IOT15597-L

Pimping Your IoT Ride

Brian Sherman
Autodesk

Learning Objectives

- Learn how to connect a sensor/device to Fusion Connect
- Learn how to start to track your product information
- Learn how to build an application based on your product
- Learn how to create a set of visual-reporting elements

Description

Join our Fusion Connect technical wizard for a hands-on, connected product lab that will lead you through the creation of your first connected product application in Autodesk, Inc.'s, new Internet of Things platform. This session features Fusion Connect and SeeControl (now Fusion Connect).

Your AU Expert(s)

Brian Sherman is a senior software engineer on the Fusion Connect team at Autodesk, based out of San Francisco. He has been with this team for over two years and joined Autodesk through the acquisition of the former SeeControl, the IoT platform that became Fusion Connect. He spends his time building new features for the cloud service, integrating the service with new devices, training partners and customers on the use of Fusion Connect, and building applications using the service itself. Brian has a Bachelor’s degree in Computer Science and Political Science from the University of California, Berkeley.

Introduction

As the learning objectives indicate, the purpose of this lab is to give you an introduction to Fusion Connect, Autodesk’s IoT cloud service, and there is no better way to do this than to try it yourself. Fusion Connect is a cloud-native IoT application development service, with a no-coding toolset that allows you to build, maintain and understand an application without needing to understand any programming language or write a single line of code. Even the simplest IoT solution has a lot of moving parts, a lot of different pieces, such as establishing communication with hardware, figuring out how all the data are stored, building business processes atop that data and analytics to coax the value out of the raw data, and building visuals to display data and analytics. Fusion Connect provides tools to accomplish all of these, to build the interconnected components that make up an IoT solution. This lab aims to explain how you can build each of these components.

In this lab, we’re going to start with a nascent application that has a bare minimum of each of these components in place; this will provide context as to how each piece fits together, and will provide a foundation atop which we can build real depth into the application. You’ll be using the

same tools that were used to build this foundation in the first place, so once you understand each of the components and how they fit together, building this foundation would be simple!

Exercise Background

The use case that we'll be implementing in this lab is as follows: we'll be assuming the perspective of a manufacturer of wood chippers, which is early on in its IoT journey. We've retrofitted some of our wood chippers with three sensors so far: one sensor monitoring how much fuel a wood chipper has left in its tank; another sensor that monitors the temperature of its engine, and one sensor that monitors the amount of time that the wood chipper has been running. A controller on the wood chipper will sample these sensors once every minute and then send a message to Fusion Connect with this information, over the HTTP protocol.

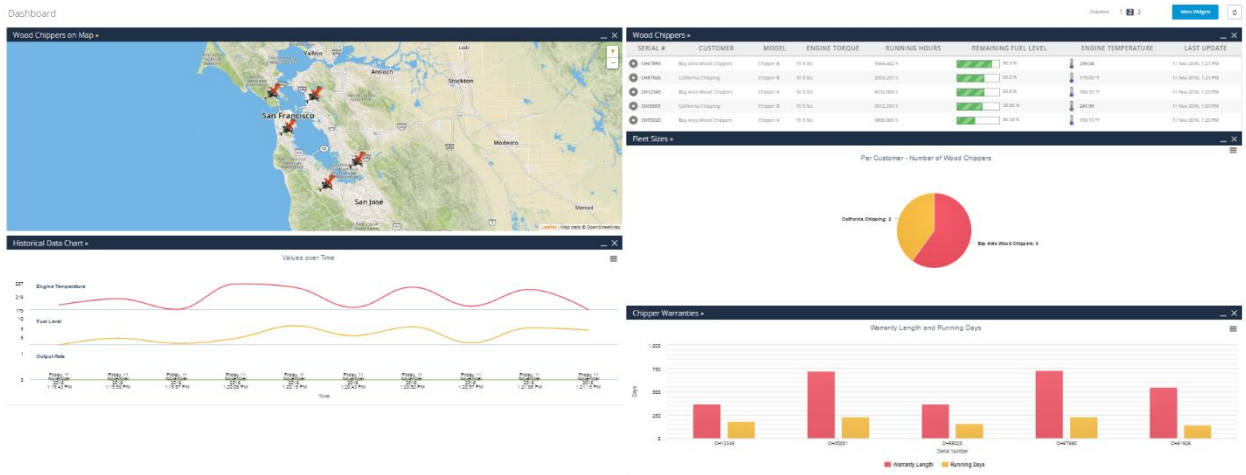


FIGURE 1: THE STARTING DASHBOARD

As it stands, the application in Fusion Connect allows us to do basic remote monitoring of the connected wood chippers we've sold; to monitor where they lie on a map, whether they're currently running, and their current statuses. We can track the values that they've sent over time, as a window into how our wood chippers actually perform in the field. We can keep track of how long these wood chippers are under warranty, and compare that to how long they've been running, to help us potentially identify sales opportunities. Finally, this application not only models the wood chippers we've sold and that are in the field, but it models the customers of ours that have bought those connected wood chippers, tracking how many wood chippers they own, and who we should contact in case of any problems.

Exercise 1 – Monitoring a New Sensor

We've just retrofitted our wood chippers with a new sensor that measures the rate at which they're outputting ground material, and their controllers have been updated with a new firmware version to poll that sensor and include the output rate in its messages to Fusion Connect. Let's integrate this new sensor with our application.

Device Connectivity / Telling the application to look for that new value in messages

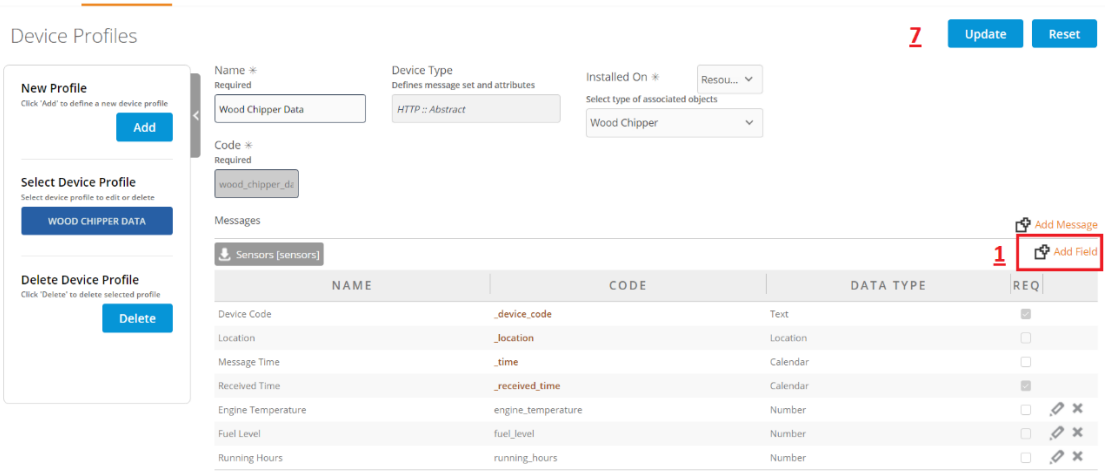
We're currently in runtime mode; what an end user of this application experiences, though different classes of users can be given the ability to see varying amounts of these pages. As administrative users, though, we have the ability to make changes to the application itself. To do this, we must click the "Design Mode" checkbox in the page header to switch into design mode.

FIGURE 2: THE DESIGN MODE CHECKBOX

The landing page of design mode is the "Account Center", which shows the skeleton of the application, all of the components we've defined that result in the finished product that our users will see on the other side of the coin, the runtime experience. The main menu of design mode shows all of the different tools you can use to configure the application, instead of the custom menu of the runtime experience. For the rest of this handout, when I refer to selecting a certain link or page from one of the tabs of the main menu, I'll use the notation ***Tab -> Page***.

FIGURE 3: THE MAIN MENU IN DESIGN MODE

From the "Connect" tab of the main menu, select the "Device Profiles" page, i.e. ***Connect -> Device Profiles***. This page allows us to define the ways that our physical devices will be communicating with our application. In this case, we have a single device profile, called "Wood Chipper Data", and all of our wood chippers will communicate using this method. This profile informs the system that messages from our wood chippers will arrive over the HTTP communication protocol, will send a type of message with the name "Sensors", which will contain three fields: engine temperature, fuel level, and running hours. In order for the system to start looking for a fourth value in the "Sensors" messages it receives, all we need to do is add another field here.



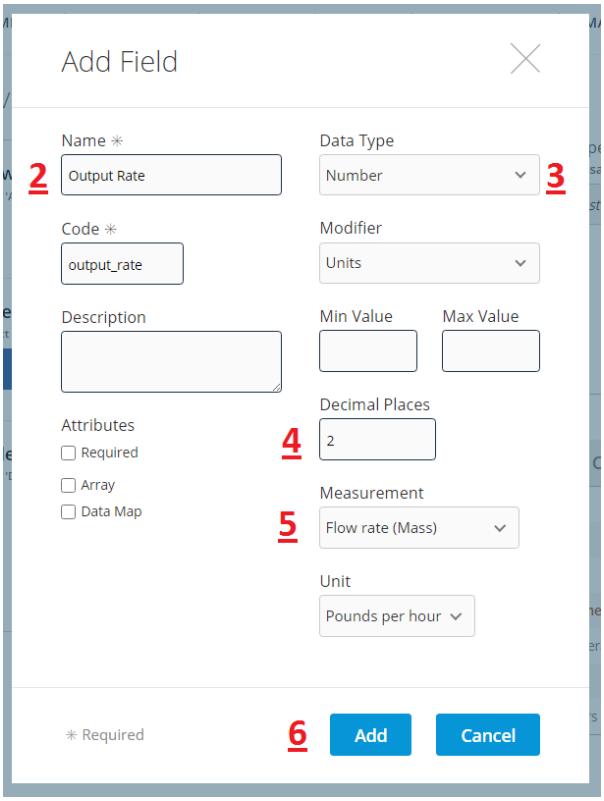


FIGURE 4: ADDING A FIELD TO THE DEVICE PROFILE

1. Press the “Add Field” button to bring up the panel to add a new field.
2. Give this field the *name* **Output Rate**; the *code* will automatically populate with **output_rate**, and do not change this. The code dictates what key the application will look for in messages.
3. Set the *data type* of the field to **Number** to inform the system that values of this field will be floating point numbers. Then, from the *modifier* drop-down, select **Units** so we can specify the meaning behind the number that will arrive.
4. For the *Decimal Places* input, enter **2**.
5. For the *measurement* of the field, select **Flow rate (Mass)** since the wood chippers are sending the rate at which they are outputting ground material. The specific *unit* “Pounds per hour” will automatically be selected, and you can leave this as is.
6. Press the “Add” button to add this field to the message profile the application will expect.
7. Press the “Update” button to save your changes to this device profile. Anywhere in Fusion Connect with buttons such as these, you can always revert your unsaved changes using the “Reset” button, and no changes will be saved without pressing “Update”.

Registering a Device with your Application

This device profile defines how the gateways will communicate with our application. Additionally, note the “Installed On” box in the top right of the page; this profile also informs the system that these devices are installed upon and monitoring wood chippers, which are modeled in the application already. We can then register each gateway that will be sending data, so that the

application will accept messages from those gateways. Go to the ***Connect -> Devices*** page from the main menu. Here you can see each of the devices that are already registered with the application; which of Fusion Connect's cloud adapters is used to receive its messages, which of the application's device profiles defines the structure of its messages, and which wood chipper – or, more accurately, the digital twin thereof – that this device is installed upon. To register a new device, we first need to create a digital twin of the wood chipper upon which that device is installed.

If you click the "Add Device" button on the left, you can see that registering a new device manually from within this administrative perspective is just as simple as providing the code with which a device will identify itself in its messages, selecting the profile defining those messages, and the digital twin of the wood chipper upon which it is installed. However, this means that we then have to separately create a new wood chipper in the application, increment the customer's count of wood chippers, and so on; plus, this is a much higher level of access than we would ever want to give out to our users; we may want them to be able to register a new wood chipper, but never to edit or delete any others.

Luckily, there is a better way. Fusion Connect comes equipped with a powerful form builder, that allows us to define which set of information a user can provide, and define exactly what the application does with that information. Go to the ***View -> Forms*** page from the main menu, and the first form that you'll see is the "Add Chipper" form. This is a form that we can safely display to our users, and when a user submits the form, the application will take care of everything: it will create a digital twin for a wood chipper with the information we provide, register a new device, and increment the customer's count of wood chippers. Fill out the form as shown below; only two of the inputs need to be specific values to make sense in this example.

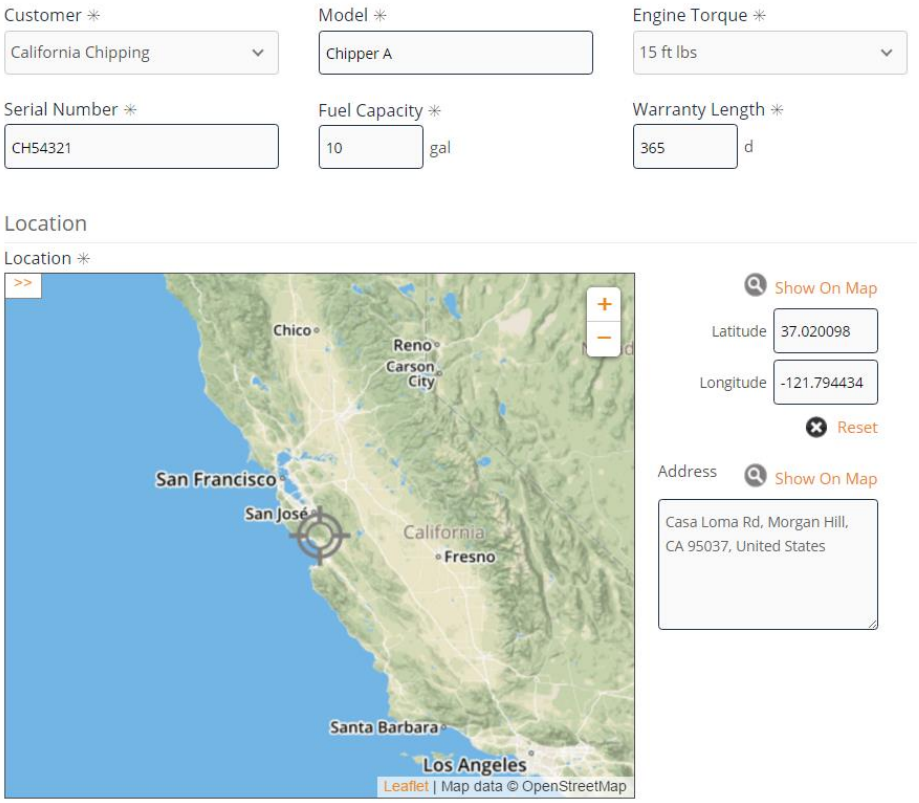


FIGURE 5: THE ADD CHIPPER FORM, FILLED OUT

1. For the *Customer* field, select either customer from the drop-down.
2. For the *Serial Number* field, enter **CH54321** into the input box.
3. For the *Model* field, enter any value you want, such as **Chipper A**.
4. For the *Fuel Capacity* field, enter **10** into the input box.
5. For the *Engine Torque* field, select either option from the drop-down box.
6. For the *Warranty Length* field, enter any number of days you want.
7. For the *Location* field, zoom and drag around on the map until the crosshair is centered on the location you want.
8. Press the "Submit" button in the top right of the form to enter these values.

You can now go back to the ***Connect -> Devices*** page from the main menu, and verify that there is a device registered with the code **CH54321**. Your application is now configured to receive messages from this new wood chipper, and it is configured to accept an output rate value from all wood chippers. We can verify these messages by going to the ***Monitor -> Device Messages*** page. If you select the **Inbound** tab along the top, you can see a list of the messages that your application has received, and from which devices. Select the magnifying glass icon in the rightmost column of the table to bring up a panel to view any individual message; the most recent ones should all have the new output rate value you've defined, since the application is now prepared to receive them!

Storing the New Information in your Application

The application is now parsing an output rate from the messages it receives from our wood chippers, but it is currently not doing anything with that information. The application is currently extracting the fuel level, engine temperature, and running hours from each message it receives, then mapping that information onto the digital twin of that wood chipper to update its current status; the application also stores those fields inside a historical record belonging to that wood chipper, a snapshot of what the chipper is at this point in time, so that we can then look at how these values change over time. So, we simply need to configure the application to do the same with the new output rate field.

First things first, let's create a place to store this information in the wood chippers we've modeled in the application, a new bucket for that piece of information. Go to the ***Model -> Resource Types*** page. To explain the nomenclature, a "resource" is an asset that is modeled in the system. Here you can see the template of a wood chipper, all of the information that we store for a wood chipper; each digital twin we create for our physical assets is built using this template. Adding a field to this resource type works exactly the same way as it did for adding to a device profile, since data are normalized so they can be seamlessly transferred from messages or forms into the actual objects and historical records we're saving in the application.

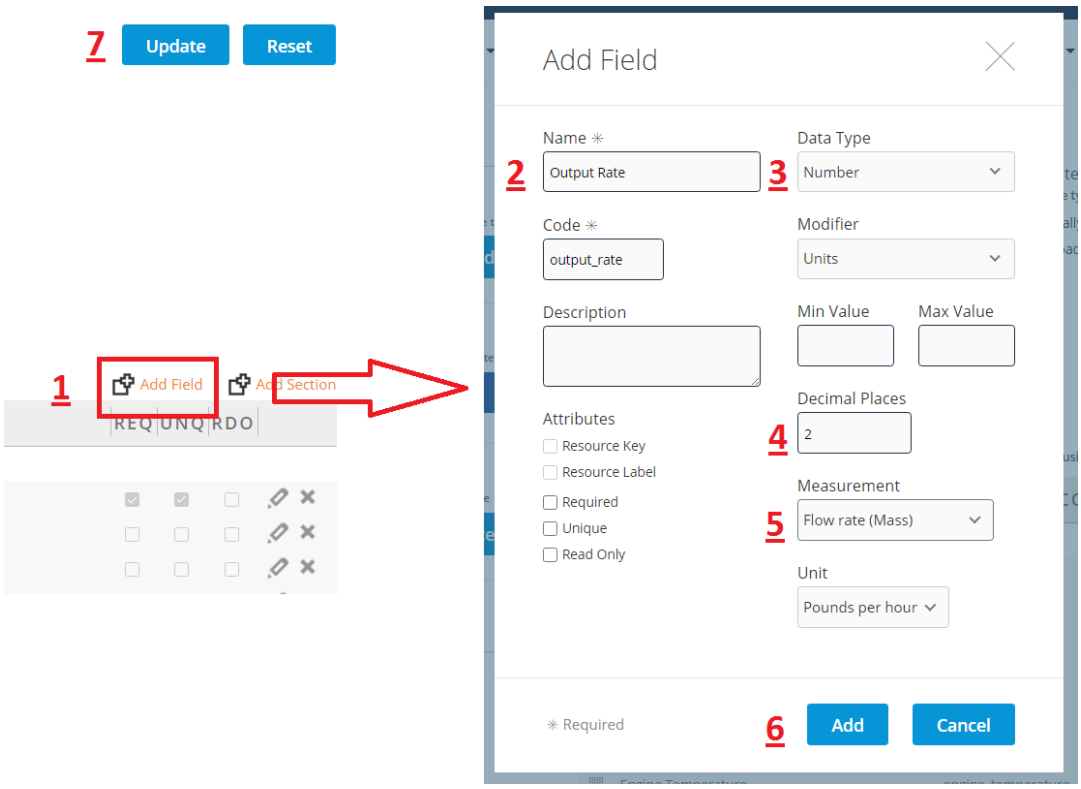


FIGURE 6: ADDING A FIELD TO THE RESOURCE TYPE

1. Press the “Add Field” button to bring up the panel to add a new field.
2. For the *name* attribute, enter **Output Rate**.
3. For the *data type* attribute, select “Number”. Then select **Units** from the *Modifier* drop-down, and as before, select **Flow rate (Mass)** for the *Measurement* drop-down that appears.
4. For the *Decimal Places* input, enter **2**. While each value is always stored with complete precision, this ensures that exactly 2 decimal places will be shown to users.
5. Press the “Add” button to add this field to the wood chipper template.
6. Press the “Update” button to save your changes to the wood chipper type.

Mapping the Message Value to the Wood Chipper

We've now configured the application to receive output rate values at the edge of the system, and we've created a place to store that information in our wood chipper records. Now all that remains is to configure the application to take the value from an incoming message and put it in that new bucket. Go to the ***Configure -> Routines*** page. A routine is a flow of business logic that is executed in response to a triggering event, such as a message being received, a form being submitted, or a timer running out; this is how an application performs all business processes, calculations and analytics. Select the **Device Message Received** routine from the "Select Routine" section on the left of the page. This is the flow of business logic that the system executes whenever it receives a message, and it is currently doing two things: the first block is an "Update" action, which updates the current status of our wood chipper with the information in the message; the second block is an "Add" action, which creates a historical record of the state of this wood chipper in time. Let's modify that first action; click the first block, and an edit **(…)** and delete **(X)** button will appear, so click the edit button, to bring up the panel shown in the screenshot below.

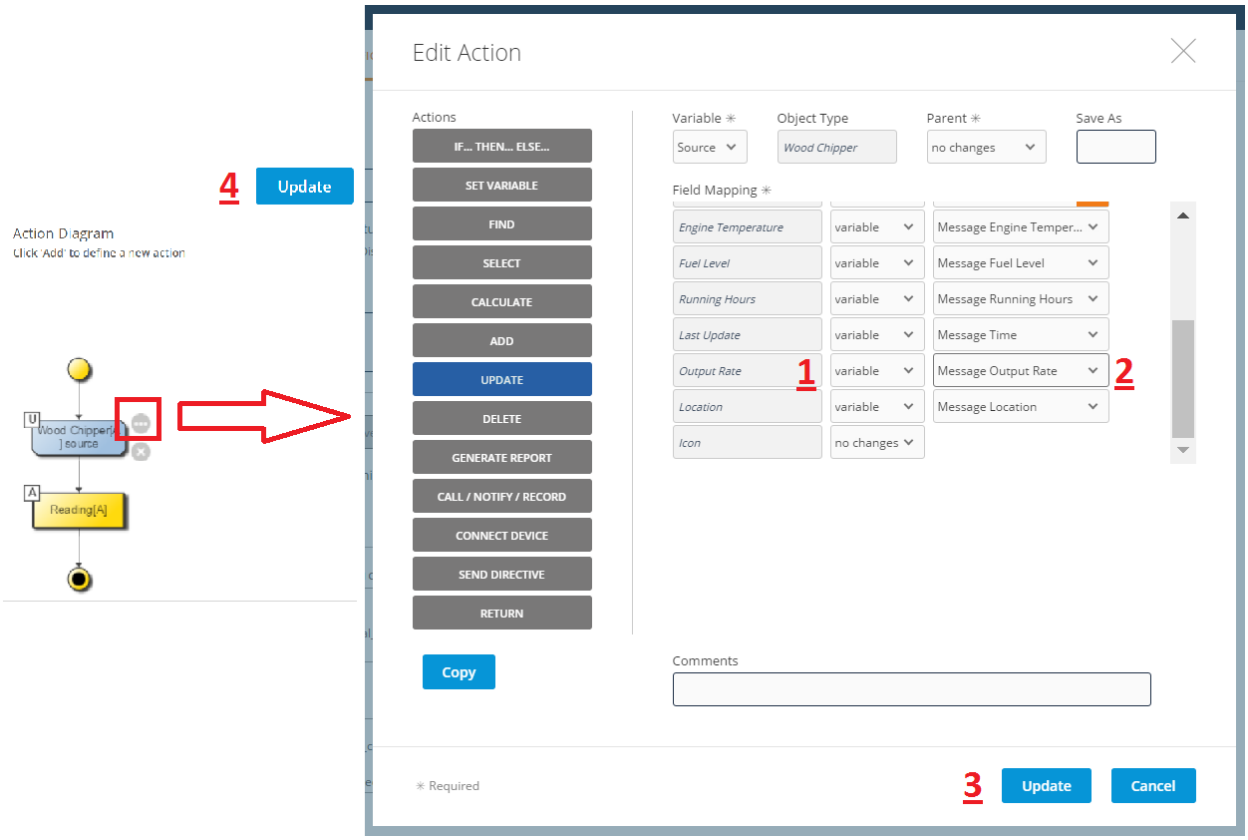


FIGURE 7: MAPPING THE OUTPUT RATE FROM MESSAGES TO WOOD CHIPPERS

On this panel, there is a list of each of the fields of the wood chipper, and what their values are being set to, if any changes are being made at all. You can see that a message doesn't change the serial number, model, etc. of the stored wood chipper, but it does update the actual data fields, and set the status of the wood chipper to true, or running. Currently, the new "Output Rate" field we've defined isn't being set to anything.

1. Select the drop-down next to the “Output Rate” box that currently is set to “no changes”. Change this to **variable**.
2. From the drop-down that appears to the right, select **Message Output Rate** to set the value of the wood chipper to the value that arrives in the message.
3. Press the “Update” button to save your changes to this routine action.
4. Press the “Add” button in the top right of the “Routines” page to save your overall changes to the routine.

The digital representations of our wood chippers should now be constantly updated with the latest information from the field. To verify this, let’s go to the ***Edit -> Resources*** page to see the direct, admin view of each of the records of the application. It may take a minute for the wood chippers to be fully populated with output data, since they’re only reporting once every minute.

Displaying Output Rate Information at Runtime

While, as high-level admin users, we can view the output rates of wood chippers here on the “Resources” page, the whole point of the application is to show this data on the reports of the runtime experience for all users. So, let’s now integrate this new information into the widget we saw earlier on the dashboard, which showed the current status of each of our wood chippers. Go to the ***Configure -> Queries & Reports*** page. This is where we define of all the visuals that display data to the end user. Each report has two parts: first, in the “Select” section, we define what data we want to include in our report. Select the **Wood Chippers** report from the “Select Query” list on the left filter. We can see in the “Select” section that this report will fetch all of the wood chippers stored in the application, and retrieve the serial number, model, fuel capacity, etc. of each; furthermore, for each wood chipper, the system automatically matches it to the customer that owns it, so that each record it retrieves will also contain the name of that customer. Let’s tell it to also retrieve the current output levels.

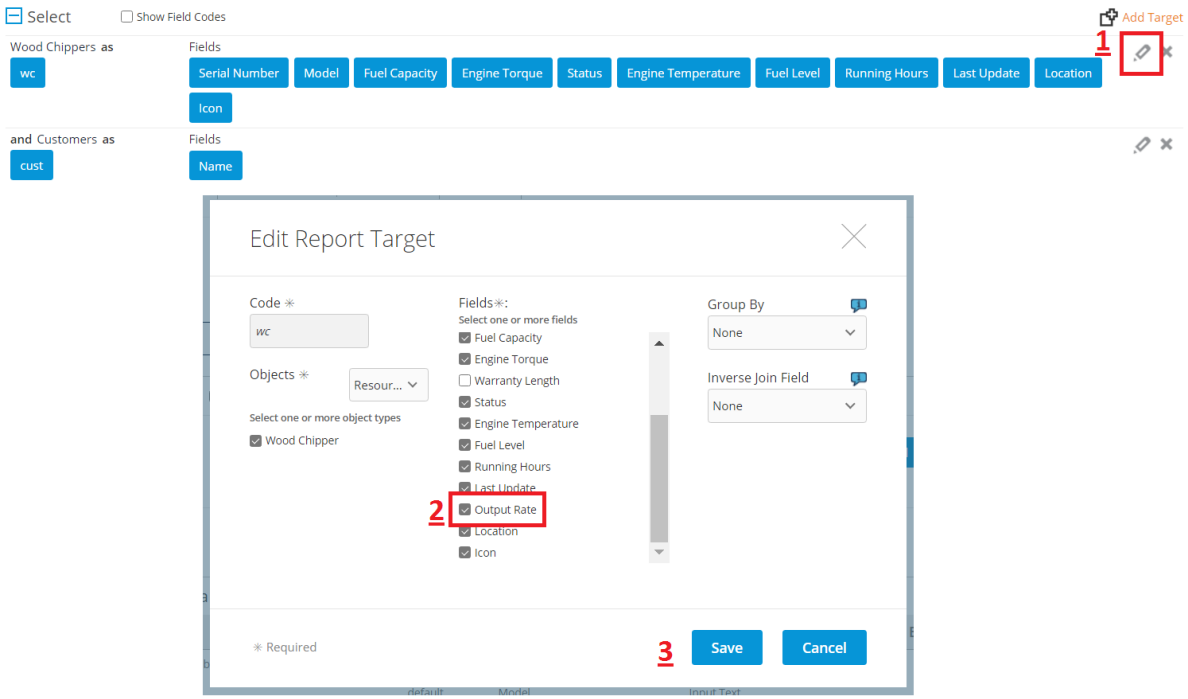


FIGURE 8: ADDING OUTPUT RATE TO THE REPORT TARGET

1. Press the “Edit” button (the button that looks like a pen) to the far right of the screen for the “Wood Chippers” target.
2. In the “Edit Report Target” panel that appears, the second column will contain a list of all of the possible fields of a wood chipper to retrieve. Scroll down this list and check the box for “Output Rate”.
3. Press the “Save” button to save the changes to this target.
4. Press the “Update” button to save your progress on this report.

Where the first part of a data query defines what data are retrieved, the second part defines how that data are displayed to the user, in the “Display As” section of the page. There are a number of built-in ways that Fusion Connect allows you to display data: tables, charts, maps, gauges, etc. Currently, this report contains two views, or ways of displaying data: a table view and a map view, both of which we saw as widgets on the dashboard. Let’s add a new column to the table to show the current output rate of each wood chipper.

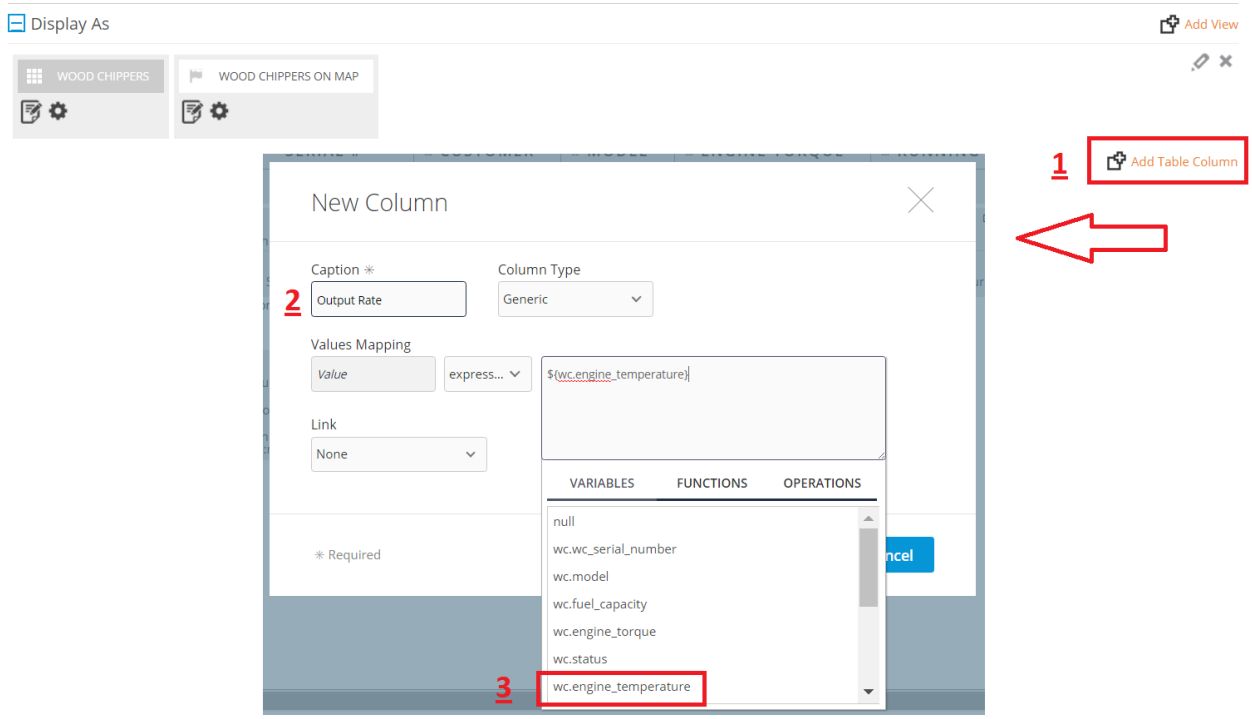


FIGURE 9: ADDING A NEW COLUMN TO THE TABLE

1. Press the “Add Table Column” button on the right side of the page in the “Display As” section.
2. Give the column a header, **Output Rate** for example, but you can choose whatever would be meaningful to you.
3. Instead of simply setting the value of this column to be a variable, as before, let’s use an *expression* instead. Fusion Connect supports a variant of Java Expression Language to allow you to build more complicated statements and values. Press the orange “F” button to bring up the expression builder; here, you can see that all the same fields are available under the “Variables” tab. For now, simply click the **wc.output_rate** option

from the “Variables” tab. You should see **${wc.output_rate}** appear in the box above, which is simply the Expression Language notation to reference a variable.
4. Click elsewhere on the “New Column” panel to minimize the function builder, then press the “Add” button to add this column to the table.
5. Press the “Update” button at the top of the page to save your changes to the report.

You can verify that this was successful by going back to the ***View -> Dashboard*** page from the main menu; you should see that the tabular dashboard widget now contains a column for the current output levels of each of the wood chippers.

Adding Output Rate to our Historical Records

While we’re now successfully monitoring the current output rates of our wood chippers in our application, we’re still only tracking engine temperature, fuel levels, and running hours historically. The process for tracking output rates historically is nearly identical to tracking their current values: add a bucket for this information in our historical record template, map the value received from messages to the new field in the historical record type, and then add this information to our runtime reports showing historical data over time.

Go to the ***Model -> Activity Types*** page; as a point of nomenclature, an “activity” in Fusion Connect is a timestamped record belonging to one of our resources. The “Reading” activity type stores a snapshot of our wood chipper data fields in time. Just like before, let’s add output rate to this template.

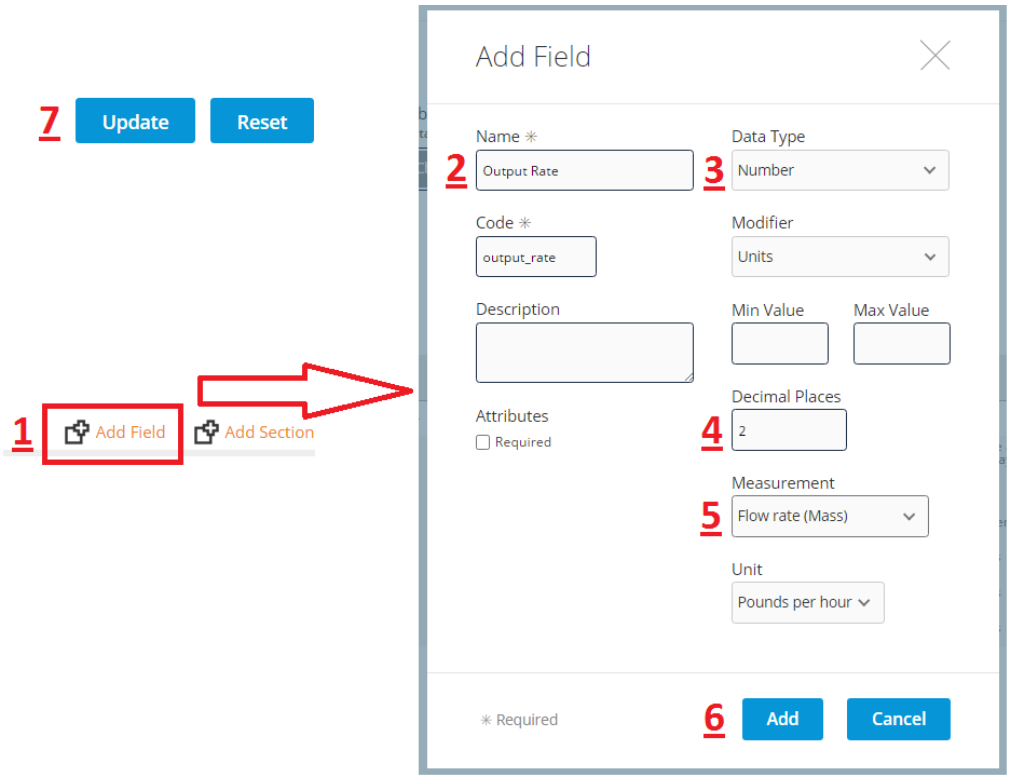


FIGURE 10: ADDING OUTPUT RATE TO THE ‘READING’ ACTIVITY TYPE

1. Press the “Add Field” button to bring up the panel to add a new field.
2. For the *Name* input, provide a value like **Output Rate**.

3. For the *Data Type* attribute, select **Number**, then select **Units** from the *Modifier* drop-down list.
4. For the *Decimal Places* attribute, enter 2.
5. For the *Measurement* attribute, select **Flow rate (Mass)**.
6. Press the "Add" button to add this field to the "Reading" template.
7. Press the "Update" button in the top right of the page to save your changes.

Mapping Output Rate from Messages to our Historical Records

Go to the ***Configure -> Routines*** page, and once again select the **Device Message Received** routine from the "Select Routine" list on the left of the page. This time, let's edit the *second* action of this routine, which is responsible for creating a historical record to store the information from the message.

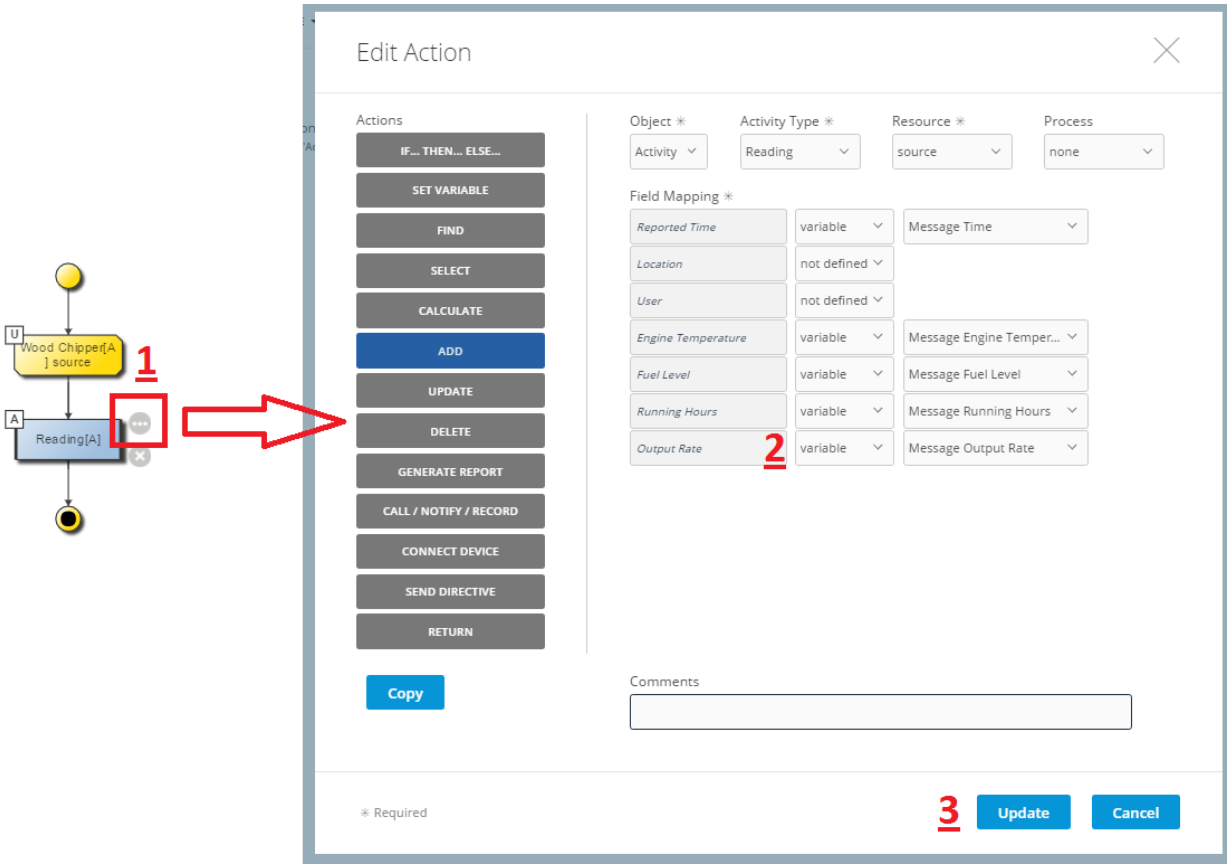


FIGURE 11: MAPPING OUTPUT RATE FROM MESSAGES TO READINGS

1. Click the second action – with the "A" tag in the top left of the block – to select it. Click the edit button **(…)** to bring up the panel to edit the routine action.
2. For the drop-down to the right of the "Output Rate" field, change "not defined" to **variable**. Select **Message Output Rate** from the drop-down that appears to the right.
3. Press the "Update" button to save your changes to this routine action and close the panel.
4. Press the "Update" button in the top right of the "Routines" page to update the routine as a whole with your changes.

Display Historical Output Rates on Reports

Our historical records will now begin to be populated by the data from the messages our application is receiving. So, we can now add this data to our reports. Go to the ***Configure -> Queries & Reports*** page. This time, select the **Historical Data** query from the list on the left side of the page. In the "Select" section, you can see that this report is fetching all the readings in the application, combining them with the serial number and model of the wood chippers to which they belong, as well as to the name of the customer that owns the matching wood chipper. In the "Display As" section, you can see that this report displays these historical readings as a table as well as a series of line charts, the latter of which we've already seen on the dashboard. As before, first we must tell the report to also fetch the output rate from our readings.

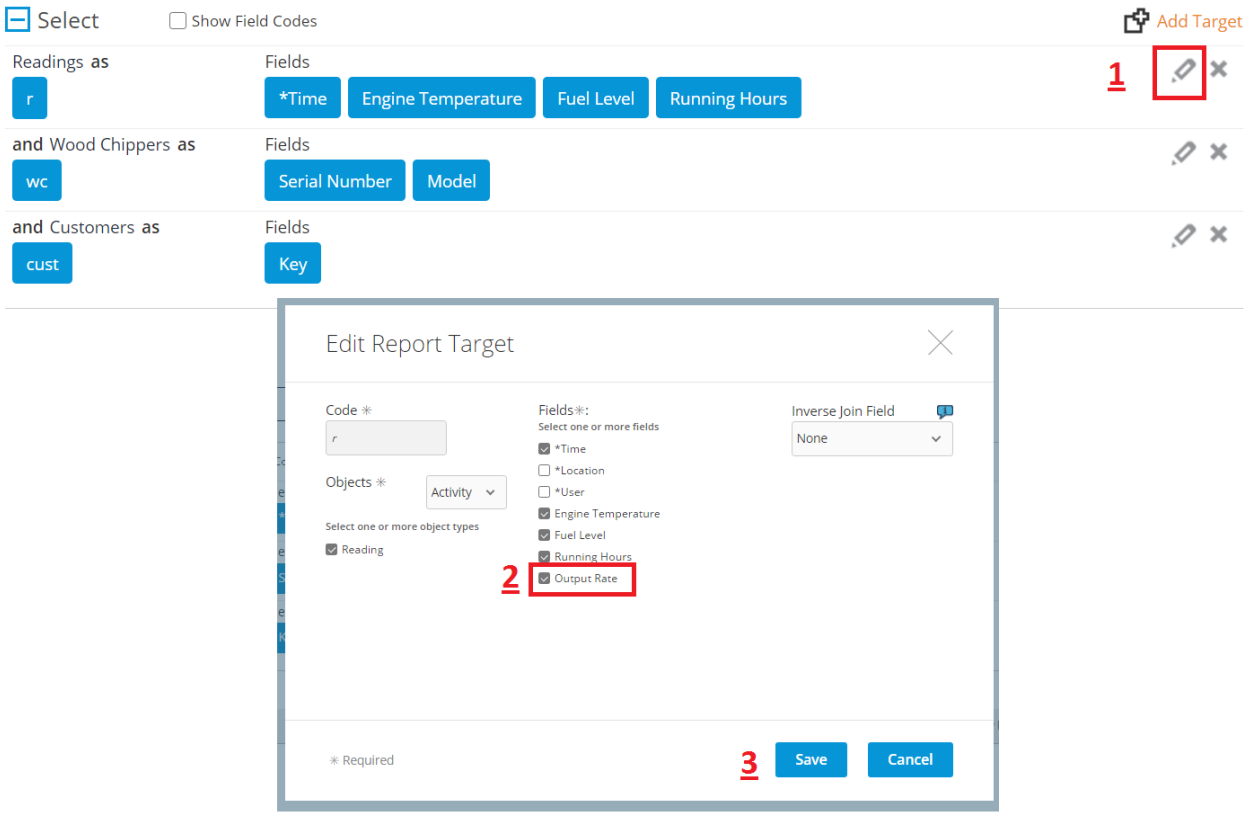


FIGURE 12: ADDING OUTPUT RATE TO THE READINGS TARGET

1. Press the edit button – the button which resembles a pen – to the right of the "Readings" target, to bring up the panel to edit the target.
2. From the second, "Fields" column of this panel, check the box for **Output Rate**.
3. Press the "Save" button to save your changes to the target, then press the "Update" button in the top right of the "Queries & Reports" page to save your progress on this report.

Now, let's add output rate as a column of the table view. Instead of doing this manually like we did before, this time let's use a shortcut, which is shown in the screenshot below.

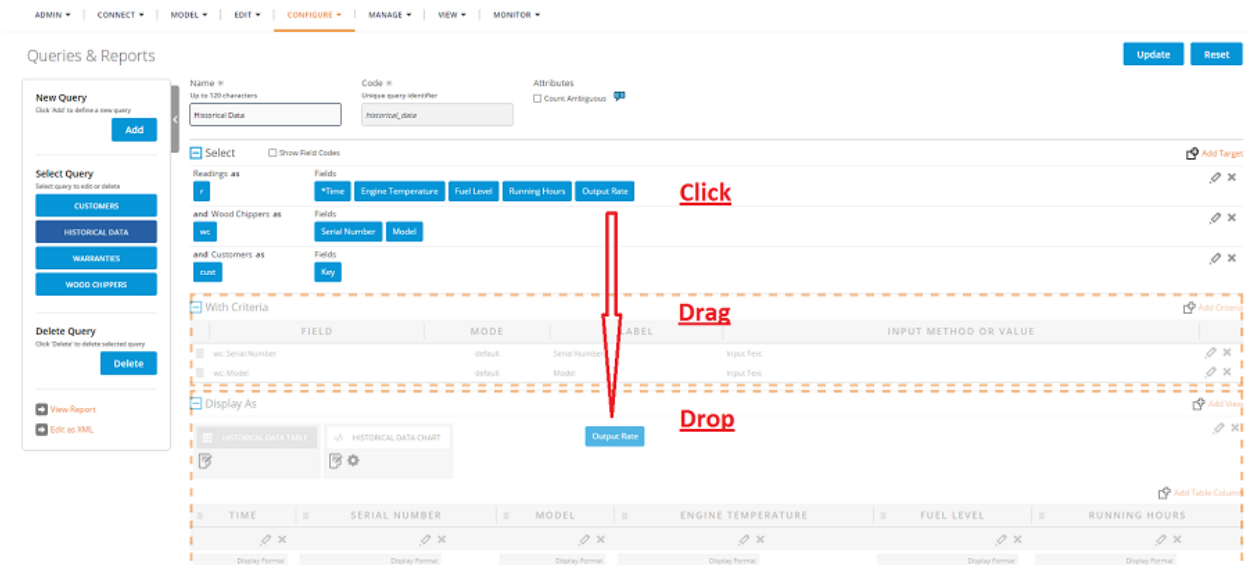


FIGURE 13: ADDING A TABLE COLUMN THROUGH DRAG AND DROP

1. Click and hold the blue box in the “Select” section that is labeled “Output Rate”
2. Drag this box into the “Display As” section of the page, which should now be bordered by a broken orange line, and drop the box in the section.
3. Press the “Update’ button in the top right of the page to save your changes.

To verify that this has succeeded, press the “View Report” button on the left side of the page, below the list of reports. This is a shortcut to view this particular report, rather than view the dashboard or going through the main menu. You should see that there is now a new column containing the output rate for each reading. Once you verify this, you can click the “Configure Report” button in a similar position on the left side of this page to return to the “Queries & Reports” page, or you can once again select ***Configure -> Queries & Reports*** from the main menu and select **Historical Data** on the left.

As a more interesting exercise, let’s now update the chart view that we saw on the dashboard. Click the box for **Historical Data Chart** at the top of the “Display As” section to select the chart view of this report. You can see that the “Display As” section now appears differently than when a table view was selected, which is because there are different components that make up a chart view. The “Chart Outline” box allows you to define whether this is a line chart, bar chart, pie chart, etc., to label the axes on the graph, to specify the title atop the graph, and so on. A chart view then has a number of data series that actually populate this graph, such as the individual lines on the chart. There are already two lines on this chart: one that uses time as an X-value and the engine temperature of readings as a Y-value; the other uses time as an X-value and fuel level as a Y-value; every reading will result in a single data point in each of these lines. Let’s add a new line to this chart.

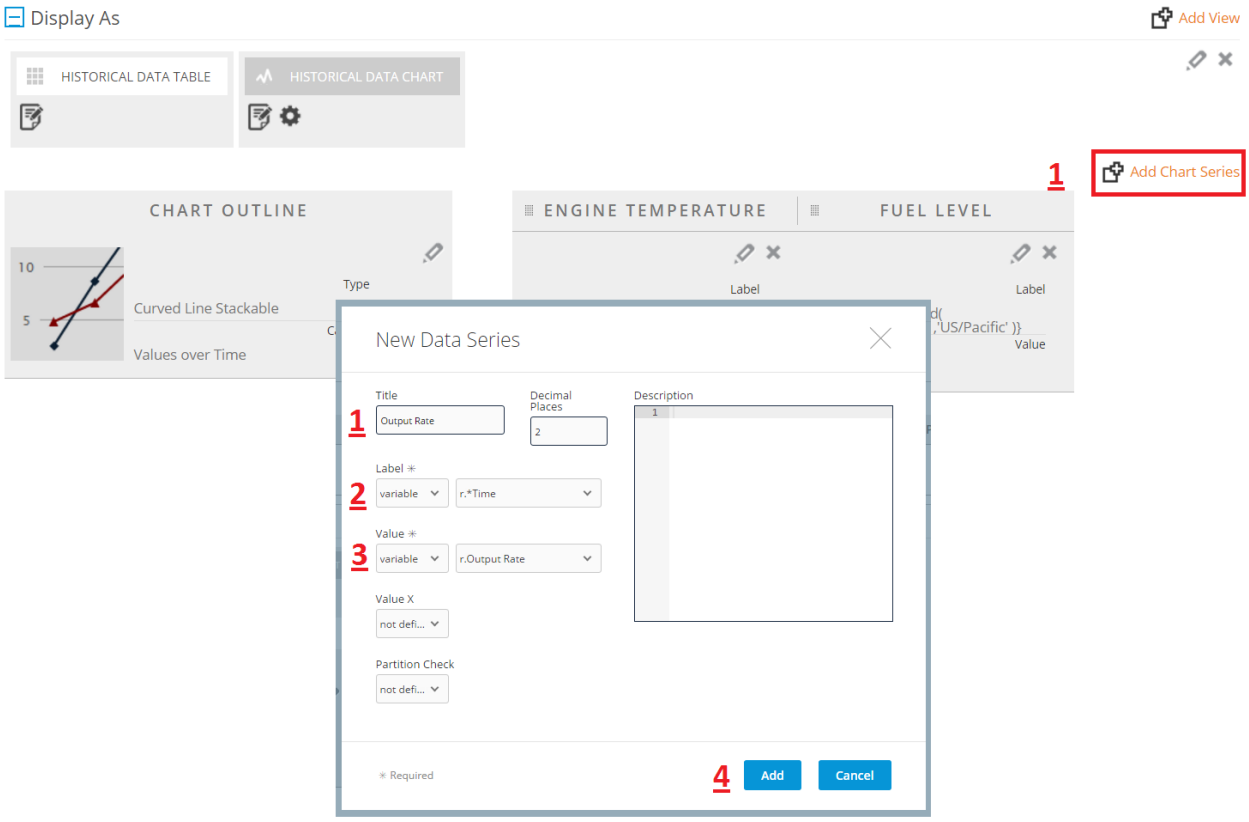


FIGURE 14: ADDING OUTPUT RATE AS A CHART SERIES

1. Press the "Add Chart Series" button on the right of the "Display As" section while you have the "Historical Data Chart" view selected.
2. For the *Title* attribute, which identifies this data series, provide a value like **Output Rate**.
3. For the *Label* attribute, which determines the x-values used by each data point in this series, switch "expression" to **variable**, and then select **r.*Time** from the drop-down that appears.
4. For the *Value* attribute, which determines the y-values used by each data point in this series, switch "expression" to **variable**, and then select **r.Output Rate**.
5. Press the "Add" button on this panel to add the data series to the chart, and then press the "Update" button on the "Queries & Reports" button to save your changes overall.

You can go to the ***View -> Dashboard*** page to see the fruits of your labor; the "Historical Data" dashboard widget should now show output rates over time!

Exercise 2 – Displaying Alerts when Problems Are Detected

Our tracking of the historical readings of wood chippers has revealed an interesting insight. Every time one of our wood chippers fails and needs to be serviced, it is preceded by a warning sign: the engine temperature spikes above 350 degrees Fahrenheit, which will never happen in normal circumstances. So, we're going to configure our application to automatically generate and display alerts when we receive a message indicating that the engine temperature has crossed this threshold, so that we can respond to problems more quickly, hopefully fix our wood

chippers before they fail completely, and minimize the amount of downtime that our customers experience.

Modeling Alert Objects

The first step to displaying alerts inside our application is to create a template for alert objects, so they can be stored and tracked in the same way that the application is currently monitoring customers, wood chippers, and historical readings. Go to the ***Model -> Resource Types*** page. Instead of modifying the "Wood Chipper" resource type, this time we're going to add a new type entirely. Click the "Add" button on the left side of the page under the "New Resource Type" header. Enter **Temperature Alert** into the *Resource Type Name* input. Now, we're going to create the following fields, using the same process that we used numerous times in Exercise 1.

1. Add a field with the name **ID** to serve as a unique identifier for each temperature alert, so the application has a guaranteed way to tell them apart.
    a. Under the *Attributes* header, check the boxes for **Resource Key** and **Resource Label** to tell the system that this will be the unique identifier.
    b. Leave the *Data Type* as **Text**, but from the *Modifier* drop-down, select **Auto Number** so that the application will automatically take care of generating unique ID's for each alert for us.
    c. For the *Pattern* input, enter **{000}**; don't forget to include the brackets.
    d. For the *Next Value* input, enter **0**.
    e. For the *Increment* input, enter **1**. This means that the first alert will have the ID "0", the next will have "1", then "2", and so on. The pattern of three zeroes means there can be up to 999 unique identifiers, which is more than enough for our purposes.
2. Add a field with the name **Time** to store the time at the engine temperature spiked above 350 F.
    a. Give this field the *Data Type* of **Calendar** to specify that it will store a date-time value.
3. Add a field with the name **Temperature** to store the actual engine temperature that resulted in the alert.
    a. Set the *Data Type* of the field to **Number**, and give it the *Modifier* of **Units**.
    b. For the *Decimal Places* attribute, enter **2**.
    c. For the *Measurement* drop-down, select **Temperature**, and for the **Units** drop-down, select **Fahrenheit**.
4. Add a field with the name **Serial Number**, which will give us easy access to the serial number of the wood chipper for which the alert occurred. This should remain a basic **Text** field.

When you're done adding all these fields, press the "Add" button in the top right of the page to add this new object type to the application. The template should look like this:

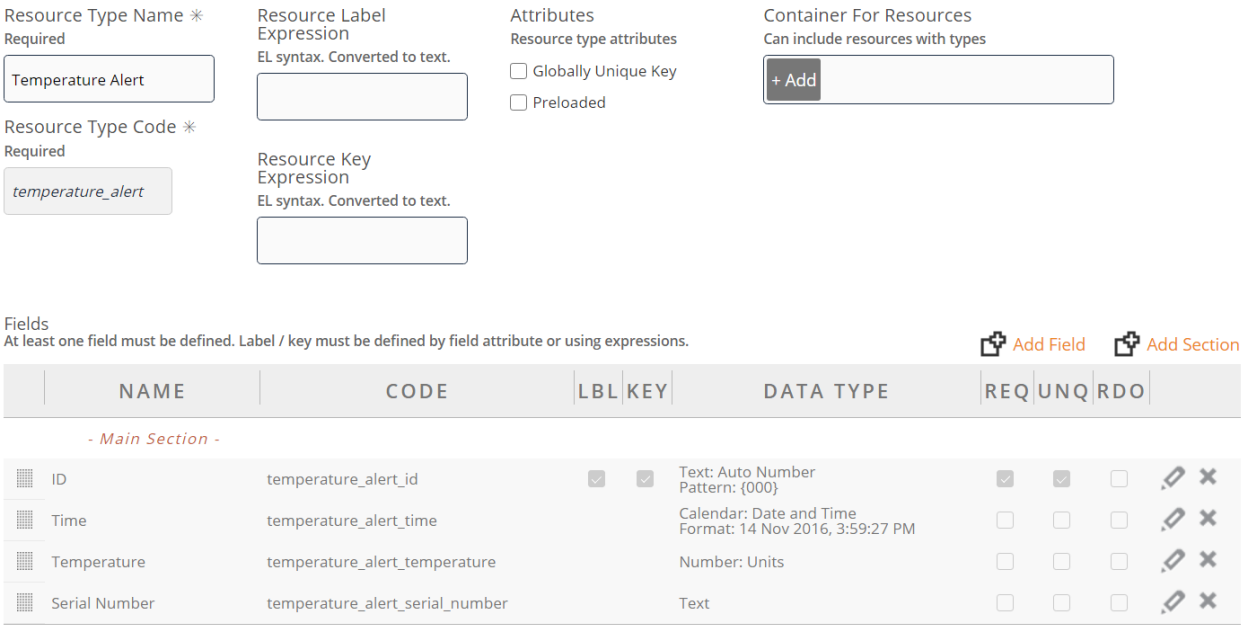


FIGURE 15: THE COMPLETE TEMPERATURE ALERT TEMPLATE

Finally, we can define a hierarchical relationship such that temperature alerts actually belong to wood chippers. Select **Wood Chipper** from the list of resource types on the left side of the page. On the top of this page, there is now a *Container for Resources* input. Press the **+Add** button and check the box for **Temperature Alert** to inform the system that wood chippers can contain temperature alerts. Press the "Update" button to save these changes.

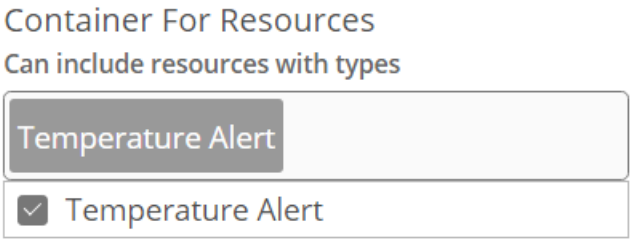


FIGURE 16: SETTING WOOD CHIPPERS AS CONTAINERS FOR ALERTS

Generating Temperature Alerts when Temperature is Too High

We've created a template for an alert that the application can create; now we want the application to generate one of these temperature alerts whenever it receives a message with an engine temperature value greater than 350. Naturally, changes to the logic with which we handle incoming messages means we need to go back to the ***Configure -> Routines*** page, and once more select the **Device Message Received** routine. We're now going to add a new action entirely, so press the "Add Action" button on the right of the page to bring up the following panel to add a new action. We're going to add an instance of the first action type: **If, Then, Else**, which will evaluate a certain condition and split the execution of the routine into two branches; if the condition is true, the routine will proceed along one path, and if false, the routine will proceed along the other path. This is what we need because we only want the system to generate an alert if the temperature is problematic, and do nothing otherwise.

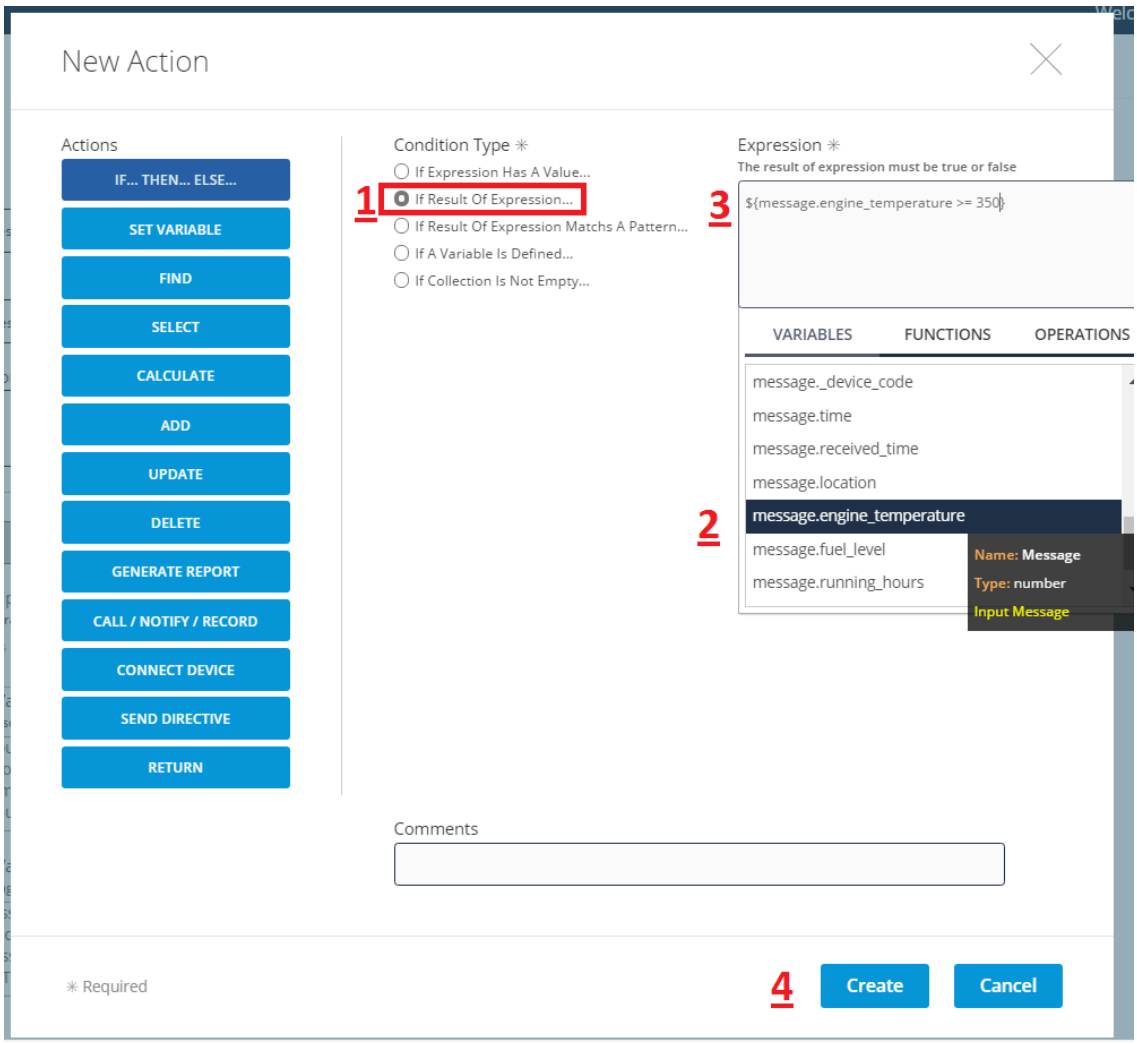


FIGURE 17: BRANCHING THE ROUTINE BASED ON ENGINE TEMPERATURE

1. Select the second option for the *Condition Type* list of radio buttons: **If Result of Expression**. This allows us to provide an expression that will be evaluated as true or false.
2. Click the orange “F” button to open the function builder. On the *Variables* tab, click the **message.engine_temperature** option.
3. Type **>= 350** to add this to the end of the expression; if you clicked away, then make sure to enter this value after the **message.engine_temperature** and before the end bracket. The final expression should read exactly:
   ${message.engine_temperature >= 350}
4. Press the “Create” button to add the action to the routine, then press the “Update” button on the “Routines” page to save your progress.

As you can see, a new action has appeared at the top of our routine, that splits the execution of the routine into two branches before bringing them back together. The first, vertical branch is now where we can add an action that will only be executed if the engine temperature of the message. So, press the “Add Action” button once more to add such an action, and from the list

of action types on the left of the resulting panel, select **Add**, so the panel appears as the following screenshot.

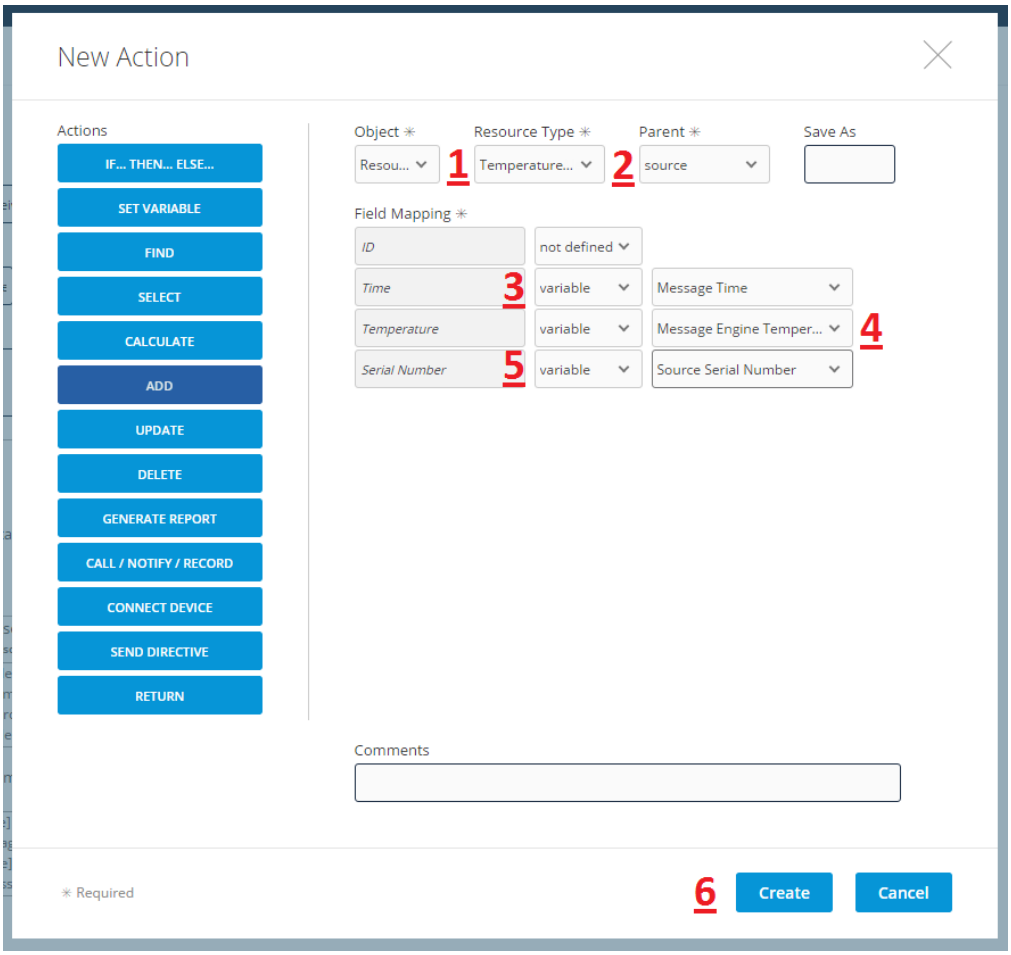


FIGURE 18: ADDING A TEMPERATURE ALERT TO THE ROUTINE

This is the same type of action we've seen before, in adding a historical record to store a snapshot of the wood chipper's state in time.

1. From the *Resource Type* drop-down on the top of the panel, select **Temperature Alert** so that will be the type of object we're adding.
2. From the *Parent* drop-down, select **source**, which is the wood chipper for which we have received a message; we want this temperature alert to belong to that wood chipper.
3. For the *Time* field in the *Field Mappings* section, change the value of the mapping type drop-down from "not defined" to **variable**. From the drop-down that appears, select **Message Time**.
4. For the *Temperature* field, switch "not defined" to "variable", and select **Message Engine Temperature** from the list of variables.
5. For the *Serial Number* field, switch "not defined" to "variable", and select **Source Serial Number** from the list of variables.
6. Press "Create" to add this action. **Do not** yet update the routine.

Before we update the routine, we now need to move this action into the correct position. As it stands, the routine will always add a temperature alert whenever it receives any message, before it even checks the engine temperature. To move this action into position, click and hold the "Add" action we've just created from the top of the routine, and drag it into the vertical branch of the "If, Then, Else" statement we created previously, then drop the action. Now, the routine should look exactly like this:

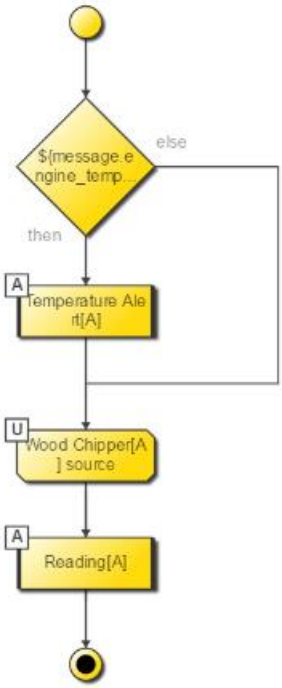


FIGURE 19: THE COMPLETE ROUTINE TO PROCESS DEVICE MESSAGES

When your routine looks exactly like this, then you can press the "Update" button in the top right. The application will now begin creating alerts any time a problematic message is received!

Displaying Alerts

As we know, data existing in the application where only an administrative user can access it is relatively useless; the real value comes from being able to display data to our users. Let's build a report so that when a user logs into this application, he or she can immediately see if there are any temperature alerts of which they need to be aware. Go to the ***Configure -> Queries & Reports*** page.

1. Press the "Add" button under the "New Query" header on the left side of the page to begin the process of creating a new report; give this new report the name **Temperature Alerts**. Next, let's define the set of data that will populate this report.

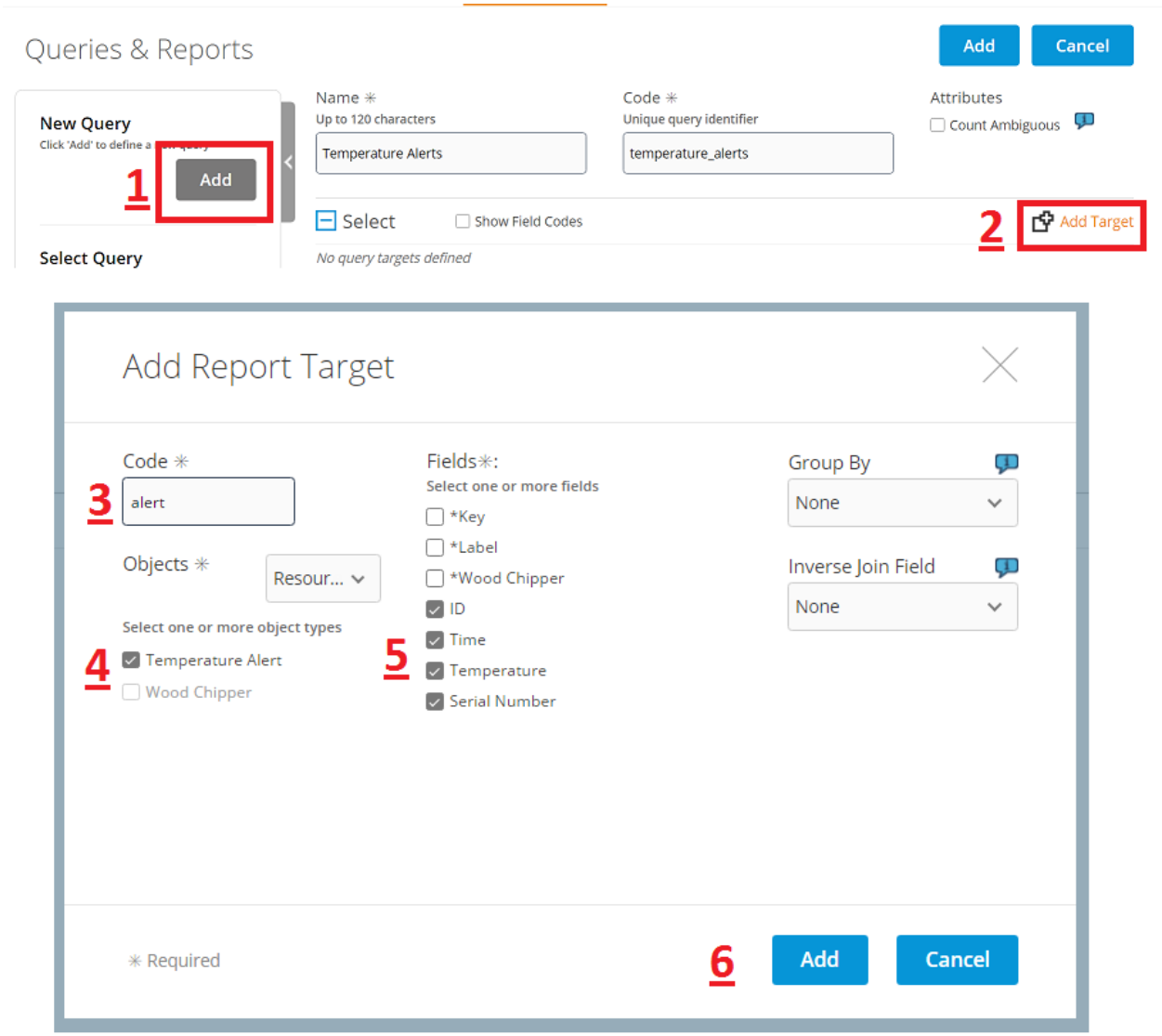


FIGURE 20: ADDING A NEW REPORT ON TEMPERATURE ALERTS

2. Press the “Add Target” button on the right side of the page in the “Select” section, to bring up the panel to add a new target from which the system should retrieve data.
3. For the *Code* input that will identify this target throughout the report, enter **alert**.
4. From the list of checkboxes on the left of the panel displaying each of the resource types defined in the application, check the box for **Temperature Alert**.
5. From the list of *Fields* in the second column of the panel, check the boxes for **ID**, **Time**, **Temperature**, and **Serial Number**.
6. Press the “Add” button to add this target to the report.

We’ve successfully told the system that for this report, it should generate a data set containing these four fields from each temperature alert stored. Now, let’s define a simple table view with which we can display our alerts. Press the “Add View” button in the right side of the “Display As” section of the page to bring up the following panel.

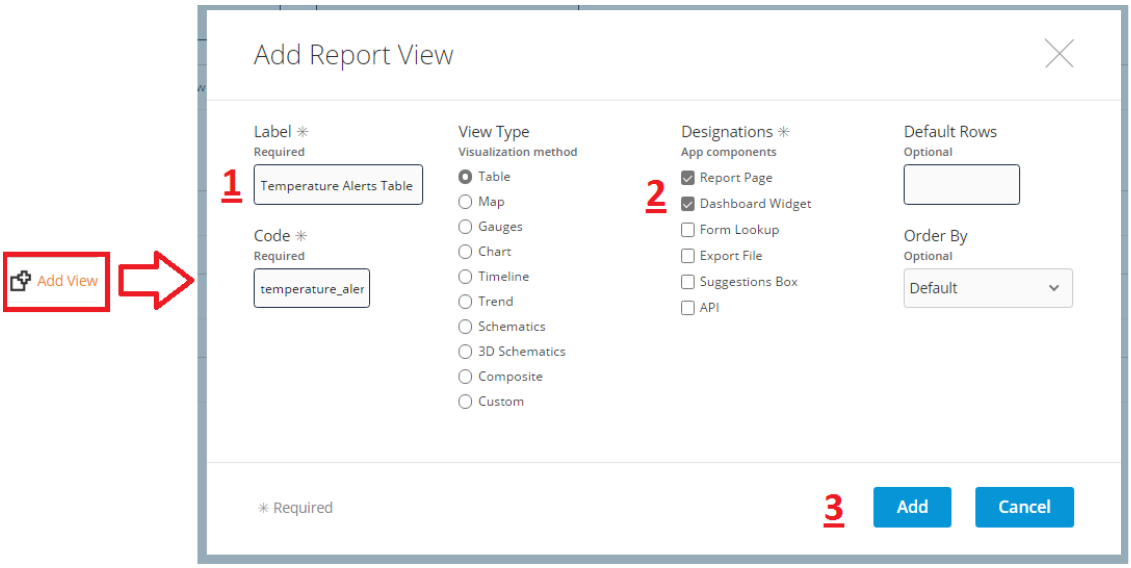


FIGURE 21: ADDING A NEW REPORT VIEW

1. Give this view a name, such as **Temperature Alerts Table**.
2. The "Table" view type is selected by default, so all we must do is choose where in the application this view is available. From the *Designations* list of checkboxes, check the boxes for **Report Page** and **Dashboard Widget** so that this view can be made available as a link on the main menu, or as a widget on the dashboard.
3. Press the "Add" button to add this view.

Now we simply need to add all of the fields from the temperature alert target as columns of this table. Since we want to add one column for each field from the target, there is an even faster shortcut than before. Rather than drag and drop each of the field boxes individually, instead drag and drop the box with the label "alert" into the "Display As" section; by dragging and dropping the box representing the entire target, the system interprets that as a drag and drop of each field contained within the target.

FIGURE 22: ADDING ALL TEMPERATURE ALERT FIELDS AT ONCE WITH DRAG AND DROP

Once this is done, you can now press the "Add" button in the top right of the page to add your new report. Go to the ***View -> Dashboard*** page. By giving this latest report the "Dashboard Widget" designation, we've added it to the pool of possible reports that can be added to the dashboard. However, each user can configure his or her own dashboard, selecting which

widgets they would like to see, and arranging them as they wish. Let's do this now. Press the **More Widgets** button in the top right of the dashboard.

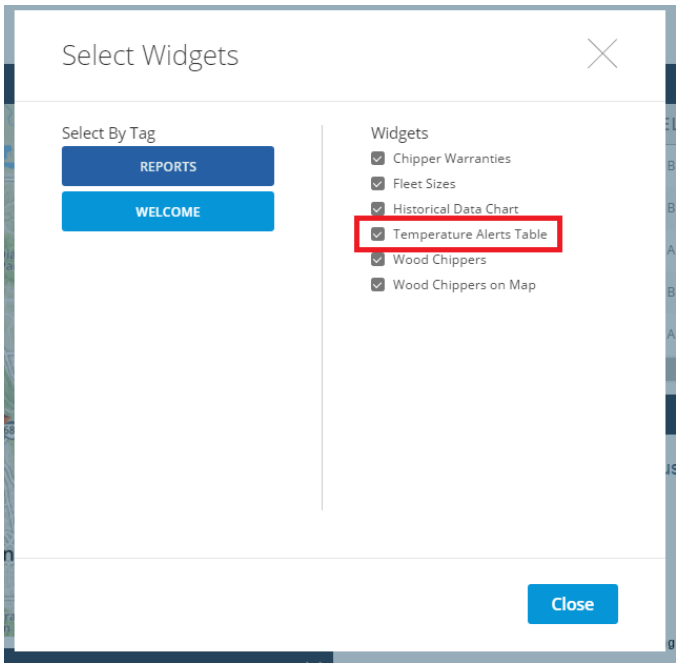


FIGURE 23: ADDING THE NEW WIDGET TO THE DASHBOARD

You can see that the box is checked for each of the widgets present on the dashboard; check the box for **Temperature Alerts Table** and it will appear at the bottom of the dashboard; you can then close this panel using either the "Close" or "X" buttons. If you want to move this widget into a new position, you can click and drag any widget by its header and drop it into a new position. You can also press any of the "Columns: 1 2 3" buttons at the top of the dashboard to change the number of columns into which the dashboard is divided.

Adding This Report to the Main Menu

We also gave this latest table the "Report Page" designation, meaning that we can make it available to users as a link on the main menu. This is helpful because a table on the dashboard will only show the latest 7 rows possible, but a table on a report page allows users to tab through the entire set of records. Go to the ***Configure -> Menu*** page. Here, you can see the main menu as it appears to a user at runtime. This menu can either contain links directly, such as the existing "Dashboard" link, or it can contain nested groups, such as "Admin", "Forms", and "Reports", which themselves can contain links. Press the "Add Item" button above the table of menu items to bring up the following panel.

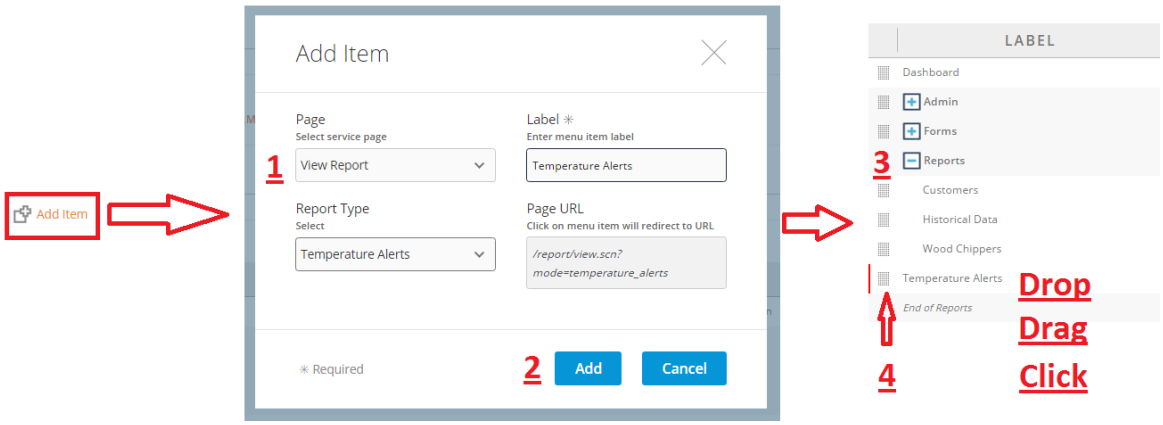


FIGURE 24: ADDING THE REPORT TO THE MAIN MENU

1. From the *Page* drop-down, select **View Report**. Then, from the *Report Type* drop-down that appears, select **Temperature Alerts**, to specify that this link will bring up our new report specifically.
2. Press the "Add" button to add this menu item.
3. The menu item now appears at the end of the list. However, it makes sense to include it within the "Reports" tab on the menu, since that is where the other application reports are listed. On the table of items, click the blue "+" button to expand the reports group.
4. Click and hold the box in the leftmost column for the new "Temperature Alerts" row at the end of the list. Drag it above the "*End of Reports*" row and drop it, so that it now is contained within the "Reports" group. You can verify this by hovering over the "Reports" group in the "Preview" section of the page.
5. Press the "Update" button in the top right of the page to save your changes.

Now, let's see the modified main menu in action by re-entering the runtime experience from design mode. Uncheck the **Design Mode** checkbox in the page header to return to the runtime that our users will experience. You should be able to verify that the new "Temperature Alerts" report is accessible both on the dashboard and from the "Reports" tab of the main menu. With this, we have successfully modified the application to display temperature alerts to users!

Exercise 3 – Sending Automatic Notifications Out from the Application

Our users can now log into our application and see if any alerts exist for our wood chippers whenever they want, or can leave the application up and set to auto-refresh; this means that we can service our machines more quickly than if we rely on them to actually fail, and for our customers to actually notice the problem however long after that. However, this still relies on a user actively logging in and/or checking the application frequently, and we can do better. We will now build an automatic email notification that the application will send to a user when it detects a temperature alert.

Building Notifications

The first component necessary for sending notifications is an "Internal Event"; on one side, an internal event can be triggered by a routine, and can have a set of fields or buckets for

information that the routine can populate; on the other side, an internal event contains the templates for notifications to be sent. In this way, an internal event acts as a conduit or bridge, taking data from a routine, using it to build notifications from these templates, and then sending those notifications. Go to the ***Model -> Internal Events*** page. Here you can see that an internal event has already been defined in this application, for the sake of time; this event, "Send Email", can currently store two pieces of information: a message and a recipient. Let's define what the system actually does with this information when the internal event is triggered. Go to the ***Configure -> Notifications*** page. This page shows all of the internal events we've explicitly defined, as well as the system default types of events, that get triggered whenever we make configuration changes to the application, or a user logs in, and the like. Select **Send Email** from the list on the left under the "Select Event" header. Press the "Add" button in the top right of the screen to create a new type of notification that will be sent when this internal event is triggered.

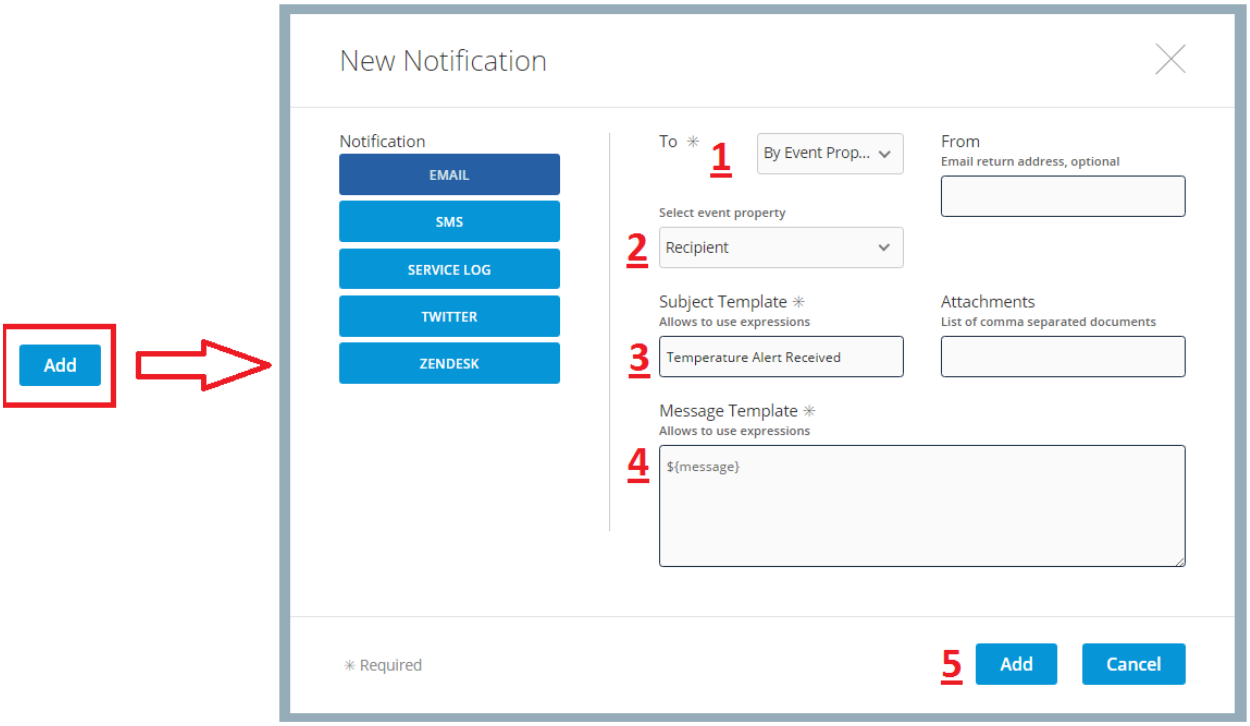


FIGURE 25: ADDING AN EMAIL NOTIFICATION TO THE 'SEND EMAIL' EVENT TYPE

1. We'll be sending out an email notification, so this type of notification should already be selected from the list of types on the left of this panel. The first step, then, is to specify to whom we'll want to be sending an email notification. Recall that the "Send Email" event type had a field to store a recipient exactly for this purpose. In the *To* drop-down at the top of the panel, change "Email Address" to **By Event Property.**
2. The free text input for recipient email addresses will now change to a list where you can select from the different fields of the event type; select **Recipient** from this drop-down.
3. For the *Subject Template* input, enter **Temperature Alert Received** as the subject line for the email.
4. For the *Message Template* input, we want to set this to the "Message" value of the event type so that we can customize the message we send from within a routine when we trigger this event type. Type the exact value **${message}** .
5. Press the "Add" button to add this notification to the internal event type.

Now, a routine can call this internal event type, provide a recipient and a message, and the application will send an email notification to that recipient containing the specific message. Let's actually trigger this event type and populate these fields with data. Go to the ***Configure -> Routines*** page and select the **Device Message Received** routine from the list on the left. In Exercise 2, we defined a branch of this logical flow that would only be executed when the engine temperature inside the message was greater than 350; this is the branch where we want to send out an email notification. Previously, we added an action, then dragged it into the proper place. Let's try a different way. Click on the line within this branch, immediately before the "Add Temperature Alert" block. You should see a "+" button appear next to this point in the routine; click this button to add a new action at this point in the routine specifically.

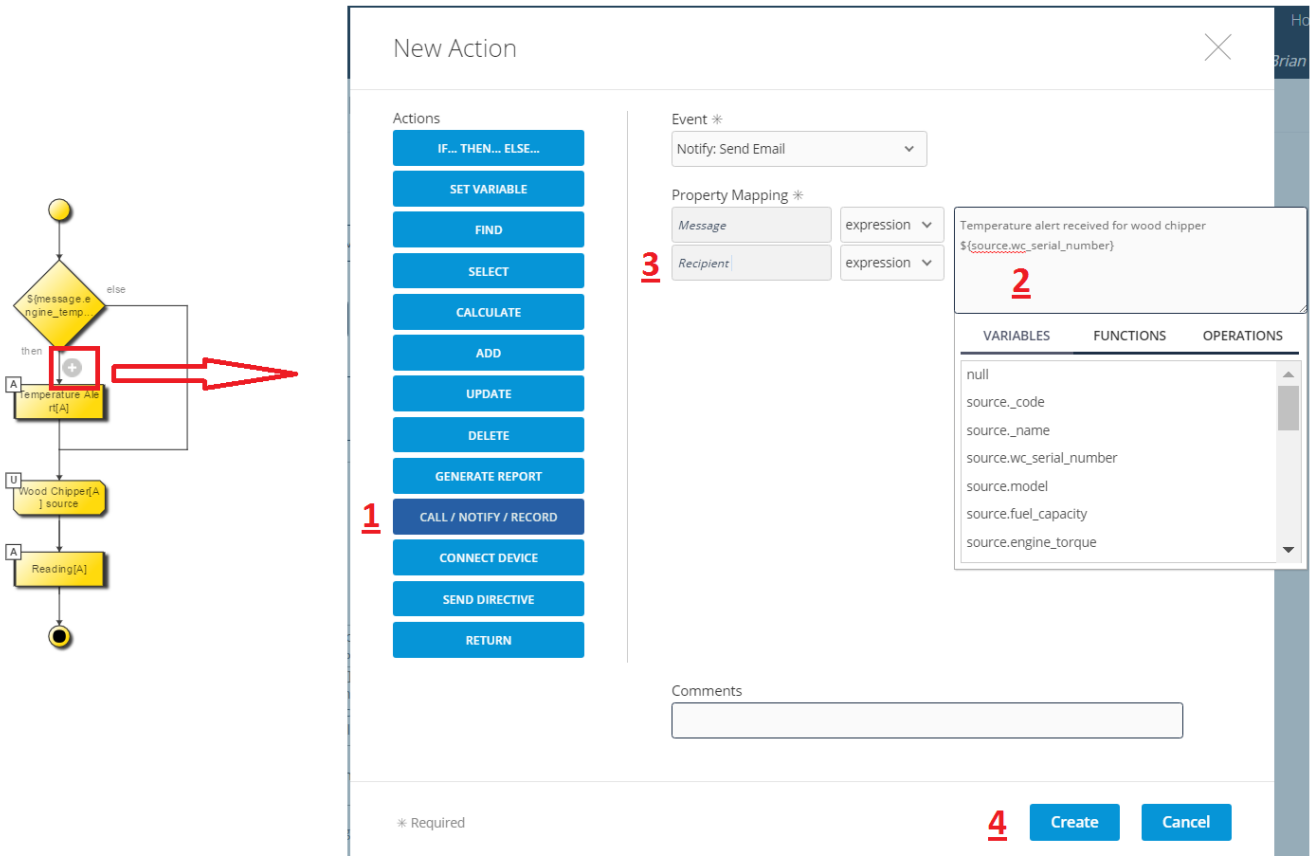


FIGURE 26: ADDING A NOTIFICATION CALL TO THE ROUTINE

1. From the list of action types on the left of the "New Action" panel, select **Call / Notify / Record**.
2. Here we see a list of all of the internal events that we can trigger, and what fields of theirs we can populate with data. Set the mapping type of the *Message* property from "not defined" to **expression**. We can build a custom expression here that will serve as the text of our email message. Type the following text into the box: **Temperature alert received for wood chipper:** , don't forget the space at the end. Then, from the *Variables* list within the function builder, click the **source.wc_serial_number** variable, so that the message will be populated with the serial number of the wood chipper for which there is an alert. The final expression should look exactly like this: **Temperature alert received for wood chipper ${source.wc_serial_number}**

3. Set the mapping type of the *Recipient* field from “not defined” to **expression**. In the box that appears, type your own email address to send yourself this test notification.
4. Press “Create” to add this routine action. **Do not** yet update the routine.

Before you actually update this routine, let me say this: I will make sure to disable all of these notifications once this class is over. In the meantime, however, the system will likely generate one of these alerts and send you an email once every couple of minutes. So, if you do not want to receive any emails whatsoever, feel free to simply not update this routine, since you’ve already seen the flow of how you *would* go about defining a notification to yourself. If you want to add this functionality for a couple of minutes to verify it, you can then come back and delete this action from the routine once you’ve received a single email, or delete the notification from the “Notifications” page. If this is not a concern for you, then press “Update” on the “Routines” page.

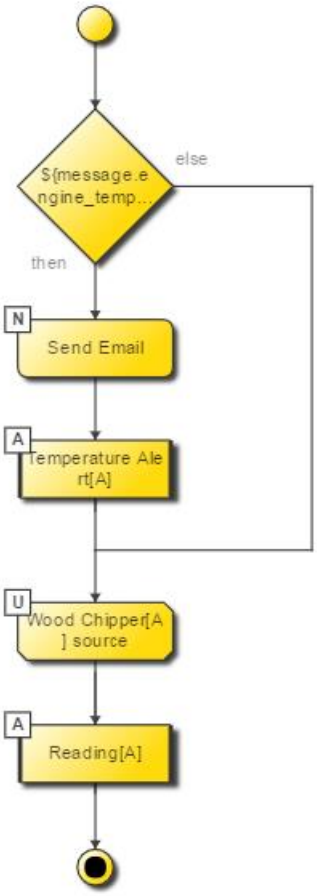


FIGURE 27: THE COMPLETE MESSAGE PROCESSING ROUTINE, WITH EMAIL NOTIFICATION

Now, now only will the application log alerts for display and future analysis, but it will also send out email notifications automatically in response to alert conditions, meaning we can minimize the time it takes for us to detect and respond to events.

Exercise 4 – Acknowledging Alerts

We're now rendering alerts to users and logging them in the application, and sending out automated notifications when they occur. However, if multiple people receive a notification, or log in and see the alert, they might simultaneously start to address the issue, unaware that anyone else has noticed the alert. In other words, simply viewing the log of alerts is helpful, but it would be more helpful to know which alerts are outstanding. So, let's add the ability for a user to acknowledge alerts, so that future users will know that a temperature alerts has been or is already being addressed.

Modifying the Data Model

Whether an alert has been acknowledged is a piece of data, and as we've seen throughout these exercises, in order to track a piece of data, we must first create a bucket to store that information. Go to the ***Model -> Resource Types*** page, and select the **Temperature Alert** resource type from the list on the left of the page. Press the "Add Field" button on the right of the screen to add a new field to this template.

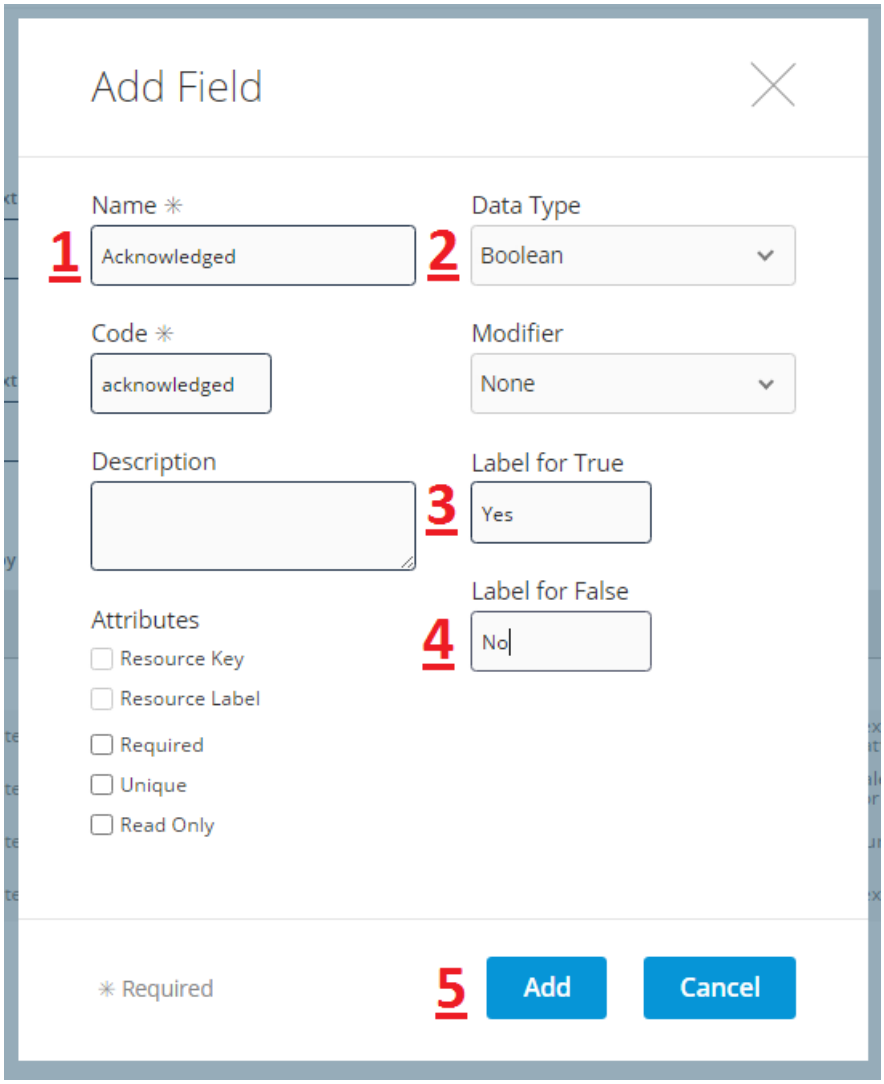


FIGURE 28: ADDING THE ACKNOWLEDGED

1. For the *name* of the field, enter a value like **Acknowledged**.
2. For the *data type* of the field, select **Boolean**, or a value that can either be true or false.
3. For the *Label for True* attribute, enter a positive value like **Yes**, **True**, or **Acknowledged**.
4. For the *Label for False* attribute, enter the opposite: **No**, **False**, or **Not Acknowledged**.
5. Press the "Add" button to add this field, then press the "Update" button on the "Resource Types" page to save these changes overall.

Our temperature alerts can now store a true-false value indicating whether they have been acknowledged. The existing alerts may have empty values for this field, but an empty value is as good as a false value.

Adding a Form to Acknowledge an Alert

Now that we have a place to store whether an alert has been acknowledged, we need to provide users with a way to actually acknowledge an alert. As indicated in Exercise 1, the primary means of allowing users to actually interact with the system and submit data is through forms. Let's configure a form that will allow a user to do this. Go to the ***Configure -> Forms*** page. Here, you can see the definition of the form we used in Exercise 1 to create a new wood chipper, and you can see that it appears nearly identical to a template for a resource type, or to a device profile. Forms have an extra display element, but behind the scenes they too are simply vessels to store and transport data, like these other components. So, building one is nearly identical to the process with which we're now familiar. Press the "Add" button under the "New Form" header on the left of the page to begin the creation process. For the *Form Name* input, enter **Acknowledge Alert** or a similar value. Now, you can repeatedly add fields to this form using the "Add Field" button on the right of the page.

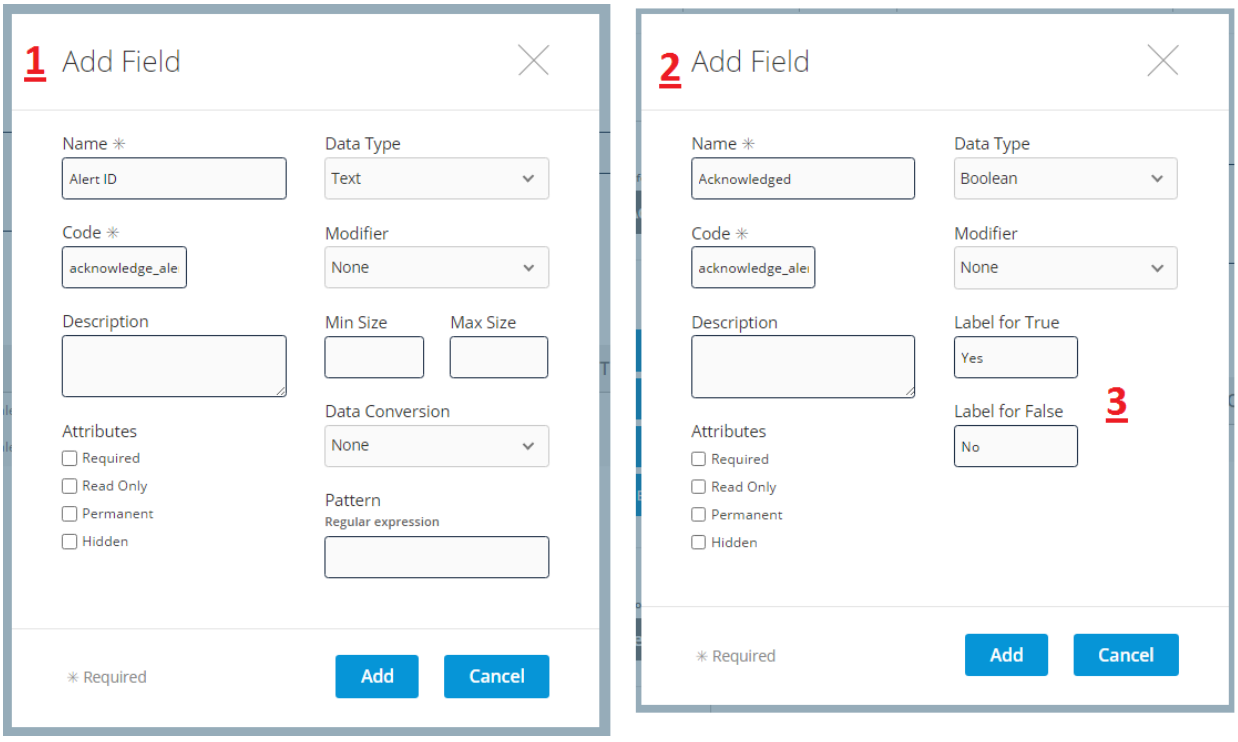


FIGURE 29: THE FIELDS TO ADD TO THE ACKNOWLEDGE ALERT FORM

1. Add a field with the *Name* **Alert ID**. Give this field the **Text** data type.

2. Add a field with the *Name* **Acknowledged**. Give this field the **Boolean** data type. Provide positive and negative labels for the *Label for True* and *Label for False*; ideally these would be the same as what you defined for the resource type field, but that's not necessary, so **Yes** and **No** will suffice, respectively.
3. Press the "Add" button in the top right of the "Design Forms" page to add this new form.

Creating a Routine to Process the Form

We've successfully created a form that allows users to enter a text and Boolean value and press "Submit". However, the application currently does nothing with that data. Forms are processed by routines, which execute any logic required by the use case of the form; for example, the routine powering the "Add Chipper" form added a wood chipper resource, defined a new device, and updated the customer's count of wood chippers. Let's create a routine to process our new form. Go to the ***Configure -> Routines*** page. Press the "Add" button under the "New Routine" header on the left of the page to begin the creation process.

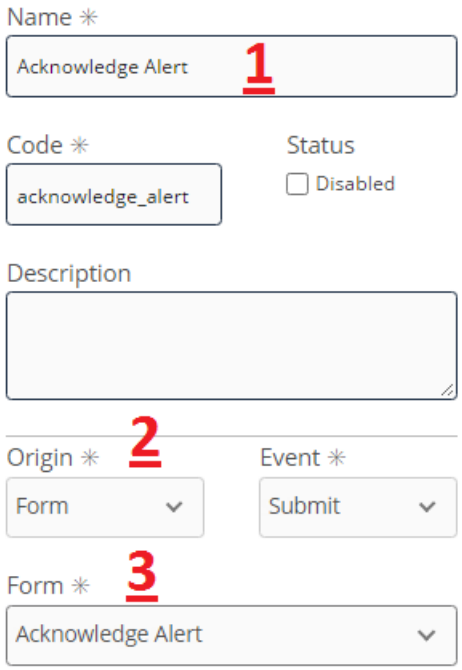


FIGURE 30: DEFINING THE ROUTINE TO PROCESS THE ACKNOWLEDGE ALERT FORM

1. Enter the name **Acknowledge Alert** or something similar, as that will be the purpose of the routine.
2. From the *Origin* drop-down, select **Form**.
3. From the *Form* drop-down that appears, select the new routine we've defined, **Acknowledge Alert**.

From here, we can actually define the logic that is executed in response to this form. The logic that we want to define is as follows: we want to find the temperature alert stored in the application with the ID provided in the form; we want to check to make sure that such an alert is found, so we don't accidentally try to update a record that doesn't exist; and if an alert is found,

we want to update it and set its “Acknowledge” field to the value in the form. So, add the following three routine actions; in this case, it is best to add them in the place we want them, so click on the routine line and use the “+” button that appears.

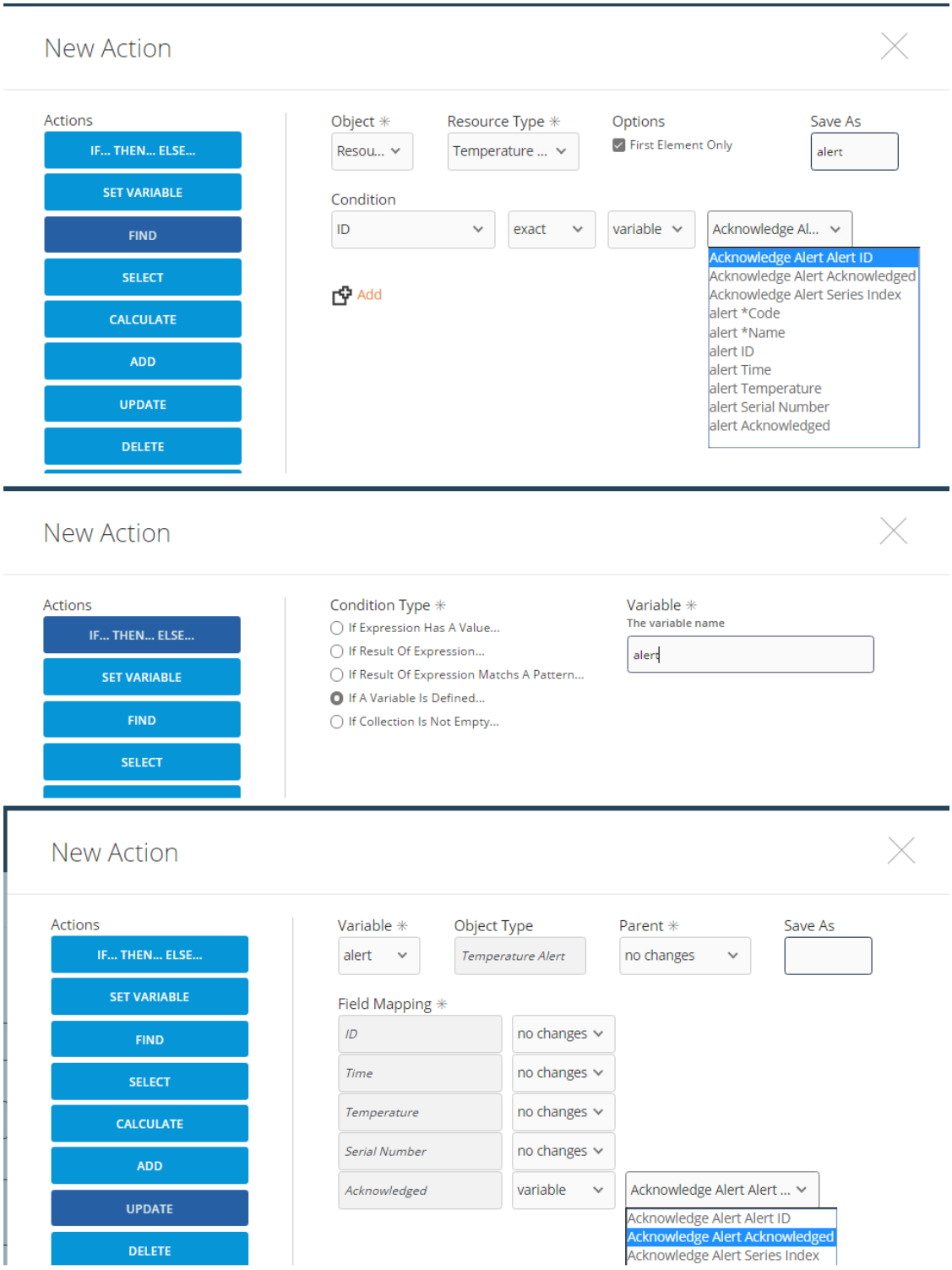


FIGURE 31: THE THREE ACTIONS TO PROCESS THE ACKNOWLEDGE ALERT FORM

1. Add an action of the **Find** type.
    a. From the *Resource Type* drop-down, select **Temperature Alert**. This indicates that the system should find a temperature alert.
    b. In the *Save As* box, enter the value **alert**. This allows us to reference the temperature alert we find using that name later in the routine.
    c. From the *Condition* drop-down, select **ID**. Two drop-down boxes and an input box will appear. Switch the second drop-down from "expression" to **variable**. Then, select **Acknowledge Alert Alert ID** from the list.
2. Add an action of the **If, Then, Else** type.
    a. Select the fourth *Condition Type* available: **If A Variable Is Defined**.
    b. In the *Variable* input box that appears, provide **alert**, the name that we assigned the temperature alert in the previous action. So, this condition will result in true if an alert was found with the given ID, and false otherwise.
3. Add an action within the vertical "Then" block of the "If, Then, Else" action, of the **Update** type.
    a. From the *Variable* drop-down at the top of this panel, select **alert**, so that we're updating the temperature alert we found.
    b. For the *Field Mapping* for the *Acknowledged* field, set the mapping type from "no changes" to **variable**. From the list that appears, select **Acknowledge Alert Acknowledged**. This will set the stored alert to acknowledged if the user checked the box, and not acknowledged if they did not; this gives us the ability to un-acknowledge alerts if we so desire.

Once this is done, your routine should look exactly like the screenshot below. If this is the case, then press the "Add" button in the top right of the page to create this routine. The form you've created is now fully functional!

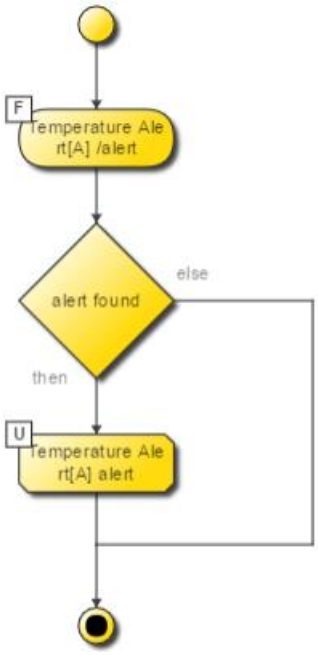


FIGURE 32: THE COMPLETE ROUTINE TO PROCESS THE ACKNOWLEDGE ALERT FORM

Adding the Acknowledged Field to the Report

We want the report we built in Exercise 2 to show whether each alert is acknowledged. Go to the ***Configure -> Queries & Reports*** page so we can add this value as a column in that table. Select the **Temperature Alerts** report from the list on the left under the "Select Query" header. As we saw in Exercise 1, adding a new field to a table report is as simple as telling the system to retrieve it, and dragging it onto the table view.

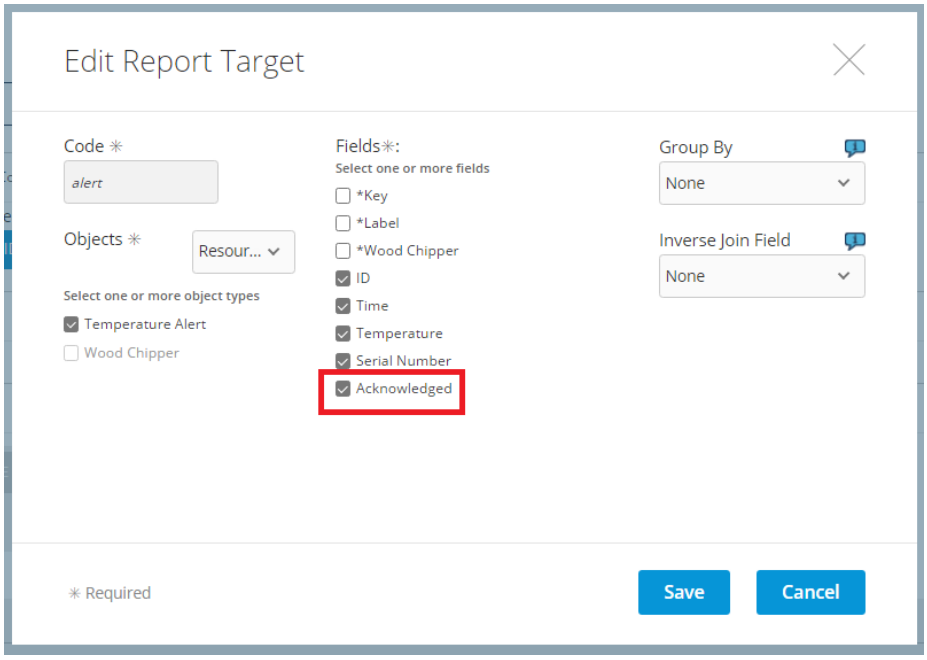


FIGURE 33: ADDING THE ACKNOWLEDGED FIELD TO THE TEMPERATURE ALERTS REPORT

1. Press the edit target button to the right of the "Temperature Alerts" target in the "Select" section of the page. In the *Fields* column of the "Edit Report Target" panel, check the box next to **Acknowledged**. Press "Save" to update the target.
2. Click and drag the box labeled **Acknowledged** from the "Select" section of the page, and drop it into the "Display As" section of the page. The acknowledged field will now be a column in the table.

It could be useful for a user to be able to filter the rows of this table report so they can see only the temperature alerts that have not been acknowledged, or those that have been, or all. This is the purpose of the "With Criteria" section of the page; we can define criteria that allow users to enter desired values for any of the fields of the selected targets, and only those rows with matching values will be shown. We can add criteria just as easily as we can add table columns: click and drag the box labeled **Acknowledged** from the "Select" section of the page, and drop it into the "With Criteria" section, which is also separately bordered by a broken orange line. Press the "Update" button in the top right to save the changes to this report.

Embedding Our Form Within This Report

The form we have built to acknowledge alerts is fully functional, and will acknowledge or un-acknowledge the stored alert with the ID that a user provides. However, this does require a user

to know the ID of the alert they want, and as we know, this is just a unique integer and is not based on the actual meaning of the alert. Luckily, we can embed the form right into this "Temperature Alerts" report, so that a user can select a row from the table, and bring up the form pre-populated with the values from that row. If you've moved away from the ***Configure -> Queries & Reports*** page, return to that page and re-select the **Temperature Alerts** report. Below the list of table columns in the "Display As" section, on the right, is an "Add Command" button. This allows us to add a button above the table that a user can press to display a panel containing a form. Press the "Add Command" button to bring up the following panel.

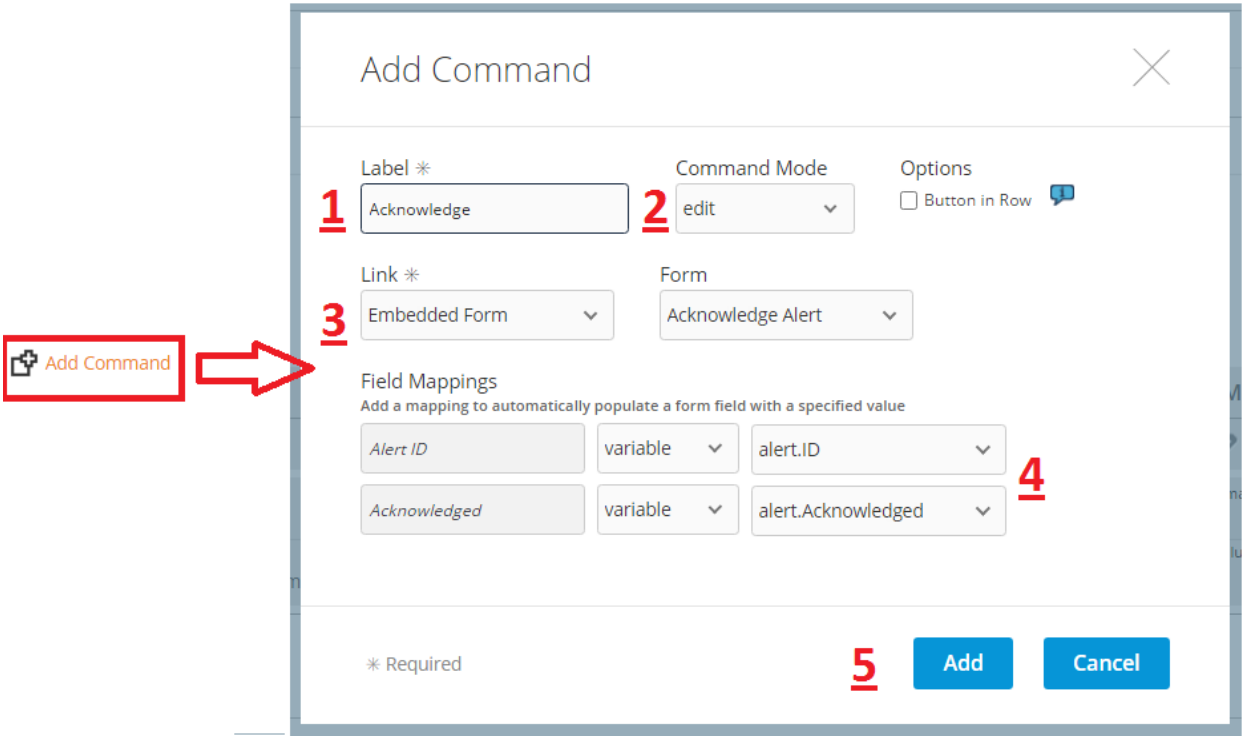


FIGURE 35: ADDING AN ACKNOWLEDGE BUTTON TO THE TEMPERATURE ALERTS REPORT

1. For the *Label* of the command, enter **Acknowledge** or something similar. This dictates the text of the button that will appear atop the report.
2. For the *Command Mode* drop-down, select **edit**. This controls the appearance of the button, and an edit button makes the most sense here because we are modifying an alert here.
3. From the *Link* drop-down, select **Embedded Form** to embed a form within this report. A *Form* input drop-down will appear, from which you can select **Acknowledge Alert**.
4. A *Field Mappings* section has appeared that will allow you to automatically populate the fields of the form with data from a row you select in the report.
    a. For the *Alert ID* field, change the mapping type to **variable**, and select **alert.ID** from the list that appears.
    b. For the *Acknowledged* field, change the mapping type **variable**, and select **alert.Acknowledged** from the list.
5. Press the "Add" button to add this command to the report view, then press the "Update" button atop the "Queries & Reports" page to save the changes to this report.

To verify whether this was successful, press the "View Report" button below the list of reports on the left side of the page. You should see that a column has appeared on the left of this table

containing checkboxes; if you click one of these checkboxes, a button should appear atop the table titled “Acknowledge”. Press this button, and the form you’ve defined should appear, pre-populated with the ID of the temperature alert you selected. Check the box for “Acknowledged”, submit the form, and you should see that the value of the “Acknowledged” column has changed for that alert.

Conclusion

Let’s take a step back and review you accomplished over these four exercises. You began with a nascent prototype application that was simply doing remote monitoring of a couple of data fields. You added the ability to track the output rate of wood chippers end-to-end, from receiving that value from devices, all the way to tracking it historically and displaying it to users. You added the ability to automatically detect a warning condition among the messages received by the application and log alerts in the application. You gave users the ability to acknowledge these alerts, the first step towards full tracking of the process of alert resolution. Congratulations! This is pretty significant development of an application in a vacuum, let alone over the course of this brief lab. We’ve only scratched the surface of the power and functionality of Fusion Connect in these exercises here; imagine what you could do when wielding the full power of the service!