'), 'col2', 'row2', 'Not Found')` will return `5`.

- `LookupStringBased(Table('testLookupTable'), 'col2', 'xyz', 'Not Found')` will return '*Not Found*'. The `YValue` is out of range.

Adaptors

P2 Verify entities offer two pairs of configuration parameters that control where the initial value of the entity comes from, and what is its destination at the end of the approval process:

– **Collector type** lets users choose a type a *collector* responsible for retrieving entity values from external systems. **Source** parameter lets users provide configuration, which is specific to the collector type, indicating what value is expected.

– **Submitter type** lets users choose a type of a submitter responsible for relaying entity values to external systems. **Destination Name** parameter lets users provide configuration, which is specific to the submitter type, indicating where the value will be forwarded.

Collectors (see page 64) and submitters (see page 66) (collectively referred to as **adaptors**) are P2 Verify extension libraries for communication with external systems. Each P2 Verify data area can contain multiple entities, all collected from and submitted to different adaptors. Each individual entity can also use a submitter of a different type than its collector.

For example, a particular entity could contain a daily production value. Its **Collector Type** property can be configured to point to P2 Server collector, with a **Source** indicating a proper P2 Server expression with a tag name. The **Submitter Type** can be configured to point to SQL submitter, with a **Destination Name** property including a database query used to insert values.

Changes to these entity properties can only be applied in the Layout Mode.

All these parameters are optional. An entity does not have to be collected - it can be populated by calculations or manual input, and it does not have to be available for submission - its value can be used for display purposes only within P2 Verify.

Collectors

Collectors are P2 Verify extension libraries for retrieving data from external systems. P2 Verify supports the following categories of collectors:

- **Data Collectors** - collects a single value from a data source.

- **Dynamic List Collectors** - collects a complex value of a dynamic list, which comprises of multiple values corresponding to its rows and columns.

Some more complex collectors accept names of other collectors in entity's **Source** parameter to extend their functionality. For example, a dynamic list collector can use an additional collector to generate rows of dynamic list columns, while all individual values could be generated by the dynamic list collector itself.

To support such scenarios, P2 Verify collectors can also support an additional category of requests:

- **Hierarchy Repository** - returns a collection of items; P2 Verify controls such as combo-boxes, or other collectors can use such collectors to retrieve zero or more objects, represented by a display name and an internal name.
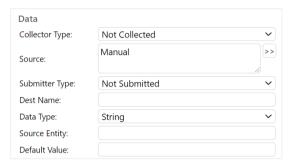
A collector must be either a data collector or a dynamic list collector to expose the hierarchy repository functionality.

Configuring Collectors

To retrieve a value for P2 Verify, you need to:

1. Configure a **Collector Type** to use one of the adaptors included with P2 Verify.

2. In Layout Mode, specify the **Collector Type** and **Source** for each entity, so that the entities are associated with the Collector Type in the previous step.

The Collector Type is selected from a drop-down list of available types, while the Source depends on the selected submitter.
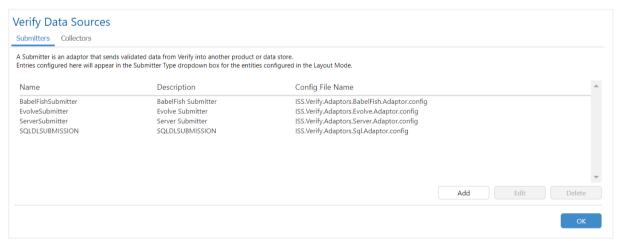
Adding a Collector Type

Collectors are configured through the **Data Sources** dialog window available in Layout Mode. These Collectors then appear in the **Collector Type** drop-down list in the Entity Editor.

The following example illustrates how to configure a collector to collect data from P2 Server. To configure other types of Collectors, refer to the documentation accompanying the Collector.

1. In Layout Mode, click **Data Sources** on the Verify toolbar.

    The **Verify Data Sources** window appears, with the **Submitters** tab active.

2. Click the **Collectors** tab.

3. Click **Add**.

The **Add Collector Type** dialog box appears.

4.   In the **Add Collector Type** dialog box, enter the following:

Name
Name of the Collector. For example, *P2 Server*

Description
Description of what the Collector is used for. For example, *Collects data from P2 Server*

Adaptor Assembly
Click the Ellipsis **...** button to select the file name of the adaptor assembly. For example, use **ISS.Verify.Adaptors.Server.dll** if you are fetching data from P2 Server.

Class Name
From the drop-down list, select the class name of the Adaptor you want to use. For example, use **ISS.Verify.Adaptors.Server.DataAdaptor** if you are fetching data from P2 Server.

Config File
Click the Ellipsis **...** button to select the configuration file you want to use. For example, the XML configuration file setup with your P2 Server installation details, based on a sample file ISS.Verify.Adaptors.Server.Adaptor.config. The default location for configuration files for adaptors is:

```
C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Server\
```

5.   Click **OK**.

The Collector Type is added.

Edit the configuration file to update the details of the data source. For some collectors, you will need to specify additional details, such as database names and network connection information.

Submitters

Submitters are P2 Verify extension libraries for relaying data to external systems. P2 Verify supports the following categories of submitters:

- **Data Submitters** - submit a single value to a destination.

- **Dynamic List Submitters** - submit a complex value of a dynamic list, which comprises of multiple values corresponding to its rows and columns.

Some more complex submitters accept names of other submitters in entity's **Destination Name** parameter to extend their functionality. For example, a dynamic list submitter can use an additional data submitter to submit individual values of a dynamic list's cells.

Configuring Submitters

To submit a daily value for P2 Verify, you need to:

1. Configure a Submitter Type to use one of the Adaptor assemblies included with P2 Verify. For details, refer to *Adding a Submitter Type* (see page 67).

    Note: While an Adaptor assembly is included with P2 Verify, the Adaptor for the data source to which you want to submit is also required. For example, if you want to submit data to P2 Server, install the Adaptor onto the same machine that P2 Verify is installed (in this case, install P2 Server and P2 Verify on the same server).

2. In the Entity Editor, specify the **Submitter Type** and **Destination Name** for each entity, so that the entities are associated with the Submitter Type in the previous step.

    The Submitter Type is selected from a drop-down list of available types, while the Destination Name depends on the selected submitter. If one of the standard submitters is used, you can enter P2 Server tag name as the data destination.
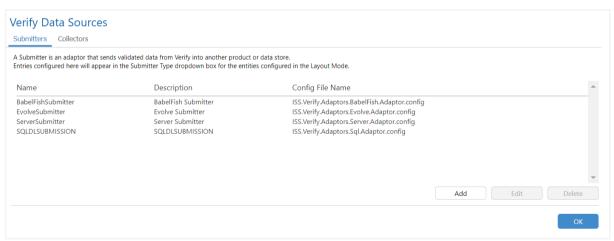
Adding a Submitter Type

Submitters are configured through the **Data Sources** dialog window available in Layout Mode. These Submitters then appear in the **Submitter Type** drop-down list in the Entity Editor.

The following example illustrates how to configure a submitter to submit data to P2 Server. To configure other types of Submitters, refer to the documentation accompanying the Submitter.

1. In Layout Mode, click **Data Sources** 🗄 on the Verify toolbar.

    The **Verify Data Sources** page appears, with the **Submitters** tab active.

2. Click **Add**.

The **Add Submitter Type** dialog box appears.

3. In the **Add Submitter Type** window, enter the following:

    Name
    Name of the Submitter. For example, *P2 Server*

    Description
    Description of what the Submitter does. For example, *Submits data into P2 Server*.

    Adaptor Assembly
    Click the Ellipsis [...] button to select the file name of the adaptor assembly. For example, use **ISS.Verify.Adaptors.Server.dll** if you are sending data to P2 Server.

    Class Name
    From the drop-down list, select the class name of the Adaptor you want to use. For example, use **ISS.Verify.Adaptors.Server.DataAdaptor** if you are sending data to P2 Server.

    Config File
    Click the Ellipsis [...] button to select the configuration file you want to use. For example, the XML configuration file setup with your P2 Server installation details, based on a sample file ISS.Verify.Adaptors.Server.Adaptor.config. The default location for configuration files for adaptors is:

    ```
    C:\Program Files (x86)\P2 Energy Solutions\P2 Verify\Server\
    ```

4. Click **OK**.

    The Submitter Type is added.

    Edit the configuration file to update the details of the data destination. For some submitters, you will need to specify additional details, such as database names and network connection information.

Standard Adaptors

P2 Verify 1.18 comes with adaptors that allow for integration with other P2 Energy Solutions Pty Ltd modules and external systems. The table below itemizes libraries, adaptors, and their supported request types:

Library	Adaptor	Supported Request Type
P2 Server 4.x Adaptor	Data Adaptor	• Data Collector • Data Submitter • Hierarchy Repository
P2 Explorer 2.6 Adaptor	Data Adaptor	• Data Collector

Library	Adaptor	Supported Request Type
		• Data Submitter • Hierarchy Repository
P2 Reconcile Adaptor	Data Adaptor	• Data Collector • Hierarchy Repository
SQL Adaptor	Data Adaptor	• Data Collector • Data Submitter • Hierarchy Repository
	Dynamic List Adaptor	• Data Collector • Data Submitter • Hierarchy Repository
Extension Adaptor	Data Adaptor	• Data Collector • Data Submitter
	Dynamic List Adaptor	• Dynamic List Collector • Dynamic List Submitter
	Periods Adaptor	• Dynamic List Collector
	Mappings Adaptor	• Dynamic List Submitter
Legacy Adaptors (see note)	-	-

Note: Legacy adaptors are provided only for backward compatibility with previous versions of P2 Verify. It is highly recommended to migrate existing configurations to other standard adaptors, as any legacy adaptors might be removed in future versions of P2 Verify.

P2 Server 4.x Adaptor

P2 Server 4.x Adaptor allows for working with data and asset hierarchies provided by P2 Server 4.x.

Data Adaptor

DATA COLLECTOR

P2 Server 4.x Data Collector evaluates a P2 Server expression and returns the result as a new value for an entity. The **Source** property of an entity must contain a valid P2 Server expression, for example:

```
SeriesExp(Random())
```

When the **Source** property represents a P2 Server tag name, it must be surrounded with curly braces, for example:

```
{TagName}
```

When used as a Hierarchy Repository, P2 Server 4.x Data Collector queries the P2 Server asset model. The **Source** property of an entity must contain a valid Hierarchy Source Pattern (see page 53), for example:

```
Hierarchy1||Entity1||Template1
```

▶ To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Server.dll
Class Name: ISS.Verify.Adaptors.Server.DataAdaptor

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <hierarchyRepository
      securityUrl="example.com"
      serverUrl="example.com"
      userName=""
      password="" />
   <dataCollector
      securityUrl="example.com"
      serverUrl="example.com"
      userName=""
      password=""
      startOffset="00:00:00"
      endOffset="00:00:00"
      rangeType="StartToStart"
      sampleMethod="LastKnownValue"
      sampleInterval="3600"
      dateFileOffsetFormat="yyyy-MM-dd HH:mm:ss"
      recordErrors="true"
      stringNullValue=""
      booleanNullValue=""
      timeSpanNullValue=""
      integerNullValue=""
      decimalNullValue=""
      errorValue=""
      dummyValue=""
      nullValue="" />
</configuration>
```

The `configuration/hierarchyRepository` element contains configuration properties required for performing hierarchy repository requests with P2 Server:

securityUrl
> The   host name of the P2 Security 4.x server integrated with P2 Server 4.x, indicated by the **serverUrl** attribute.

serverUrl
> The host name of P2 Server 4.x server.

userName
> The name of a user defined in P2 Security 4.x indicated by **securityUrl** attribute.

password
> The password of the user indicated by the **userName** attribute.

The `configuration/dataCollector` element contains configuration properties required for performing get-data requests with P2 Server:

securityUrl
> The host name of the P2 Security 4.x server integrated with P2 Server 4.x indicated by the **serverUrl** attribute.

serverUrl
> The host name of P2 Server 4.x server.

userName

   The name of a user defined in P2 Security 4.x indicated by **securityUrl** attribute.

password

   The password of the user indicated by the **userName** attribute.

startOffset

   An additional offset of the start time of the request, from the report start time.

endOffset

   An additional offset of the end time of the request, from the report end time.

rangeType

   Indicates which report times are used as request times.
   Options are: *StartToStart*, *EndToEnd*, *StartToEnd*

sampleMethod

   Indicates the sampling mode used in the request.
   Options are: *Raw*, *LastKnownValue*, *Average*, *LinearInterpolate*

sampleInterval

   Indicates the sampling interval, in seconds, used in the request.

dateTimeOffsetFormat

   Indicates the date format to which collected P2 Server date/time values should be
   converted in P2 Verify. See Appendix A (see page 105) for valid date/time formats.

recordErrors

   Indicates whether any operation errors should be returned as entity errors.

stringNullValue

   The text value to be used when a *null* string value has been returned from P2 Server.

booleanNullValue

   The text value to be used when a *null* boolean value has been returned from P2 Server.

timeSpanNullValue

   The text value to be used when a *null* time span value has been returned from P2 Server.

integerNullValue

   The text value to be used when a *null* integer has been returned from P2 Server.

decimalNullValue

   The text value to be used when a *null* decimal value has been returned from P2 Server.

errorValue

   The text value to be used when an error value has been returned from P2 Server.

dummyValue

   The text value to be used when a dummy value has been returned from P2 Server.

nullValue

   The text value to be used when a null value has been returned from P2 Server.

A sample file **ISS.Verify.Adaptors.Server.Adaptor.config** is provided in the installation directory.

DATA SUBMITTER

P2 Server 4.x Data Submitter allows for saving a value into a P2 Server tag. The **Source** property of an entity must contain a valid P2 Server tag, for example:

`Entity1[Template1]:Attribute1`

▸ To use this submitter, add it as a Submitter into P2 Verify. To do this, follow the steps in *Adding a Submitter Type* (see page 67), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Server.dll
Class Name: ISS.Verify.Adaptors.Server.DataAdaptor

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <dataSubmitter
      securityUrl="example.com"
      serverUrl="example.com"
      userName=""
      password=""
      timestampOffset="00:00:00"
      timestampType="Start"
      dateFileOffsetFormat="yyyy-MM-dd HH:mm:ss"
      stringNullValue=""
      booleanNullValue=""
      timeSpanNullValue=""
      integerNullValue=""
      decimalNullValue=""
      errorValue=""
      dummyValue=""
      nullValue=""
      />
</configuration>
```

The `configuration/dataSubmitter` element contains configuration properties required for performing put-data requests with P2 Server:

securityUrl
The host name of the P2 Security 4.x server integrated with P2 Server 4.x indicated by the **serverUrl** attribute.

serverUrl
The host name of P2 Server 4.x server.

userName
The name of a user defined in P2 Security 4.x indicated by **securityUrl** attribute.

password
The password of the user indicated by the **userName** attribute.

timestampOffset
Indicates an additional offset for a sample's timestamp.

timestampType
Indicates which report time is used as a sample's timestamp.
Options are: *Start*, *End*, *Now*

dateTimeOffsetFormat

Indicates the date format from which P2 Verify date/time values should be converted before being submitted to P2 Server. See Appendix A (see page 105) for valid date/time formats.

stringNullValue

The text value to be used when a *null* string value has been returned from P2 Server in an error response.

booleanNullValue

The text value to be used when a *null* boolean value has been returned from P2 Server in an error response.

timeSpanNullValue

The text value to be used when a *null* time span value has been returned from P2 Server in an error response.

integerNullValue

The text value to be used when a *null* integer has been returned from P2 Server in an error response.

decimalNullValue

The text value to be used when a *null* decimal value has been returned from P2 Server in an error response.

errorValue

The text value to be used when an error value has been returned from P2 Server in an error response.

dummyValue

The text value to be used when a dummy value has been returned from P2 Server in an error response.

nullValue

The text value to be used when a null value has been returned from P2 Server in an error response.

The sample file **ISS.Verify.Adaptors.Server.Adaptor.config** is provided in the installation directory.

P2 Explorer 2.6 Adaptor

P2 Explorer 2.6 Adaptor allows for working with data and asset hierarchies provided by P2 Explorer 2.6.8 (or newer), which support extended security features protecting data requests.

Data Adaptor

DATA COLLECTOR

P2 Explorer 2.6 Data Collector allows for retrieving a P2 Explorer tag as a new value for an entity. The **Source** property of an entity must contain a valid P2 Explorer tag name, for example:

```
CALC.RANDOM.NORMAL
```

When used as a Hierarchy Repository, P2 Explorer 2.6 Data Collector allows for querying P2 Explorer Data Dictionary. The **Source** property of an entity must contain a valid Hierarchy Source Pattern (see page 53), for example:

```
Hierarchy1||Entity1||Template1
```

▸ To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Explorer.dll
Class Name: ISS.Verify.Adaptors.Explorer.DataAdaptor

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <hierarchyRepository
      hostName="example.com"
      port="8135"
      userName=""
      password="" />
   <dataCollector
      hostName="example.com"
      port="8135"
      userName=""
      password=""
      startOffset="00:00:00"
      endOffset="00:00:00"
      rangeType="StartToStart"
      sampleMethod="LastKnownValue"
      sampleInterval="3600"
      recordErrors="true"
      stringNullValue="" />
</configuration>
```

The `configuration/hierarchyRepository` element contains configuration properties required for performing hierarchy repository requests with P2 Explorer:

hostName
A network host name of P2 Explorer 2.6 server.

This property is optional. If the **hostName** and the **port** properties are omitted, or their values are empty, a default instance of P2 Explorer 2.6 is identified based on the P2 Explorer SDK registry settings.

port
A network port of P2 Server 4.x server.

This property is optional. If the **hostName** and the **port** properties are omitted, or their values are empty, a default instance of P2 Explorer 2.6 is identified based on the P2 Explorer SDK registry settings.

userName
A name of a user defined in P2 Explorer 2.6 who can perform requests.

password
A password of the user indicated by the **userName** attribute.

The `configuration/dataCollector` element contains configuration properties required for performing get-data requests with P2 Explorer:

hostName

A network host name of P2 Explorer 2.6 server.

This property is optional. If the **hostName** and the **port** properties are omitted, or their values are empty, a default instance of P2 Explorer 2.6 is identified based on the P2 Explorer SDK registry settings.

port

A network port of P2 Server 4.x server.

This property is optional. If the **hostName** and the **port** properties are omitted, or their values are empty, a default instance of P2 Explorer 2.6 is identified based on the P2 Explorer SDK registry settings.

userName

A name of a user defined in P2 Explorer 2.6 who can perform requests.

password

A password of the user indicated by the **userName** attribute.

startOffset

An additional offset of the start time of the request from the report start time.

endOffset

An additional offset of the end time of the request from the report end time.

rangeType

Indicates which report times are used as request times.

Options are: *StartToStart*, *EndToEnd*, *StartToEnd*

sampleMethod

Indicates the sampling mode used in the request.

Options are: *Raw*, *LastKnownValue*, *Average*, *LinearInterpolate*

sampleInterval

Indicates the sampling mode used in the request.

recordErrors

Indicates whether any operation errors should be returned as entity errors.

stringNullValue

A text value to be used when a *null* string value has been returned from P2 Explorer.

A sample file **ISS.Verify.Adaptors.Explorer.Adaptor.config** is provided in the installation directory.

DATA SUBMITTER

P2 Explorer 2.6 Data Submitter allows for saving a value into a P2 Explorer tag. The **Source** property of an entity must contain a valid P2 Explorer tag (in a normal or templated format), for example:

```
Entity1[Template1]:Attribute1
```

▸   To use this submitter, add it as a Submitter into P2 Verify. To do this, follow the steps in *Adding a Submitter Type* (see page 67), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Explorer.dll
Class Name: ISS.Verify.Adaptors.Explorer.DataAdaptor

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <dataSubmitter
     hostName="example.com"
     port="8135"
     userName=""
     password=""
     timestampOffset="00:00:00"
     timestampType="Start" />
</configuration>
```

The `configuration/dataSubmitter` element contains configuration properties required for performing get-data requests with P2 Explorer:

hostName

A network host name of P2 Explorer 2.6 server.

This property is optional. If the **hostName** and the **port** properties are omitted, or their values are empty, a default instance of P2 Explorer 2.6 is identified based on the P2 Explorer SDK registry settings.

port

A network port of P2 Server 4.x server.

This property is optional. If the **hostName** and the **port** properties are omitted, or their values are empty, a default instance of P2 Explorer 2.6 is identified based on the P2 Explorer SDK registry settings.

userName

A name of a user defined in P2 Explorer 2.6 who can perform requests.

password

A password of the user indicated by the **userName** attribute.

timestampOffset

Indicates an additional offset of a sample's time stamp.

timestampType

Indicates which report time is used as a sample's time stamp.

Options are: *Start*, *End*, *Now*

A sample file **ISS.Verify.Adaptors.Explorer.Adaptor.config** is provided in the installation directory.

BabelFish 2.6 Adaptor

BabelFish 2.6 Adaptor allows for working with data and asset hierarchies provided by P2 Explorer 2.6.4   (or older; in older versions known as *ISS BabelFish*), which ***DO NOT*** support extended security features protecting data requests.

> **Note:** This adaptor is provided for backward compatibility with legacy systems only. In solutions using P2 Explorer 2.6.8 or newer, use P2 Explorer 2.6 Adaptor (see page 73).

Data Adaptor

DATA COLLECTOR

P2 Explorer 2.6 Data Collector allows for retrieving a P2 Explorer tag as a new value for an entity. The **Source** property of an entity must contain a valid P2 Explorer tag name, for example:

`CALC.RANDOM.NORMAL`

When used as a Hierarchy Repository, P2 Explorer 2.6 Data Collector allows for querying P2 Explorer Data Dictionary. The **Source** property of an entity must contain a valid Hierarchy Source Pattern (see page 53), for example:

`Hierarchy1||Entity1||Template1`

▶ To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
  Adaptor Assembly: ISS.Verify.Adaptors.BabelFish.dll
  Class Name: ISS.Verify.Adaptors.BabelFish.DataAdaptor

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <hierarchyRepository />
   <dataCollector
     startOffset="00:00:00"
     endOffset="00:00:00"
     rangeType="StartToStart"
     sampleMethod="LastKnownValue"
     sampleInterval="3600"
     recordErrors="true"
     stringNullValue="" />
</configuration>
```

The `configuration/hierarchyRepository` element is a placeholder for configuration properties required for performing hierarchy repository requests with P2 Explorer. Currently, it requires no configuration attributes.

The `configuration/dataCollector` element contains configuration properties required for performing get-data requests with P2 Explorer:

startOffset
  An additional offset of the start time of the request from the report start time.

endOffset

An additional offset of the end time of the request from the report end time.

rangeType

Indicates which report times are used as request times.

Options are: *StartToStart*, *EndToEnd*, *StartToEnd*

sampleMethod

Indicates the sampling mode used in the request.

Options are: *Raw*, *LastKnownValue*, *Average*, *LinearInterpolate*

sampleInterval

Indicates the sampling mode used in the request.

recordErrors

Indicates whether any operation errors should be returned as entity errors.

stringNullValue

A text value to be used when a *null* string value has been returned from P2 Explorer.

A sample file **ISS.Verify.Adaptors.BabelFish.Adaptor.config** is provided in the installation directory.

DATA SUBMITTER

P2 Explorer 2.6 Data Submitter allows for saving a value into a P2 Explorer tag. The **Source** property of an entity must contain a valid P2 Explorer tag (in a normal or templated format), for example:

`Entity1[Template1]:Attribute1`

▸   To use this submitter, add it as a Submitter into P2 Verify. To do this, follow the steps in *Adding a Submitter Type* (see page 67), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.BabelFish.dll
Class Name: ISS.Verify.Adaptors.BabelFish.DataAdaptor

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <dataSubmitter
      timestampOffset="00:00:00"
      timestampType="Start" />
</configuration>
```

The `configuration/dataSubmitter` element contains configuration properties required for performing get-data requests with P2 Explorer:

timestampOffset

Indicates an additional offset of a sample's time stamp.

timestampType

Indicates which report time is used as a sample's time stamp.

Options are: *Start*, *End*, *Now*

A sample file **ISS.Verify.Adaptors.BabelFish.Adaptor.config** is provided in the installation directory.

P2 Reconcile Adaptor

P2 Reconcile Adaptor allows for working with P2 Reconcile Evolve asset hierarchies and P2 Reconcile Core Expression Resolver expressions.

Data Adaptor

DATA COLLECTOR

P2 Reconcile Data Collector allows for evaluating a P2 Reconcile Core Expression Resolver expression and returning the result as a new value for an entity. The **Source** property of an entity must contain a valid P2 Reconcile Core Expression Resolver expression, for example:

`{#Entity1:Attribute}`

When used as a Hierarchy Repository, P2 Reconcile Data Collector allows for querying P2 Reconcile Evolve. The **Source** property of an entity must contain a valid Hierarchy Source Pattern (see page 53), for example:

`RelationGroup1||Entity1||Template1`

▸ To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Evolve.dll
Class Name: ISS.Verify.Adaptors.Evolve.DataAdaptor

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <hierarchyRepository />
   <dataCollector
     startOffset="00:00:00"
     endOffset="00:00:00"
     rangeType="StartToStart"
     sampleMethod="LastKnownValue"
     sampleInterval="0"
     dateFileOffsetFormat="yyyy-MM-dd HH:mm:ss"
     recordErrors="true"
     stringNullValue=""
     booleanNullValue=""
     timeSpanNullValue=""
     integerNullValue=""
     decimalNullValue=""
     errorValue=""
     dummyValue=""
     nullValue="" />
</configuration>
```

The `configuration/hierarchyRepository` element is a placeholder for configuration properties required for performing hierarchy repository requests with P2 Explorer. Currently, it requires no configuration attributes.

The `configuration/dataCollector` element contains configuration properties required for performing get-data requests with P2 Reconcile:

startOffset
> An additional offset of the start time of the request from the report start time.

endOffset
> An additional offset of the end time of the request from the report end time.

rangeType
> Indicates which report times are used as request times.
>
> Options are: *StartToStart*, *EndToEnd*, *StartToEnd*

sampleMethod
> Indicates the sampling mode used in the request.
>
> Options are: *Raw*, *LastKnownValue*, *Average*, *LinearInterpolate*

sampleInterval
> Indicates the sampling mode used in the request.

dateTimeOffsetFormat
> Indicates a date format to which collected P2 Server date/time values should be converted in P2 Verify.

recordErrors
> Indicates whether any operation errors should be returned as entity errors.

stringNullValue
> A text value to be used when a *null* string value has been returned from P2 Reconcile.

booleanNullValue
> A text value to be used when a *null* Boolean value has been returned from P2 Reconcile.

timeSpanNullValue
> A text value to be used when a *null* time span value has been returned from P2 Reconcile.

integerNullValue
> A text value to be used when a *null* integer has been returned from P2 Reconcile.

decimalNullValue
> A text value to be used when a *null* decimal value has been returned from P2 Reconcile.

errorValue
> A text value to be used when an error value has been returned from P2 Reconcile.

dummyValue
> A text value to be used when a dummy value has been returned from P2 Reconcile.

nullValue
> A text value to be used when a null value has been returned from P2 Reconcile.

A sample file **ISS.Verify.Adaptors.Evolve.Adaptor.config** is provided in the installation directory.

SQL Adaptor

SQL Adaptor allows for working directly with Microsoft SQL Server databases.

Data Adaptor

DATA COLLECTOR

SQL Data Collector allows for executing a SQL query and returning the first value of a first column as a new value for an entity. The **Source** property of an entity must contain a valid SQL query, which may optionally contain `startDate` and `endDate` parameters *(DateTimeOffset* type) representing report dates, optionally prefixed with a name of the connection string from the configuration file, for example:

```
DataCollectorOnly||SELECT value FROM myTable WHERE startDate = @startDate;
```

It is possible to specify an additional formatting sequence `/* merge */` to help P2 Verify merge queries that use the same connection string, share the tables, and conditions, from multiple entities into a single query. For instance, queries of different entities:

```
DataCollectorOnly||SELECT /* merge */ column1 FROM myTable WHERE startDate =
@startDate;
DataCollectorOnly||SELECT /* merge */ column2 FROM myTable WHERE startDate =
@startDate;
```

may be executed by the collector as a single database query:

```
DataCollectorOnly||SELECT column1, column2 FROM myTable WHERE startDate = @startDate;
```

When used as a Hierarchy Repository, SQL Data Collector allows for returning collections of elements based on SQL queries. The **Source** property of an entity must contain a valid SQL query with two columns representing an internal name (`internal_name`, alternatively `name`) and a display name (`external_name`, alternatively `display_name`), optionally prefixed with a name of the connection string from the configuration file, for example:

```
HierarchyRepositoryOnly||SELECT internal_name, external_name FROM myTable;
```

▸   To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
    Adaptor Assembly: ISS.Verify.Adaptors.Sql.dll
    Class Name: ISS.Verify.Adaptors.Sql.DataAdaptor

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <hierarchyRepository>
     <connectionString
       name="HierarchyRepositoryOnly"
       value="Server=[SERVER];Database=[DATABASE];User Id=[USER
ID];Password=[PASSWORD];" />
   </hierarchyRepository>
   <dataCollector>
     <connectionString
       name="DataCollectorOnly"
       value="Server=[SERVER];Database=[DATABASE];User Id=[USER
ID];Password=[PASSWORD];" />
   </dataCollector>
   <connectionString
```

```
      name=""
      value="Server=[SERVER];Database=[DATABASE];User Id=[USER ID];Password=[PASSWORD];"
/>
</configuration>
```

The `configuration/hierarchyRepository` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are used only when the adaptor is performing hierarchy repository requests.

The `configuration/dataCollector` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are used only when the adaptor is performing get-data requests.

The `configuration` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are can be used for all the requests. If a connection string with the same name has been also defined in an element for a particular request type, that connection string takes precedence.

A connection string that has an empty name is considered default. This connection string is used when the **Source** property of an entity is not prefixed with a different connection string name.

DATA SUBMITTER

SQL Data Submitter allows for executing a SQL insert, update or delete statement to submit a single value of an entity. The **Destination Name** property of an entity must contain a valid SQL statement, which may optionally contain `startDate` and `endDate` parameters *(DateTimeOffset* type) representing report dates, and a `value` parameter *(Integer, Decimal* or *String*, depending on the entity data type) representing a value of an entity, optionally prefixed with a name of the connection string from the configuration file, for example:

`DataSubmitterOnly||INSERT INTO myTable(date, value) VALUES(@startDate, @value);`

▶   To use this submitter, add it as a Submitter into P2 Verify. To do this, follow the steps in *Adding a Submitter Type* (see page 67), using the following settings:
     Adaptor Assembly: ISS.Verify.Adaptors.Sql.dll
     Class Name: ISS.Verify.Adaptors.Sql.DataAdaptor

A configuration file must have the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <dataSubmitter>
     <connectionString
       name="DataSubmitterOnly"
       value="Server=[SERVER];Database=[DATABASE];User Id=[USER
ID];Password=[PASSWORD];" />
   </dataSubmitter>
   <connectionString
     name=""
     value="Server=[SERVER];Database=[DATABASE];User Id=[USER ID];Password=[PASSWORD];"
/>
</configuration>
```

The `configuration/dataSubmitter` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are used only when the adaptor is performing put-data requests.

The `configuration` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are can be used for all the requests. If a connection string with the same name has been also defined in an element for a particular request type, that connection string takes precedence.

A connection string that has an empty name is considered default. This connection string is used when the **Destination Name** property of an entity is not prefixed with a different connection string name.

Dynamic List Adaptor

DYNAMIC LIST COLLECTOR

SQL Dynamic List Collector allows for executing a SQL query and returning the result as the new contents of a dynamic list entity. The **Source** property of an entity must contain a valid SQL query, which may optionally contain `startDate` and `endDate` parameters *(DateTimeOffset* type) representing report dates, optionally prefixed with a name of the connection string from the configuration file, for example:

```
DataCollectorOnly||SELECT column1, column2, column3 FROM myTable WHERE startDate =
@startDate;
```

The **Source** property of individual columns should contain a name of a matching column from the resulting data set.

When used as a Hierarchy Repository, SQL Dynamic List Collector allows for returning collections of elements based on SQL queries. The **Source** property of an entity must contain a valid SQL query with two columns representing an internal name (`internal_name`, alternatively `name`) and a display name (`external_name`, alternatively `display_name`), optionally prefixed with a name of the connection string from the configuration file, for example:

```
HierarchyRepositoryOnly||SELECT internal_name, external_name FROM myTable;
```

▸　　To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
　　　　Adaptor Assembly: ISS.Verify.Adaptors.Sql.dll
　　　　Class Name: ISS.Verify.Adaptors.Sql.DynamicListAdaptor

A configuration file must have the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <hierarchyRepository>
    <connectionString
      name="HierarchyRepositoryOnly"
      value="Server=[SERVER];Database=[DATABASE];User Id=[USER
ID];Password=[PASSWORD];" />
  </hierarchyRepository>
  <dynamicListCollector>
    <connectionString
      name="DataCollectorOnly"
      value="Server=[SERVER];Database=[DATABASE];User Id=[USER
ID];Password=[PASSWORD];" />
```

```
    </dynamicListCollector>
    <connectionString
       name=""
       value="Server=[SERVER];Database=[DATABASE];User Id=[USER ID];Password=[PASSWORD];"
    />
</configuration>
```

The `configuration/hierarchyRepository` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are used only when the adaptor is performing hierarchy repository requests.

The `configuration/dynamicListCollector` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are used only when the adaptor is performing get-data requests.

The `configuration` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are can be used for all the requests. If a connection string with the same name has been also defined in an element for a particular request type, that connection string takes precedence.

A connection string that has an empty name is considered default. This connection string is used when the **Source** property of an entity is not prefixed with a different connection string name.

DYNAMIC LIST SUBMITTER

SQL Dynamic List Submitter allows for executing a SQL insert, update or delete statement to submit values for a dynamic list entity. The **Destination Name** property of an entity must contain a valid SQL statement, which may optionally contain `startDate` and `endDate` parameters *(DateTimeOffset* type) representing report dates, optionally prefixed with a name of the connection string from the configuration file. As the query is executed for every row in a dynamic list entity, values of individual entities in a row are added as parameters with the names corresponding to the columns' Destination Name settings, andwith the types depending on the column data type (*Integer*, *Decimal* or *String*), for example:

`DynamicListSubmitterOnly||INSERT INTO myTable(date, value1, value2) VALUES(@startDate, @column1, @column2);`

It is not uncommon for a dynamic list entity to execute statements for an entire dynamic list before (`{BEFORE}`) or after (`{AFTER}`) executing statements for individual rows (`{PER ROW}`), for example to delete all previous values with a single query based on report dates. For this reason the submitter supports also extended syntax:

`DynamicListSubmitterOnly|| {PER ROW} /*pre*/ {BEFORE} /*post*/ {AFTER}`

All three statements may contain `startDate` and `endDate` parameters, however only `{PER ROW}` statement can use value parameters.

▸   To use this submitter, add it as a Submitter into P2 Verify. To do this, follow the steps in *Adding a Submitter Type* (see page 67), using the following settings:
     Adaptor Assembly: ISS.Verify.Adaptors.Sql.dll
     Class Name: ISS.Verify.Adaptors.Sql.DynamicListAdaptor

A configuration file must have the following format:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<configuration>
  <dynamicListSubmitter>
    <connectionString
      name="DataSubmitterOnly"
      value="Server=[SERVER];Database=[DATABASE];User Id=[USER
ID];Password=[PASSWORD];" />
  </dynamicListSubmitter>
  <connectionString
    name=""
    value="Server=[SERVER];Database=[DATABASE];User Id=[USER ID];Password=[PASSWORD];"
/>
</configuration>
```

The `configuration/dynamicListSubmitter` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are used only when the adaptor is performing put-data requests.

The `configuration` element contains `connectionString` elements with a unique `name` and a `value` that represents a connection string in ADO.NET format. These connection strings are can be used for all the requests. If a connection string with the same name has been also defined in an element for a particular request type, that connection string takes precedence.

A connection string that has an empty name is considered default. This connection string is used when the **Destination Name** property of an entity is not prefixed with a different connection string name.

Extension Adaptors

Extension adaptors are the adaptors that do not offer connectivity with external data sources directly. Instead, they either transform the format data is organised in, or represent virtual data sources by generating values based on a defined algorithm. All the standard extension adaptors provided with P2 Verify use other adaptors to perform their core functionality.

Data Adaptor

DATA COLLECTOR

Data Collector is an extension adaptor that acts as a proxy, and it collects an entity using another data collector. The **Source** property of an entity must contain a valid Source property for a target data collector, and be prefixed with a name of that target collector:

`{collector}||{source}`

This adaptor can be useful in places, where the name of a target collector must be resolved dynamically, for example as a part of a dynamic list column definition.

When used as a Hierarchy Repository, Data Collector acts as a proxy, and it allows for querying another data collector that supports hierarchy repository requests. The **Source** property of an entity must contain a valid Source property for a target Hierarchy Repository, and be prefixed with a name of that target collector:

`{collector}||{source}`

▶   To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Extension.dll
Class Name: ISS.Verify.Adaptors.Data.DataAdaptor

A configuration file is ignored by this adaptor.

Dynamic List Adaptor

DYNAMIC LIST COLLECTOR

Dynamic List Collector is an extension adaptor that acts as a dynamic list collector that uses a Hierarchy Repository specified in the Source of a dynamic list entity to generate rows, and Source properties of its columns to generate individual entities within each row.

The **Source** property of an entity must contain a valid Source property for a target collector supporting hierarchy repository requests, and be prefixed with a name of that target collector:

`{collector}||{source}`

Source properties of all the columns must contain valid Source properties for data collectors for particular columns, and be prefixed with names of these data collectors:

`{collector}||{source}`

The Ad Hoc Combo Box (see page 42) components significantly extend the functionality of dynamic lists, and therefore they extend the syntax of the Source property values. Columns using this component provide drop-down lists of selectable options, and if other columns reference the combo-box, their value will be refreshed when selection changes. To reference a combo-box, use the "at" (`@`) symbol followed by a number indicating a column with the combo-box.

For example, `@0` will be replaced by the selected value of the Ad Hoc Combo Box in the first column. To display the *Choke* attribute value of the selected well from a collector *MyServerCollector* in a text box, the **Source** property of that text box has to be set to `MyServerCollector||{@0:Choke}`. The resulting attribute/property will be retrieved from a selected collector.

Note: If there is no corresponding control, that is, @9 where there are only 5 columns in the dynamic list, the pattern will not be substituted.

▶   To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Extension.dll
Class Name: ISS.Verify.Adaptors.DynamicList.DynamicListAdaptor

A configuration file is ignored by this adaptor.

Periods Adaptor

DYNAMIC LIST COLLECTOR

Periods Adaptor is an extension adaptor that - when used as a dynamic list collector - behaves exactly like Dynamic List Adaptor (see page 86).

When used as a Hierarchy Repository, Periods Adaptor generates a collection of date periods, for example past months in the current year, based on the special syntax provided in the **Source** parameter:

```
{period}||{cutoff}||{allowFuture}||{format}||{periodMonths}||{offset}||{offsetMonths}||
{trim}||{startAt}
```

The value can include all arguments in the sequence presented above. Alternatively, arguments can be prefixed with their names if they are used in a different order, like the `offset` parameter in the example below (`period`, `cutoff`, `allowFuture` arguments are following the default order and do not need a prefix):

```
1.00:00:00||month||true||offset:08:00:00||trim:month
```

The example above defines a collection of daily values (`period`) from the beginning of the month (`trim`) to the end of the month (`cutoff`), including future dates (`allowFuture`).

Full description of the available arguments is presented below:

period
> Time span increment of each period. For example, `1.02:03:04` represents 1 day, 2 hours, 3 minutes, 4 seconds.

cutoff
> A number representing a maximum number of generated periods, or a name of a period type which all periods must not exceed.
>
> Options are: a *positive integer number*, or one of the following values: `year`, `month`, `day`, `hour`, `minute`, `second`, `millisecond`.

allowFuture
> Indicates whether period dates can exceed the current date.
>
> Options are: `true`, `false`.

format
> Indicates a date formatting string, For example, `yyyy-MM-dd hh:mm`.

periodMonths
> Additional number of full months to be added to a **period** argument.

offset
> Time span increment of each date generate for the period. For example, `1.02:03:04` represents 1 day, 2 hours, 3 minutes, 4 seconds.

offsetMonths
> Additional number of full months to be added to an **offset** argument.

trim

A name of a period type, to which an initial period date should be rounded down.

Options are: year, month, day, hour, minute, second, millisecond.

offsetMonths

Additional number of full months to be added to an **offset** argument.

▸ To use this collector, add it as a Collector into P2 Verify. To do this, follow the steps in *Adding a Collector Type* (see page 65), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Extension.dll
Class Name: ISS.Verify.Adaptors.Periods.DynamicListAdaptor

A configuration file is ignored by this adaptor.

Mappings Adaptor

DYNAMIC LIST SUBMITTER

Mappings Adaptor is an extension adaptor that acts as a dynamic list submitter that uses the column mappings provided in a configuration file to compose time-series triplets [*tag*, *time stamp*, *value*], and submit them to a data submitter, which name has been used as a prefix in each tag.

▸ To use this submitter, add it as a Submitter into P2 Verify. To do this, follow the steps in *Adding a Submitter Type* (see page 67), using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Extension.dll
Class Name: ISS.Verify.Adaptors.Mappings.DynamicListAdaptor

The **Destination Name** parameter must match a table mapping in the configuration file. If no matching table mapping is found, a default table mapping from the configuration file is used.

A configuration file must have the following format:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <dynamicListSubmitter>
     <tables>
       <table name="test1">
         <column name="value" tagColumn="tag" timeColumn="time" />
         <column name="time" source="startDate" />
         <column name="tag" />
       </table>
       <table name="test2">
         <column tagColumn="tag" timeColumn="time" />
         <column name="time" source="column" timeOffset="-00:00:01"
canConvertTimeToUtc="true" />
         <column name="tag" />
         <column name="value" tagColumn="tag" source="const">CONST1</column>
       </table>
       <table>
         <column tagColumn="tag" timeColumn="time" />
         <column name="time" source="column" timeOffset="-00:00:01"
canConvertTimeToUtc="true" />
         <column name="tag" />
       </table>
     </tables>
   </dynamicListSubmitter>
```

```
</configuration>
```

The `configuration/dynamicListSubmitter/tables` element contains `table` elements, which provide the table mapping   information. A `name` of a `table` element indicates a name of a table mapping, and it should match the **Destination Name** property of a dynamic list entity, or remain empty to indicate a default mapping for any dynamic list without a matching table mapping. Within each table mapping there is a collection `column` elements with its column mappings.

A `name` of a `column` element indicates a name of a column mapping, and it should match the **Destination Name** property of a dynamic list's column, represent a new name to indicate a virtual column (for example with a constant, literal value), or remain empty to indicate a default mapping for any column without a matching column mapping.

Each time-series triplet contains a *tag*, *value*, and *time-stamp*. The adaptor forms time-series triplets are for each column mapping with a `tagColumn` attribute. This attribute indicates that the current column represents a value, a column indicated in the `tagColumn` attribute represents a tag name (which should be prefixed by a name of an appropriate data submitter), while a column indicated in a `timeColumn` attribute represents a time-stamp (defaulting the the report's start date when not provided).

A column mapping for triplet *values* can specify the following attributes:

name
> A name of a column that contains values. Matches all columns without explicit mappings when this attribute is not provided.

tagColumn
> A name of a column mapping that indicates a column containing tag names.

timeColumn
> A name of a column mapping that indicates a column containing time-stamps. A start date of a report is used when this attribute is not provided.

source
> Indicates what is the source of the column mapping's data.

> Options are:

> `const` (a constant, literal value provided are the `column` element's value)

> `column` (a value from the underlying column of a dynamic list with a matching **name**)

> `startDate` (a report's start date in the format `yyyy-MM-dd'T'HH:mm:ss`)

> `endDate` (a report's end date in the format `yyyy-MM-dd'T'HH:mm:ss`)

> `entity` (a name of the dynamic list entity being processed)

> `tag` (a **Destination Name** of the underlying column of a dynamic list with a matching **name**)

A column mapping for triplet *time-stamps* can specify the following attributes:

name
> A name of a column that contains time-stamps. Matches all columns without explicit mappings when this attribute is not provided.

timeOffset
> Additional time span offset added to the time-stamp.

canConvertTimeToUtc

Indicates whether the value should be converted to UTC.

Options are: *false*, *true*.

source

Indicates what is the source of the column mapping's data.

Options are:

`const` (a constant, literal value provided are the `column` element's value)

`column` (a value from the underlying column of a dynamic list with a matching **name**)

`startDate` (a report's start date in the format `yyyy-MM-dd'T'HH:mm:ss`)

`endDate` (a report's end date in the format `yyyy-MM-dd'T'HH:mm:ss`)

`entity` (a **Destination Name** of the dynamic list entity being processed)

`tag` (a **Destination Name** of the underlying column of a dynamic list with a matching **name**)

A column mapping for triplet *tags* can specify the following attributes:

name

A name of a column that contains time-stamps. Matches all columns without explicit mappings when this attribute is not provided.

source

Indicates what is the source of the column mapping's data.

Options are:

`const` (a constant, literal value provided are the `column` element's value)

`column` (a value from the underlying column of a dynamic list with a matching **name**)

`startDate` (a report's start date in the format `yyyy-MM-dd'T'HH:mm:ss`)

`endDate` (a report's end date in the format `yyyy-MM-dd'T'HH:mm:ss`)

`entity` (**Destination Name** of the dynamic list entity being processed)

`tag` (**Destination Name** of the underlying column of a dynamic list with a matching **name**)

Additionally, for all column mappings, **source** attributes with `entity` and `tag` values allow for special syntax of **Destination Name** values. The following placeholder are allowed:

{column:*name*}

This placeholder will be substituted with a value from the column with a **Destination Name** equal to **name**.

{index:*index*}

This placeholder will be substituted with a value from the column with an index indicated by **index**.

{param:*name*}

This placeholder will be substituted with a value of a parameter with a specific **name**.

Available parameters are:

`destination` (a **Destination Name** of a dynamic list entity)

`tag` (a tag portion of **Destination Name** of a dynamic list entity; **Destination Name** without a prefix)

`adaptor` (an adaptor portion of **Destination Name** of a dynamic list entity; **Destination Name** prefix)

`row` (an index of the current row)

`value` (a value from the underlying column of a dynamic list)

`column` (**Destination Name** of the underlying column of a dynamic list with a matching **name**)

Legacy Adaptors

The key factor differentiating the legacy adaptors and other standard adaptors is that it is not possible to identify whether an adaptor acts as a collector or a submitter, or whether it is dedicated for working with simple data or dynamic lists. The legacy adaptors also do not support hierarchy requests. It is necessary to refer to the documentation to configure that adaptor properly.

▶   To use an adaptor as a collector, follow the steps in *Adding a Collector Type (see page 65)*. To use an adaptor as a submitter, follow the steps in *Adding a Submitter Type (see page 67)*. *In both cases* use the following **Adaptor Assembly**: ISS.Verify.Adaptors.Legacy.dll . For the information about the **Class Name**, refer to the listing below.

P2 Verify has a collection of the legacy adaptors in the **ISS.Verify.Adaptors.Legacy.dll** assembly. The five time-series adaptors below are the ones most commonly used:

ISS.Verify.Adaptors.GetTimeSeriesData.GetAdaptor

The basic adaptor used to retrieve a value from a time-series data source, this a historian database.

ISS.Verify.Adaptors.GetTimeSeriesData.GetCalcAdaptor

Allows calculations to be performed on the tags; e.g. TAG_1 + TAG_2. The calculation must be specified in the expression editor.

ISS.Verify.Adaptors.GetTimeSeriesDataAdvanced.GetAdaptor

This adaptor is similar to the basic **GetTimeSeriesData.GetAdaptor**, but has advanced features such as *Minimum*, *Maximum*, *Delta*, *Average*, and *Sum*, which can be performed on a tag over a specified period of time. The configuration for the advanced features is specified in the adaptor's configuration file.

ISS.Verify.Adaptors.GetTimeSeriesDataAdvanced.GetCalcAdaptor

This adaptor has the combined ability of both **GetTimeSeriesData.GetCalcAdaptor** and **GetTimeSeriesDataAdvanced.GetAdaptor**.

ISS.Verify.Adaptors.PutTimeSeriesData.PutAdaptor

The adaptor used to submit data. The adaptors mentioned above can be configured to use the P2 Explorer 2.6 Data Broker as its central data source, or independent adaptors (such as the PI Adaptor or Relational Adaptor) can be installed and configured to work with P2 Verify. The adaptor's configuration file can specify how the data is retrieved.

P2 Verify can also directly invoke other adaptors such as PI Adaptor, PHD Adaptor, and Relational Adaptor. To enable P2 Verify to use these adaptors, they need to be installed on the same machine as P2 Verify. Then the configuration file will need to be changed to reference the adaptor (For example, Relational Adaptor) with that adaptor's particular settings.

Below is part of a configuration file for the Relational Adaptor, showing the configuration to be used with a collector for P2 Verify. The `InitData` element contains the properties required to initialise the adaptor and use it as a collector. This property list is provided with the adaptor's documentation and is different for every adaptor.

```xml
<GetTimeSeriesData>
   <Debug>true</Debug>
   <IsAdaptor>true</IsAdaptor>
   <ProgId>ISS.DataSources.Relational</ProgId>
   <TypeName></TypeName>
   <InitData><![CDATA[
     <Initialise>
       <Profile>
         <Property name="DBCONNECTIONCLASSS"
type="STRING">MSSQLDBCONNECTION</Property>
         <Property name="SERVER" type="STRING">SERVER_NAME</Property>
         <Property name="USER_ID" type="STRING">LOGIN_ID</Property>
         <Property name="PASSWORD" type="STRING">LOGIN_PASSWORD</Property>
         <Property name="SourceTable" type="STRING">TABLE_NAME</Property>
         <Property name="TagColumn" type="STRING">TAGNAME</Property>
         <Property name="TimestampColumn" type="STRING">TIMESTAMP</Property>
         <Property name="ALLOW_WRITE" type="STRING">Y</Property>
         <Property name="USE_MERCURY" type="STRING">y</Property>
         <Property name="ALLOW_DUPLICATE_TIMESTAMPS" type="STRING">n</Property>
         <Property name="EXT_REF_URL" type="STRING"></Property>
         <Property name="HOSTED_PROG_ID"
type="STRING">ISS.DataSources.Relational</Property>
         <Property name="POOLED_PROG_ID" type="STRING">ISS.AdaptorHost</Property>
         <Property name="INACTIVITY_TIMEOUT_SECS" type="STRING">3600</Property>
         <Property name="INIT_FAILURE_LOCK_SECS" type="STRING">60</Property>
         <Property name="INIT_TIMEOUT" type="STRING">60</Property>
         <Property name="LOG_LEVEL" type="STRING">ERROR</Property>
         <Property name="LOG_TIMING" type="STRING"></Property>
         <Property name="POOL_SIZE" type="STRING">4</Property>
         <Property name="POOLED_PROG_ID" type="STRING">ISS.AdaptorHost</Property>
         <Property name="RAW_RETURN" type="STRING">n</Property>
         <Property name="SUBMIT_TIMEOUT" type="STRING">120</Property>
         <Property name="TokenProperties" type="STRING"></Property>
         <Property name="ValueColumn" type="STRING">VALUE</Property>
       </Profile>
     </Initialise>
   ]]></InitData>
   <SampleMethod>LastKnownValue</SampleMethod>
   <SampleFrequency>3600</SampleFrequency>
   <SinglePoint>false</SinglePoint>
   <MultiFetch>false</MultiFetch>
   <UseReportEndTime>false</UseReportEndTime>
   <StartOffset>0</StartOffset>
   <EndOffset>0</EndOffset>
</GetTimeSeriesData>
```

When the P2 Verify Server Windows service collects the initial data for a report, or someone who is using the P2 Verify web interface recollects data, the settings in this configuration file will be read.

ProgID *or* **TypeName**

This setting is used to create an instance of the adaptor, and only one of these keys should be filled out. Use the **TypeName** for .NET adaptors, and the **ProgID** for COM adaptors and P2 Explorer 2.6 Data Broker.

For example, to collect from the P2 Explorer 2.6 Data Broker, use `BFDataBroker.DataBroker.1` as the value for **ProgID**. For the correct value, refer to the installation documentation of the adaptor.

Debug

This setting controls whether debugging statements are printed. If this is set to `true`, P2 Verify will log extra statements when it runs a collection.

InitData

This setting is used to give an adaptor the adaptor-specific initialisation data. When collecting from the Data Broker, no **InitData** is required.

IsAdaptor

This setting controls whether to use an adaptor or P2 Explorer 2.6 Data Broker. When collecting from the Data Broker, set **IsAdaptor** to `false`. Where connection to an adaptor is required, set this to `true`.

SampleMethod

The sampling method to use for the query. Possible values are `Raw`, `LastKnownValue`, `LinearInterpolate`, and `Average`. The default is `Raw`.

SampleFrequency

The sampling frequency to be used in the request. The default is `3600` (seconds).

SinglePoint

Whether to use a single point fetch or not, with the default being `true`. If this is set to `true`, the end time of the request will be the same as the start time (adjusted for offsets), so that in the final request the start and end time values will be the same. Thus, only a single point value can be returned.

MultiFetch

This setting controls whether to use multiple requests and responses for the collection process. Setting this to `false` will cause a separate request to be performed for every tag entity using this Data Source. Setting it to `true` will result in a single request containing all of the entities. The default is `true`, which is recommended for performance reasons.

UseReportEndTime

If this is set to `true`, the report end time will be used as the start time in the request for data. Setting this to `false` will use the report start time as the start time of the request.

FetchType

The type of aggregation function applied to the returned results.

Options are:

`Normal`: Returns the first value in the data set (this is the default).

`Minimum`: Returns the smallest value from the data set.

`Maximum`: Returns the largest value from the data set.

`Delta`: Returns the difference between the start and end values. The earliest value is always subtracted from the later value.

`Average`: Calculates the mean of the returned values.

StartOffset

An offset in seconds to be added to a report start date (or the end date, if **UseReportEndTime** is `true`) to get the start date for the request. This value can be negative, and the default is `0`.

EndOffset

An offset in seconds to be added to a report end date to get the request end date. This value can be negative, and the default is `0`. This value will not be used if the **SinglePoint** setting is set to `true`.

PutTimeSeriesDataOffset

An offset in seconds to be added to a report start date to get the timestamp used when *submitting* data. This value can be negative, and the default is `0`.

Table Submitter

The **Table Submitter** is a legacy adaptor frequently used with previous versions of P2 Verify. This adaptor is used to submit the contents of a dynamic list directly to the destination table in a database.

> **Note:** Consider replacing this adaptor with one of the standard extension adaptors, for example with SQL Adaptor.

The columns to be submitted are configurable, and it is possible to submit fixed values that are not contained in the dynamic list as well.

A single submitter can be used to submit data from multiple dynamic lists to multiple tables, but the destination tables must reside within the same database.

Before the data is inserted into the database table, the submitter will delete existing data, to avoid having duplicate or deleted data in the destination table.

▸ To use the **Table Submitter**, you must first add it as a new submitter type. Follow the steps in *Adding a Submitter Type* , using the following settings:
Adaptor Assembly: ISS.Verify.Adaptors.Legacy.dll
Class Name: ISS.Verify.Adaptors.DynamicList.TableSubmitter.Adaptor

---

In the configuration file, every entity in the dynamic list has to be added to the **ConfigColumns** section in the same sequence as it exists in the dynamic list.

Additional **Column** sections are added after that, for saving fixed values to the destination table (this is optional).

There are four possible configurations for the submitter, depending on the usage scenario. These are described in the following sections.

Configuration 1

Submit data using the report date as the timestamp for the submitted values (affected by the **UseReportEndTime** and **DateOffset**).

The timestamp column name specified in **DateColumn** is the only key column in the destination table. When deleting existing data, only data with timestamp on the report date will be deleted.

Note: The passwords used in all configuration files, including submitter configuration files, must be encrypted. Use the Secure Setting Builder to encrypt the password.

The example configuration file for this case is:

```xml
<PutDynamicListData>
  <Debug>true</Debug>
  <DatasourceConfig>
    <DatasourceType>[SQLServer/Oracle]</DatasourceType>
    <Datasource>[db_servername]</Datasource>
    <UserId>[db_owner]</UserId>
    <Password>[password]</Password>
  </DatasourceConfig>
  <Tables>
    <TableConfig name="[dynamic_list_name1]">
      <TableName>[destination_table_name]</TableName>
      <!-- submit at the end of the day -1 seconds -->
      <UseReportEndTime>True</UseReportEndTime>
      <DateOffset>-1</DateOffset>
      <!-- data store date column is in UTC time -->
      <UTC>True</UTC>
      <DateColumn>Date</DateColumn>
      <ConfigColumns>
        <Column>
          <Name>NODE</Name>
          <Type>String</Type>
          <Submit>True</Submit>
        </Column>
        <Column>
          <Name>VALUE</Name>
          <Type>Number</Type>
          <Submit>True</Submit>
        </Column>
        <!-- Store the confidence as a fixed value of 100 -->
        <Column>
          <Name>CONFIDENCE</Name>
          <Type>Number</Type>
          <Submit>True</Submit>
          <Value>100</Value>
        </Column>
      </ConfigColumns>
    </TableConfig>
    <TableConfig name="[dynamic_list_name2]">
      ...
    </TableConfig>
  </Tables>
</PutDynamicListData>
```

DatasourceType

This can be SQLServer or Oracle, depending on the destination database.

Datasource

The destination database server name.

UserId

The user account used to access the database.

Password

The password for the user account.

TableConfig Name

> The name of the dynamic list to be submitted.

TableName

> The name of the destination table where the data is to be submitted.

UseReportEndTime*

> This flag determines if the P2 Verify report end time is to be used as the timestamp for the values. This value is set to either `true` or `false`. The default is `false` if this node is not specified, that is, the report start date is used by default.

DateOffset

> The date offset in seconds to be applied to the submitted data (from the report start date).

UTC (TableConfig)*

> Determines if the date value should be converted to UTC time. By default local time is used. This value is set to either `true` or `false`.

DateColumn

> The name of the date or timestamp column in the destination table.

Column Name

> The name of the column to write to.

Type

> The data type of the column, may be `String`, `Number` or `Date`.

Submit

> This flag determines if the value in the dynamic list is to be written to the destination table. This value is set to either `true` or `false`.

Value*

> The fixed value to be stored to the destination table. Additional fixed value nodes must be placed last in the **ConfigColumns** node, after all the dynamic list columns.

Format*

> Specify only if the **Type** has been set to `Date`. It is used to specify the .NET Format string to convert the Date String value into a valid DateTime object.

UTC   (Column)*

> Optionally specify only if the **Type** has been set to `Date`. Determines if the date will be converted to UTC before saving.

* Items marked with an asterisk (*) are optional.

Configuration 2

Submit data using the report date as the timestamp for the submitted values (affected by the **UseReportEndTime** and **DateOffset**).

Note: The passwords used in all configuration files, including submitter configuration files, must be encrypted. Use the Secure Setting Builder to encrypt the password.

There is more than one key column in the destination table, including the timestamp column. For example, the Date and CommentType columns are the key columns in this example:

```
<PutDynamicListData>
  <Debug>true</Debug>
  <DatasourceConfig>
    <DatasourceType>[SQLServer/Oracle]</DatasourceType>
    <Datasource>[db_servername]</Datasource>
    <UserId>[db_owner]</UserId>
    <Password>[password]</Password>
  </DatasourceConfig>
  <Tables>
    <TableConfig name="[dynamic_list_name1]">
      <TableName>[destination_table_name]</TableName>
      <!-- submit at the start of the day with no offset -->
      <UseReportEndTime>False</UseReportEndTime>
      <DateOffset>0</DateOffset>
      <!-- table stores data in local time -->
      <UTC>False</UTC>
      <DateColumn>Date</DateColumn>
      <ConfigColumns>
        <Column>
          <Name>COMMENT</Name>
          <Type>String</Type>
          <Submit>True</Submit>
        </Column>
        <!-- Store the comment type as a fixed value 'Test Comment'. When deleting,
only rows with Date = [report date] and COMMENTTYPE = 'Test Comment' are deleted -->
        <Column>
          <Name>COMMENTTYPE</Name>
          <Type>String</Type>
          <Submit>True</Submit>
          <Value>Test Comment</Value>
          <IsGroup>True</IsGroup>
        </Column>
      </ConfigColumns>
    </TableConfig>
    <TableConfig name="[dynamic_list_name2]">
      ...
    </TableConfig>
  </Tables>
</PutDynamicListData>
```

The additional configuration parameters presented in the example are as follows:

IsGroup

> Determines if the data is grouped by this column. When the existing data is deleted, only data that matches this value will be deleted.

Configuration 3

Submit data using a column value in the dynamic list as the timestamp for the submitted values; the report date is not used. When deleting existing data, all data within the specified **UserSelectedDate** period will be deleted.

Note: The passwords used in all configuration files, including submitter configuration files, must be encrypted. Use the Secure Setting Builder to encrypt the password.

```
<PutDynamicListData>
  <Debug>true</Debug>
  <DatasourceConfig>
```

```
    <DatasourceType>[SQLServer/Oracle]</DatasourceType>
    <Datasource>[db_servername]</Datasource>
    <UserId>[db_owner]</UserId>
    <Password>[password]</Password>
  </DatasourceConfig>
  <Tables>
    <!--
      the list stores the data at the datetime the user selects
      the list will delete all data where timestamp >= report start date and <= report
start date + 23 hours 59 minutes 59 seconds
    -->
    <TableConfig name="[dynamic_list_name1]">
      <TableName>[destination_table_name]</TableName>
      <!-- table stores data in UTC time -->
      <UTC>True</UTC>
      <DateColumn>Timestamp</DateColumn>
      <UserSelectedDate>
        <!-- the offset in seconds to be applied to the report date when deleting data.
the data will be deleted within this time range -->
        <StartDateOffset>0</StartDateOffset>
        <EndDateOffset>86399</EndDateOffset>
      </UserSelectedDate>
      <ConfigColumns>
        <Column>
          <Name>Timestamp</Name>
          <Type>Date</Type>
          <Submit>True</Submit>
          <Format>dd/MM/yyyy HH:mm:ss</Format>
          <!-- Date will be converted to UTC before saving -->
          <UTC>True</UTC>
        </Column>
        <Column>
          <Name>Value</Name>
          <Type>Number</Type>
          <Submit>True</Submit>
        </Column>
      </ConfigColumns>
    </TableConfig>
    <TableConfig name="[dynamic_list_name2]">
      ...
    </TableConfig>
  </Tables>
</PutDynamicListData>
```

The additional configuration parameters presented in the example are as follows:

UserSelectedDate

> The UserSelectedDate node should be present if the dynamic list will provide the timestamp to be used instead of using the report date.

StartDateOffset

> The start offset in seconds from the report date. Used to configure the start of the range that will be deleted when submitting the data.

EndDateOffset

> The end offset in seconds from the report date. Used to configure the end of the range that will be deleted when submitting the data.

Configuration 4

Submit data using a column value in the dynamic list as the timestamp for the submitted values; the report date is not used. A user configurable SQL string may be used to determine which data is to be deleted, in addition to the time period configured in the **UserSelectedDate** node. The custom SQL clause is configured in the **CustomDeleteClause**.

> **Note:** The passwords used in all configuration files, including submitter configuration files, must be encrypted. Use the Secure Setting Builder to encrypt the password.

```
<PutDynamicListData>
  <Debug>true</Debug>
  <DatasourceConfig>
    <DatasourceType>[SQLServer/Oracle]</DatasourceType>
    <Datasource>[db_servername]</Datasource>
    <UserId>[db_owner]</UserId>
    <Password>[password]</Password>
  </DatasourceConfig>
  <Tables>
    <!--
      the list stores the data at the datetime the user selects
      the list will delete all data where Sample Time >= report start date and <=
report start date + 23 hours 59 minutes 59 seconds
      where the node matches the specified criteria
    -->
    <TableConfig name="[dynamic_list_name1]">
      <TableName>[destination_table_name]</TableName>
      <!-- table stores data in UTC time -->
      <UTC>True</UTC>
      <DateColumn>Sample Time</DateColumn>
      <UserSelectedDate>
        <!-- the offset in seconds to be applied to the report date when deleting data.
the data will be deleted within this time range -->
        <StartDateOffset>0</StartDateOffset>
        <EndDateOffset>86399</EndDateOffset>
      </UserSelectedDate>
      <CustomDeleteClause>
        <![CDATA[
          Sample Point LIKE 'A_%' AND Sample Point LIKE 'B_%' AND Sample Point NOT LIKE
'C_%' AND Sample Point NOT LIKE 'D_%'
        ]]>
      </CustomDeleteClause>
      <ConfigColumns>
        <Column>
          <Name>Sample Point</Name>
          <Type>String</Type>
          <Submit>True</Submit>
        </Column>
        <Column>
          <Name>Sample Time</Name>
          <Type>Date</Type>
          <Submit>True</Submit>
          <Format>dd/MM/yyyy HH:mm:ss</Format>
          <!-- Date will be converted to UTC before saving -->
          <UTC>True</UTC>
        </Column>
        <Column>
          <Name>Sample Value</Name>
          <Type>Number</Type>
          <Submit>True</Submit>
```

```
            </Column>
        </ConfigColumns>
    </TableConfig>
    <TableConfig name="[dynamic_list_name2]">
     ...
    </TableConfig>
  </Tables>
</PutDynamicListData>
```

The additional configuration parameters presented in the example are as follows:

CustomDeleteClause

> The Custom Delete Clause is an additional SQL clause to be used when deleting existing data. In the example above, the SQL statement executed to delete the existing data becomes:

```
DELETE FROM [destination_table_name]
WHERE Sample Time >= [report date]
AND Sample Time <= [report date]+86399
AND Sample Point LIKE 'A_%'
AND Sample Point LIKE 'B_%'
AND Sample Point NOT LIKE 'C_%'
AND Sample Point NOT LIKE 'D_%';
```

Post-submit Notification

The Post-submit Notification system may be used to trigger an action when a report is submitted and reaches a pre-determined state. Some examples of post-submit actions are:

- Sending an email to a group of users.

- Calling a web service or application to run.

The name of the data area and the report date may be passed to the application. For details, refer to *Configuring a Post-submit Action* (see page 101).

The status of post-submit actions may also be monitored from the *Post-submit State status page* (see page 101).

Configuring a Post-submit Action

Post-submit Actions are functions that are automatically executed after a report has been submitted. The *Post-submit State* (see page 101) screen shows details about each execution.

These functions should be contained in external assembly files and referenced from P2 Verify database.

It is the responsibility of the function to return appropriate messages to indicate the progress and/or success of the execution.
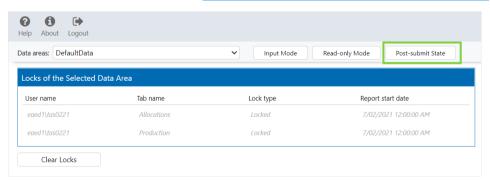
Post-submit State

The Post-submit State screen provides information on the execution outcome of post-submit actions that are configured in the `SUBMIT_NOTIFICATION` table in the Verify database.

For details on the use, setup, and configuration of post-submit actions, refer to *Post-submit Notification* (see page 101).

To view the post-submit states:
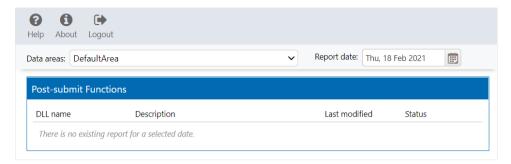
1. Go to the Area Picker.

    The URL will look something like: http://localhost/verify/AreaPicker.aspx

2. Click **Post-submit State**.

The Post-submit State screen appears, listing the results of the post-submit actions that have been executed.

3.   You can filter the results by date and data area.

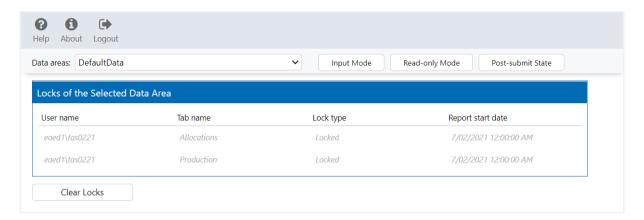If there is more than one shift in a day, you can also select the shift start time.

4.   Examine the **Post-submit Functions** section that lists details of the post-submit actions.

5.   Click any of the rows in the post-submit functions section to display further information in the Post-submit Log (such as success, failure, exception details).

This information is retrieved from the P2 Verify database, and it is saved there by each post-submit action when it runs.

Clearing Locks

For Administrators, the Area Picker page also shows a list of the users currently editing the selected area.

In certain circumstances, you may want to clear the locks that the users have in an area. A lock prevents more than one user entering data at the same time into an item, such as a tab. You may want to clear a lock if a user has left their browser open for an extended period of time, or to give one user emergency priority over another.

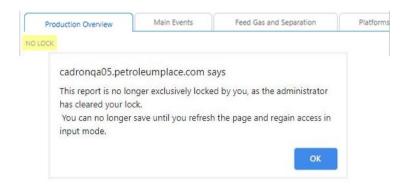▶ To clear **all** locks in the selected data area, click **Clear Locks**.

Note: For any user editing reports in this data area, their session will end immediately and **they will not be able to save their changes**. All the locks in the area will be cleared and other users will then be able to edit the same pages.

When locks are cleared, the message *NO LOCK* appears in the upper left corner of the page, to all users who have been affected by the lock clearing.

▶   Click the NO LOCK text to view the details of the message.

For example, if user A is in *Input mode* of a report before the administrator clears all locks, the NO LOCK message appear on user A's screen. At this time, user A will no longer be able to save any changes.

Tip: The recommended course of action is to note down all the changes made, refresh the report, and re-enter the data again.

In the meantime, if another user saves or submits the same report, a message will appear to user A, informing them that the report has already been modified by another user.

Data collection events that happen while a user is accessing a report in input mode will also trigger the same message.

Appendix A. Date/Time Format Strings

The **VFYDateTime** entity can be customised to use a custom Date/Time format string. This can be done by creating a custom format string using a combination of any of the following format strings to represent each of the Date/Time parts:

d

Represents the day of the month as a number from 1 through 31. A single-digit day is formatted without a leading zero.

dd

Represents the day of the month as a number from 01 through 31. A single-digit day is formatted with a leading zero.

h

Represents the hour as a number from 1 through 12, that is, the hour as represented by a 12-hour clock that counts the whole hours since midnight or noon. Consequently, a particular hour after midnight is indistinguishable from the same hour after noon. The hour is not rounded, and a single-digit hour is formatted without a leading zero.

hh

Represents the hour as a number from 01 through 12, that is, the hour as represented by a 12-hour clock that counts the whole hours since midnight or noon. Consequently, a particular hour after midnight is indistinguishable from the same hour after noon. The hour is not rounded, and a single-digit hour is formatted with a leading zero.

H

Represents the hour as a number from 0 through 23, that is, the hour as represented by a zero-based 24-hour clock that counts the hours since midnight. A single-digit hour is formatted without a leading zero.

HH

Represents the hour as a number from 00 through 23, that is, the hour as represented by a zero-based 24-hour clock that counts the hours since midnight. A single-digit hour is formatted with a leading zero.

m

Represents the minute as a number from 0 through 59. The minute represents whole minutes passed since the last hour. A single-digit minute is formatted without a leading zero.

mm

Represents the minute as a number from 00 through 59. The minute represents whole minutes passed since the last hour. A single-digit minute is formatted with a leading zero.

M

Represents the month as a number from 1 through 12. A single-digit month is formatted without a leading zero.

MM

Represents the month as a number from 01 through 12. A single-digit month is formatted with a leading zero.

MMM

Represents the abbreviated name of the month.

MMMM

Represents the full name of the month.

s

Represents the seconds as a number from 0 through 59. The second represents whole seconds passed since the last minute. A single-digit second is formatted without a leading zero.

ss

Represents the seconds as a number from 00 through 59. The second represents whole seconds passed since the last minute. A single-digit second is formatted with a leading zero.

t

Represents the first character of the A.M./P.M. designator.

tt

Represents the A.M./P.M. designator.

yyyy

Represents the year as a four-digit number.