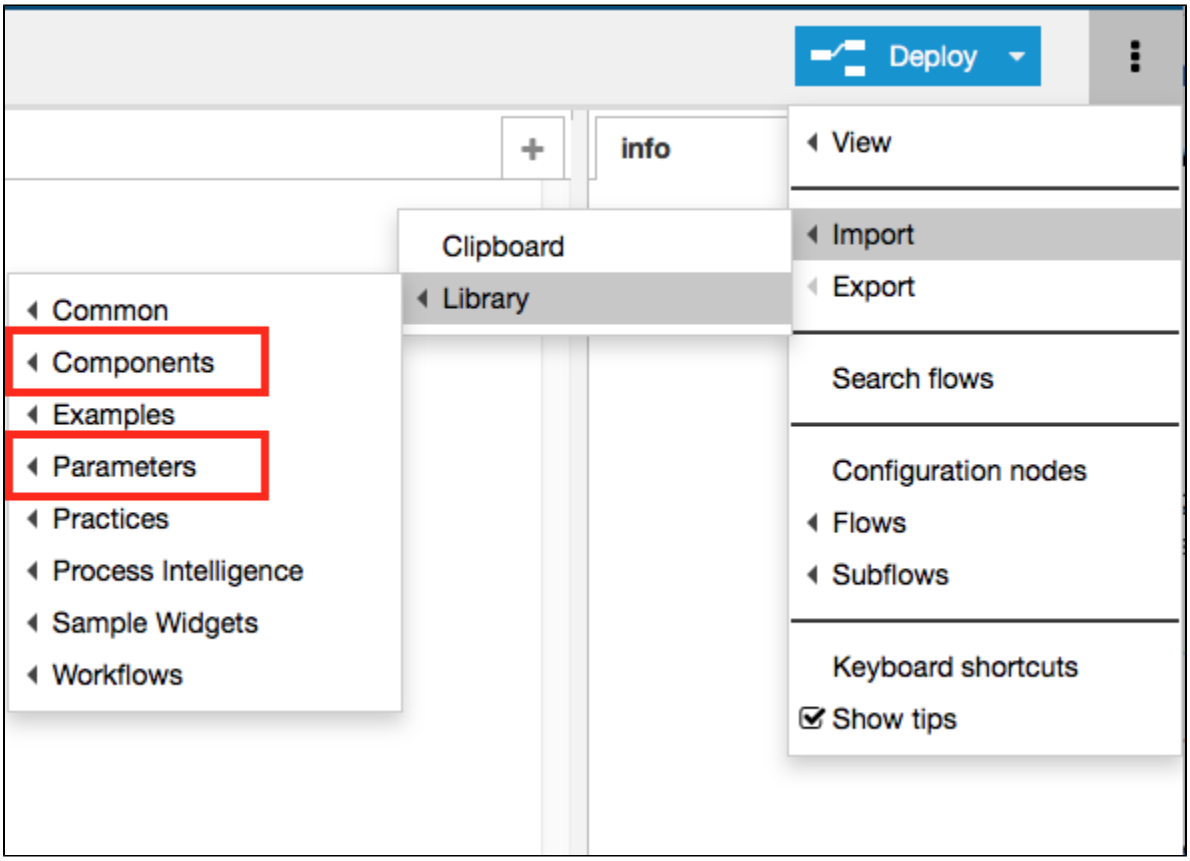
Creating Custom DTP Widgets Using Extension Designer

In this tutorial:

Introduction

You can create a flow using an endpoint node and the component and parameter nodes located in the Extension Designer library. Widget definitions are embedded into the component and parameter nodes, and the endpoint node can choose which widgets to create. To import component and parameter nodes, choose **Import> Library> Components** or **Parameters> <*node*>** from the Extension Designer menu.



The endpoint node provides data for the widget so that Extension Designer can provide a full definition of the widget and the data. This enables a smooth user experience when viewing the data in a dashboard because you don't have to remember specific endpoints or settings.

See Working with Nodes for additional information about these and other nodes.

Configuring Component Nodes

Components store a custom widget and report's structure and definition. It consists of three definition parts:

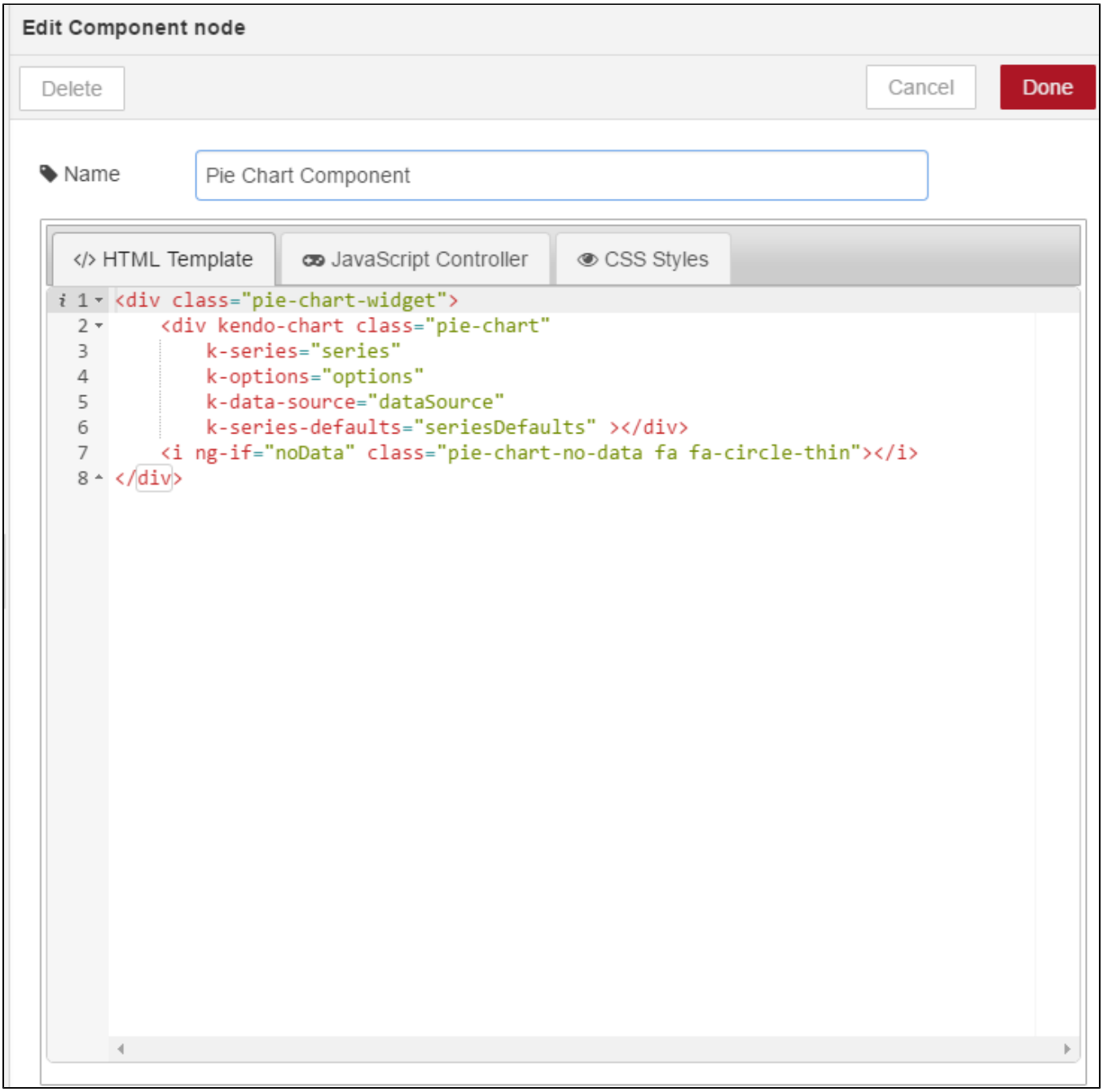
- HTML Template
- Javascript Controller
- CSS Style

This node is based on the Kendo UI Chart

The Kendo UI Chart is an AngularJS 1.4 implementation. For more information, please refer to http://demos.telerik.com/kendo-ui/ and review each chart's AngularJS section, such as http://demos.telerik.com/kendo-ui/pie-charts/angular for the Pie chart. DTP includes Kendo UI version 2015.3.1201. Make sure to use charts from this version.

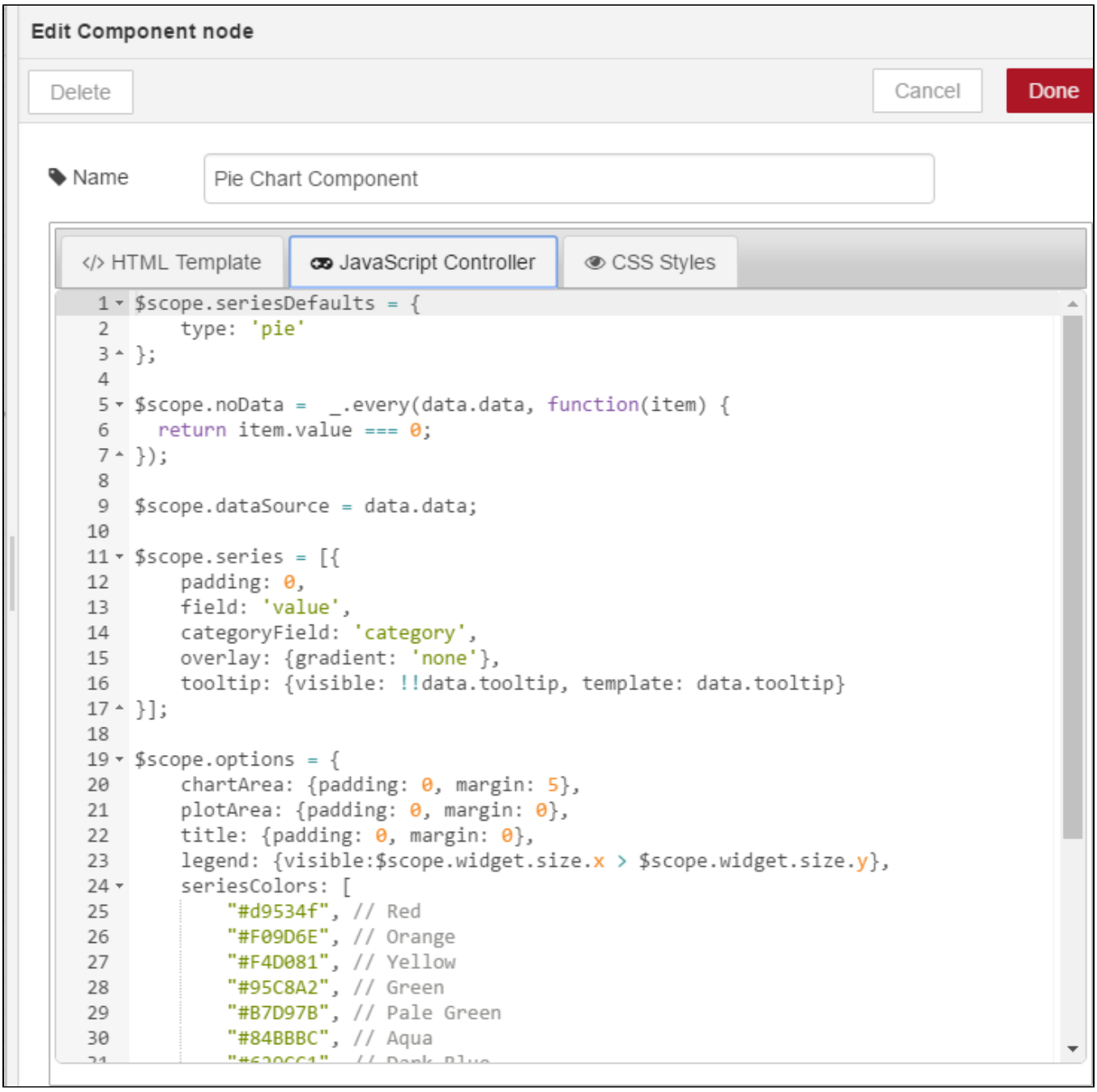
HTML Template

The HTML Template defines how the component should look inside of the dashboard or custom report. It defines what properties that Javascript Controller should have under the `$scope` object.



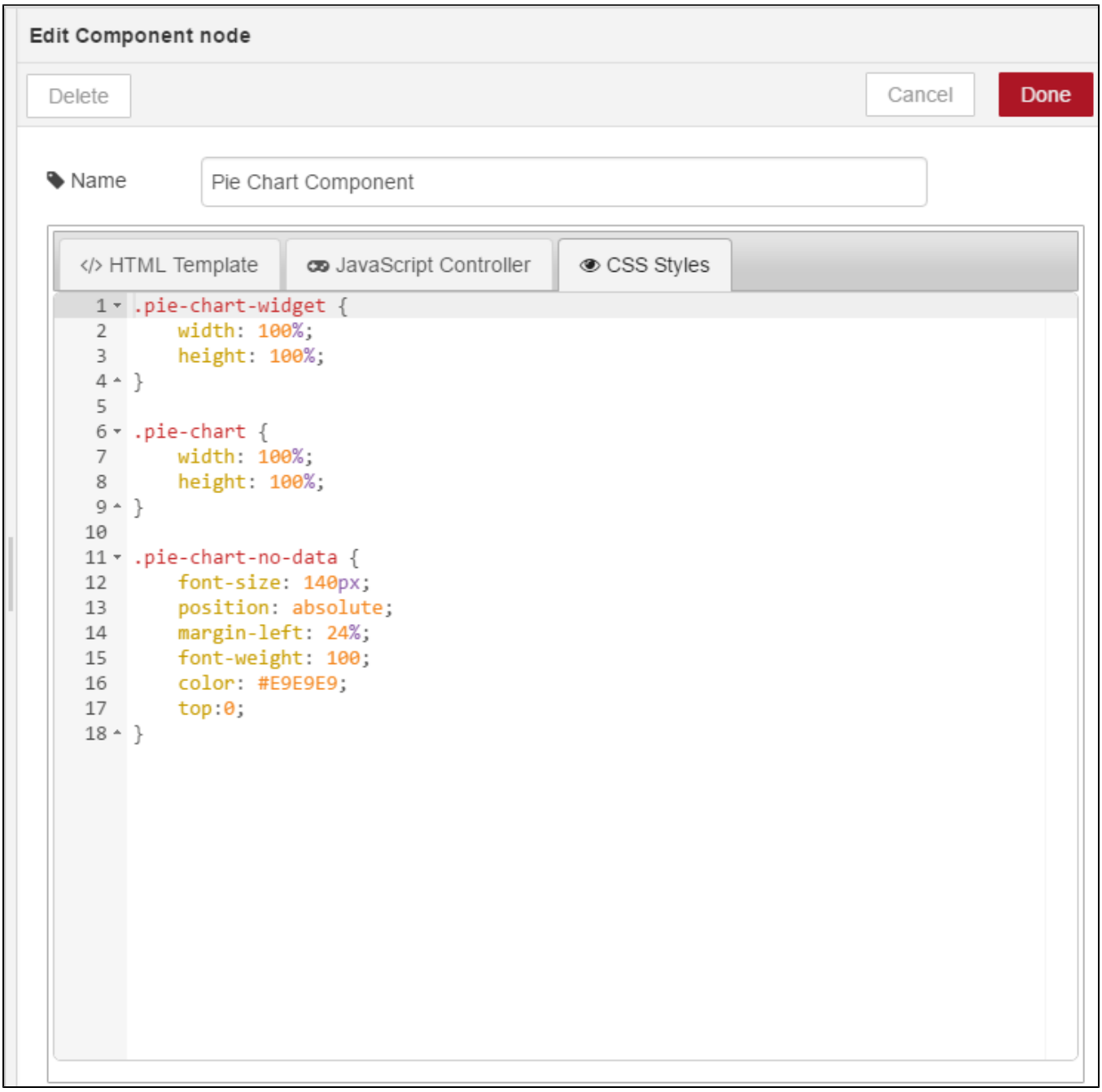
JavaScript Controller

The JavaScript Controller implements how to represent the chart according to the HTML Template. It controls the color of the chart and the drill-down operation.



CSS Style

This tab defines the CSS styling unique to the component. We highly recommend providing unique prefixes if you want to customize the CSS, otherwise, the style definitions may conflict with other widgets in same the dashboard.

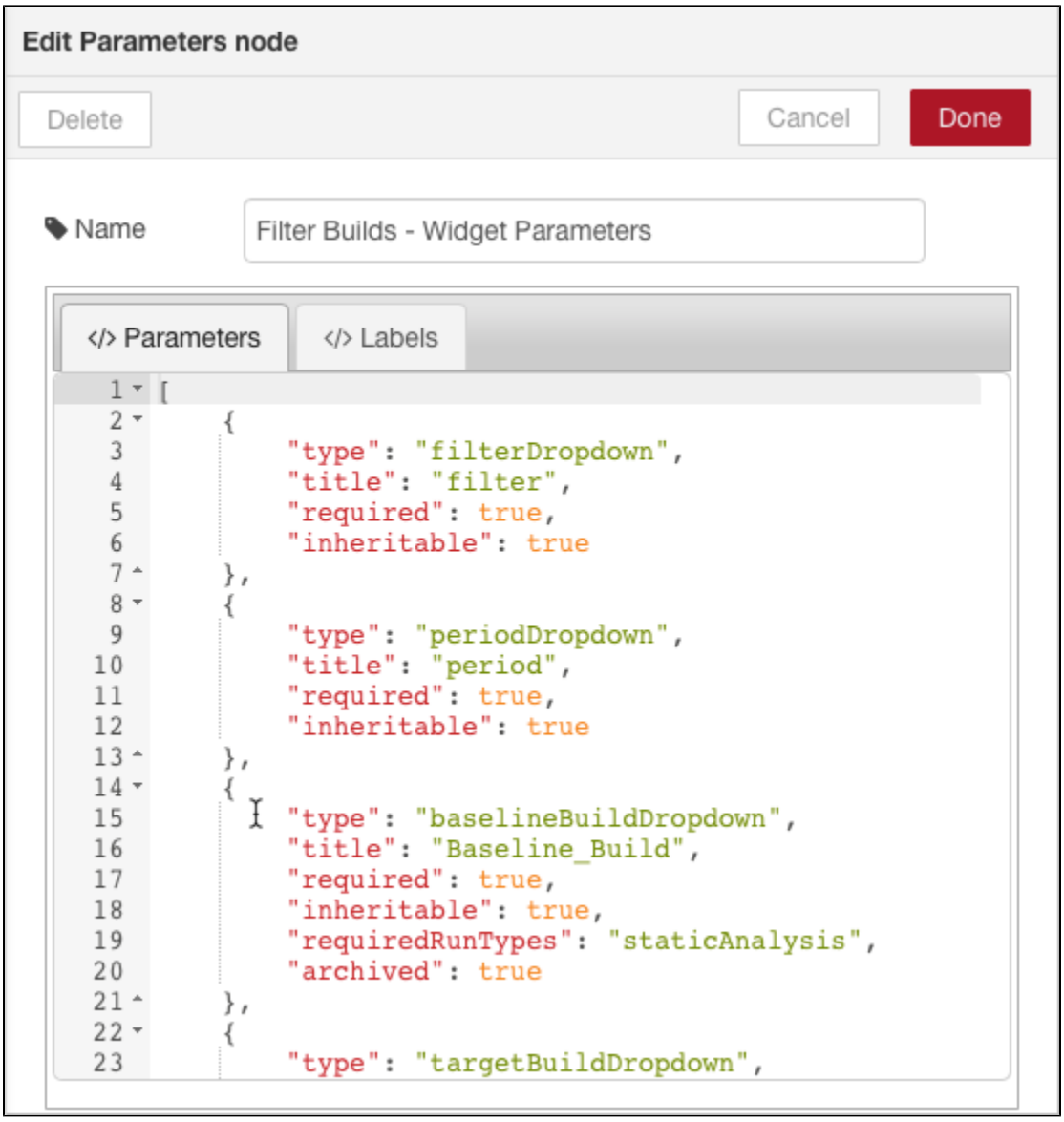


Configuring Parameter Nodes

Parameters define what options the widget can include under the Add Widget page. Parameters a Parameters tab and a Labels tab. You can also control which options are available using the Parameter Values Endpoint (see Using a Parameter Values Endpoint to Create Custom Parameters), as well as filter builds that appear in the drop-down menus (see Filtering Target and Baseline Build Settings).

Parameters Tab

This tab defines list of parameters and its data source. Some parameters can also link the dependencies so that filters, such as the build ID, can be set.

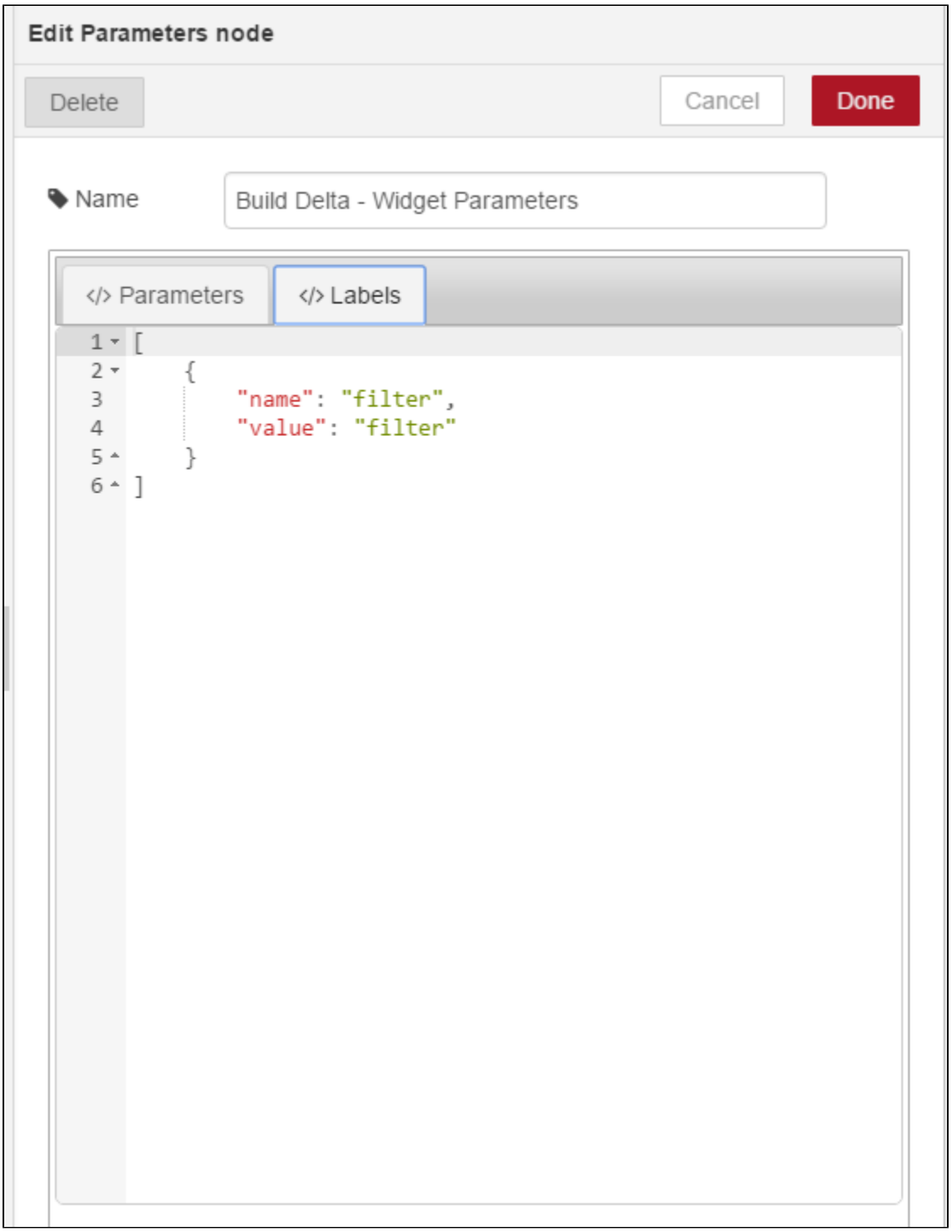


Always set the "required" parameter to "true"

When configuring widget settings, set the `required` parameter to `true`. If you set this parameter to `false`, an undefined all parameter is added to the widget by default, which will result in unexpected behavior.

Labels Tab

This tab defines how the parameter label is presented in the Add or Edit Widget page.



Parameters are tightly defined under the DTP dashboard. You can use a combination of known parameters, such as filters, periods, or baseline/target build, but you can only define text box parameters as needed. Some parameter nodes defined the Marketplace artifacts have a profile parameter definition (see Working with Model Profiles).

Pre-built Components and Parameters

Out of the box, Parasoft provides useful component and parameter definitions for creating your own custom widgets:

Components	Description

Bubble Chart	This component renders concentrations of data points as bubbles along an x- and y-axis. The Modules - Bubble widget is an example of a bubble chart.
Donut Chart	This component renders the data as proportioned segments and includes an overall value. The Policy Center Gate Summary Widget is an example of a donut chart.
Percentage Chart	This component renders the data as an overall percentage. The Coverage - Percent widget is an example of a percentage chart.
Pie Chart	This component renders the data as a pie chart with a legend. The Severities - Pie widget is an example of a pie chart.
Summary Chart	This component renders the data as a single summary value. The Metrics - Summary widget is an example of a summary chart.
Table Chart	This component renders the data as a table with five rows and an optional link to an additional report showing the complete data. The Authors - Top 5 Table widget is an example of the table chart.
TreeMap Chart	This component renders the data into tiles with sizes proportional to the data point values. The Modules - Top 10 Tree Map widget is an example of the tree map chart.

Parameters	Description
Build Delta - Report	This node specifies fields for a filter, time period, baseline build, and target build as parameters to be passed to a report. The node is useful for calculating the difference between two builds. It also allows you to resolve the build ID of the first or latest build within a time period for baseline builds, as well as resolve the build IDs of the latest build for target builds.
Build Delta - Widget	This node specifies fields that can be configured in the widget creation edit dialog. This parameter node allows for filter, time period, baseline build, and target build to be configurable fields. This parameter node is useful for calculating the difference between two builds. It allows you to resolve the build ID of the first or latest build within a time period for baseline builds, as well as resolve the build IDs of the latest build for target builds.
Build Delta with Profile - Report	This node is similar to the "Build Delta - Report" parameter node, but it also specifies a parameter for a profile that a report can use to access static data stored in Extension Designer.
Build Delta with Profile - Widget	This node is similar to the "Build Delta - Widget" parameter node, but it also adds an input field for a profile that the widget can use to access static data stored in Extension Designer.

The same set of parameters are defined per widget and report. This is because a different set of parameters are required for the drill down report.

Each component node is configured to take a specific payload to render the chart. All definitions are provided under **Import> Library> Sample Widgets**. Import an example widget and review the **Sample Data** node for details.

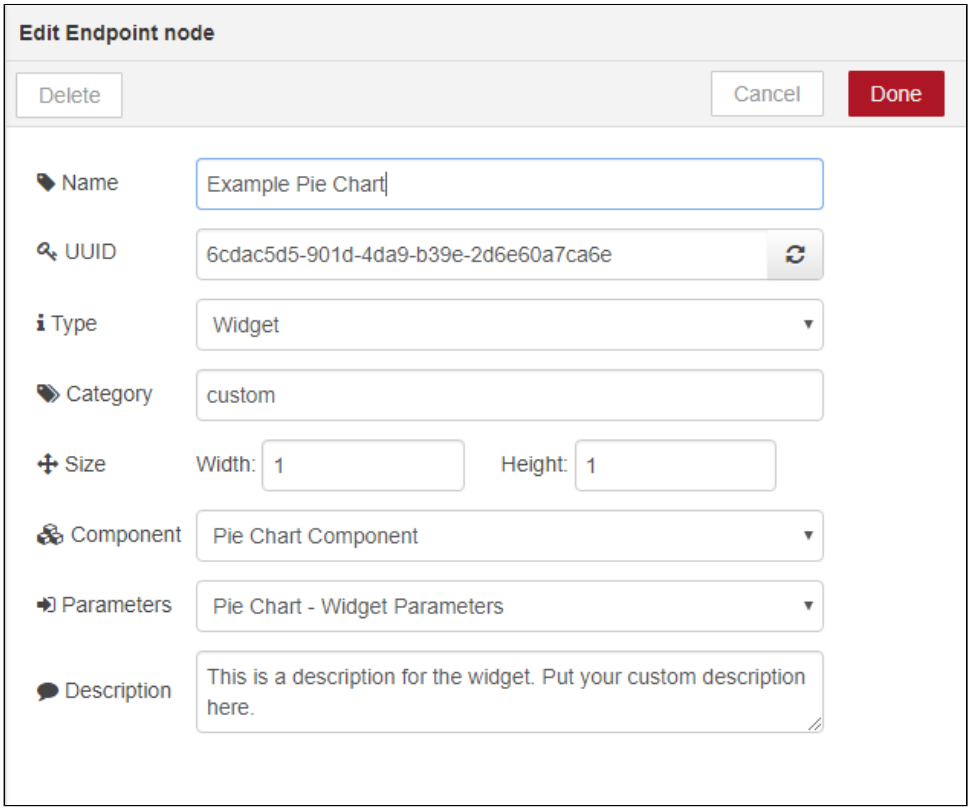
Every time you deploy a widget or a report flow, you must perform a full refresh of the dashboard browser to see the update. The component and parameter definitions are only updated when the dashboard is loaded. Refreshing the widget only won't reflect the changes to the component and parameters. However, if you are working on a data flow, you can refresh only the widget to see the changes.

Creating new Pie Chart widget

This section explains how to create new chart widget in DTP using the Pie chart sample widget.

1. Choose **Import> Library> Sample Widgets> Pie Chart** example from the Extension Designer menu to import the example to a new flow tab in any service.

2. Double-click the **Example Pie Chart** endpoint node and review the configuration. This node defines the widget implementation.

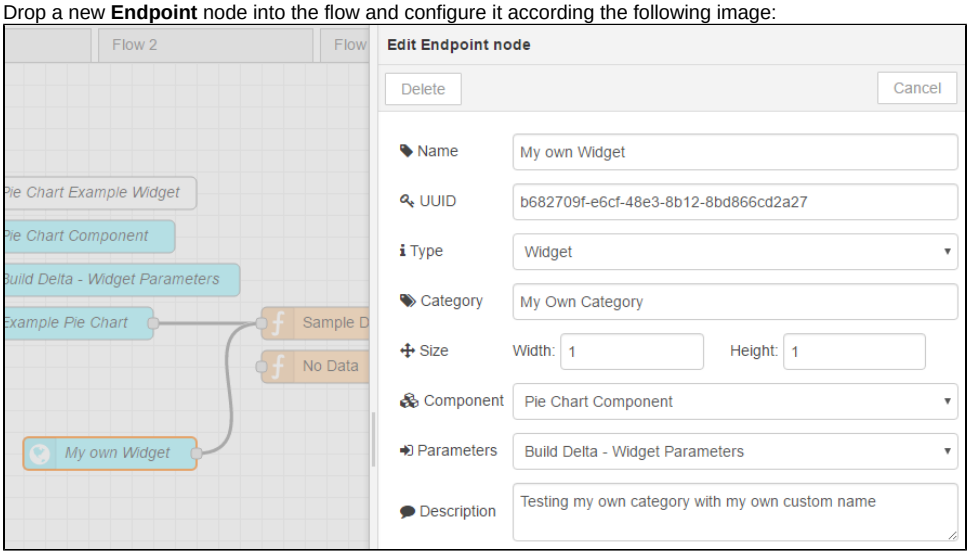


You can change the following fields to configure the example widget:

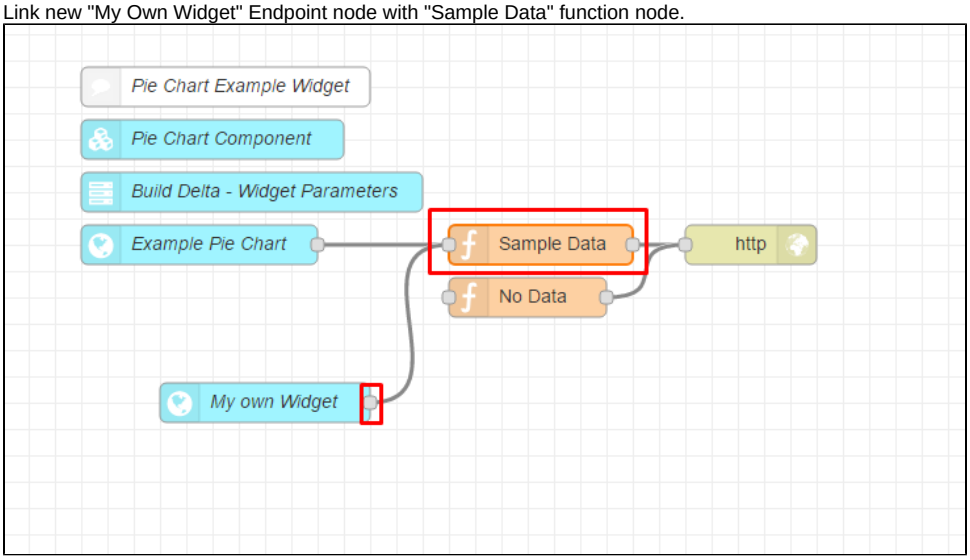
Name	Name of the widget displayed in the DTP dashboard
UUID	Unique identifier for the endpoint that is automatically generated when you drop the endpoint node into the flow canvas.
Type	Choose an endpoint type from the Type drop-down menu: • Widget: Specifies a DTP dashboard widget. Once this value is selected, the size, category, component, and parameter fields will be available. See About the Dashboard Grid for additional information about sizing widgets. • Report: Specifies a custom report for the DTP dashboard. Once this value is selected, the component and parameter fields will be available • Practice: Specifies a Policy Center practice (see Defining Policies). Once this value is selected, the component and parameter fields will be available. • General: Specifies a general REST endpoint. Both GET and POST operations will be available for the endpoint. To get the endpoint URL, open the Service Category Page and copy the URL.
Category	Defines the DTP dashboard widget category. In general, the value should be either "custom" or "process intelligence". You can enter a new name to create a new category. The widget header color will be gray if you create your own category name.
Component:	Specifies the component to provide. Any components deployed to the flow canvas should be available from the drop-down menu.
Parameter	Specifies the set of parameters to provide to the DTP dashboard. All parameters deployed to the flow canvas should be available from the drop-down menu.
Description	Specifies a description of the endpoint. This description will be used on DTP dashboards, the Add Widget page (see Adding Widgets), and the Extension Designer's category page as the endpoint description.

3. Click the refresh button at the UUID field. This will ensure that the new widget does not conflict with any other endpoint.
4. Deploy the widget.

5. Drop a new **Endpoint** node into the flow and configure it according the following image:

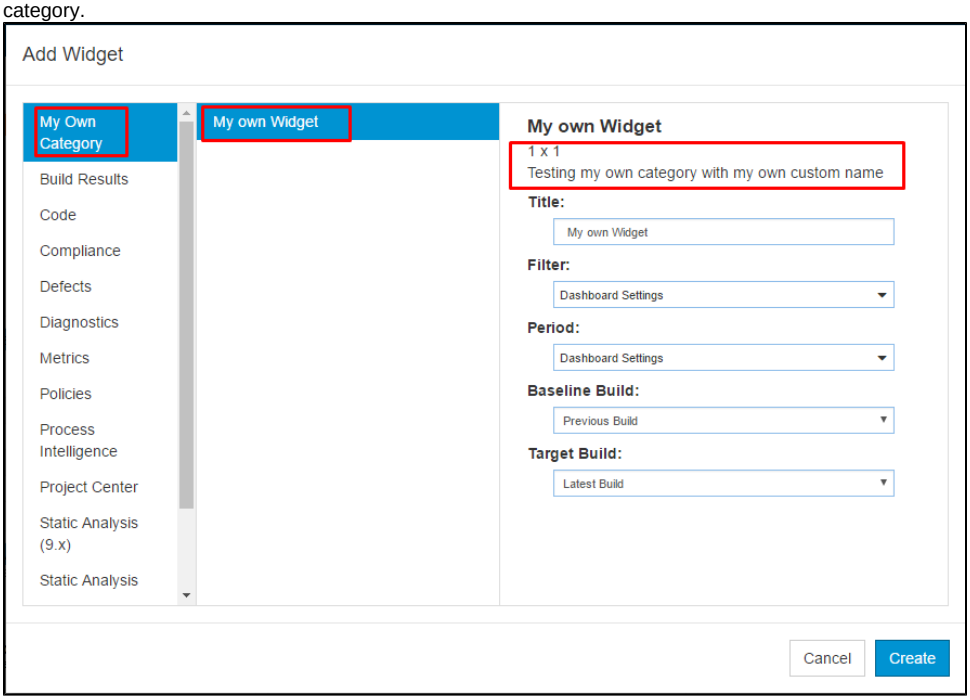


6. Link new "My Own Widget" Endpoint node with "Sample Data" function node.

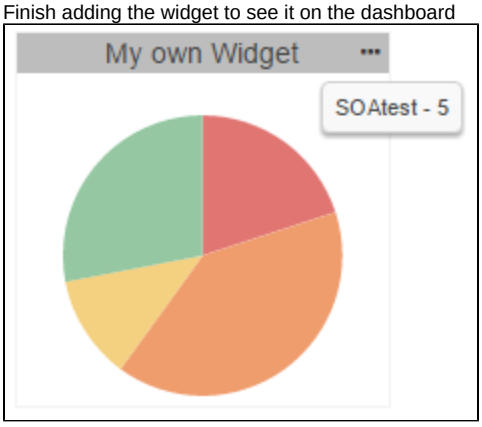


7. Deploy the flow.

8. In Report Center, refresh the dashboard (if already open) and click **Add Widget**. The Add Widget overlay will display your widget on its own category.

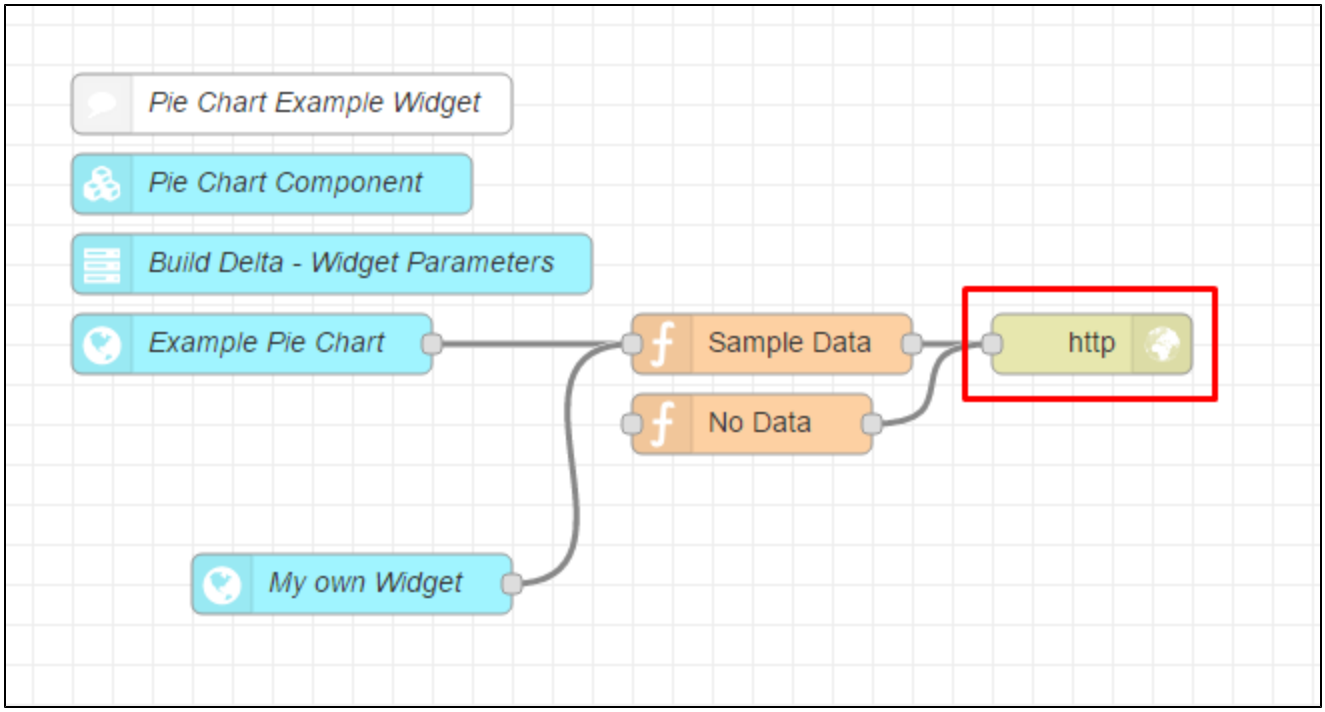


9. Finish adding the widget to see it on the dashboard



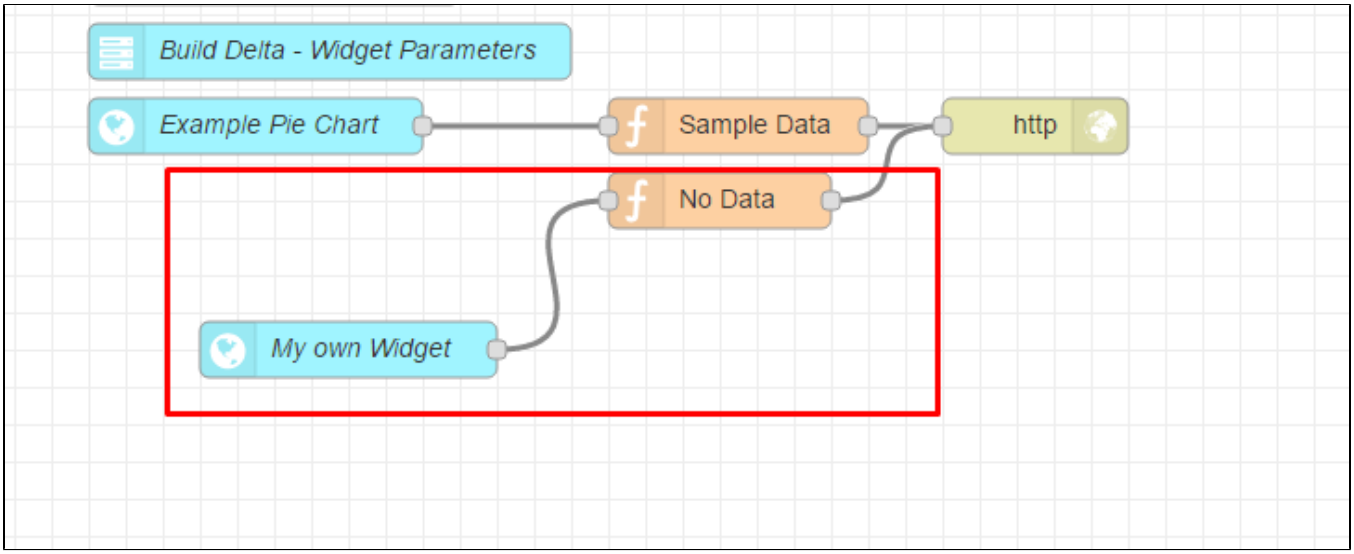
About the Endpoint Output Link

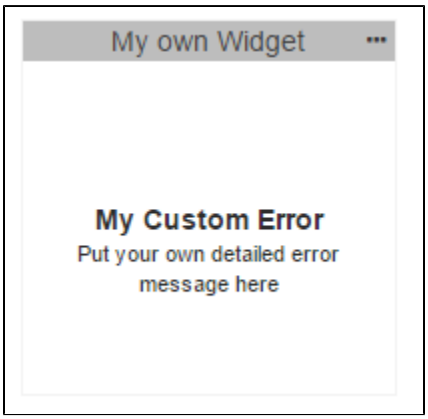
The Endpoint output node links to the data source for the widget. You should use the **http response** node in output category as the output.



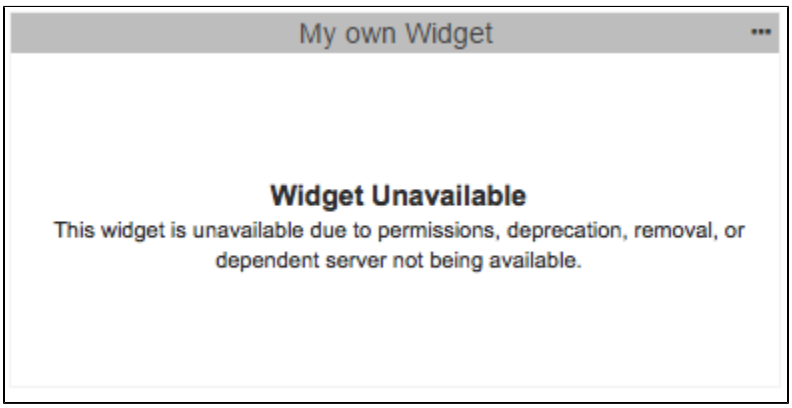
Widget Error Messages

The nodes provide a structured msg.payload as the output to the endpoint. The DTP dashboard will use the response as data for the widget. In every sample widget, we provide a **Sample Data** function node (with actual data) and a **No Data** function node for possible payload schema. The No Data node demonstrate how to send an error message back to the widget so the widget can display the message properly.





If you delete the endpoint after adding the widget to the dashboard, DTP will detect that the widget definition is no longer available and show following message:



Restoring the endpoint node back (with same UUID), and refresh the dashboard, you will see your widget back.

Configuring Error Messages

Error messages use the following Standard Error Object format, which applies to widgets and any flows that communicate with it.

```
msg.payload = {
    error: {
        title: "error message title",
       message: "detailed error message"
    }
}
```

If this error object returns to a custom widget or report, it should add following to msg object:

```
msg.statusCode = 400;
```

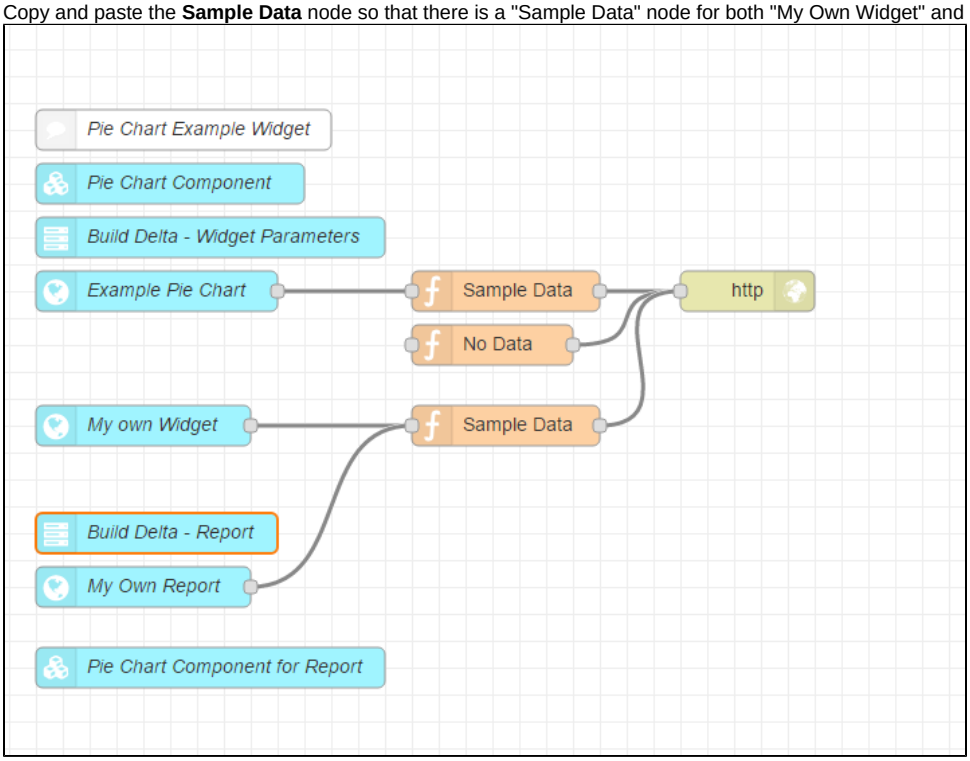
Creating a Custom Pie Report

The report concept is the same as the widget, but the report endpoint type doesn't have size limitations and can accommodate much more information. You should be able to mix one or more components into a single "Report" component to draw more complicated reports.

All components are optimized for the widgets and in this tutorials. We will explain how to add a new component for report and create a custom report using Pie Chart.

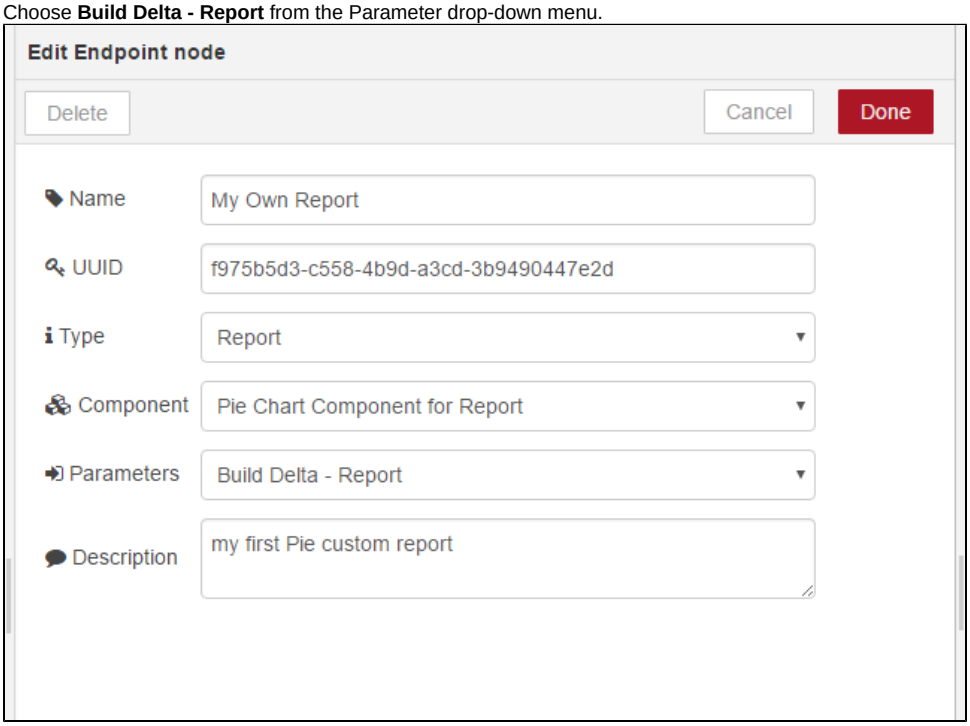
To create new custom report, we need to add additional parts to the service:

1. Import a new "Build Delta - Report" Parameter node from **Import> Library> Parameters> Build Delta - Report**. This parameter node is not used, but you will see the difference in the content.
2. Copy the **Pie Chart** component node (Ctrl/Command + c) and paste it (Ctrl/Command + v) to the flow canvas. Rename the node to "Pie Chart Component for Report".
3. Copy and paste the **My own Widget** endpoint node and rename to "My Own Report."
4. Copy and paste the **Sample Data** node so that there is a "Sample Data" node for both "My Own Widget" and "My Own Report"



5. Double-click the **My Own Report** node and choose **Pie Chart Component for Report** from the Component drop-down menu.

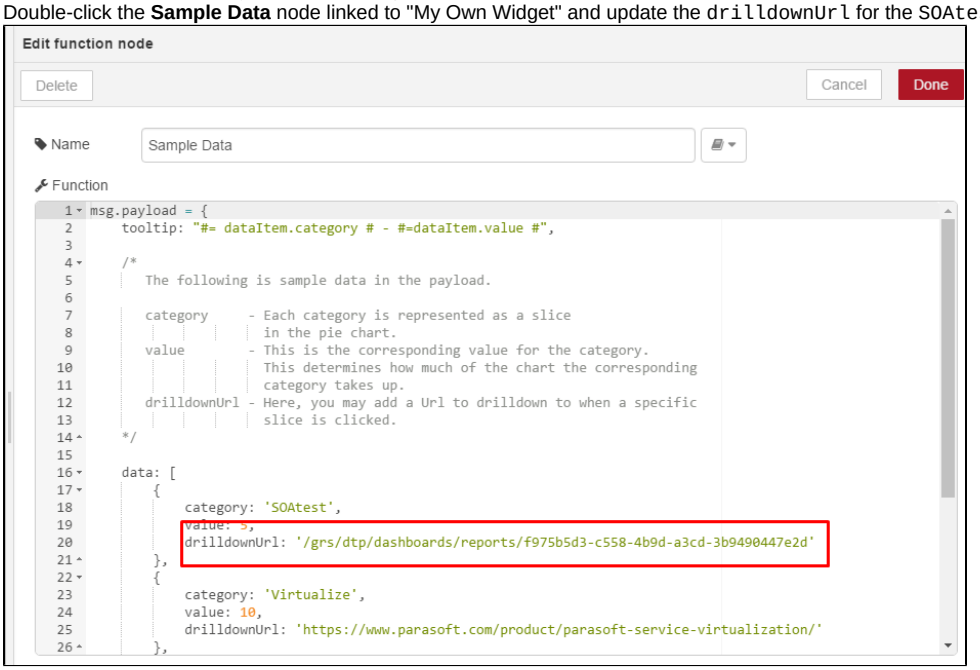
6. Choose **Build Delta - Report** from the Parameter drop-down menu.



7. Copy the endpoint node's UUID, which is part of the URL for the report:

   <DTP URL>**/grs/dtp/dashboards/reports/<Report Endpoint UUID>**.

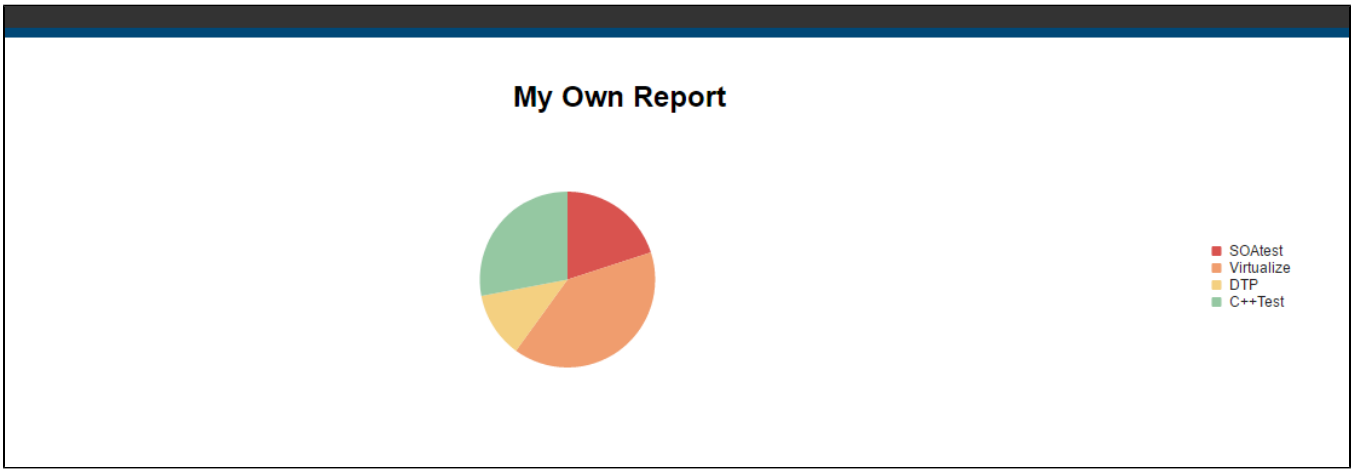
   You can use this URL into your drill-down report.
8. Double-click the **Sample Data** node linked to "My Own Widget" and update the `drilldownUrl` for the `SOAtest` category with the endpoint URL.



When a user clicks on the SOAtest section of Pie Chart, it will drill down to your own custom report. If necessary, you can your own parameter list into this URL. If you test the report at this point, it will throw an "Controller" exception because some parts of component are designed for the widget. We can fix this issue easily:

1. Double-click the **Pie Chart Component** node for the report.
2. Click JavaScript Controller tab
   The source of the error occurs at line 23:
   `$scope.options.legend`
   The component is looking for widget size. The logic from the widget is to add a legend if the X-axis size is larger than the Y-axis. Since this is a report, it can show a legend without any space limitation.
3. Change line 23 to legend:
   `{ visible: true}`
4. Click **Done** and **Deploy.**

Your simple custom report looks like following:

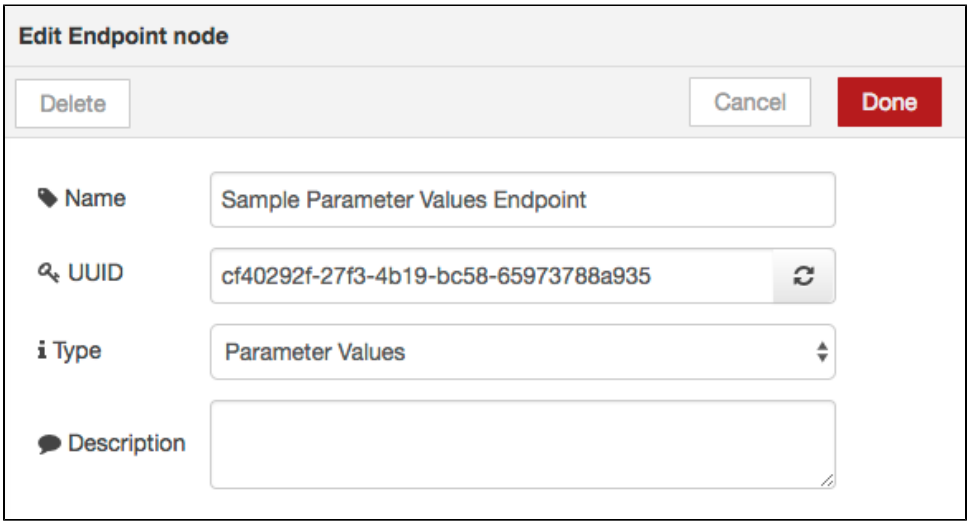


You can add additional information to the "Pie Chart Component for Report." For more advanced custom reports, review the endpoints from the Test Impact Analysis, DTP File Based Licensing Report, Modified Coverage or Test Failures by Build artifacts.

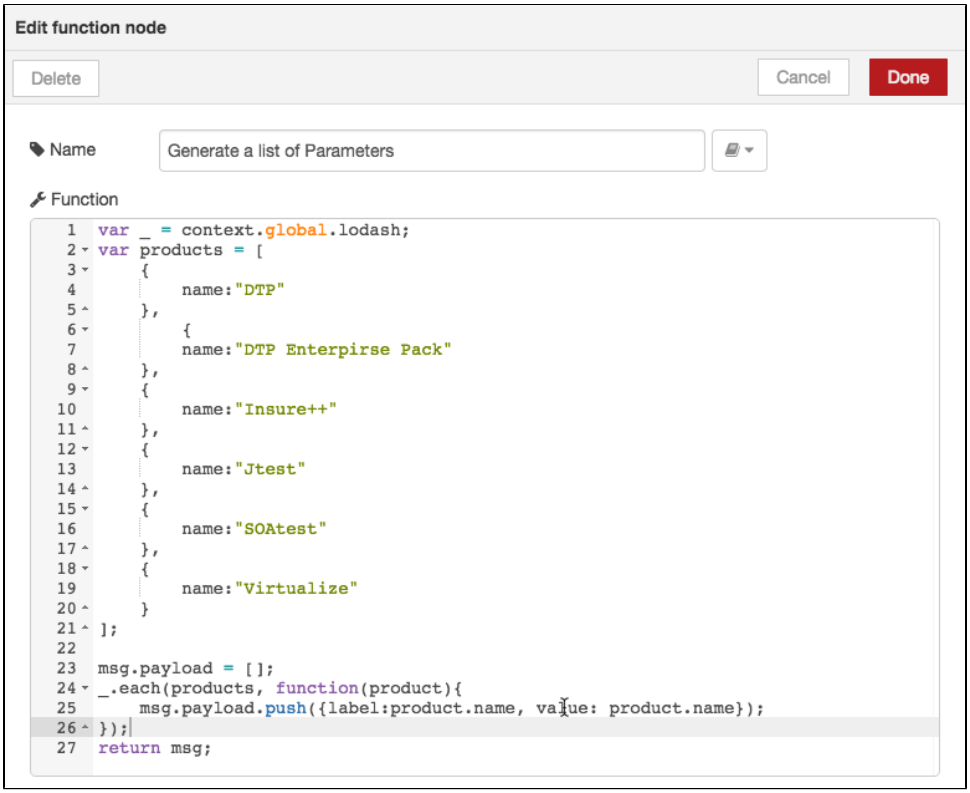
Creating Custom Parameters

You can create a flow that uses the Parameter Values Endpoint to expand the available parameters that can be sent to a Widget endpoint. You can control the output of the Parameter Values Endpoint in a variety of ways. In this tutorial, we will demonstrate how to generate a static list of values.

1. Add an Endpoint node to your flow and double-click it to configure its settings.
2. Choose **Parameter Values** from he Type drop-down menu.



3. Add a Function node to your flow and attach it to the Parameter Values Endpoint.
4. Double-click the Function node to edit it contents. In the following screenshot, we have created an array of Parasoft products to be returned from the endpoint.



The return payload must be formatted into an array where each element is an object containing values for the parameter **label** and the parameter **value.** The **label** of the parameter determines what is displayed in the resulting dropdown menu in the widget configuration. The **value** of the parameter determines the actual value of the dropdown option that is sent back to the widget endpoint.

5. Add an HTTP Response node after creating the Function node.

> **Use API nodes to dynamically generate values for the Parameter Values Endpoint**
>
> For example, if you have a model with many different profiles under it, you can configure a Parameter Values Endpoint to perform a Profile Search for all profiles under a model.

6. Create a widget by importing the example Component and Parameters nodes (see Configuring Component Nodes). No action needs to be taken in the Component node to use the Parameter Values Endpoint.
7. Open the Parameters node and add the following JSON object in the Parameters tab.

```
[
        {
                "apiParameter":"processIntelligence",
                "type":"dropdown",
                "title":"Product",
                "name":"product",
                "uuid":< uuid of the Parameter Values Endpoint created in step 2 >
        }
]
```

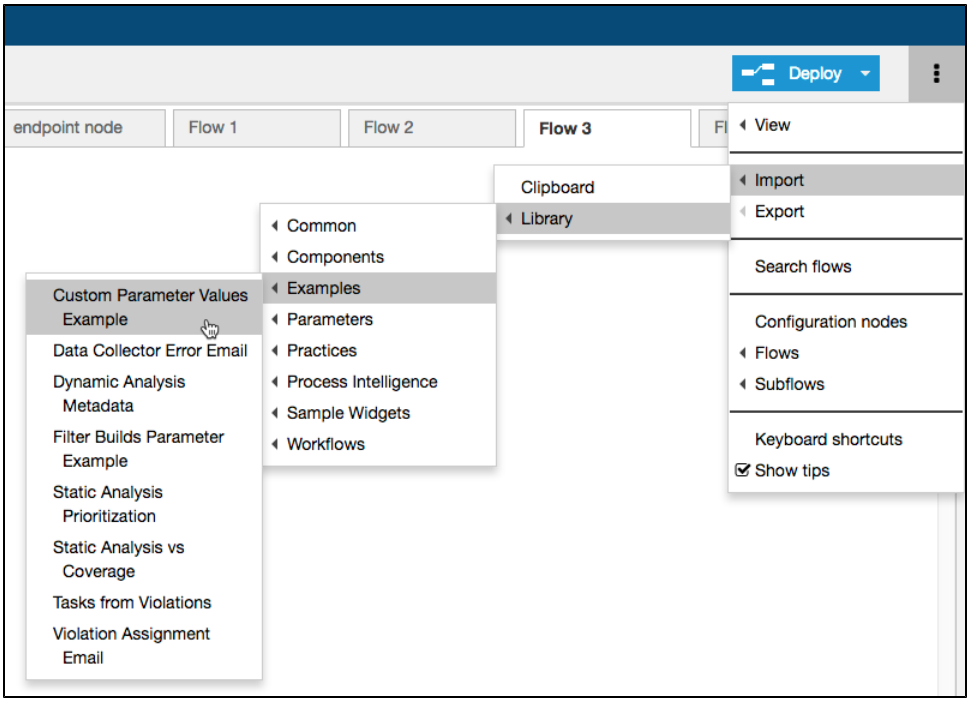
The `processIntelligence` value assigned to `apiParameter` denotes the use of a custom parameter dropdown. You can get the value for `uuid` by opening the endpoint node. Without the correct UUI, the widget will not be able to retrieve the list of parameter values.

8. Deploy the flow to DTP.

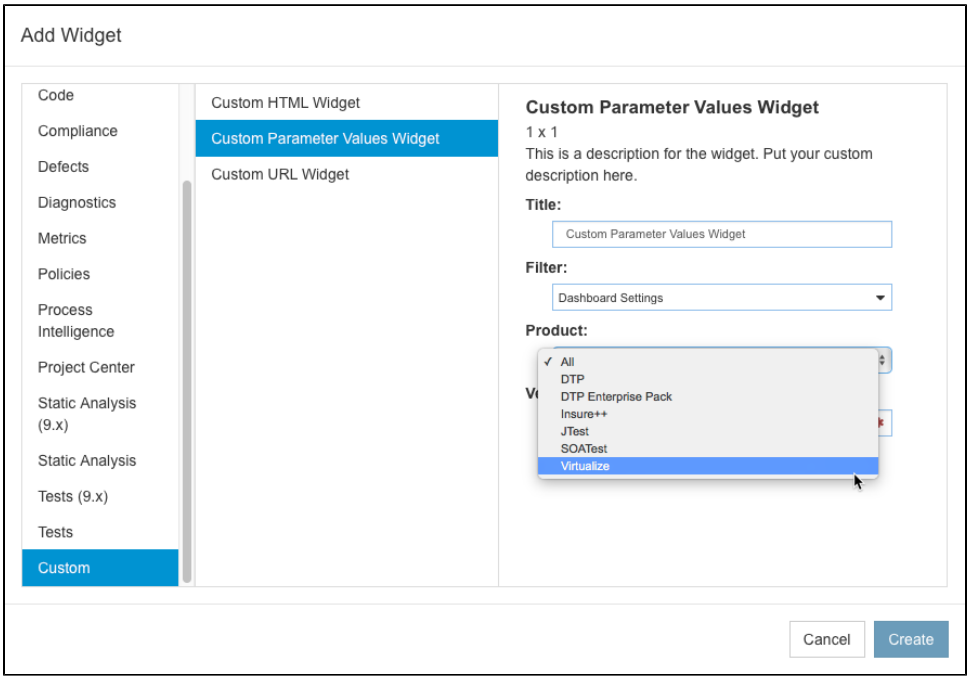
Using the Custom Parameters Value Example

For demonstration purposes, Extension Designer ships with a functioning example of a widget using a Parameter Values Endpoint.

1. Choose **Import>Library>Examples> Custom Parameter Values Example** from the library to import the example into your flow.



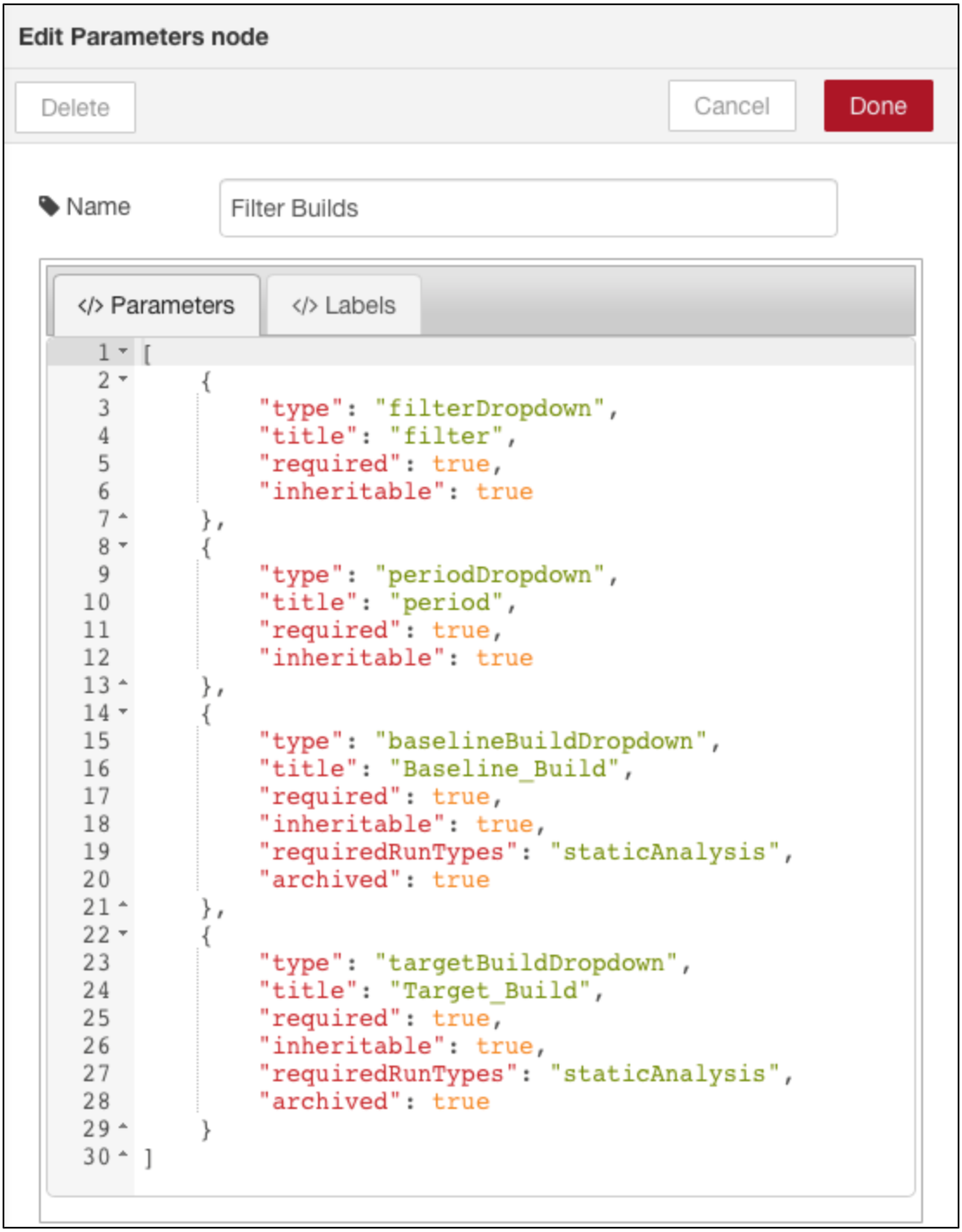
2. Click **Deploy** and refresh the DTP dashboard.
3. Add the Custom Parameter Values Widget to the dashboard from the Custom category. In the configuration, you can see that the statically generated list of Parasoft products is available for selection in the Product dropdown menu.



Filtering Target and Baseline Build Settings

You can configure your custom widgets to filter the target and baseline builds settings so that more concise options are presented when adding the widgets to your dashboard. Your DTP project should have at least two builds, one for the target and one for the baseline, in order to filter the build settings. Additionally, the data for builds must be archived (see Locking and Archiving Builds for details). DTP archives the two most recent builds by default, but you should manually archive any builds you want to include in your settings.

Add a Parameters node to your flow and include a `type` set to `targetBuildDropdown` and/or `baselineBuildDropdown` JSON object in the Parameters tab.



You can configure the `targetBuildDropdown` and `baselineBuildDropdown` parameters with additional options that enable more refined filtering.

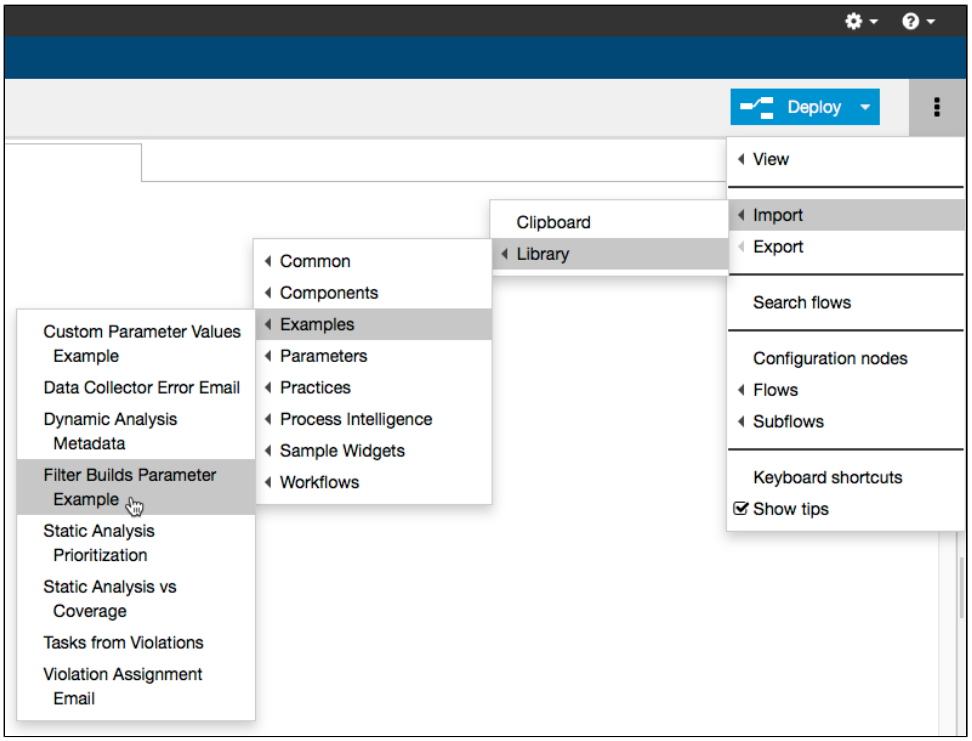
requiredRunTypes	This property accepts one comma-separated string. The string is a list of all run types contained in the build that you want to focus on. The following run types can be specified: • `dynamicAnalysis` • `coverage` • `staticAnalysis` • `metrics` Example: `"requiredRunTypes":"metrics, coverage"`
detailedRunTypes	

	This property accepts one comma-separated string. The string is a list of all run types contained in the build that you want to focus on. The run types specified in the string must "have details," which refers to data that has not been pruned from the database as part of regular cleanup activities. You can archive builds to ensure that specific builds always have details (see Using Build Administration for additional information). By default, DTP stores details for the last two runs of a build. The following run types can be specified: • `dynamicAnalysis` • `coverage` • `metrics` The runtype `staticAnalysis` is not supported as a `detailedRunType` option because all staticAnalysis runs always have details. Example: `"detailedRunTypes":"metrics, coverage"`
archived	Includes/excludes archived builds in the filter. Setting to `true` excludes unarchived builds, whereas setting to `false` excludes archived builds.
locked	Includes/excludes locked builds in the filter. Setting to `true` excludes unlocked builds, whereas setting to `false` excludes locked builds.

Using the Filter Builds Parameter Example

For demonstration purposes, Extension Designer ships with a functioning example of a widget configured to filter builds.

1. Choose **Import> Library> Examples> Filter Build Example** from the library to import the example into your flow.



2. Make sure you have archived at least two builds associated with your project as described in Filtering Target and Baseline Build Settings.
3. Double-click the **Filter Builds - Widget Parameters** node and make any changes to the JSON objects in the Parameters tab.
4. Deploy the example.

5. Add the widget to your dashboard. Only builds that match the settings in the Parameters node will be listed in the widget settings.