Course Exercises

IBM Jazz for Service Management 1.1 Dashboarding Made Simple

Course code TN700 ERC 1.0

IBM Training

December 2016 edition

Contents

About these exercises

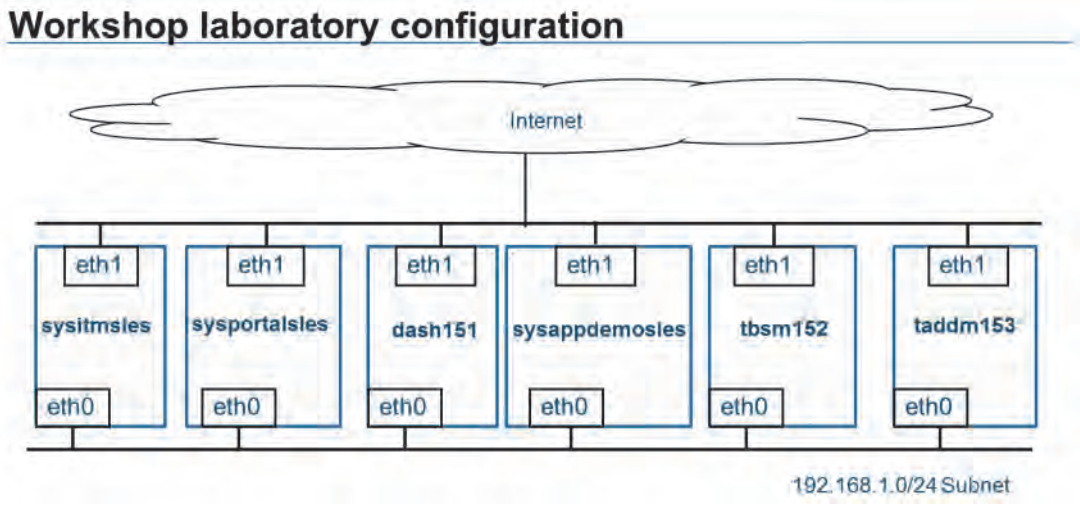
Lab environment

The lab environment consists of six Linux virtual images. The images and installed components are summarized in the following table.

Image label	Components
dash151	IBM® Jazz for Service Management 1.1.2.0 OMNIbus ObjectServer 8.1.0.3 Web GUI 8.1.0.2 DB2® 10.1.0.5 Tivoli Directory Integrator 7.1.1.4 Firefox 24.5.0 ESR
tbsm152	IBM Tivoli® Business Service Manager 6.1.1.3 IBM Netcool/Impact 6.1.1.2 Web GUI 7.4.0.3 Tivoli Integrated Portal 2.2.0.15 Firefox 31.2.0 ESR Smart Cloud Application Performance Manager 7.7.0.2 DB2 10.1.0.5
taddm153	Tivoli Application Dependency Discovery Manager 7.3.0.1 DB2 10.1.0.5 IBM Netcool/Impact 7.1
sysitmsles	APM Smart PoC Demo VM1 IBM Monitoring 6.3.0.5 Agent Support - DB2 9.7.0.10

Image label	Components
sysportalsles	APM Smart PoC Demo VM3 APM UI 7.7 IF1 DB2 9.7.0.10 IBM Monitoring 6.3.0.5
sysappdemosles	APM Smart PoC Demo VM4 Applications: • WebSphere • MQ • IIB • IHS • WSRR DB2 9.7.0.10 IBM Monitoring 6.3.0.5

The following diagram shows the logical connections between the applications and virtual images in the laboratory exercise environment.



The image host names, static IP addresses, primary user names, and corresponding passwords that are used with each virtual image are listed in the following table.

Image host name	eth0 IP Address	Primary user name	Password
dash151.poc.ibm.com	192.168.1.151	root	object00
tbsm152.poc.ibm.com	192.168.1.152	tbsmadm	object00
taddm153.poc.ibm.com	192.168.1.153	root	Smartp0c
sysitmsles.poc.ibm.com	192.168.1.121	root	Smartp0c

Image host name	eth0 IP Address	Primary user name	Password
sysportalsles.poc.ibm.com	192.168.1.123	root	Smartp0c
sysappdemosles.poc.ibm.com	192.168.1.124	root	Smartp0c

Unit 1 Dashboard design fundamentals, DASH overview, and strategy exercises

In the exercises in this unit, you develop and review basic dashboard development skills. You begin the process of designing a set of connected dashboard pages for a customer. You use the design to develop and present the dashboards at the end of the workshop when you develop skills in mining and showing data from many different sources. Next, you start the DASH server components and review the basic elements of the DASH server console. You then add custom web content to the myBox web server object that is installed in the DASH server. The myBox web server is used to service custom graphics, web pages, and JSP pages.

Exercise 1 Creating dashboard prototype diagrams

In this exercise, you create at least two dashboard page prototype diagrams. The design of the pages is up to each student, but should meet the following criteria:

- Targeted at a particular user persona (such as Operator, Executive)
- Provide a work solution for the target persona that can be demonstrated
- Include at least one high-level summary view page that includes drill-down to a more detailed view

The dashboard pages can exceed two pages, but you should keep the focus narrow, so that the dashboards can be developed and demonstrated by the end of the workshop. You should draw on your own customer experiences to come up with some general idea for your dashboard design or you can get suggestions from the instructor.

The virtual machines in your laboratory environment include application data for the following programs:

- Tivoli Business Service Manager
- IBM Monitoring
- Application Performance Manager
- Tivoli Application Dependency Discovery Manager
- APM SmartPoC Demo
- APM SmartPoC AppDemo
- Netcool/OMNIbus
- Netcool/Impact
- Full access to the Internet, but not the IBM intranet

For this exercise, you are not concerned with the specific methods for retrieving data. Your design document should be a high-level, general view of the dashboard pages. These design documents do not have to be a final version. As you learn different data-mining skills over the remainder of the workshop, you might decide to modify the design.

1. Use presentation software, such as Microsoft PowerPoint or OpenOffice Impress, on your local workstation to develop your page designs.

   a. Use blocks to show each general page layout, with the size of each block used to show the relative size of a dashboard widget.

   b. Write a short description for each element in each page view, and a short description of the purpose of the dashboard page. For example, “This page shows a summary view of the servers that exceed recommended disk space thresholds”.

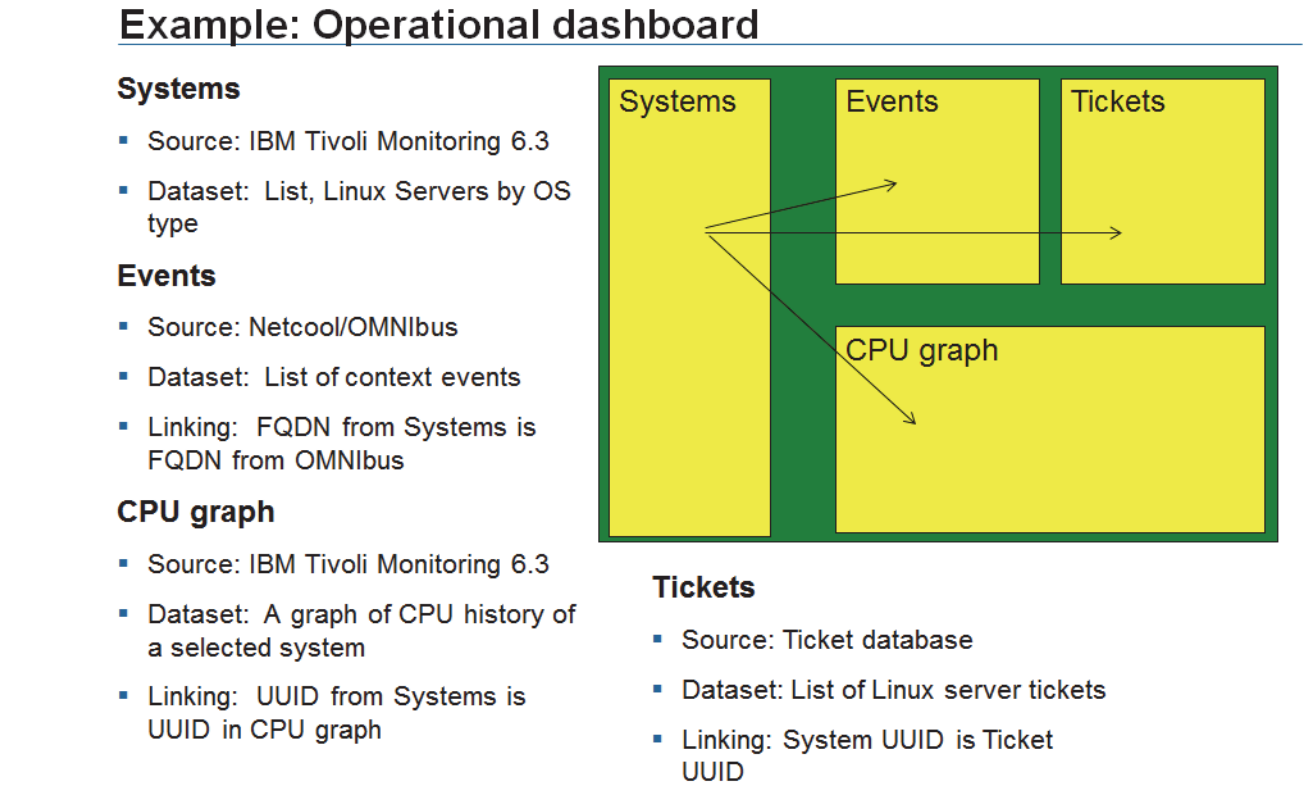
   c. Each block should represent a dashboard widget and should have a simple label that describes the function.

   d. Use directional arrows to indicate context or navigational movement from one widget to another or one page to another.

   e. Identify, at a high level, the source of the data for each widget block. For example, is the source from an application database or is it context data from a widget selection?

Hint: Do not document a specific method for data retrieval for any widget block in your design document in this exercise. You have an opportunity to supplement your design document as you learn different data mining techniques.

The following image shows an example for a dashboard page design document:



Exercise 2 Creating a DASH connection document

In this exercise, you create a connection document to the Tivoli Business Service Manager server. You use the UI data provider connection in Unit 1, Exercise 3 on page 1-10. You complete the following tasks:

- Start the DASH server and Netcool/OMNIbus ObjectServer on the dash151 virtual image
- Start the Tivoli Business Service Manager services on the tbsm152 virtual image
- Create a DASH connection document to the Tivoli Business Service Manager server

Starting the DASH services

Instructions for logging on to the virtual images were sent to each student before the start of class.

1. Switch to the **dash151** virtual image desktop. The image is configured to automatically log in to the **root** user ID. You do not need to enter the password, **object00**.
2. Double-click **Start DASH Services** on the virtual image desktop.

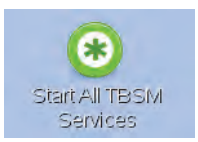
A custom start script opens in a command window. The Netcool/OMNIbus ObjectServer starts, followed by the DASH server. The script takes 30 - 45 seconds to complete.

```
********************************************************************************
Starting Jazz for Service Management profile
********************************************************************************

Running command: /opt/IBM/JazzSM/profile/bin/startServer.sh server1

ADMU0116I: Tool information is being logged in file
           /opt/IBM/JazzSM/profile/logs/server1/startServer.log
ADMU0128I: Starting tool with the JazzSMProfile profile
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is 14984

Jazz for Service Management startup complete

Start commands output in /root/jazz_components_start.log file...

Press the Enter key to close window...
```

3. When the command script is completed, close the command window. Click the window and press the **Enter** key.

Starting the Tivoli Business Service Manager services

1. Switch to the tbsm152 virtual image desktop. The image is configured to automatically log on as the user ID **tbsmadm** with the password **object00**.
2. Double-click **Start All TBSM Services** at the left of the virtual image desktop.



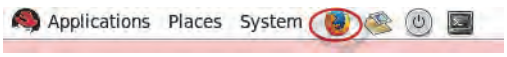
A custom start script opens in a command window. The Tivoli Business Service Manager data server starts, followed by the Tivoli Integrated Portal server. The script takes 30 - 45 seconds to complete.

```
************************************************************************
Starting TBSM Dashboard Server
************************************************************************

Running command: /opt/IBM/tivoli/tipv2/profiles/TIPProfile/bin/startServer.sh se
rver1

ADMU0116I: Tool information is being logged in file
           /opt/IBM/tivoli/tipv2/profiles/TIPProfile/logs/server1/startServer.lo
g
ADMU0128I: Starting tool with the TIPProfile profile
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is 54829
Starting XML Toolkit...

Start commands output in /home/tbsmadm/tbsm_suite_start.log file...

Press the Enter key to close window...
```

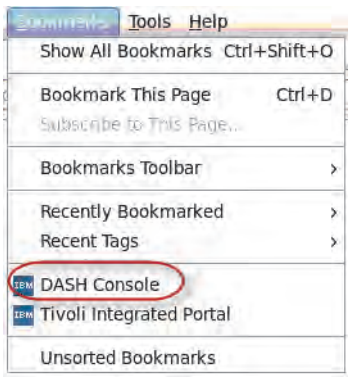
3. When the command script is completed, close the command window. Click the window and press the **Enter** key.

Logging on to the Dashboard Application Services Hub (DASH) console

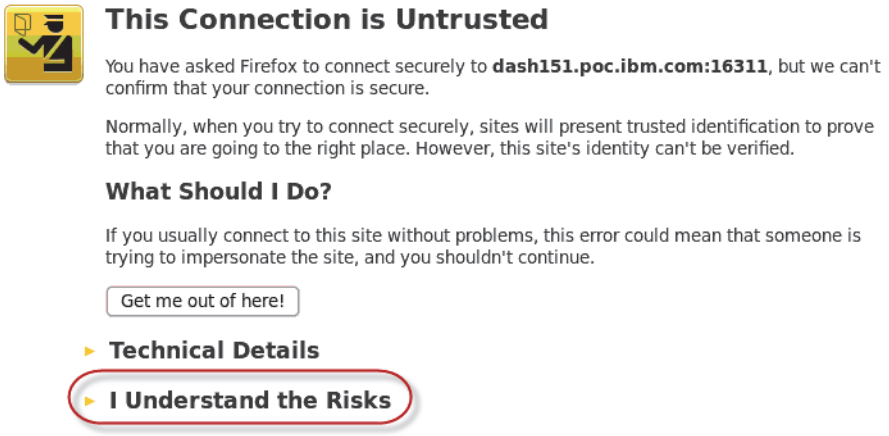
1. Switch to the **dash151** virtual image desktop.
2. Start the dash151 image browser application. Click the Firefox browser icon in the taskbar at the top of the virtual image desktop.



3. A bookmark is configured in the browser for the DASH server console login page. Select **Bookmarks > DASH Console**.

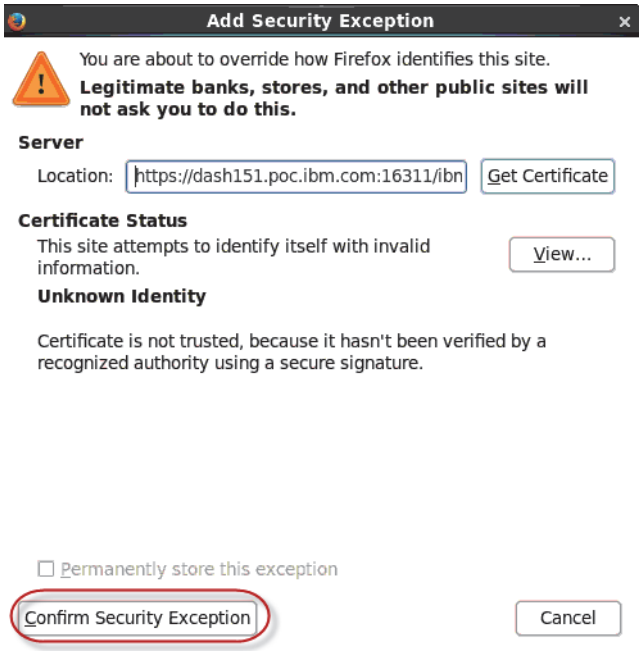


4. You see an untrusted connection window. Click **I Understand the Risks**.

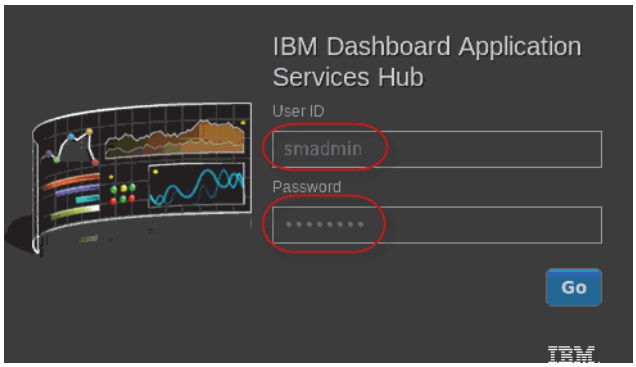


5. Click **Add Exception**.

6. Click **Confirm Security Exception**.



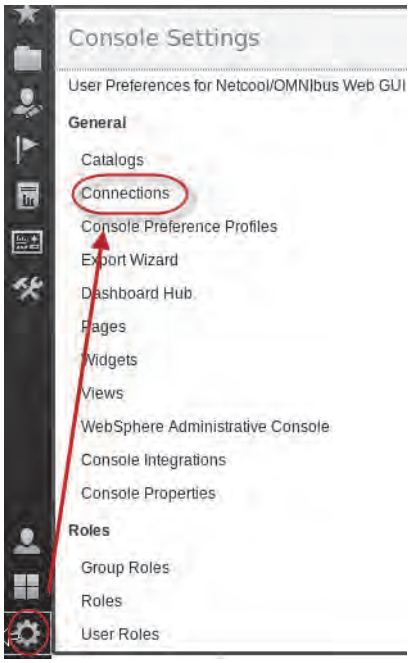
7. Log on to the DASH console with the user ID **smadmin** and the password **object00**.



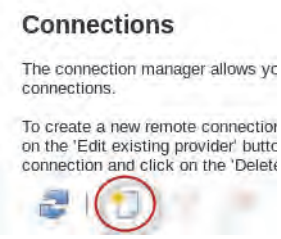
Creating a connection document

Complete the following steps to create the connection document:

1. Create the UI data provider connection document for the Tivoli Business Service Manager REST interface.
   a. Select the **Console Settings > General > Connections** task in the taskbar on the left side of the console.



   b. Click the **Create new remote provider** icon to start the connection wizard.

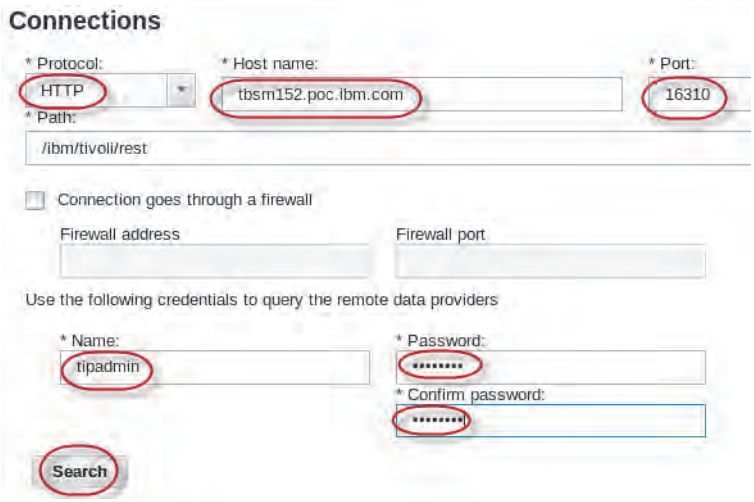


   c. Configure the form to query the Tivoli Business Service Manager UI data provider REST interface. Use the information in the following table to prepare the connection query.

Parameter name	Value
Protocol	HTTP
Host name	tbsm152.poc.ibm.com
Port	16310
Path	/ibm/tivoli/rest
Name	tipadmin

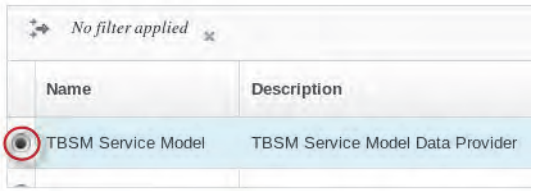
Parameter name	Value
Password	tipadmin
Confirm Password	tipadmin

d. Query the UI data provider. Click **Search**.



The list of available UI data providers is shown in a list. Multiple data providers are listed because the tbsm152 virtual image includes the Tivoli Business Service Manager dashboard server and the embedded Netcool/Impact server.

e. Select **TBSM Service Model**.



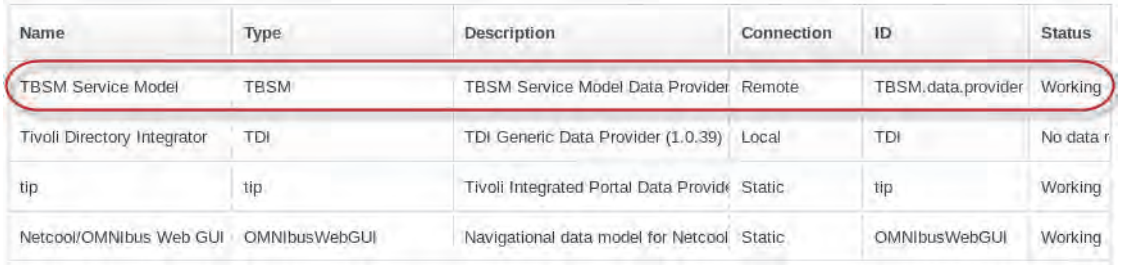
The UI data provider connection information is shown in the **Connection Information** section of the form.

f. Change the default provider ID value. Enter **TBSM.data.provider** in the **Provider ID** field.

Hint: By default, the provider ID value is set in the form **TBSM.<fully-qualified-host-name>** (for example, TBSM.tbsm152.poc.ibm.com). When the dashboards are developed in a test environment for later use in a production environment, a good practice is to use a common, generic provider ID. In this exercise, a generic provider ID, **TBSM.data.provider,** is used. When the default values are used in both environments, the dashboard export archive from one environment must be modified to match the target environment.

g. Do not select **Use the credentials of the user (requires SSO Configuration)** and click **OK**.

h. Verify that the connection document status is listed as **Working** in the Status column.



Name	Type	Description	Connection	ID	Status
TBSM Service Model	TBSM	TBSM Service Model Data Provider	Remote	TBSM.data.provider	Working
Tivoli Directory Integrator	TDI	TDI Generic Data Provider (1.0.39)	Local	TDI	No data r
tip	tip	Tivoli Integrated Portal Data Provid	Static	tip	Working
Netcool/OMNIbus Web GUI	OMNIbusWebGUI	Navigational data model for Netcool	Static	OMNIbusWebGUI	Working

i. Close the Connections page in the console. Click the **X** symbol in the Connections page tab at the top of the console.

Exercise 3 Dashboard creation and management basics

In this exercise, you review the basic tasks that are required to build and manage dashboard pages. You are tasked with creating two dashboard pages that show status and detail information for the ABCBank trouble-ticket system. The dashboard pages are viewed by executives with mobile devices. The company has three geographic regions, Asia, Europe, and US. The sum of the number of currently open high and critical severity trouble tickets are tracked, by region, by a Tivoli Business Service Manager business service. To simplify the exercise, you create two pages. The top-level page shows a summary of the status of the trouble ticket system, by region. If a region entry is selected, a second-level page is opened that shows detailed information of the number of

open high-level severity trouble tickets. A logical diagram of the dashboard navigation is shown in the following diagram.

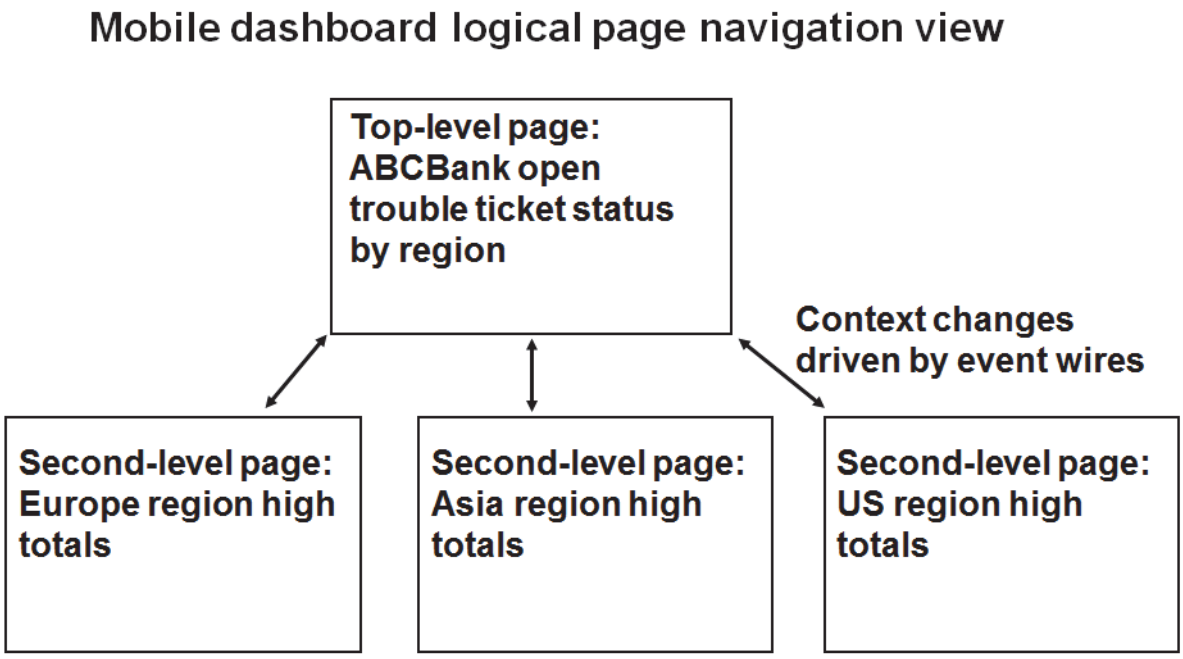


Figure 1 Dashboard page navigational diagram

To build the dashboard structure that is shown in Figure 1, you must create two separate dashboard pages and connect them with an *event wire*. Event wires are logical connections that pass navigational event messages between dashboard objects. For example, when an object is clicked, the object is configured to send a **NodeClickedOn** event to other objects on the dashboard page or to other pages. This mechanism allows context information to be passed from one dashboard object to another.

The following high-level tasks that must be completed:

- **Task 1**: You create the second-level dashboard page that is shown in Figure 2 on page 1-12. This page uses one **Analog Gauge** widget to show the total number of open high-severity trouble tickets, by region, for the ABCBank company. Only one page is required to show the values for each of the ABCBank regions. When a gauge in the top-level page is clicked, the regional context value is used to select the data that is shown in the second-level page. Estimated time to complete: 20 minutes.
- **Task 2**: You create the top-level dashboard page that is shown in Figure 4 on page 1-24. This dashboard page uses a **List** widget to show the overall trouble ticket system status for the three ABCBank geographical regions. Estimated time to complete: 10 minutes.
- **Task 3**: You create an event wire connection between the top-level and second-level pages and page objects. The event wire passes the selected context information between the dashboard pages. Estimated time to complete: 5 minutes.

- **Task 4**: You create and populate a mobile dashboard view with the dashboard top-level page. A view is a software object that is used to contain and organize dashboard pages. Estimated time to complete: 5 minutes.
- **Task 5**: You create and configure a console preference profile to use the view that contains the dashboard page. A console preference profile is a software object that controls user access to views and console navigational options that are related to the use of dashboards. Estimated time to complete: 5 minutes
- **Exercise 6**: You test the dashboard operation. Estimated time to complete: 5 minutes.

Creating the second-level dashboard page

In this task, you create the second-level page that is shown in the Figure 2 design diagram. The second-level page shows detailed trouble ticket information for a selected region of the company. The information includes the following detail:

- The number of open trouble tickets that are assigned a high severity

The Figure 2 logical diagram shows the arrangement and types of widgets that are used on the second-level page.

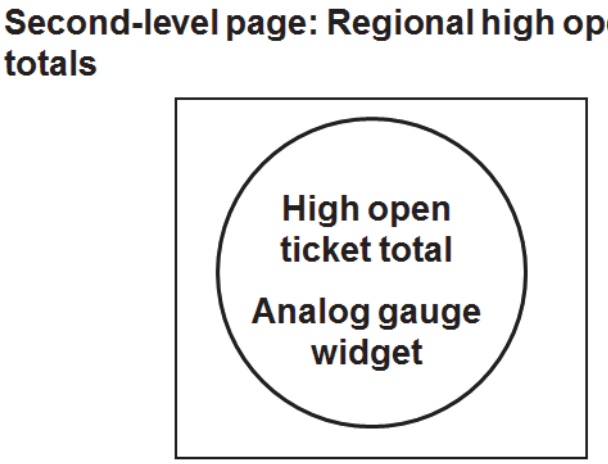


Figure 2 Second-level page logical diagram

You must complete the following high-level steps to complete this task:

- **Review the Tivoli Business Service Manager service model**: Jazz for Service Management provides tools to visualize data from multiple applications and business data sources. You

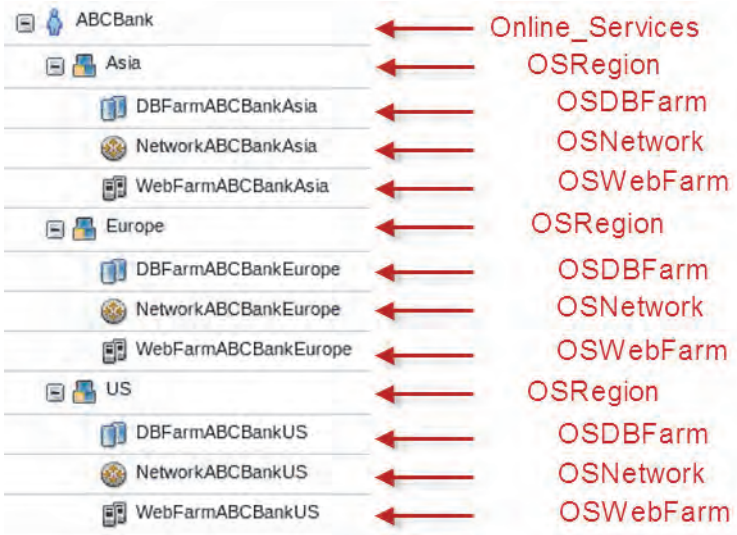
review, at a high level, how Tivoli Business Service Manager tracks and provides data that is used in the dashboard pages.

- **Configure the second-level page settings**: You configure the name, layout mode, and access authorization for the page.
- **Add widgets to the page**: You add an analog gauge widget to the page.
- **Configure widget properties**: You configure the properties for the widget on the page.
- **Arrange the widgets on the page**: You arrange the configured widget to match the logical design that is shown in Figure 2 on page 1-12.

Reviewing the Tivoli Business Service Manager service model

In this step, you review the configuration of a ABCBank business *service model*. A Tivoli Business Service Manager service model represents a business service. A service model consists of one or more service *templates* that are configured to track some aspect of a business. Templates model the properties and behavior of a service with one or more *rules*. Rules calculate the business service status or some key performance indicator (KPI) for a business service. *Template models* consist of one or more pairs of related templates that are connected in a parent-child configuration.

Service *instances* are modeled business services that are based on the behavior that is defined in the associated templates. The following logical diagram shows the ABCBank service model instances. The hierarchical relationships are shown in the tree view on the left. The templates that each service uses is shown in red on the right.



The rule names and attribute values are sent to a dashboard widget with the Tivoli Business Service Manager UI data provider. You configure how the rules and attributes are used in each Jazz for Service Management widget in a dashboard page. The service model that is used in these exercises tracks the open trouble tickets in a problem management database.

The configuration properties of the OSRegion template are shown in the Service Editor portlet. The template contains several rules. Each template rule tracks a business service status or a key

performance indicator (KPI). One of the rules calculates the number of open high severity problem tickets in a region of the ABCBank trouble ticket database. Another rule calculates the number of open critical severity problem tickets in a region.

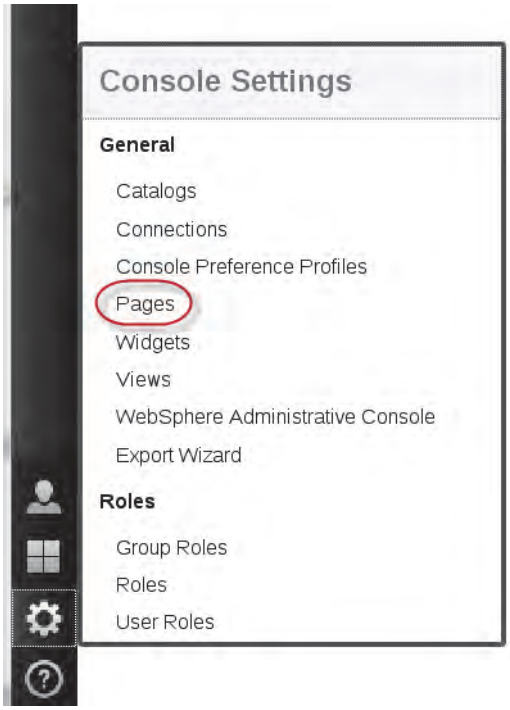
You use the following template rule in the DASH analog gauge widget:

RegSumHigh: This rule calculates the total number of open High severity trouble tickets for a region of the company.

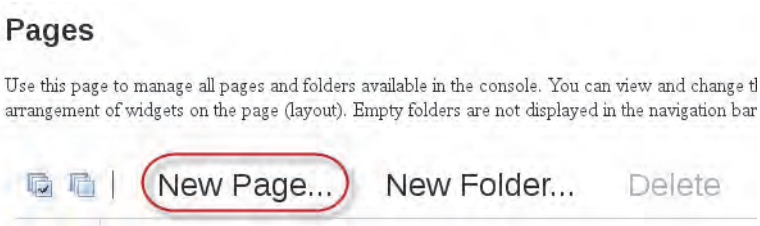
Configuring the second-level page settings

In this step, you start the page creation process for the second-level dashboard page. You assign the page name, select the page layout mode, and assign page-level access authorization.

1. Switch to the DASH console window or login with the user ID **smadmin** and the password **object00**.
2. Create and configure the settings for the second-level page.
   a. Select the **Console Settings > General > Pages** task in the taskbar.



   b. Click **New Page**.



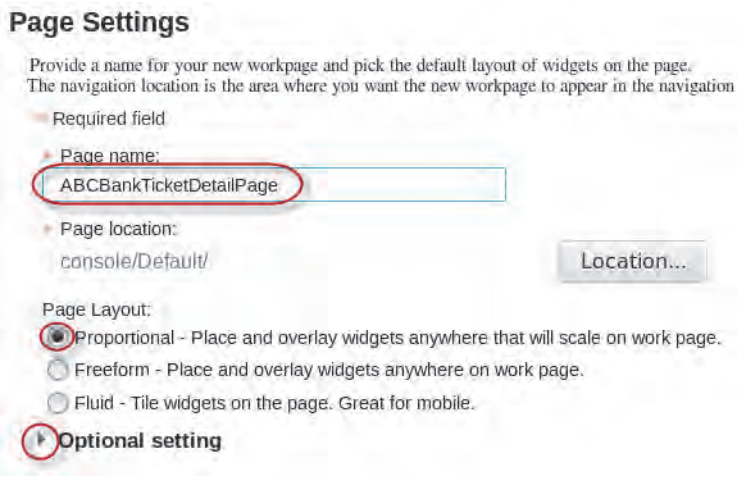
c. Assign the page name. Enter **ABCBankTicketDetailPage** in the **Page name** field.

d. Do not change **console/Default** in the **Page location** field.

e. Select the page layout mode.

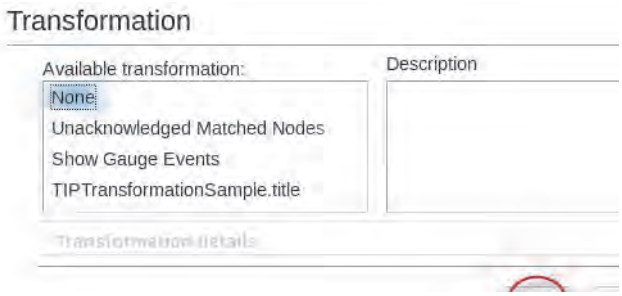
The second-level page is created with the *Proportional* page layout method. The proportional layout method controls the proportions and arrangement of canvas widgets on a page. This layout method is useful with the varying screen sizes and orientations of mobile devices.

f. Select **Proportional** in the Page Layout list and click **Optional setting**.

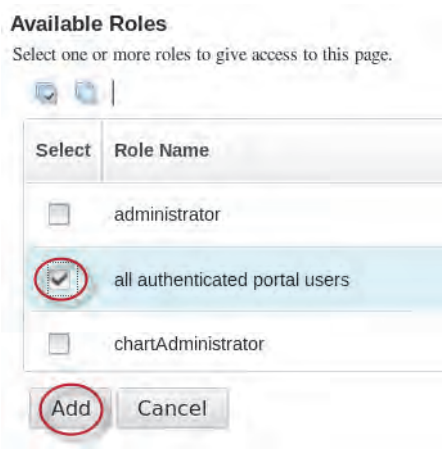


3. Assign authorization roles to the page. Only users or group members with matching assigned roles can access the page.

a. Click **Add** in the Optional setting section of the page.

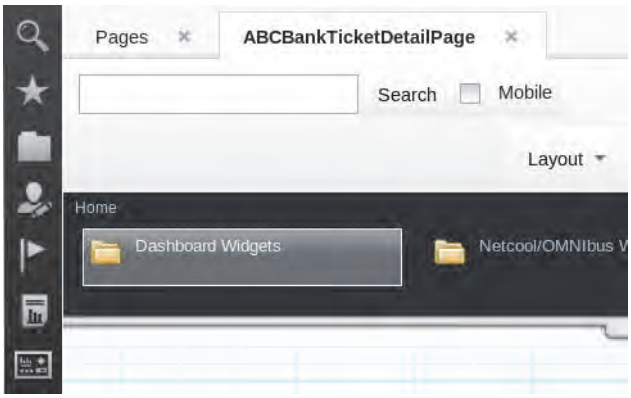


b. Select **all authenticated portal users** and click **Add**.



The selected role is shown in the **Role Name** column. The *Access Level* is an extra level of object access control. The User access level prevents a user from modifying page settings.

c. Use the default access level. Click **OK** to save the page settings and open the page editor workspace.



Adding widgets to the page

In this step, you add an analog gauge widget to the page. Figure 2 on page 1-12 shows a logical view of the second-level page.

The basic elements of the Jazz for Service Management page editor workspace are shown in Figure 3.

- The top section of the workspace contains page and widget control icons.
- The middle section is the widget ***palette***, which contains a horizontally arranged list of available widget objects.
- The lower section of the workspace is the page ***canvas***. You use this section to create, configure, and arrange dashboard widgets.

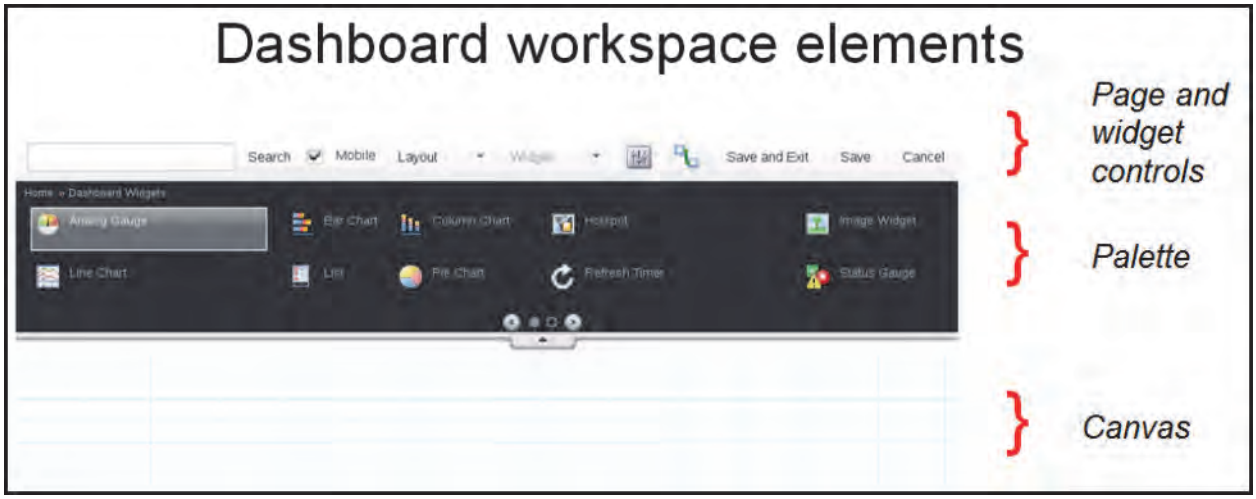


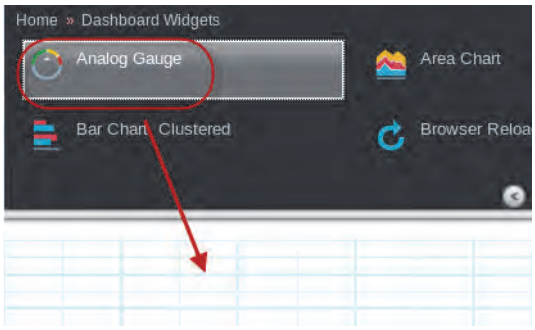
Figure 3 Dashboard workspace elements

You place widget instances on the canvas by clicking a widget icon in the palette and dragging it to the canvas. You arrange the canvas widgets by clicking and dragging the widget title bar. You resize canvas widgets by clicking the edge of a widget to activate the widget anchor points and dragging a widget corner. You collapse the palette view to proportionally increase the canvas space view.

The initial folder icons in the page palette section are referred to as widget *catalogs*. A catalog is a customizable container that is used to organize and group widgets.

4. Click **Dashboard Widgets** in the page palette to show a list of widgets that are used in these exercises.

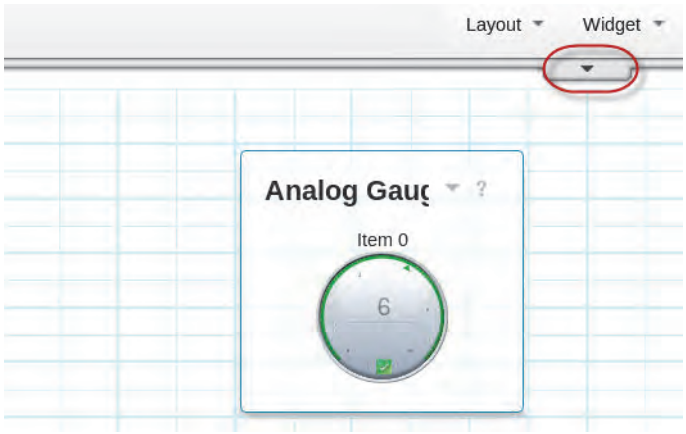
5. Locate the Analog Gauge widget icon in the palette. The analog gauge widget is used to show a single value against a range of configured minimum and maximum values. Click the **Analog Gauge** icon in the palette and drag the icon to the canvas.



Configuring widget properties

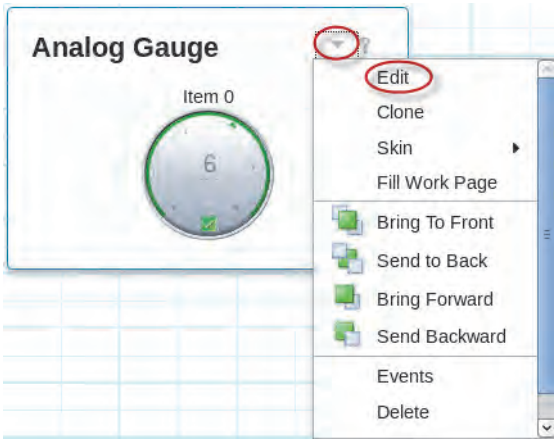
In this step, you configure the properties for the analog gauge widget on the page.

6. Close the widget palette. Click the **Hide Palette** icon in the top center of the canvas.



Hint: The canvas background grid is always 20 units wide by 20 units tall. Closing the palette does not change the grid unit count. Closing the palette changes the canvas proportions so that the widget properties and layout are easier to see.

7. Configure the first analog gauge widget to show the count of open high severity trouble tickets for a region of the ABCBank company.
   a. Click the **Edit options** icon in the upper right of the widget.
   b. Select the **Edit** menu.

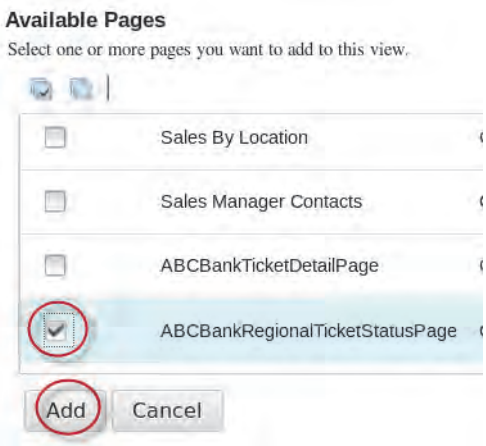


   c. Select the widget data set. Tivoli Business Service Manager provides the data with a template rule in the OSRegion template. Enter **OSRegion** in the **Select a Dataset** Search field and click **Search**.

Important: The console might initially indicate that no data sets are found. A short delay causes the error as Jazz for Service Management establishes a persistent connection with the data source. Click the Search icon again and the data sets are then listed. Alternatively, wait several seconds and the connection is automatically established. Subsequent search requests of the data source do not show any error messages.

The Tivoli Business Service Manager UI data provider is queried and a list of data sets is shown. OSRegion is listed in the *TBSM Primary Templates* and *TBSM Templates* data set. The difference between the two data sets is not important for this exercise.

d. Click **OSRegion** in the **TBSM Primary Templates** section. The order of the data sets might be different in your lab configuration.



The Visualization Settings form is shown.

e. The Tivoli Business Service Manager OSRegion template rule values are shown in the **Value** menu. The data set determines the values that are shown in the menu. Use the value that is calculated with the Tivoli Business Service Manager **RegSumHigh** rule. You reviewed the list of template rules in the step, “Reviewing the Tivoli Business Service Manager service model” on page 1-13. Select **RegSumHigh** in the **Value** menu.

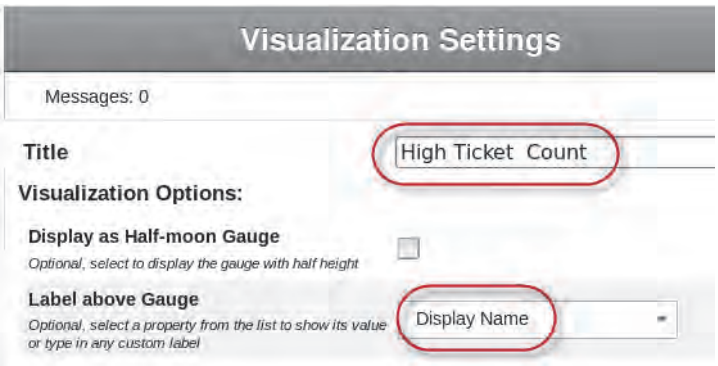
f. Configure the Optional Settings section of the form. Click **Optional Settings** to expand the section.

g. Configure how the RegSumHigh data is shown in the analog gauge widget. Enter **High Ticket Count** in the **Title** field.

h. The context for the analog gauge data is provided when a widget in the top-level page is clicked. The region name is also used as the Tivoli Business Service Manager **Display Name** attribute in the OSRegion template.

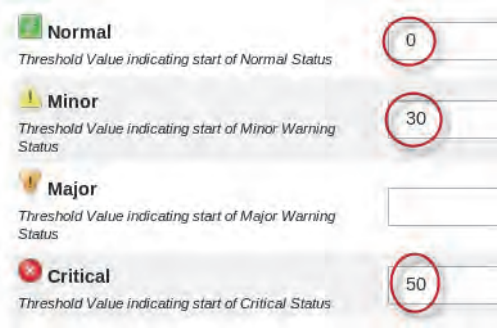
i. Select **Display Name** in the **Label above Gauge** menu.

j. Do not configure a value for the **Label at leading edge** menu.



k. Configure the threshold information for the gauge. The thresholds are shown with the widget to provide more visual information. Use the information in the following table to set the threshold values.

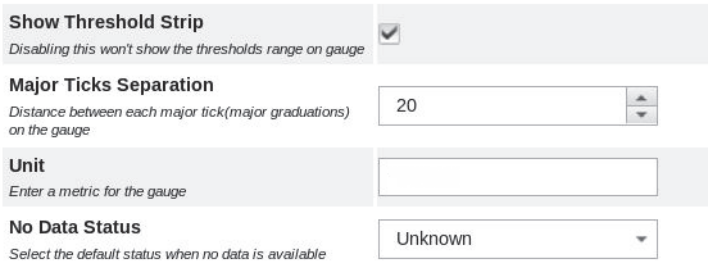
Threshold	Value
Normal	0
Minor	30
Critical	50



Hint: Technically, setting widget threshold values are optional. You do not have to specify a value in the threshold fields. For example, to configure the gauge to show only the current value, with no threshold indications, leave all the threshold values blank.

l. Configure the widget to show a color-coded threshold strip on the outer edge of the gauge. Select **Show Threshold Strip**.

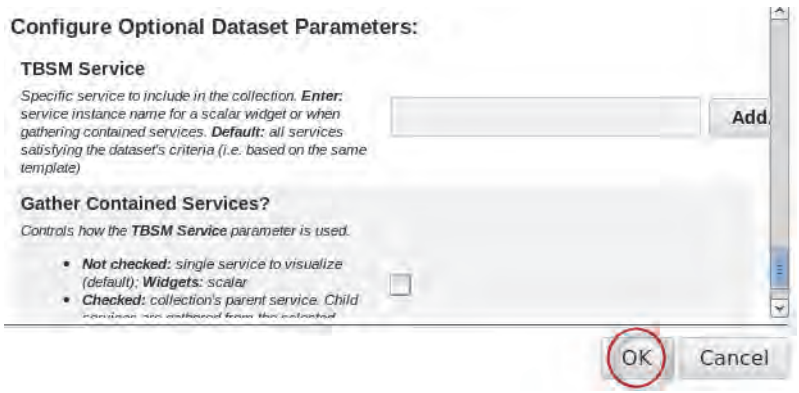
m. Use the default values for the **Minimum Value**, **Maximum Value**, **Major Ticks Separation**, Unit and **Minor Ticks Separation** fields. The RegSumHigh value corresponds to the calculated number of open high severity tickets in the region.



n. Use the default vales for the **No Data Status** and **Page to Launch** menus.

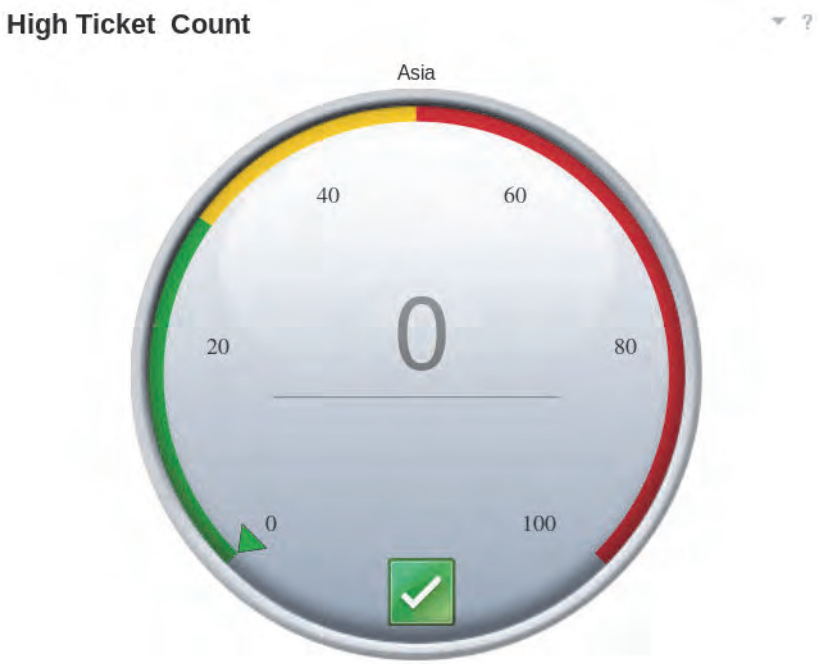
o. Do not change the default settings for the **Configure Optional Dataset Parameters** section. This section is optionally used to specify one or more values from the selected data set. The TBSM Service value is sent with a context event from another page.

p. Save the widget configuration. Click **OK**.



Because no service was specified in the **Configure Optional Dataset Parameters** section, the widget uses the first Tivoli Business Service Manager service that matches the configured parameters. In this laboratory environment, AsiaABCBank is the first business service match.

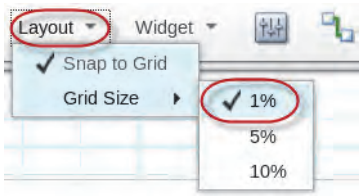
You configure the mechanism to pass contextual information to this widget in “Linking pages with wires” on page 1-30.



Arranging the widgets on the page

In this task, you arrange the widgets on the page to correspond to the logical design that is shown in Figure 2 on page 1-12. The page canvas background consists of a grid of rectangles, 20 units wide, and 20 units tall. You adjust how objects on the page snap to the grid with the **Layout** menu above the widget palette section.

1. Set the alignment to 1%. Click **Layout**, click **Grid Size**, and select > **1%**.



The default size for the analog gauge widgets is six grid units wide by ten grid units tall. Resize the gauge widgets so that the widget uses a 10x10 section of the canvas grid.

2. Move your mouse cursor over the lower right corner of the High Ticket Count widget until the cursor shape changes to a resize icon. Click the widget and the widget borders are highlighted with anchor points.

3. To change the size of a widget, click and drag an anchor point to the required dimensions. Alternatively, simultaneously hold the keyboard Shift key and use the keyboard cursor keys to change the widget size. The widget changes proportionally to fit the new size.

Hint: The Hide and Show Palette icon is at the top middle of the page canvas. The arrow symbol in the icon is aligned with the center line of the canvas grid. Use the icon to quickly find the center line of the grid.

4. Move the widget so that it is centered on the page canvas.
5. Configure the gauge widget to not show the widget frame. Click the **Edit Options** icon in the analog gauge widget and select **Skin > Transparent**.
6. Save the page configuration. Click **Save and Exit** in the menu bar above the palette.

Creating the top-level dashboard page

In this task, you create the top-level page for the ABCBank trouble ticket dashboard as shown in Figure 4. The page contains the following information:

- The status of the Asia region, which is based on the sum of open high and critical severity tickets
- The status of the Europe region, which is based on the sum of open high and critical severity tickets
- The status of the US region, which is based on the sum of open high and critical severity tickets

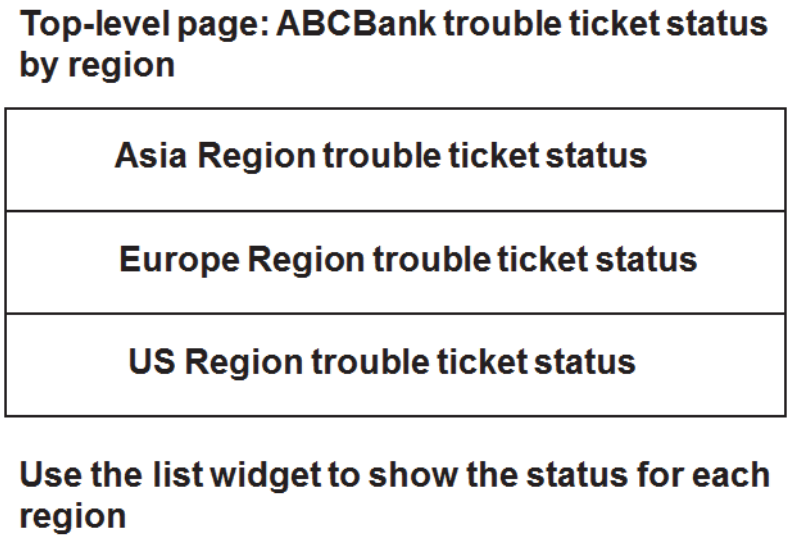


Figure 4 Top-level page diagram

This task consists of the following high-level steps:

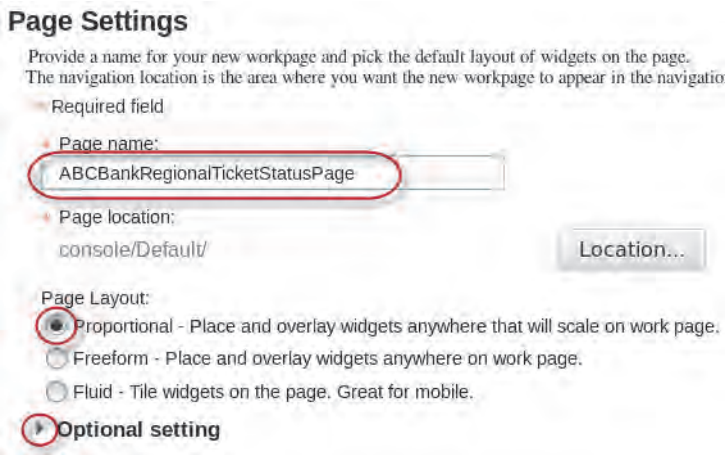
- **Configure the top-level page settings**: You configure the name, layout mode, and access authorization for the page.
- **Add widgets to the page**: You add the list widget to the page.
- **Configure widget properties**: You configure the properties for each of the widgets on the page.
- **Arrange the widgets on the page**: You set the size of the list widget on the page, as shown in the Figure 4 diagram.

Configuring the top-level page settings

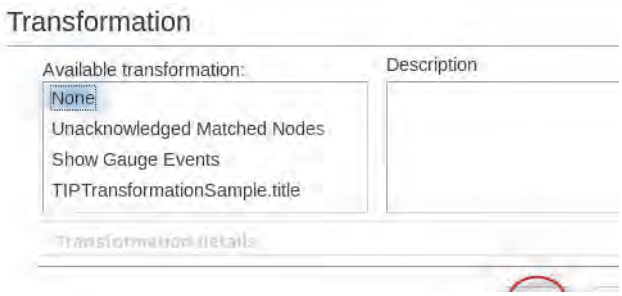
You can create a page in various ways. You can use the **Pages** task in the **Console Settings** taskbar folder, like you did in “Creating the second-level dashboard page” on page 1-12. You use a second method in this task.

1. Click the plus symbol (+) in the upper right of the console workspace.

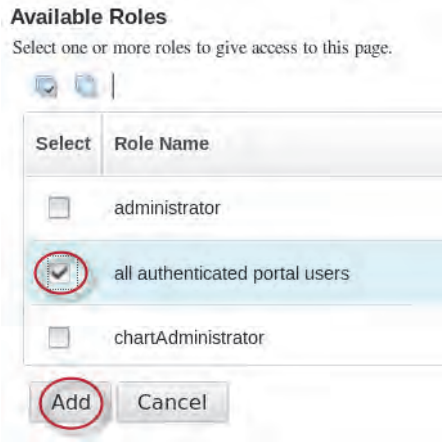
2. Assign the page name. Enter **ABCBankRegionalTicketStatusPage** in the **Page name** field.
3. Do not change **console/Default** in the **Page location** field.
4. Select the page layout mode. Select **Proportional** in the **Page Layout** list.



5. Assign an authorization role to the page. Only users or group members with a matching assigned role can access the page.
   a. Click **Optional setting** to expand the section.
   b. Click **Add**.



   c. Select **all authenticated portal users** and click **Add**.

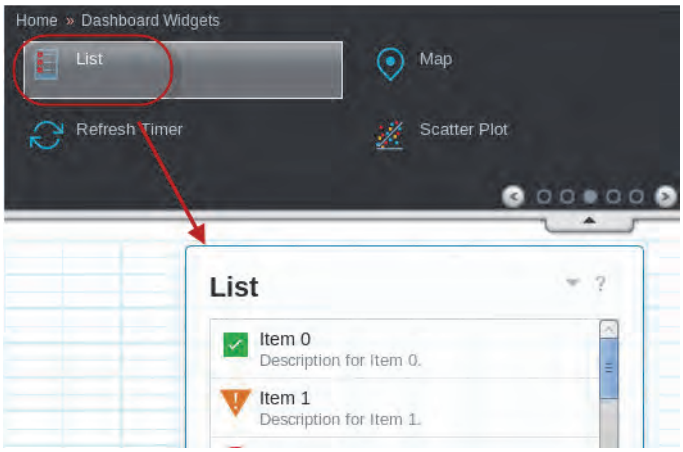


   The selected role is shown in the **Role Name** column.

   a. Use the default access level. Click **OK** to save the page settings and open the page editor workspace.

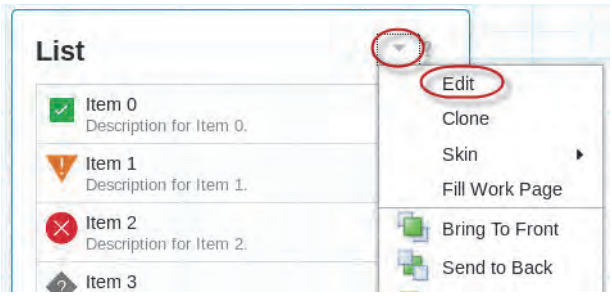
Adding widgets to the page

1. Add the List widget to the canvas.
   a. Click **Dashboard Widgets** in the page palette.
   b. Click the **List** icon in the palette and drag the icon to the page canvas.

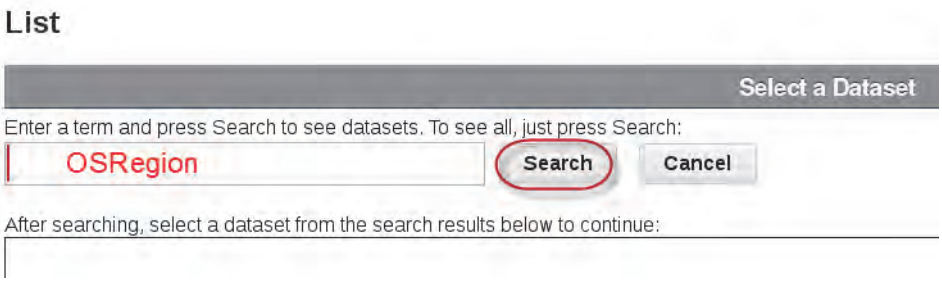


Configuring widget properties

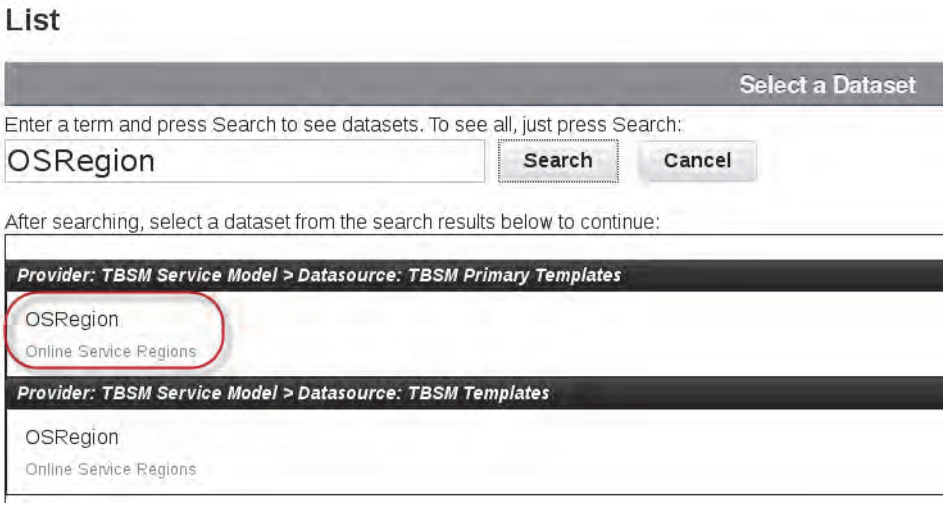
1. Configure the list widget to show the overall trouble ticket status for each geographic region in the ABCBank company. Click the **Edit options** icon in the upper right of the list widget and select **Edit**.



2. Select the widget data set. Use the Tivoli Business Service Manager OSRegion template.
   a. Enter **OSRegion** in the **Select a Dataset** search field and click **Search**.



b. Click **OSRegion** in the **TBSM Primary Templates** section.

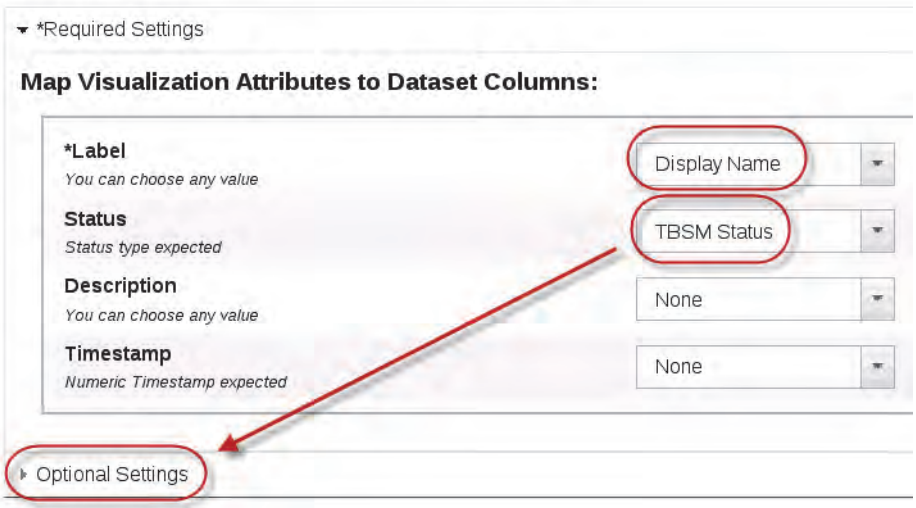


3. Configure how the provided data set is shown in the list widget. The Display Name is a Tivoli Business Service Manager service template attribute that corresponds to the geographic region. Select **Display Name** in the **Label** menu in the **Map Visualization Attributes to Dataset Columns** section.

4. Configure the value that is used to evaluate the status of the trouble ticket system in a region. For this exercise, use the Tivoli Business Service Manager status evaluation value. Select **TBSM Status** in the **Status** menu.

5. Use the default values for the **Description** and **Timestamp** menus.

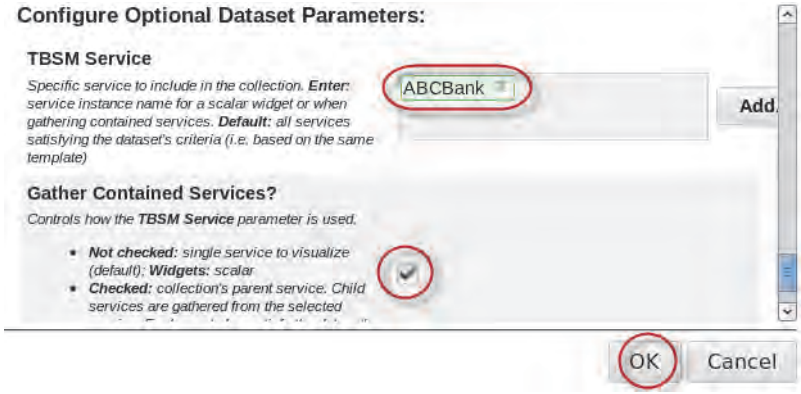
6. Configure optional settings for the List widget. Click **Optional Settings** to expand the section.

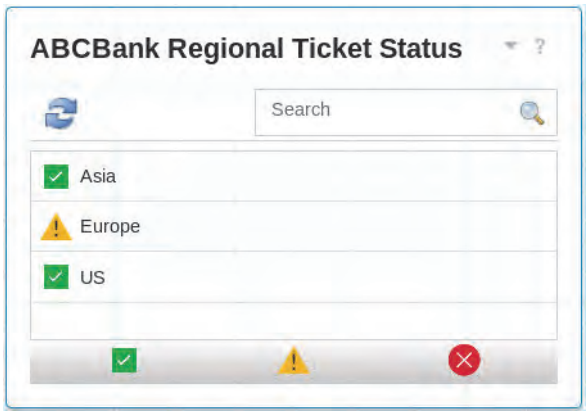


7. Add a title to the List widget. Enter **ABCBank Regional Ticket Status** in the **Title** field.

8. Use the default values for the **Group-by-Attribute**, **Sort-by-Attribute**, **Sort Order**, **Page Size**, **Automatic Refresh**, and **Page to Launch** options.

9. Complete the **Configure Optional Dataset Parameters** section. This section is optionally used to specify a Tivoli Business Service Manager service or a subset of services. For this exercise, you configure the widget to use the OSRegion child services (AsiaABCBank, EuropeABCBank, and USABCBank) that are configured for the ABCBank parent service. Enter **ABCBank** in the **TBSM Service** field.
10. Select the **Gather Contained Services** option. This option passes all matching OSRegion child services of the ABCBank service model to the list widget.
11. Save the list widget configuration. Click **OK**.

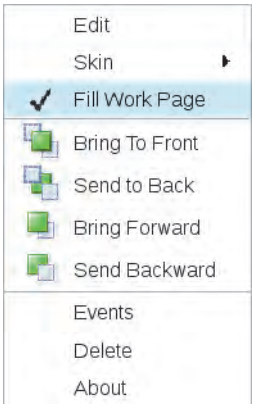




Arranging the widgets on the page

For this task, you set the list widget to fill the top-level page space. The configuration matches the design that is shown in Figure 4 on page 1-24.

12. Configure the list widget to fill the available page canvas. Click the widget **Edit Options** icon and select **Fill Work Page**.



13. Save, but do not close the page. You save the page after you configure the navigational event wire in “Linking pages with wires” on page 1-30. Click **Save** in the widget menus above the widget palette.

Linking pages with wires

In this task, you create an event wire that sends NodeClickedOn events from the top-level page to the second-level page. The wire is configured to deliver context information to the second-level page, not to specific widgets on the second-level page. Events that are delivered at the page level are automatically sent to all widgets on the target page. You create and configure only one wire with this method.

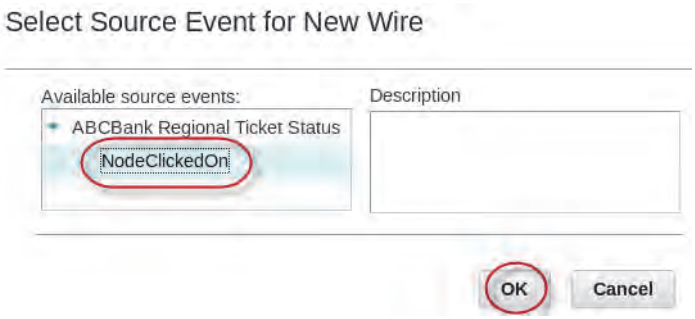
1. Click the **Show Wires** icon in the page and widget control menu bar in the upper right of the page.

The **Summary of wires** table is shown at the top of the page. By default, no wires are defined. You configure the event wire to send a NodeClickedOn event from the selected entry in the list

widget in the ABCBankRegionalTicketStatusPage page to the widgets in the ABCBankTicketDetailPage page.

a. Click **New Wire** in the **Summary of wires** table.

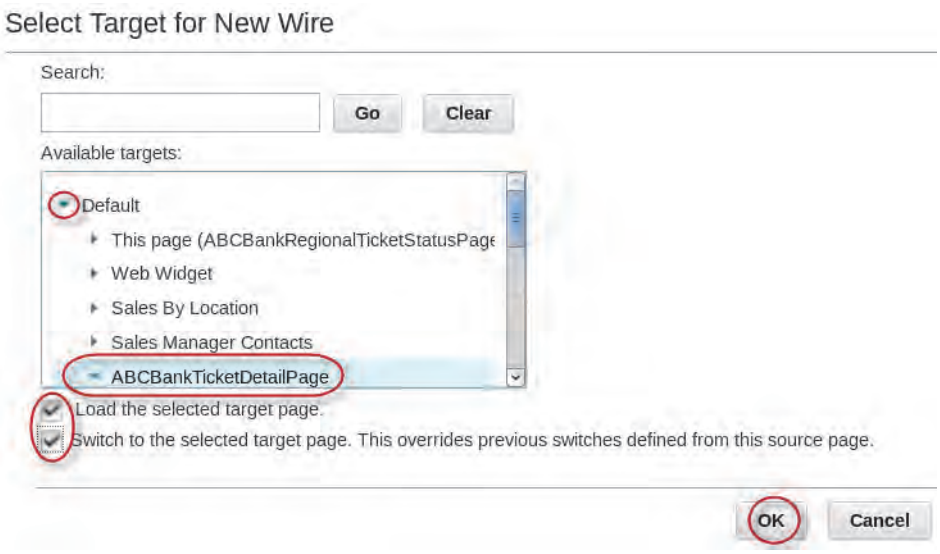
b. Select the source event for the wire. Click **ABCBank RegionalTicket Status** in the **Available source events** column, select **NodeClickedOn**, and click **OK**.



c. Select the event wire target. Send the NodeClickedOn event to all widgets in the second-level page. Click **Default** and select **ABCBankTicketDetailPage**.

Important: Do not select a single widget as the wire target. The page is selected to send the source event to all widgets in the page. You select a specific widget only if you require the source event to apply to only that widget.

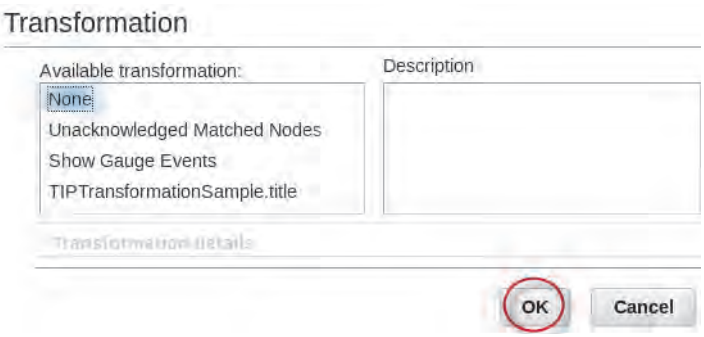
d. Configure the wire to automatically open and show the target page. Select **Load the selected target page**, **Switch to the selected target page**, and click **OK**.



e. Do not select an event transformation option. Click **None** in the **Available transformation** column.

Note: Transformations are used to intercept and modify a source event before delivery to a target object. You do not configure transformations in this laboratory.

f. Complete the event wire configuration. Click **OK**.

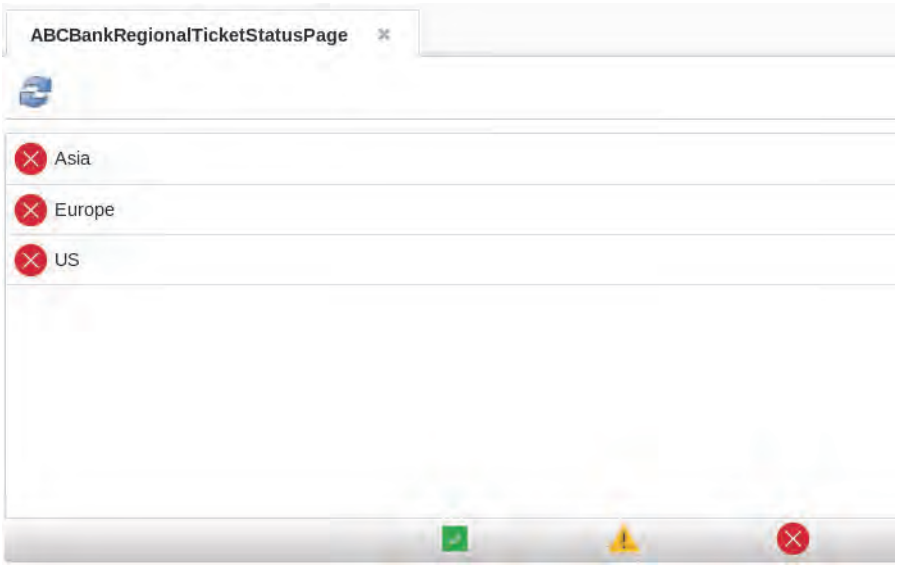


The event wire configuration is shown in the **Summary of wires** table.

2. Close the **Summary of wires** table. Click the Close Table (**X**) icon in the upper right of the table. **Do not click the X icon that closes the browser**.

3. Save the **ABCBankRegionalTicketStatusPage** page. Click **Save and Exit** in the page and widget control section, above the page palette section.

   The final page looks like the following image.



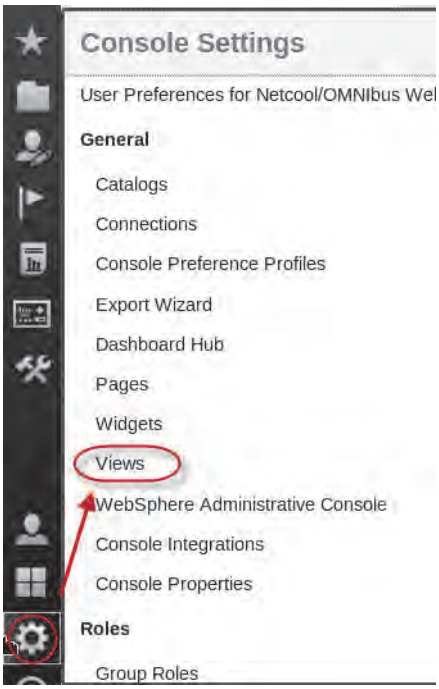
Creating views

The Jazz for Service Manager visualization services includes a tool to organize pages and folders into lists, called ***Views***. Authorization roles control access to views and to pages within a view. In this exercise, you create a view that is called mobileDashboardUsersView. Mobile devices can show dashboard pages only in views that are configured for mobile access.

You include only the top-level page in the view. You restrict access to the second-level page so that the page is seen only when a user selects a list entry in the top-level page. To configure the view, you add the top-level page to the view and assign the mobileDashboardUsersRole authorization role to control access to the view. You assigned page-level authorization when you created each

dashboard page. The mobileDashboardUsersRole was created during the configuration of the laboratory images.

1. Create the **mobileDashboardUsersView** view. Click **Console Settings > General > Views** in the taskbar on the left of the console.

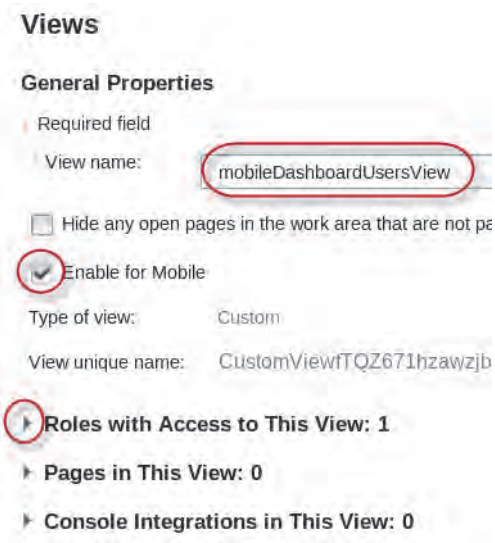


2. Start the view creation process. Click **New**.

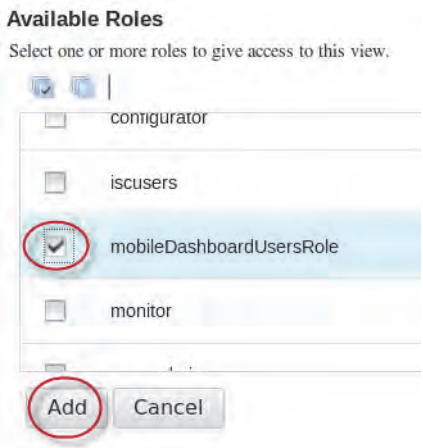
3. Enter **mobileDashboardUsersView** in the **View name** field.
4. Configure the view for mobile access. Select **Enable for Mobile**.

Important: If you do not select **Enable for Mobile**, mobile devices cannot view any pages in the view.

5. Assign authorization roles to the view. Only users or group members with the matching assigned role can access the view. Expand the **Roles with Access to This View** section.



6. Add a role to control view access. Click **Add**.
7. Scroll through the list of available roles and select **mobileDashboardUsersRole**. Click **Add**.

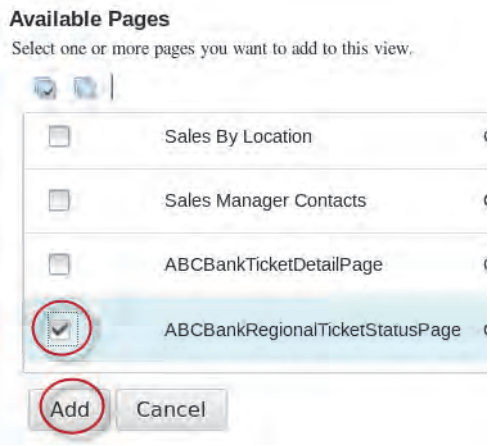


8. Add the **ABCBankRegionalTicketStatusPage** page to the view.
   a. Expand the **Pages in This View** section.
   b. Click **Add**.

c. Expand the **Default** folder

d. Select **ABCBankRegionalTicketStatusPage**.

e. Click **Add** at the bottom of the form.

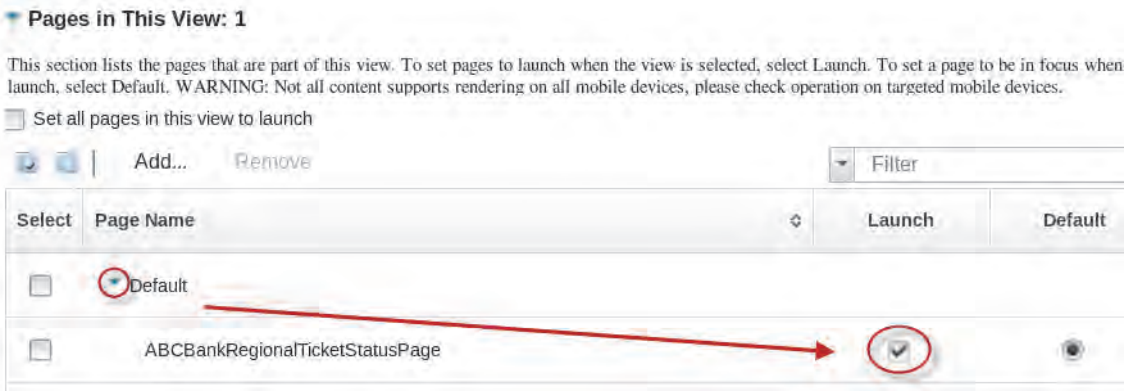


9. Configure the view to automatically open the **ABCBankRegionalTicketStatusPage** page when the view is opened.

   a. Scroll down in the form to see the **Pages in This View** section.

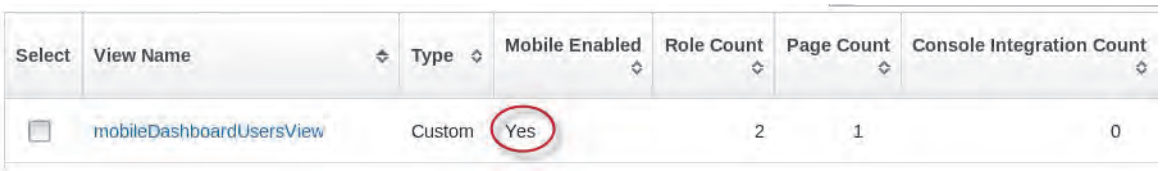
   b. Expand the **Default** folder.

   c. Select the option in the **Launch** column. With only one page in the view, **Default** is automatically selected.



10. Save the view configuration. Click **Save** at the bottom of the page.

The view is shown in the view list. The Mobile Enabled field must be set to **Yes**.



Creating console preference profiles

In this task, you create a console preference profile to control which views and console navigation options are available to users who are assigned the mobileDashboardUsersRole role. You configure the profile to use the Tivoli Dark console color theme. The Tivoli Dark theme shows dark console backgrounds with white text.

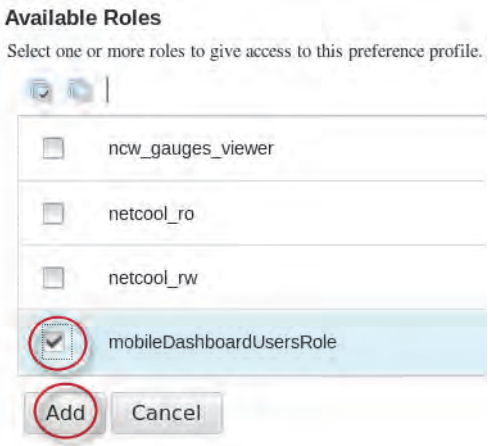
1. Create the **mobileDashboardUsersProfile** profile. Click **Console Settings > General > Console Preference Profiles** in the taskbar on the left of the console.
2. Click **New**.
3. Complete the General Properties section of the form.
   a. Enter **mobileDashboardUsersProfile** in the **Preference profile name** field.
   b. Configure the profile to use a color theme with dark backgrounds and white text. Select **IBM Design** in the **Theme** menu.

Note: The IBM Design theme is configured to work well with mobile devices.

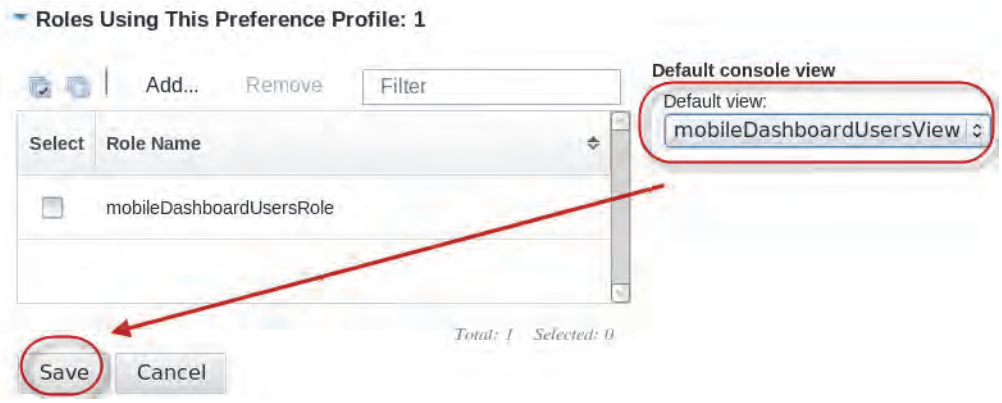
   c. Use default values for all other fields and selections in **General Properties** section.

4. Assign the **mobileDashboardUsersRole** authorization role to the profile. Adding this role ensures that only users that are assigned the role can use this profile.
   a. Click **Roles Using This Preference Profile** to expand the form section.
   b. Click **Add**.

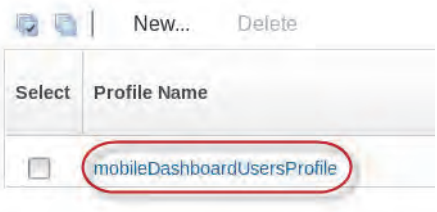
   c. Scroll down in the list, select **mobileDashboardUsersRole**, and click **Add** to complete the role selection.



5. Scroll down in the form to see the expanded Roles Using This Preference Profile section. The mobileDashboardUsersRole entry is shown in the **Role Name** column. Configure the profile to use the view you created in “Creating views” on page 1-33.
   a. Select **mobileDashboardUsersView** in the **Default console view** menu.
   b. Click **Save** to complete the profile configuration.



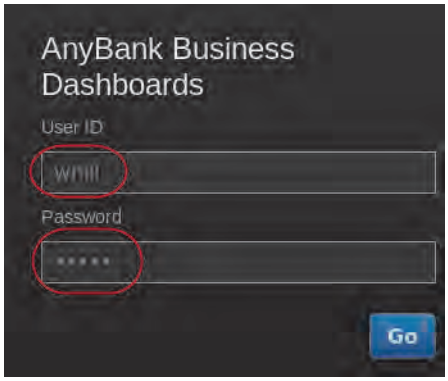
The completed profile is listed in the **Profile Name** column.



Testing the dashboard design

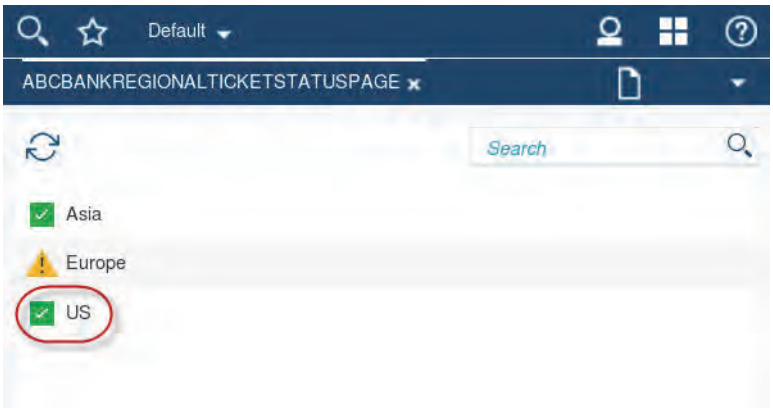
In this task, you log on to the Jazz for Service Management console with a user ID that is configured to connect to the console with a mobile device. The user ID, **whill**, was assigned the **mobileDashboardUsersRole** authorization role during the laboratory setup.

1. Log off the Jazz for Service Management console. Click the person icon in the console taskbar and select the **Log out** menu.
2. Log in to the console with the user ID **whill** and the password **object00**.

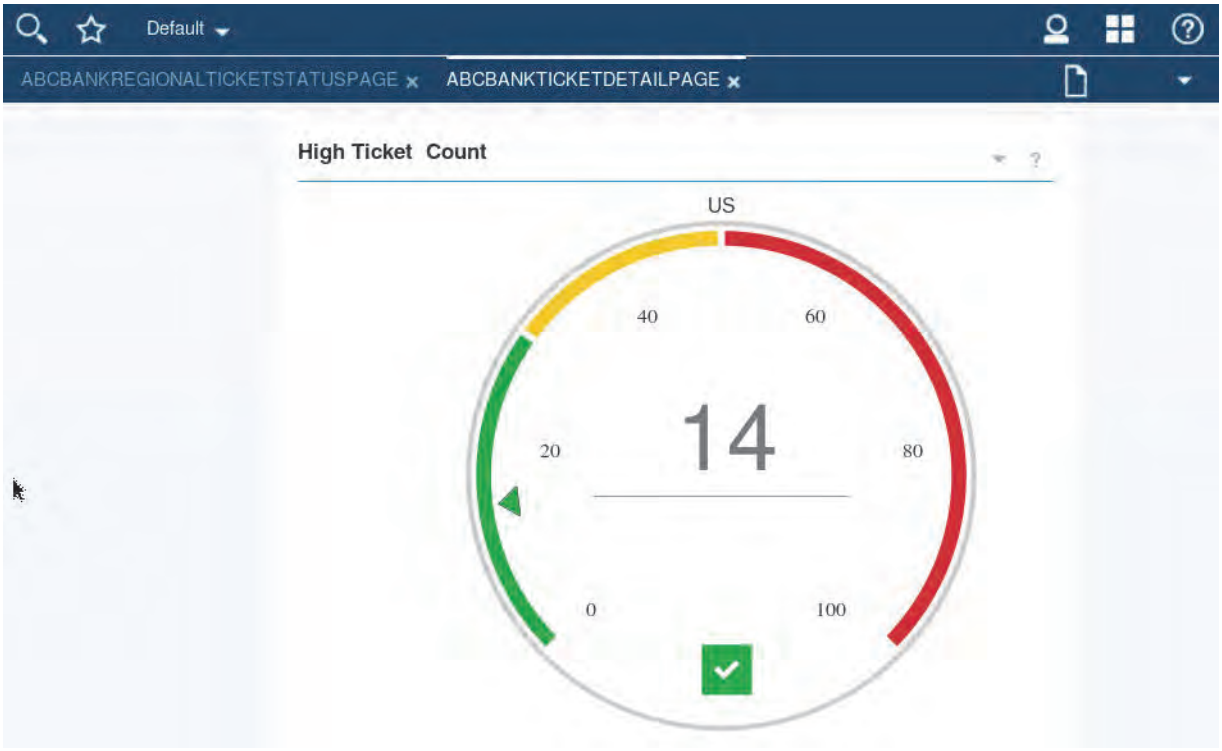


The **mobileDashboardUsersView** view that you created in “Creating views” on page 1-33 is automatically opened.

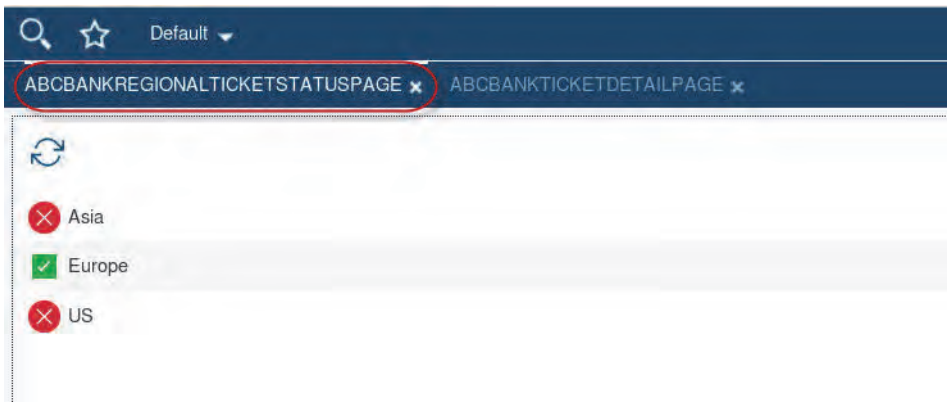
3. Test the navigational configuration between dashboard pages.
   a. Click **US** in the **ABCBankRegionalTicketStatusPage** page.



b. Verify that the second-level page is shown, with open ticket counts from the US region.



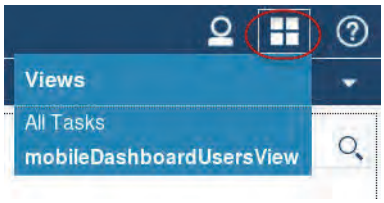
c. Switch to the top-level page. Click the page **ABCBANKREGIONALTICKETSTATUSPAGE** tab above the list widget.



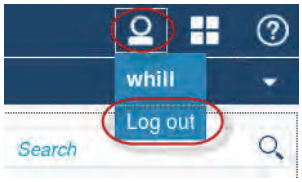
Note: Because the page theme is designed for mobile devices, the behavior is slightly different when used with a desktop browser. When connected with a mobile device browser, the page shows an arrow icon to back out from the detail page to the status page.

d. Click the **Europe** and **Asia** regions in the top-level page. Verify that the second-level page shows open ticket counts for the selected region.

e. Click the View icon to see the list of available views. In a typical production deployment, more pages and views would be configured.



4. Log off the console. Click the person icon at the top right of the console page and select **Log out**.



Exercise 4 Providing custom dashboard content with ContentBox

In this exercise, you add web content files to the myBox web server object in the dash151 virtual image. The myBox web server provides a URL-addressable path to your graphics files on the DASH server. You use several of the graphics in subsequent exercises.

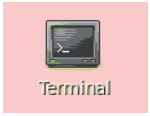
Modifying the myBox web server requires completing two tasks:

- Adding the web content files to the myBox web server directory
- Update the myBox web server

The myBox web server is included with the application of Jazz for Service Management Version 1.1 Fix Pack 3. The default installation directory is **/opt/IBM/JazzSM/ui/myBox**. Non-secure web content is placed under the **/opt/IBM/JazzSM/ui/myBox/web_files** directory. Secure web content is placed under the **/opt/IBM/JazzSM/ui/myBox/web_files/secure** directory. Secure web content is only viewed by authenticated DASH server users.

Adding web content to the myBox web server

1. Minimize the web browser and open a command window on the dash151 virtual image. Double-click the **Terminal** icon on the left of the console desktop.



2. Add the web content files to the non-secure directory in the myBox container. The files that are to be added are stored in the **/opt/labfiles/dash/myBox** directory on the dash151 virtual image during the image configuration. The files are organized in four subdirectories: **icons**, **backgrounds**, **jsp**, and **maps**. Enter the following commands:

```
cd /opt/IBM/JazzSM/ui/myBox/web_files
cp -R /opt/labfiles/dash/myBox/* .
```

Important: You must include a space between the asterisk and the period.

Updating the myBox web server

1. Update the myBox web server to include the new data content. Enter the following command:

```
/opt/IBM/JazzSM/ui/myBox/deployMyBox.sh -v -username smadmin -password object00
```

The script can take up to 5 minutes to complete, but it does not require a manual restart of the dash server. When the script is completed, you see the message, "**The operation completed successfully**".

```
I: The application isc is configured in the WebSphere Application Server reposit
ory. ADMA5005I: The application isc is configured in the WebSphere Application S
erver repository. ADMA5113I: Activation plan created successfully. ADMA5005I: Th
e application isc is configured in the WebSphere Application Server repository.
ADMA5011I: The cleanup of the temp directory for application isc is complete. AD
MA5079I: Update of isc has ended. The application or its web modules may require
 a restart when a save is performed. SUCCESSFUL
The operation completed successfully.
[root@dash151 myBox]#
```

2. Verify that the images are accessible from a browser. Because the files were added to the non-secure directory of the web server, you do not need to log on to the DASH console.
   a. Open a new browser tab. Restore the browser and select the **File > New Tab** menu.
   b. Enter the following URL:
   http://dash151.poc.ibm.com:16310/myBox/backgrounds/AnyCorpLogo.png

3. Close the browser tab.

Exercise 5 Customizing the console log on page

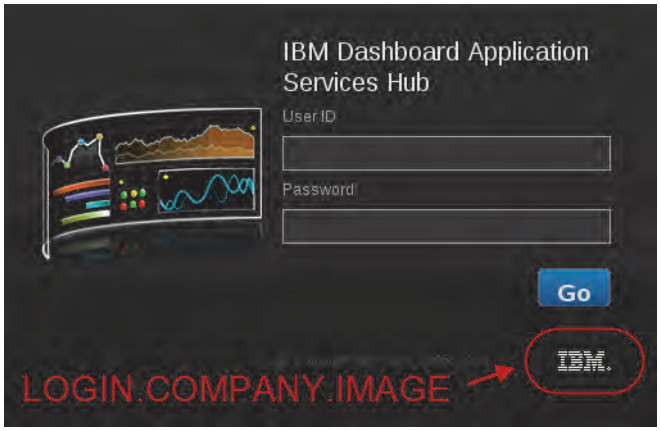
In this exercise, you customize elements of the Jazz for Service Management console logon screen. The customizations are all controlled with parameters in the **/opt/IBM/JazzSM/profile/config/cells/JazzSMNode01Cell/applications/isc.ear/deployments/isc/isclite.war/WEB-INF/consoleProperties.xml** file. Starting with Jazz for Service Management version 1.1.0.3, modifications to the customization parameters are done with a task in the DASH console. You use custom images that are served with the custom ContentBox module that you created in Exercise 4, “Providing custom dashboard content with ContentBox”.

For this exercise, you complete the following four tasks:

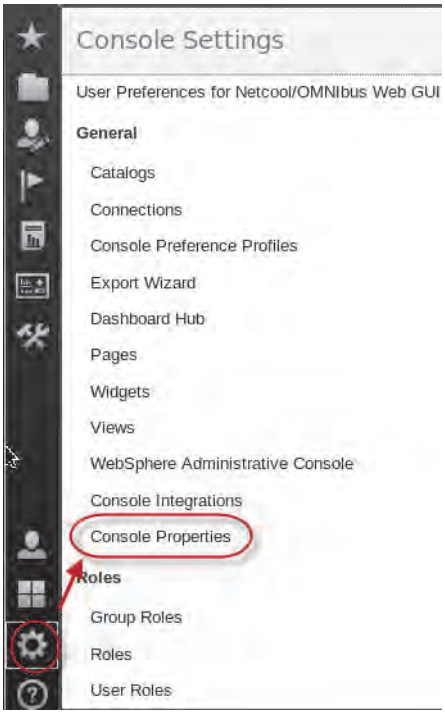
- Customize the logon page logo image
- Customize the main logon page image
- Customize the logon title
- Stop and start the Jazz for Service Management server

Customizing the login company image

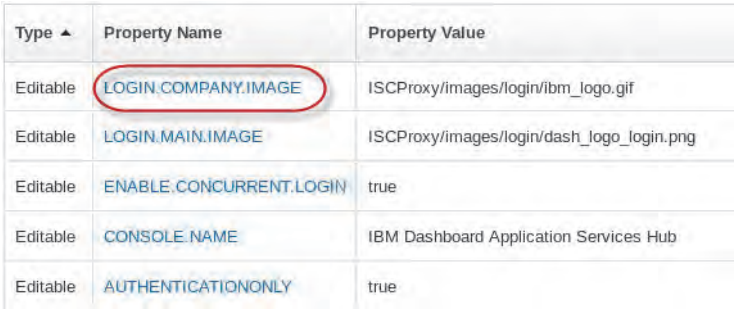
To customize the main login image, complete the steps for this task. The main logon image is shown in the following screen capture.



1. Edit the Jazz for Service Management console properties. Select the **Console Settings > General > Console Properties** task.



2. Click **LOGIN.COMPANY.IMAGE** in the Property Name column.



Type ▲	Property Name	Property Value
Editable	LOGIN.COMPANY.IMAGE	ISCProxy/images/login/ibm_logo.gif
Editable	LOGIN.MAIN.IMAGE	ISCProxy/images/login/dash_logo_login.png
Editable	ENABLE.CONCURRENT.LOGIN	true
Editable	CONSOLE.NAME	IBM Dashboard Application Services Hub
Editable	AUTHENTICATIONONLY	true

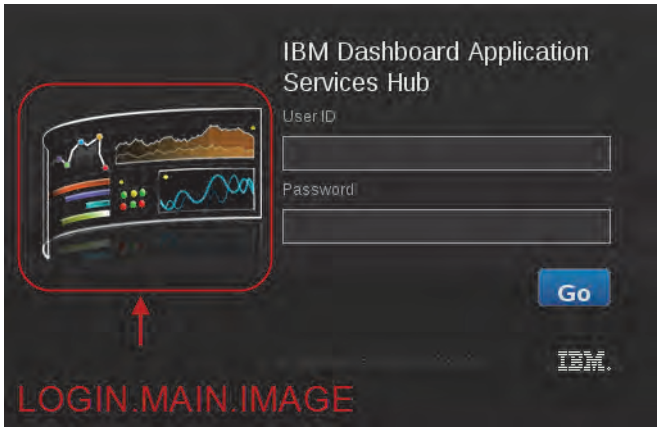
3. A window pops up with the current setting. Replace the current string with **/myBox/icons/AnyCorp.png** and click **Save**.

Important: If the property window shows an indicator that the value is undefined, you must log off the console, then stop and start the DASH server, by using the task icons on the virtual image desktop.

Instructor: Be sure to emphasize...

Customizing the login main image

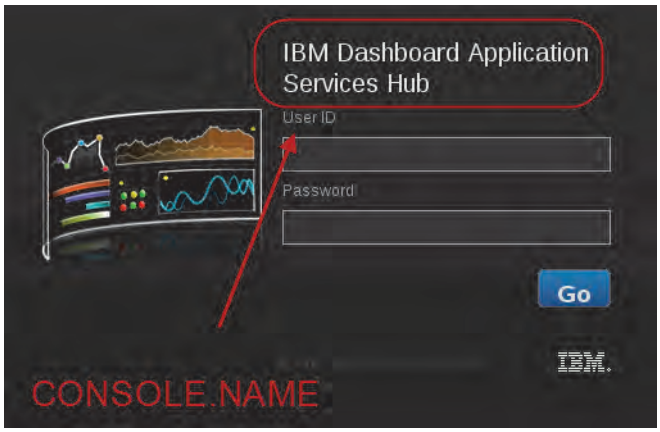
To customize the login main image, complete the steps for this task. The login main image is shown in the following screen capture.



1. Click **LOGIN.MAIN.IMAGE** in the Property Name column of the Console Properties list.
2. Replace the current string with **/myBox/icons/AnyBank_logo.png** and click **Save**.

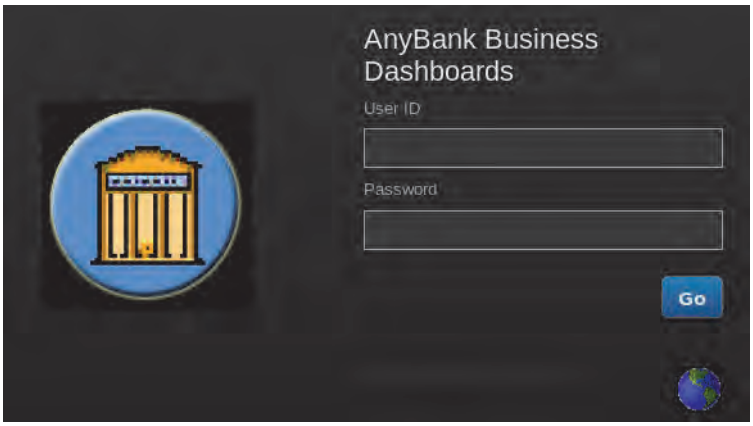
Customizing the login page title

Use the steps in this section to customize the login page title.



1. Click **CONSOLE.NAME** in the Parameter Name column of the Console Properties list.
2. Replace the current string with **AnyBank Business Dashboards** and click **Save**.
3. Log off the console. Click the **Person** icon in the console taskbar and select **Log out**.
4. Verify that the customizations are in effect for the console login screen. Restart the web browser and select the **Bookmarks > DASH Console** menu.

The results are like the following screen capture.



Note: If you do not see the changes, stop and start the DASH server. Use the **Stop All TBSM Services** and **Start All TBSM Services** at the left of the virtual image desktop.

Unit 2 Tivoli Directory Integrator and DASH basics exercises

In these exercises, you learn the basic skills to use Tivoli Directory Integrator to extract data from simple data sources (a database, a CSV file). You create a UI data provider to show database and CSV file data in a DASH widget. You also learn how to create and debug Tivoli Directory Integrator projects and assembly lines with the Configuration Editor (CE).

Exercise 1 Creating a UI data provider data set from a database

In this exercise, you create a Tivoli Directory Integrator project that is used as a UI data provider data source to the DASH server. You then add an assembly line to your Tivoli Directory Integrator project that reads data from a database. Finally, you add the project to the Tivoli Directory Integrator runtime server and then use the assembly line UI data provider data set to show the database data in a chart widget in a DASH dashboard page. The following screen image shows the database structure and data.

Data Type Name: SALESDATA

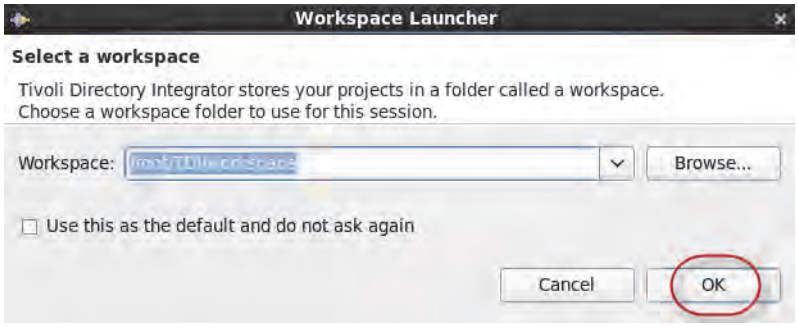
Number of Objects: 5

Select:(all)	WEEKDATE (key)	VANS	SEDANS	PICKUPS	Links
☐	2012-10-05	5	10	8	
☐	2012-10-12	3	8	11	
☐	2012-10-19	4	7	10	
☐	2012-10-26	5	8	7	
☐	2012-11-02	2	9	8	
Delete					

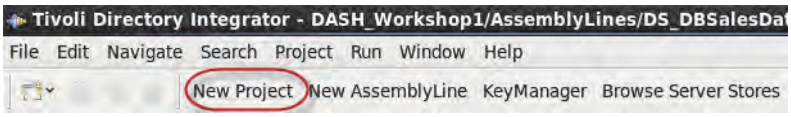
1. Open the Tivoli Directory Integrator Configuration Editor (CE). Double-click **Start TDI Editor** at the left of the dash151 virtual image desktop.

Note: The task launcher icon starts a script that issues the command **/opt/IBM/TDI/ibmditk**.

2. Click **OK** to confirm the default location of the project configuration files.

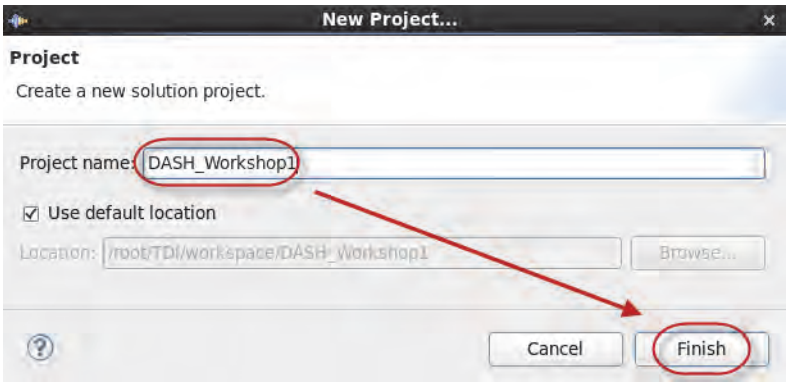


3. Create a Tivoli Directory Integrator project.
   a. Click **New Project** above the Navigator section of the console.

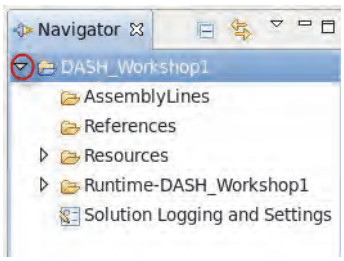


   b. Enter **DASH_Workshop1** in the **Project Name** field and click **Finish**.

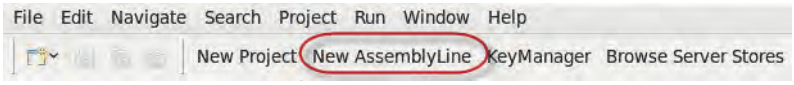
Important: You must prefix DASH project names with the string prefix **DASH_**. If you do not use this prefix, the DASH server does not recognize the project as a DASH data source.



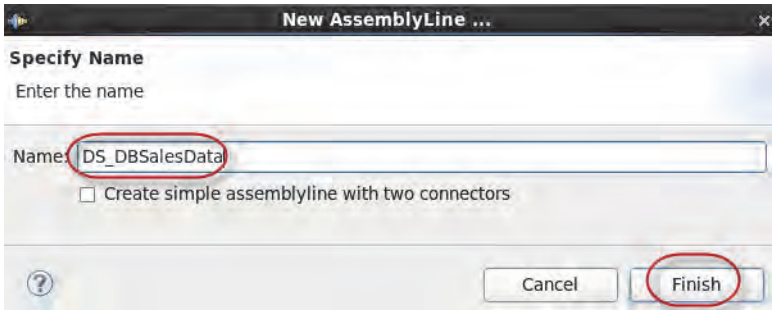
Default values and elements are automatically created for the project and shown in the Navigator section of the console.



4. Create an assembly line in the project. Click **New AssemblyLine** at the top of the configuration editor.

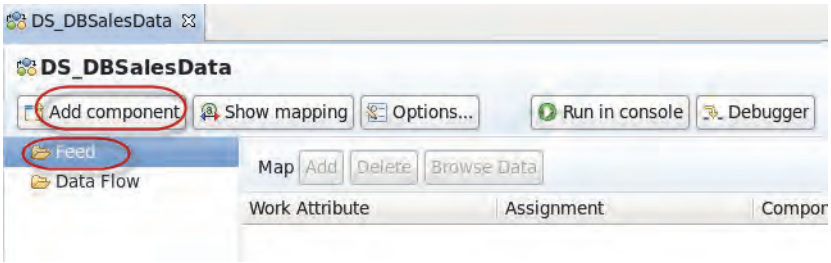


5. Enter **DS_DBSalesData** in the **Name** field and click **Finish**.



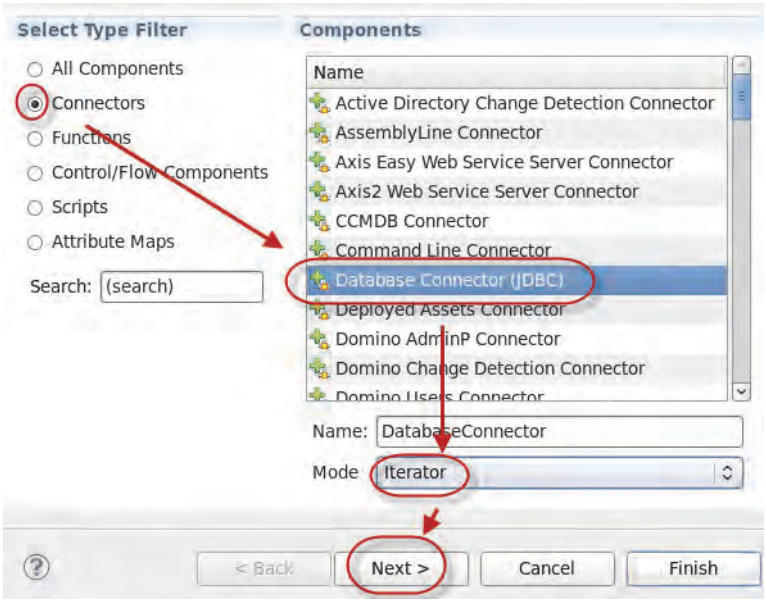
Important: You must prefix DASH assembly line names with the string prefix **DS_**. If you do not use this prefix, the DASH server does not recognize the project as a DASH data set.

6. You now configure the connection to the database data source. Select **Feed** in the workspace and click **Add component**.



7. You add and configure a database connector. Select **Connectors** to filter the Components list.
8. Click **Database Connector (JDBC)** in the Components list.

9. You configure this assembly line to iterate through the database table records. Select **Iterator** in the **Mode** menu and click **Next**.

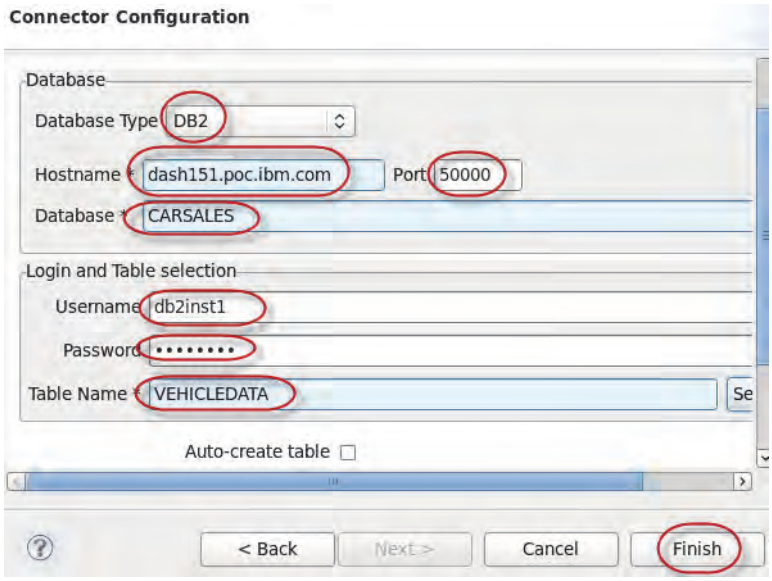


10. Use the following table to configure the database connector details.

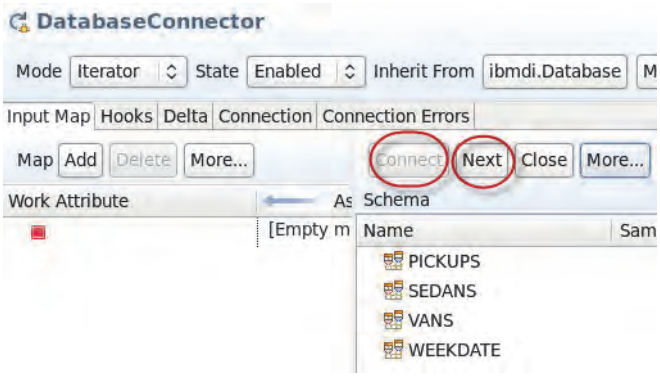
Table 1

Parameter Name	Value
Database Type	DB2
Hostname	dash151.poc.ibm.com
Port	50000
Database	CARSALES
Username	db2inst1
Password	object00
Table Name	VEHICLEDATA

11. Click **Finish** to complete the configuration.

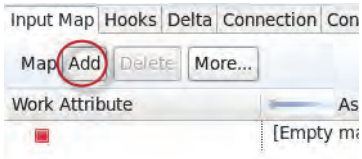


12. Click **Connect** to connect to the database and click **Next** to verify that the connector can iterate through the table.

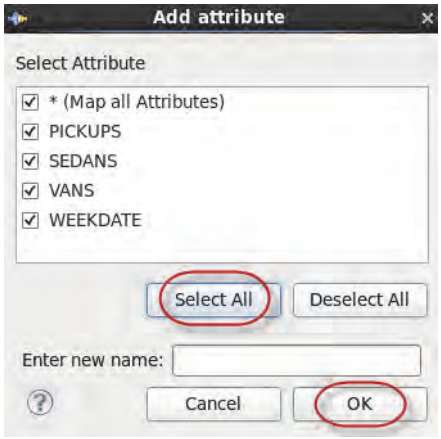


The connector automatically discovers the column data types from the database schema.

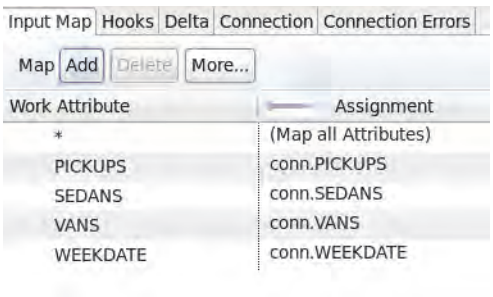
13. You need to map the data connector values to the work object. The work object is used as the output map, or data set, to the DASH server. Click **Add** to map the column values.



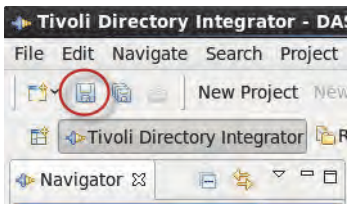
14. Click **Select All** and click **OK** to map all column values to the work object.



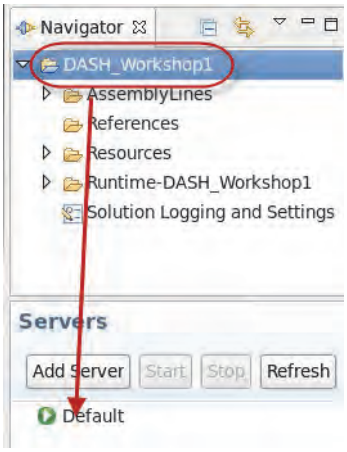
The mapped columns are automatically assigned variable names that can be used in JavaScript programs. For this simple assembly line, no further data enhancement is required.



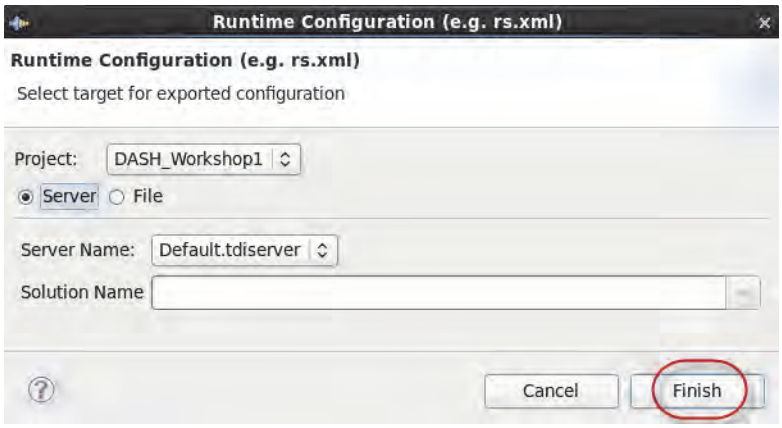
15. Save the project. Click the **Save** icon at the upper left of the configuration editor or select the **File > Save** menu.



16. Add the project to the Tivoli Directory Integrator runtime server. Drag **DASH_Workshop1** from the **Navigator** section onto the **Default** entry in the **Servers** section.

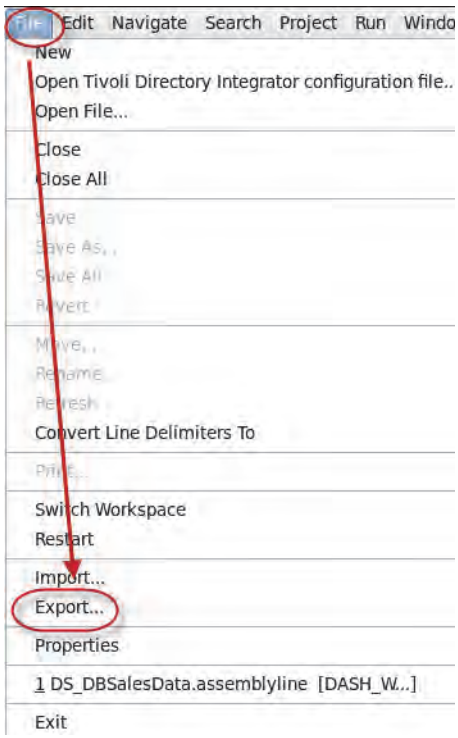


17. As the project is added to the server, a window pops up requesting a runtime configuration confirmation. Use the default values and click **Finish**.

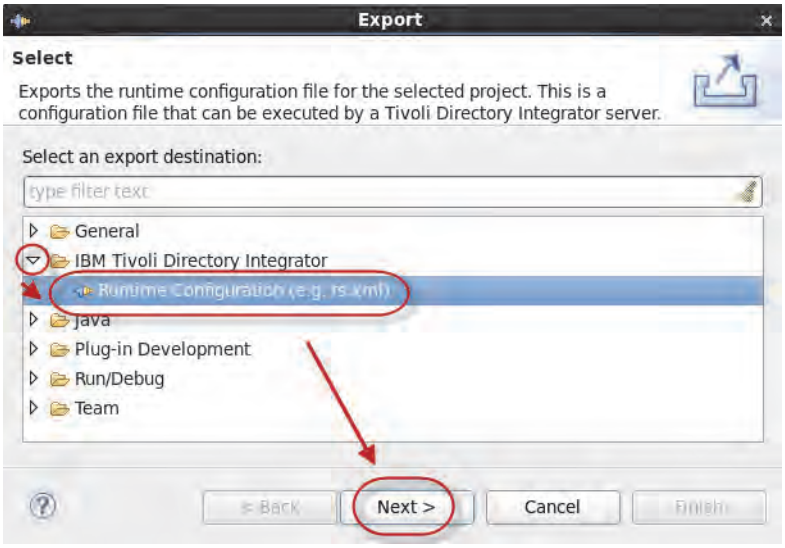


Hint: The drag mechanism does not always work. When working properly, the icon shape changes when the project name is selected and dragged. When the dragged object is moved over the Default entry, the row color shade changes. When the object is released, a confirmation window pops up. If none of those steps are seen, the drag action failed. If you are unable to drag the project to the Default server icon, use the following alternative method to add a project file to the runtime server:

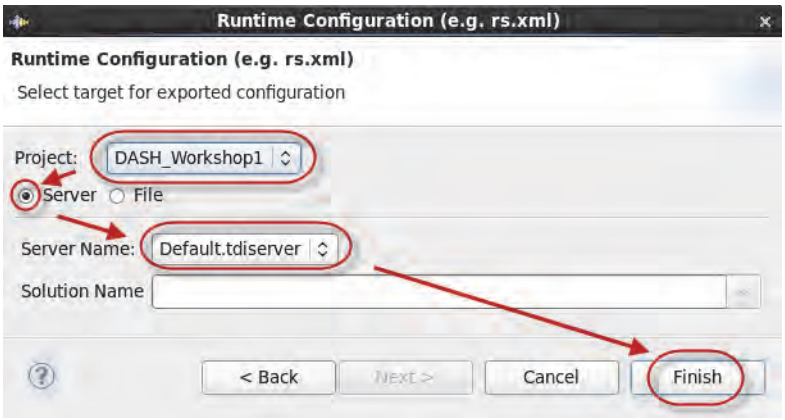
a. Select the **File > Export** menu.



b. Expand **IBM Tivoli Directory Integrator**, select **Runtime Configuration**, and click **Next**.



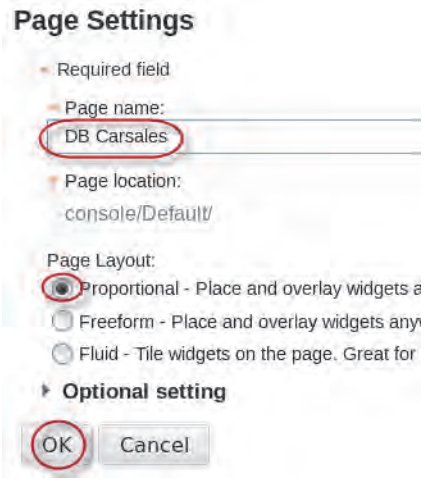
c. Select **DASH_Workshop1** in the **Project** menu, select **Server**, verify that the **Server Name** value is **Default.tdiserver**, and click **Finish**.



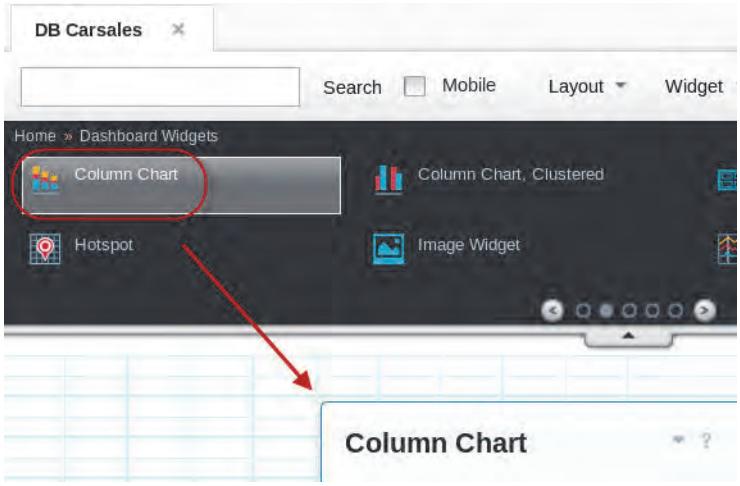
18. You are now ready to use the UI data provider and data set. Switch to the DASH console or login with the user ID **smadmin** and the password **object00**.

19. Create a dashboard page. Click the plus symbol (+) to open the page configuration.

20. Enter **DB Carsales** in the **Page name** field, select **Proportional** in the **Page Layout** section, and click **OK**.



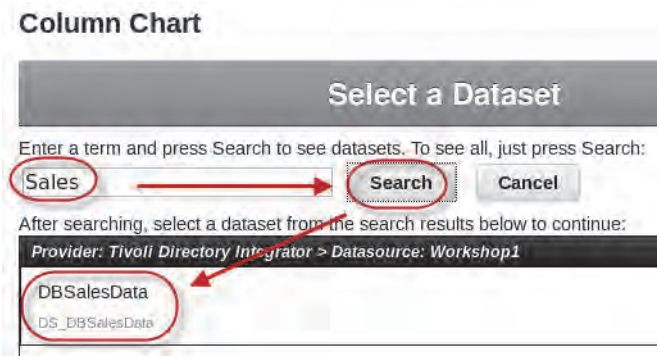
21. Click **Dashboard Widgets** in the widget palette and drag a **Column Chart** widget object to the page canvas.



22. Configure the Column Chart widget. Click the **Edit options** icon and select **Edit**.

23. The Tivoli Directory Integrator assembly line was named DS_DBSalesData. Enter **Sales** in the search field and click **Search**.

24. Click **DBSalesData** in the search result list.

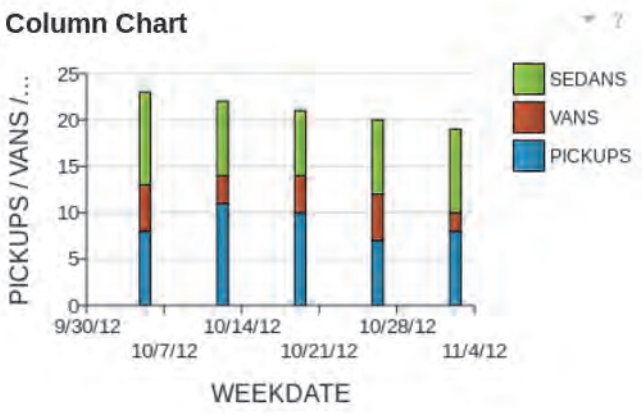


25. Configure the column chart widget. Select **WEEKDATE** for the **X-axis field** value.

26. Select **PICKUPS** for the first **Y-axis value field** value, click **Add**, then select **VANS** for the second value. Finally, click **Add** and select **SEDANS** for the third **Y-axis value field** value.

27. Scroll down in the configuration page and select **Side** in the **Legend** field. Use default values for all other options and click **OK**.

28. You should see the configured chart, with information for SEDANS, VANS, and PICKUPS.

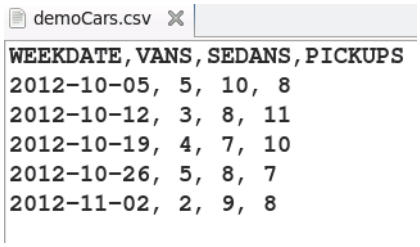


29. Save the page. Click **Save and Exit** above the widget palette.

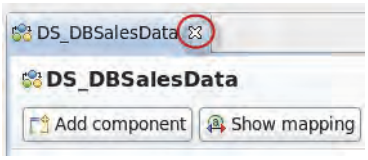
Exercise 2 Creating a UI data provider data set from a CSV file

In this exercise, you add an assembly line to your Tivoli Directory Integrator project that reads data from a CSV file. The CSV file data is similar to the database data that you configured in Unit 2, Exercise 1 on page 2-1, but does not include schema data. You add a file connector in the assembly line Feed section and modify the mapped work object values. You then use the assembly

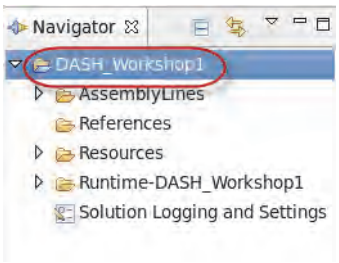
line data set to show the CSV data in a chart widget in a DASH dashboard page. The following screen image shows the contents of the CSV file.



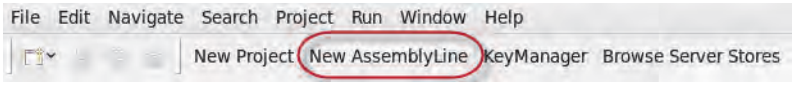
1. Close the **DS_DBSalesData** assembly line in the configuration editor workspace. Click the **X** icon in the **DS_DBSalesData** tab in the workspace.



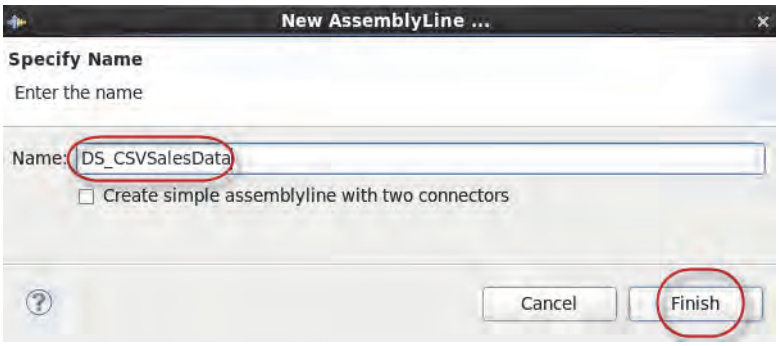
2. Select **DASH_Workshop1**in the Navigator section.



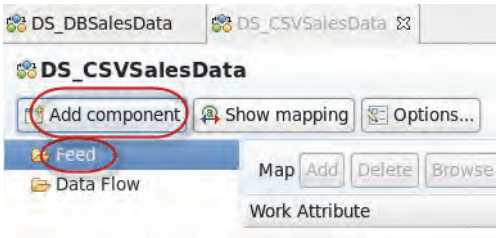
3. Click **New AssemblyLine** in the toolbar.



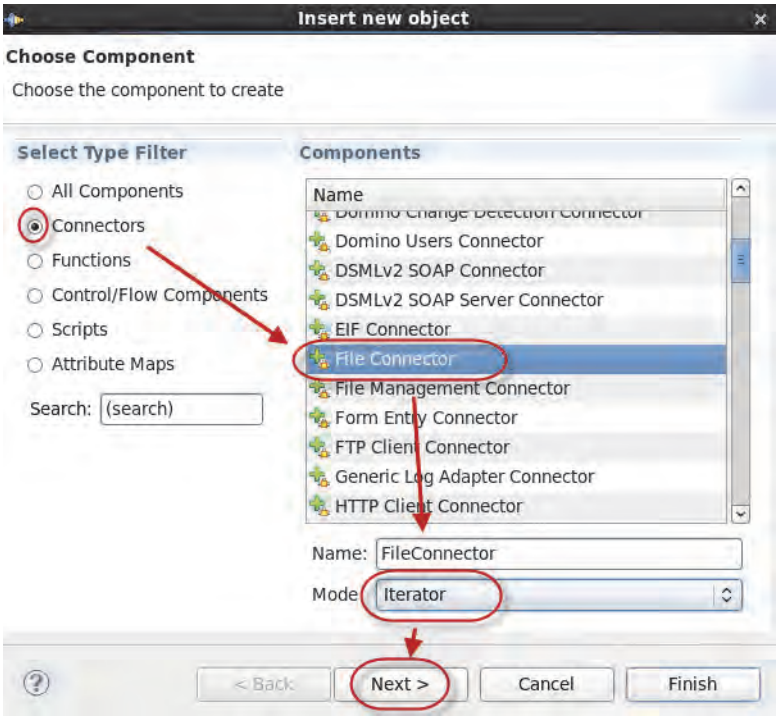
4. Enter **DS_CSVSalesData** in the **Name** field and click **Finish**.



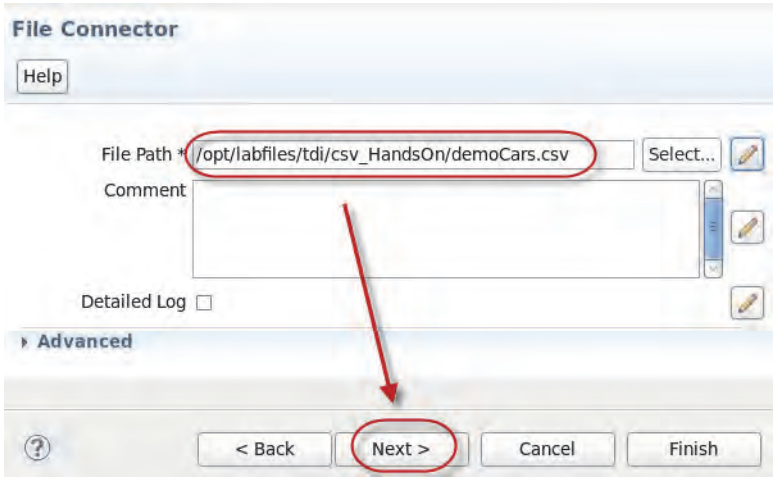
5. You must add a file connector to read data from the CSV file. Click **Feed** in the assembly line workspace and click **Add component**.



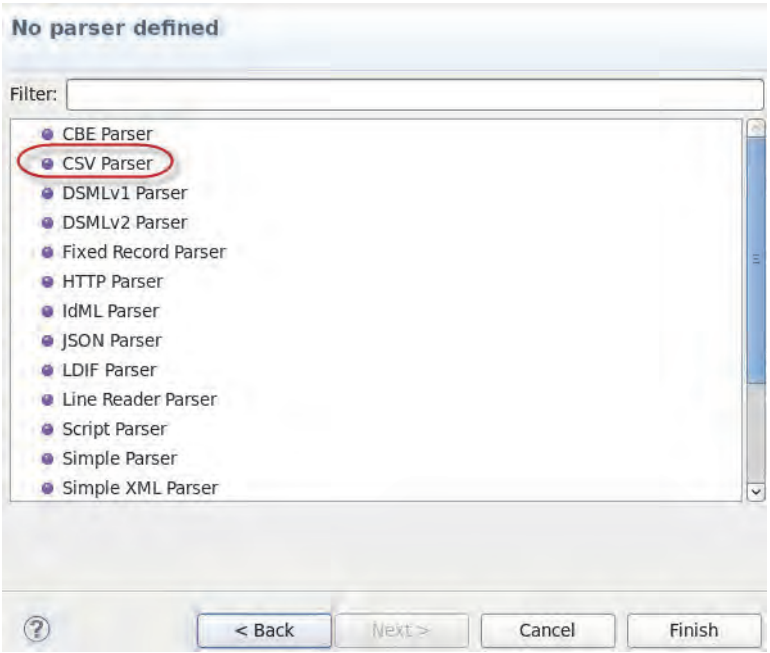
6. Select **Connectors** in the **Select Type Filter** section, click **File Connector** in the **Components** list, set the **Mode** to **Iterator**, and click **Next**.



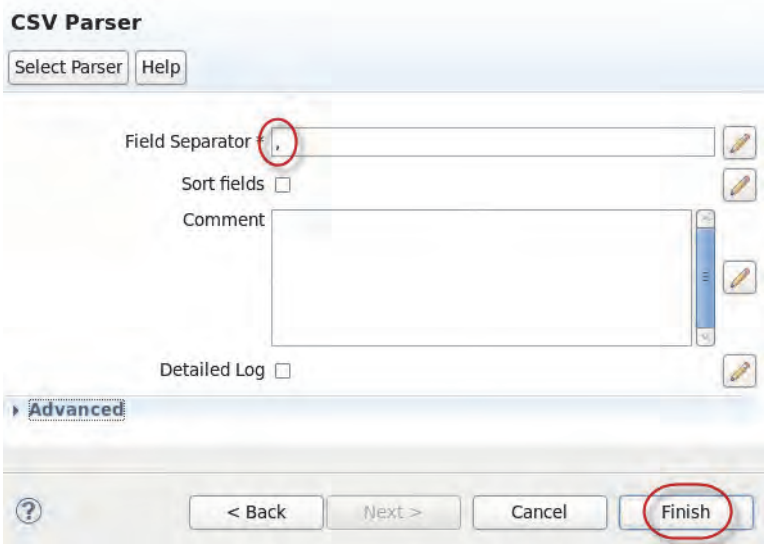
7. Configure the location of the CSV file. Click **Select**, then browse to and select the file **/opt/labfiles/tdi/csv_HandsOn/demoCars.csv**, and click **Next**.



8. The connector must be configured to parse the text file. Click **CSV Parser** in the parser list.

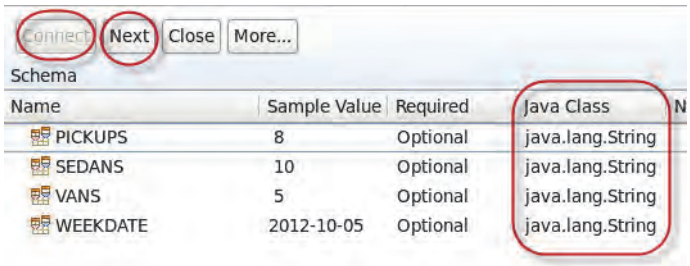


9. The default CSV field separator is the semi-colon (;) character, but the laboratory file uses a comma (,) character. Enter the comma character (**,**) in the **Field separator** field and click **Finish**.

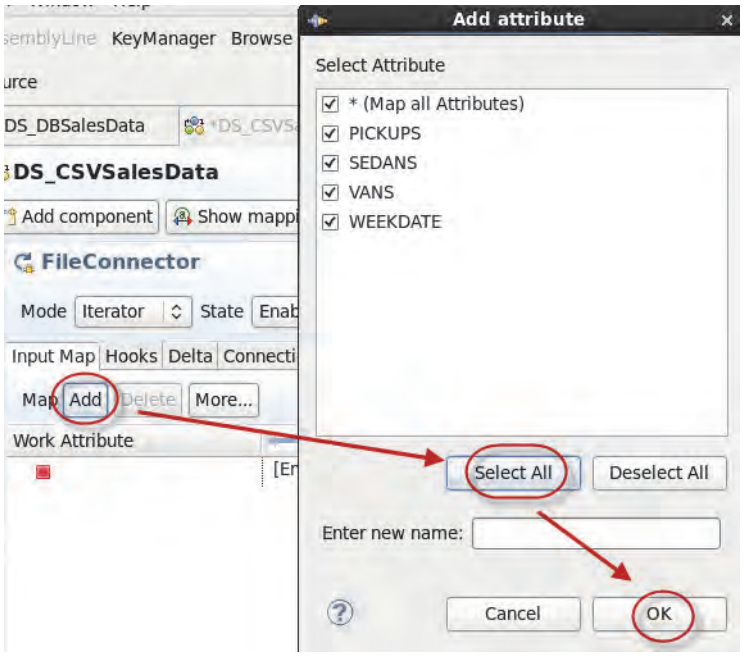


10. You now connect to the CSV file to configure the input map. Click Connect and then Next to begin iterating through the file. The column names are included in the first line of the CSV file and are used as the column headers. If the file did not include column header information, you configure header values in the **Advanced** section of the CSV Parser page.

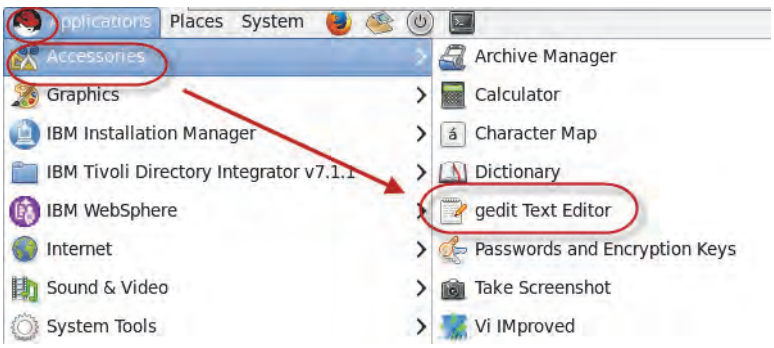
    Field types are all interpreted as character strings, as seen by the values in the **Java Class** column. You modify the field types in a subsequent step in this exercise.



11. Map the input values to the output work object. Click **Add** in the **Input Map** tab, click **Select All**, and click **OK**.

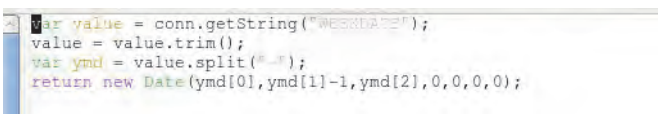


12. The default output value variable for each column value is in the form **conn.<COLUMNNAME>**. You must replace the default value with JavaScript code that converts the string value to the appropriate data type (Integer or Date). The JavaScript code is available in a file on the virtual image. Open the first file with a text editor. Select **Applications > Accessories > gedit Text Editor** in the desktop menu.



13. Select **File > Open** and browse to select the file **File System > opt > labfiles > tdi > csv_HandsOn > stringToDate.js** file.

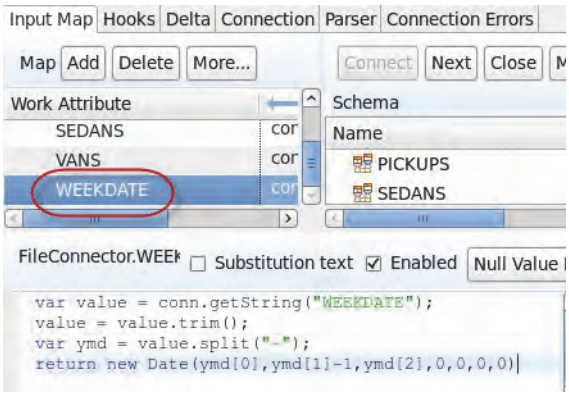
14. Select and copy the contents of the file. Select **Edit > Select All** and **Edit > Copy**.



```
var value = conn.getString("WEEKDATE");
value = value.trim();
var ymd = value.split("-");
return new Date(ymd[0],ymd[1]-1,ymd[2],0,0,0,0);
```

15. Switch back to the configuration editor and double-click the row to the left of the WEEKDATE work attribute.

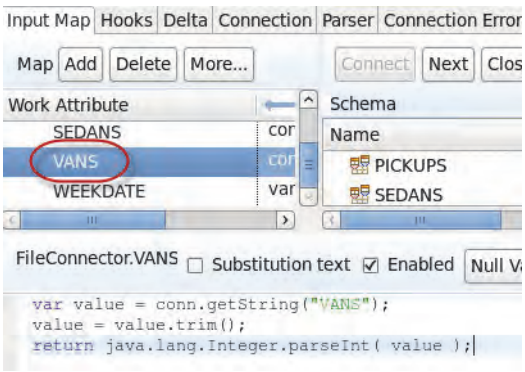
16. Replace the conn.WEEKDATE value with the contents of the text file. Press the Ctrl + V key combination.



17. Switch to the text editor and open the file **/opt/labfiles/tdi/csv_HandsOn/stringToInt.js**.

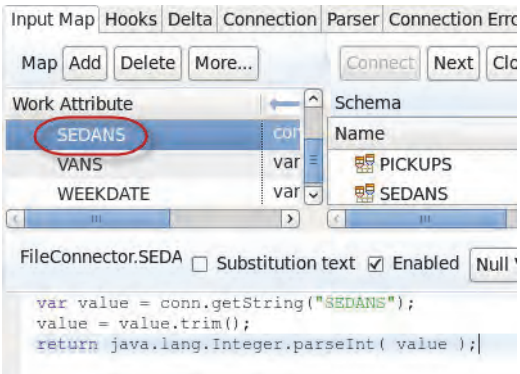
18. Copy the contents of the file.

19. Click **VANS** in the **Work Attribute** column and replace the default **conn.VANS** value with the text that you copied from the **stringToInt.js** file.



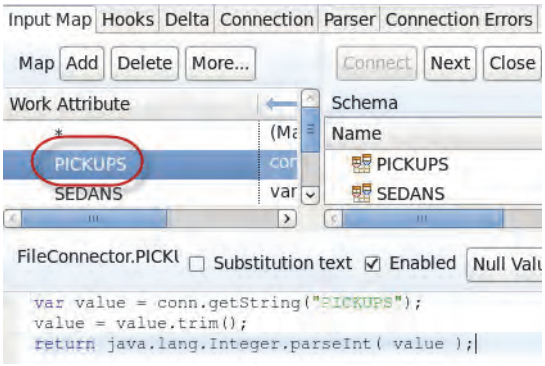
20. Click **SEDANS** in the **Work Attribute** column and replace the default **conn.SEDANS** value with the text that is copied from the **stringToInt.js** file.

21. Replace the word **VANS** with **SEDANS**.

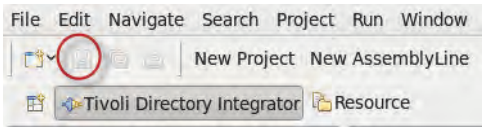


22. Click **PICKUPS** in the **Work Attribute** column and replace the default **conn.PICKUPS** value with the text that is copied from the **stringToInt.js** file.

23. Replace the word **VANS** with **PICKUPS**.

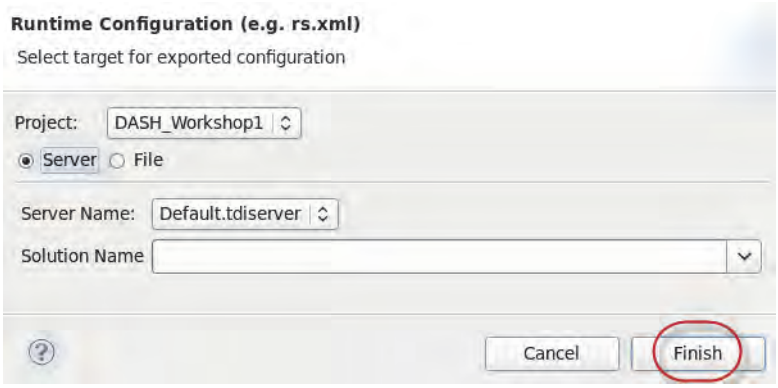


24. Save the updated assembly line and project file. Click the **Save** icon at the top of the configuration editor or select the **File > Save** menu.

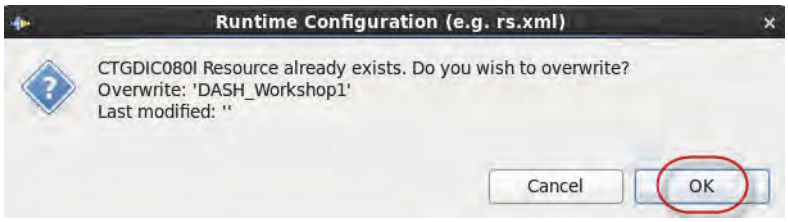


25. Drag **DASH_Workshop1** from the Navigator section onto the Default entry in the Servers section to add the updated project to the runtime server or use the **File > Export** menu method that is shown in the hint on page 2-8.

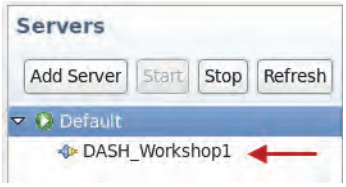
26. Use the default values for the runtime configuration and click **Finish**.



27. You see a warning message that you are overwriting an existing configuration file. This message is an indicator that the configuration is being updated and is normal. Click **OK**.

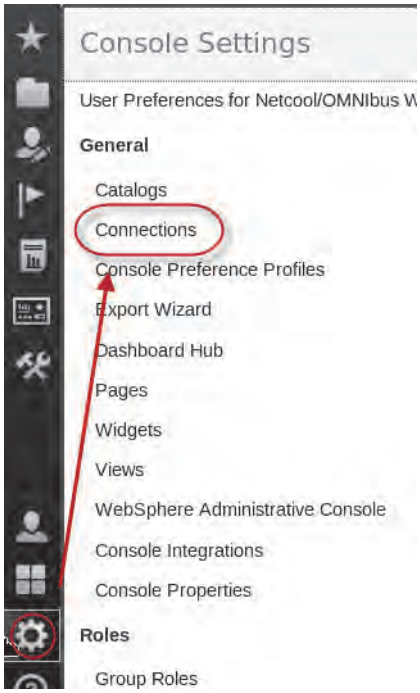


Important: When you configured the chart widget in Unit 2, Exercise 1 on page 2-1, the Tivoli Directory Integrator assembly line is triggered and the assembly line is shown in the Servers section. Because of the way the data is cached between Tivoli Directory Integrator and the DASH server, you must force the Tivoli Directory Integrator server to refresh the cache before you can use the updated assembly line.

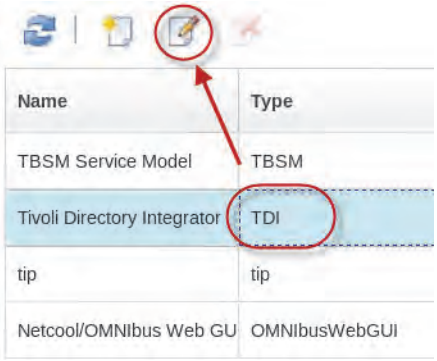


Refreshing the Tivoli Directory Integrator data cache

1. Select **Console Settings > General > Connections** in the DASH server console.

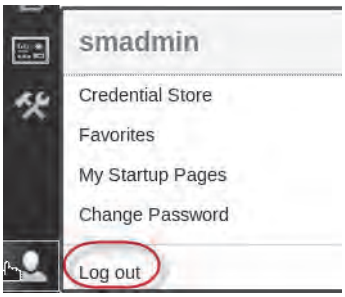


2. Click to select the **Tivoli Directory Integrator** row and click the **Edit** icon above the connection list.



3. Do not change any values that are configured in the connection document. Clicking **OK** is sufficient to trigger a refresh of the data cache.

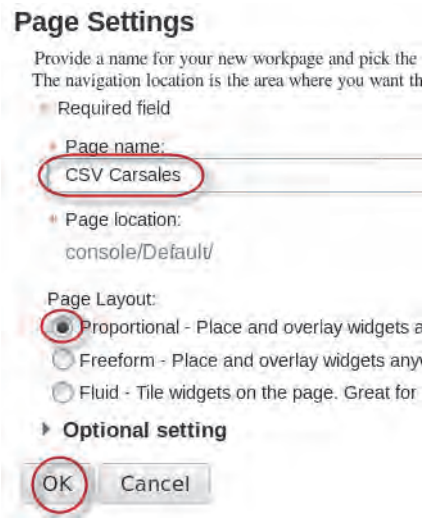
4. Finally, log off and login again to the DASH console.



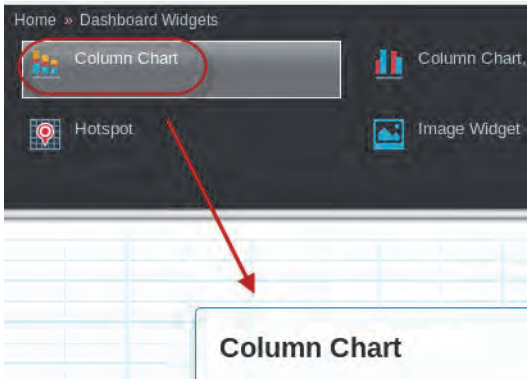
Hint: If you switch back to the configuration editor, you see that the DASH_Workshop1 project is no longer shown under the Default server entry. The missing entry indicates that the cache was successfully refreshed. If you still see an entry for DASH_Workshop1, repeat the process to refresh the cache.

5. Switch back to the DASH console and create a new page. Click the plus symbol (+) at the upper right of the console.

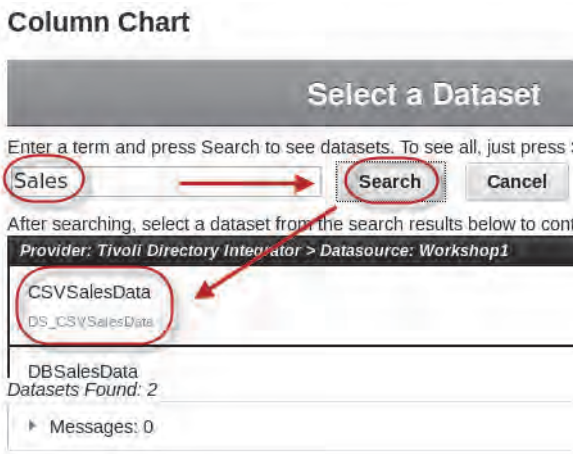
6. Enter **CSV Car Sales** in the **Page name** field, select **Proportional** for the **Page Layout**, and click **OK**.



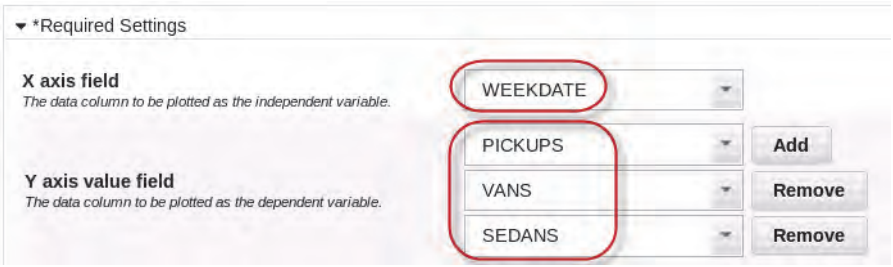
7. Click **Dashboard Widgets** and drag a **Column Chart** widget object onto the page canvas.



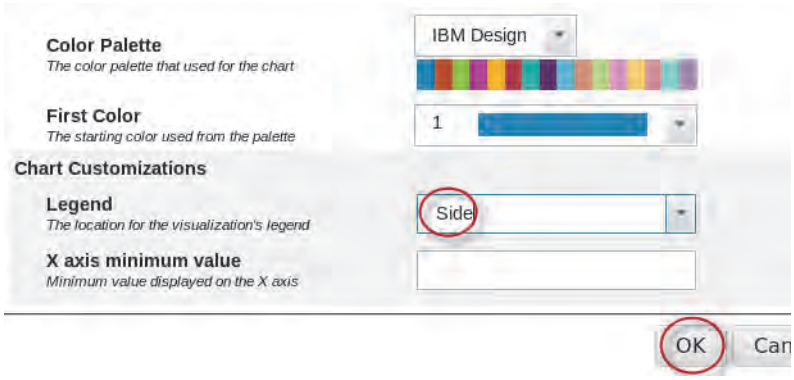
8. Edit the widget properties. Enter **Sales** in the Search field, click **Search**, and select **CSVSalesData**.



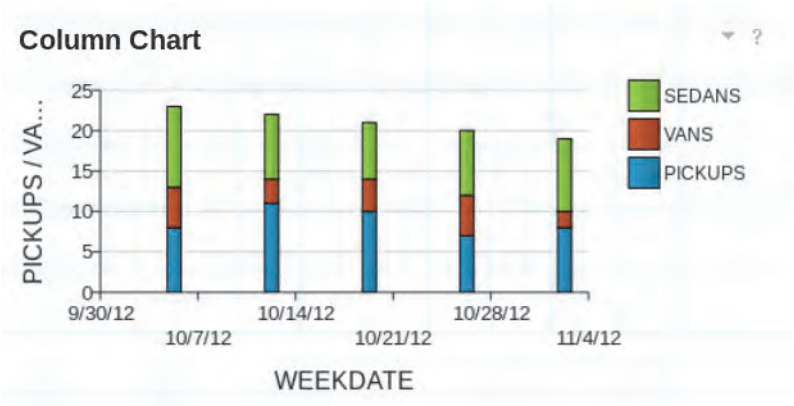
9. Select **WEEKDATE** for the **X-axis field value**, and add **PICKUPS**, **VANS**, and **SEDANS** as **Y-axis value field** values.



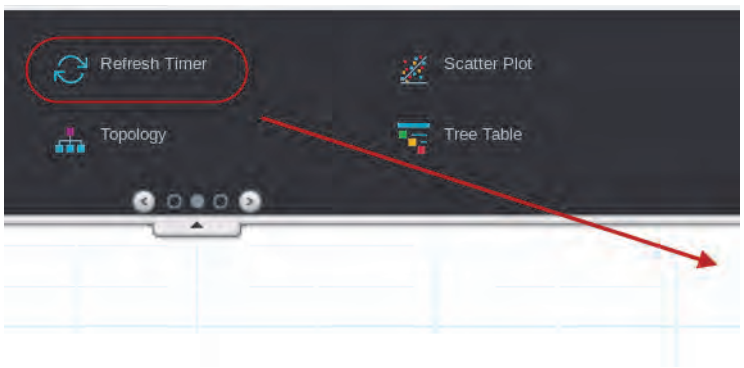
10. Click to expand the Optional Settings section, scroll down in the configuration form, select **Side** in the **Legend** menu, and click **OK**.



The new chart is shown on the page.

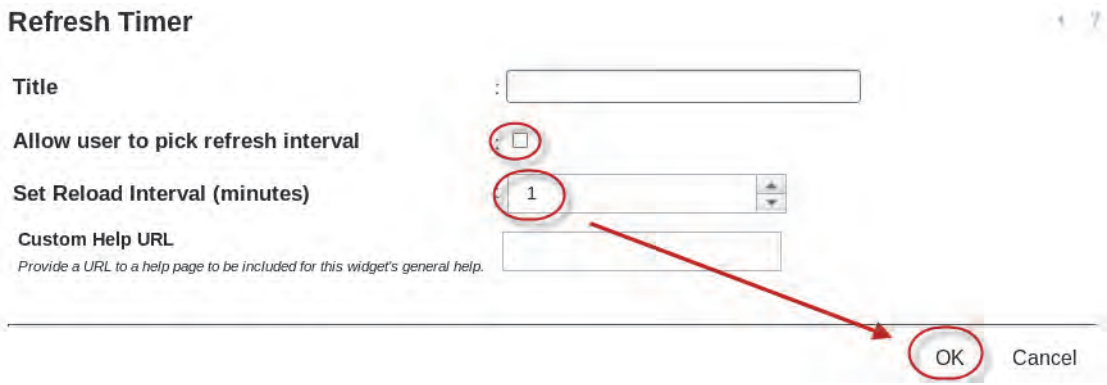


11. Tivoli Directory Integrator assembly lines are triggered when a configured dashboard widget is opened in a dashboard page. Changes in the Tivoli Directory Integrator data source are not automatically pushed to the widget. To see data changes in the dashboard page, you add and configure a *Refresh Timer* widget on the page. Locate the Refresh Timer widget in the widget palette and drag the widget to the upper right of the page canvas.

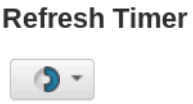


12. Click the **Refresh Timer** widget and select the **Widget > Edit** menu.

13. Do not enter a value in the Title field, remove the selection for **Allow user to pick refresh interval**, set the **Reload Interval** to **1**, and click **OK**.

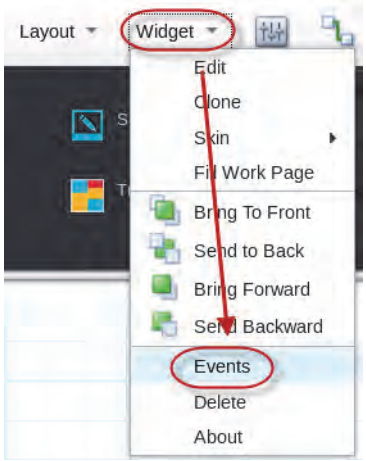


You see the refresh timer countdown progress in the widget icon.

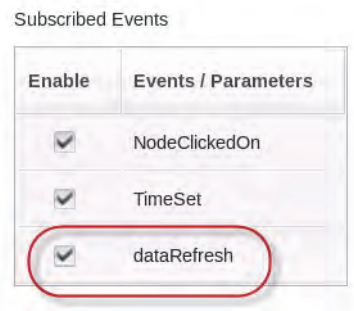


14. Remove the visible frame from the Refresh Timer widget. Click the widget and select **Widget > Skin > Transparent no title**.

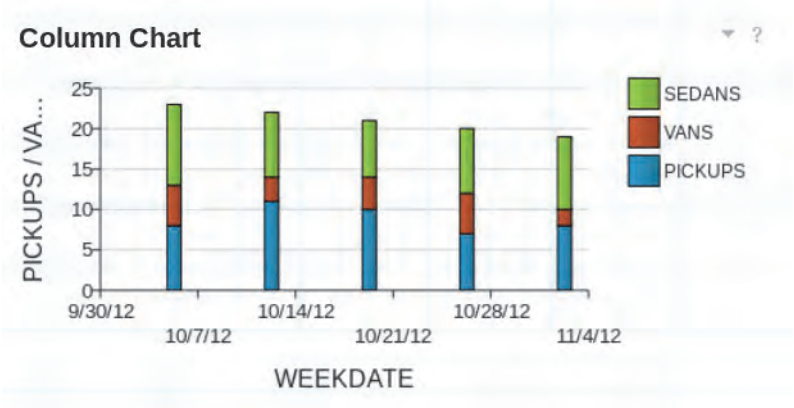
15. When the Refresh Widget timer reaches zero, the widget publishes a **dataRefresh** event to all of the widgets on the page. Only widgets that are configured as subscribers to dataRefresh events request an update from the configured data source. Verify that the chart widget is configured to subscribe to dataRefresh events. Click the chart widget and select the **Widget > Events** menu.



16. Verify that **dataRefresh** is selected in the **Subscribed Events** section and click **Close**.



17. Save the page. Click **Save and Exit** in the menu above the widget palette.



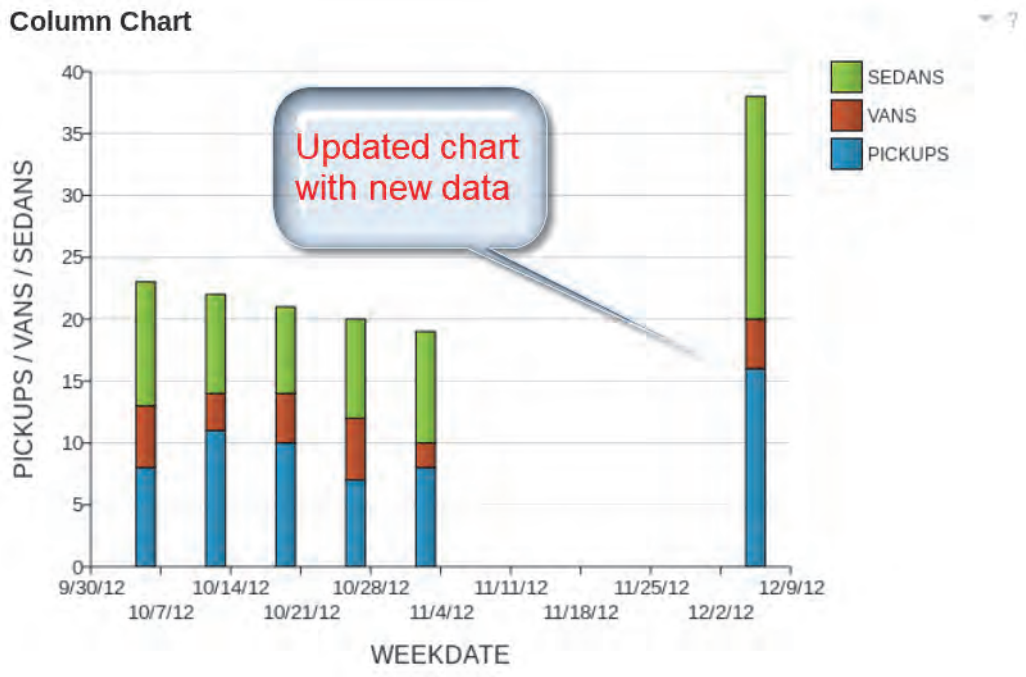
18. Update the content of the demoCars.csv file and verify that the data changes are shown in the chart widget when the Refresh Timer publishes a dataRefresh event.
    a. Open a command window on the dash151 image and enter the following command to append data to the demoCars.csv file:

```
echo "2012-12-05, 4, 18, 16" >> /opt/labfiles/tdi/csv_HandsOn/demoCars.csv
```

```
[root@dash151 ~]# echo "2012-12-05, 4, 18, 16" >> /opt/labfiles/tdi/csv_HandsOn/
demoCars.csv
```

Important: You must use *two* right-arrow characters (**>>**) in the command. Two right-arrow characters append the echo string to the demoCars.csv file. If you use a single right-arrow character (>), you overwrite the contents of the file and replace it with the single line of text.

b. Switch to the DASH console and the CSV Car Sales page. Watch the countdown timer icon in the Refresh Timer window. When the timer reaches zero, the dataRefresh event is published, triggering an update to the chart data.



Important: Some application UI data providers, such as Netcool/OMNIbus and Tivoli Business Service Manager, automatically push data changes to any connected dashboards. Never publish dataRefresh events to widgets that are automatically refreshed by the data provider. You must unsubscribe dataRefresh events for widgets that automatically updated the UI data provider. Failure to unsubscribe the events can cause dashboard instability when the page is opened.

Unit 3 Tivoli Directory Integrator and DASH advanced topics exercises

In these exercises, you learn advanced techniques to retrieve data and create UI data providers with Tivoli Directory Integrator. You learn how to enhance UI data providers with helper assembly lines that provide parameter selections and status evaluation. You learn how to save and use cached data to provide offline demonstrations and how to evaluate and present metric data from a data source.

Tivoli Directory Integrator project files can be saved and ported to many different environments. This portability means that you can easily reuse and share previously tested projects and adapt them to different customer environments. In these exercises, you use example templates to quickly develop data mining solutions.

Exercise 1 Using a helper assembly line to add parameters to a data set

In the Unit2 exercises, you created UI data providers to serve sales data for a car company. The company had offices in North Carolina and California. In this scenario, the company has gathered more detailed sales information, including the tracking of sales by office location. In this exercise, you reuse an example assembly line that to provide parameter selection data. This assembly line is not a primary assembly line, but is what is referred to as a *helper* assembly line. Helper assembly lines supplement a primary assembly line data set to provide more configuration selections when configuring a DASH widget. In this exercise, you create a helper assembly line that adds the ability to select sales data by location.

You must complete the following high-level tasks:

- **Import example assembly line project file**: You import a Tivoli Directory Integrator project file into the configuration editor. The project file includes several assembly lines and scripts that can be copied into your projects and quickly reconfigured for your requirements.
- **Create a primary assembly line to retrieved enhanced sales data**: This assembly line shows car sales data from an enhanced sales database. The new sales database contains

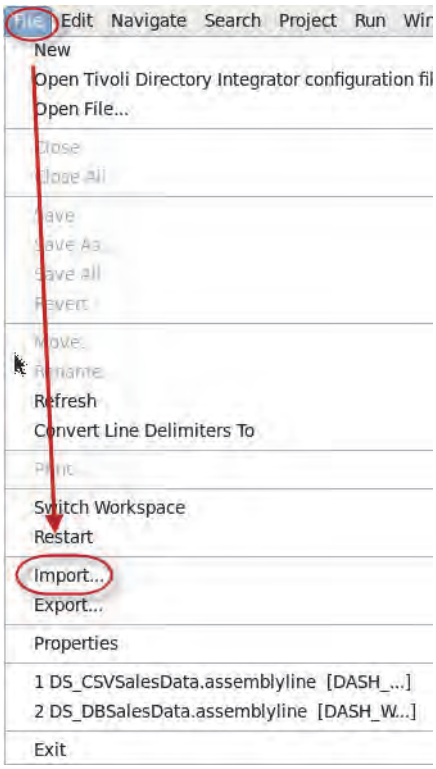
more detailed information about the date, number sold, types of cars, vans, and trucks, and the location of each vehicle that was sold.

- **Create a helper assembly line to support selecting sales data by location**: This assembly line evaluates the database to determine where each vehicle was sold. The data is correlated with the information in the primary assembly line, and it is automatically added as a widget parameter selection in the widget quick-edit form.
- **Test assembly lines with widgets in a dashboard page**: You create a dashboard page that uses the new assembly lines. You edit a page that contains a map image with two hotspot widgets that correspond to the North Carolina and California sales offices. Also included is a table widget that shows detailed sales data, by location. You configure the hotspot widgets so that you can select a hotspot widget to see sales data by location.

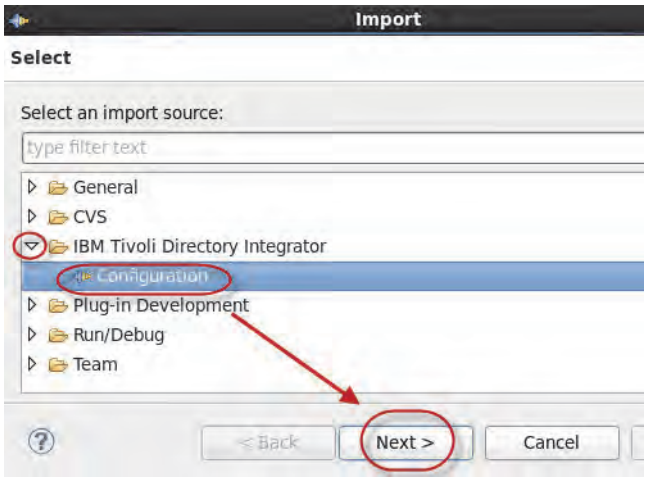
Importing example assembly line project file

In this task, you import a Tivoli Directory Integrator project file into the configuration editor. The project file contains several example assembly lines that you use in this and other exercises for this unit.

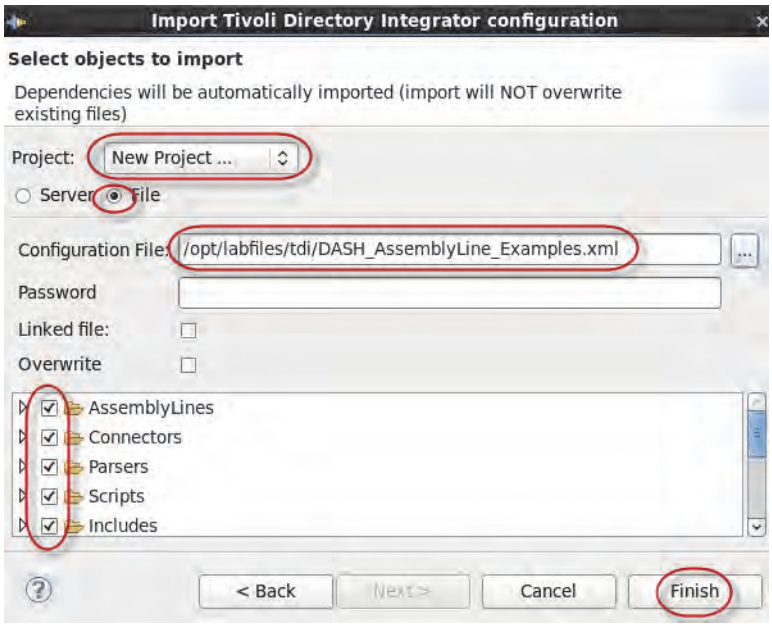
1. Close any assembly lines that are open in the configuration editor workspace. Click the **X** icon in any tabs in the configuration editor workspace.
2. Import the example project file. Switch to the Tivoli Directory Integrator configuration editor and select the **File > Import** menu.



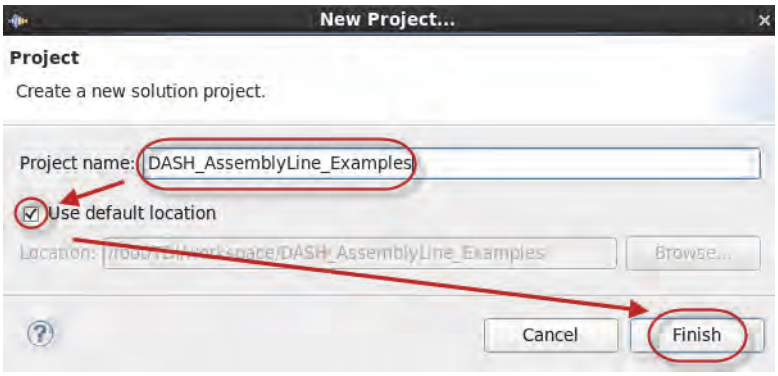
3. Click the **arrow icon** to the left of **IBM Tivoli Directory Integrator**, select **Configuration**, and click **Next**.



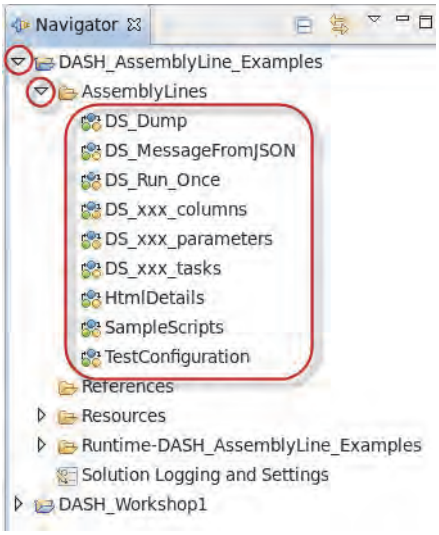
4. Select **New Project** in the Project menu, select **File**, enter **/opt/labfiles/tdi/DASH_AssemblyLine_Examples.xml** in the **Configuration File** field, and click **Finish**. All of the project components are selected by default.



5. Enter **DASH_AssemblyLine_Examples** in the **Project name** field, use the default location selection, and click **Finish**.



6. The imported project is shown in the Navigator section of the configuration editor. Expand **DASH_AssemblyLine_Examples > AssemblyLines** to see the list of examples.



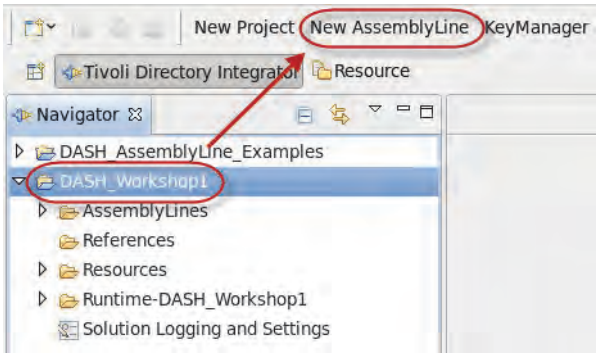
Creating a primary assembly line to retrieve enhanced sales data

First, you create an assembly line that serves the data from the enhanced car sales database. The new data is maintained in the **BREAKDOWNSALESDATA** table in the **CARSALES** database on the dash151 virtual image. The updated database schema is shown in the following screen capture.

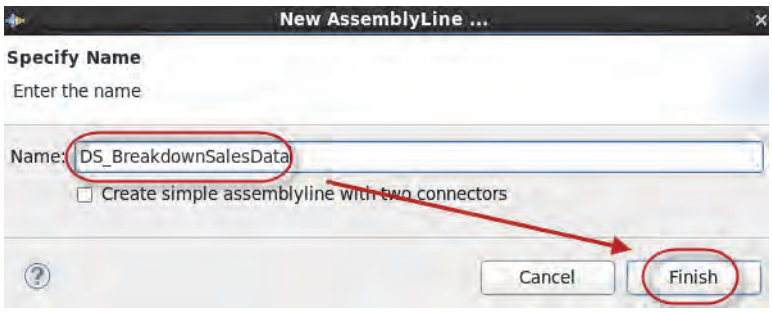
Schema

Name	Sample Value	Required	Java Class	Native Syntax
LABEL		Optional		VARCHAR
LOCATION		Optional		VARCHAR
MAJORTYPE		Optional		VARCHAR
MINORTYPE		Optional		VARCHAR
SALES		Optional		INTEGER
WEEKDATE		Optional		DATE

1. Create an assembly line. Select the **DASH_Workshop1** project in the **Navigator** and click **New AssemblyLine**.

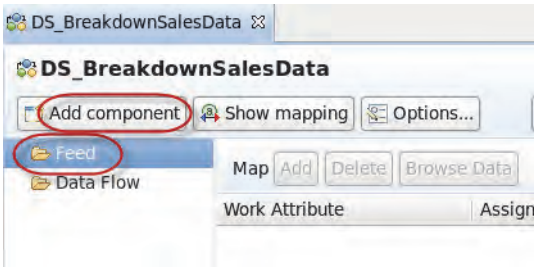


2. Enter **DS_BreakdownSalesData** in the **Name** field and click **Finish**. You must include the prefix **DS_** in the name.

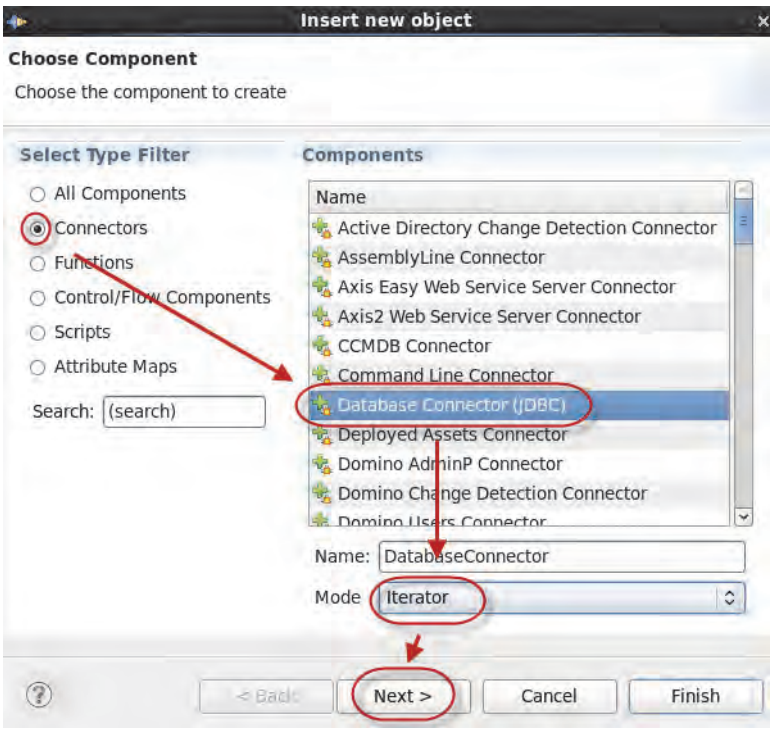


The new assembly line is created and shown in the configuration editor workspace and added to the DASH_Workshop1 project.

3. Add a database connector to the assembly line Feed section. Click **Feed** and click **Add component**.



4. Select **Connectors** to filter the Components list, select **Database Connector (JDBC)**, select **Iterator** in the Mode menu, and click **Next**.



5. Configure the database connector. Use the information in the following table to complete the form:

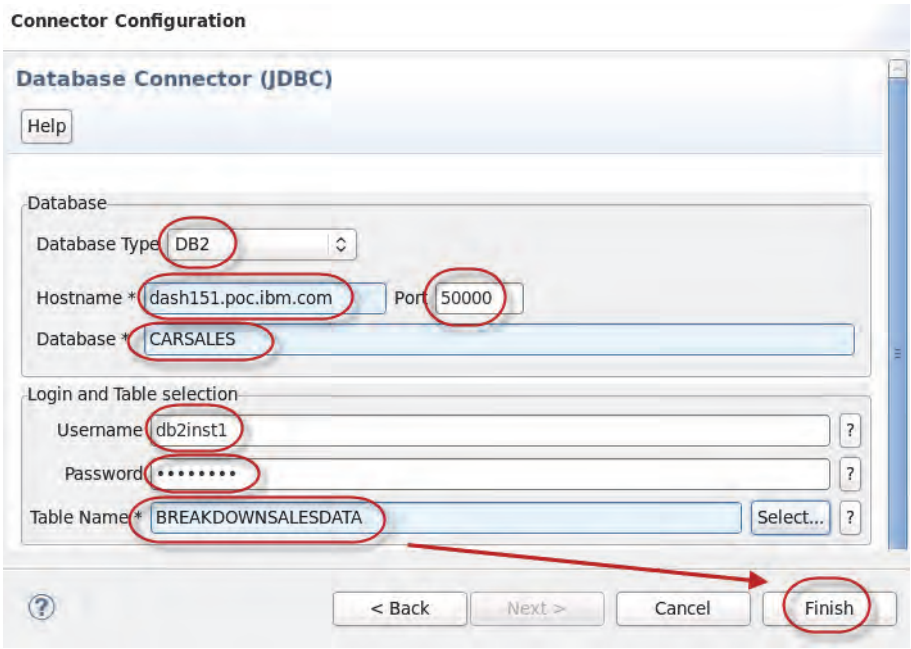
Table 1

Parameter Name	Value
Database Type	DB2
Host name	dash151.poc.ibm.com
Port	50000
Database	CARSALES
User name	db2inst1

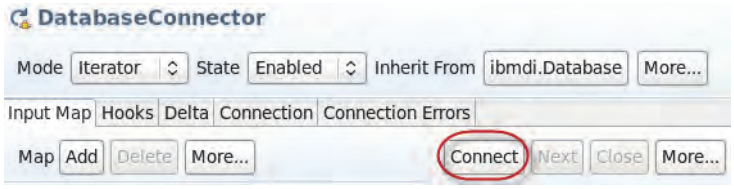
Table 1

Parameter Name	Value
Password	object00
Table Name	BREAKDOWNSALESDATA

6. Click **Finish**.

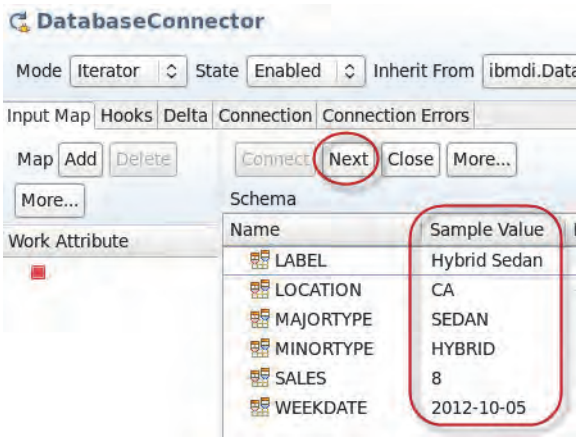


7. Verify the database connector configuration. Click **Connect** in the **Database Connector** section.



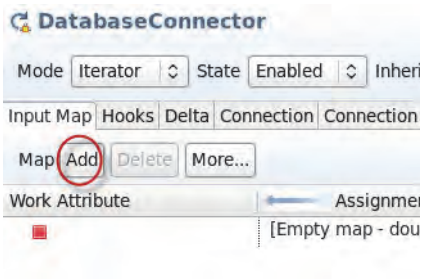
The assembly line connects to the configured database and shows the database schema.

8. Verify that the database contains valid data. Click **Next** above the **Schema** section.

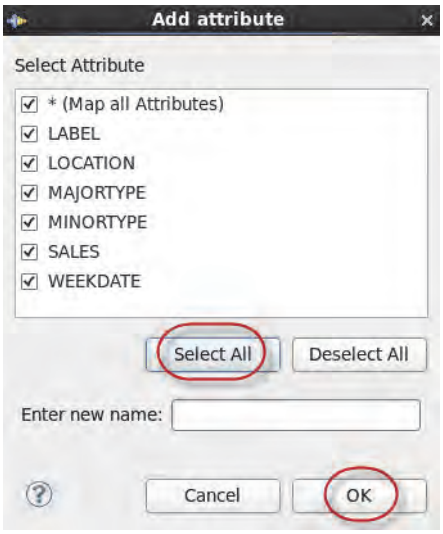


You see data values in the Sample Value column each time you click Next.

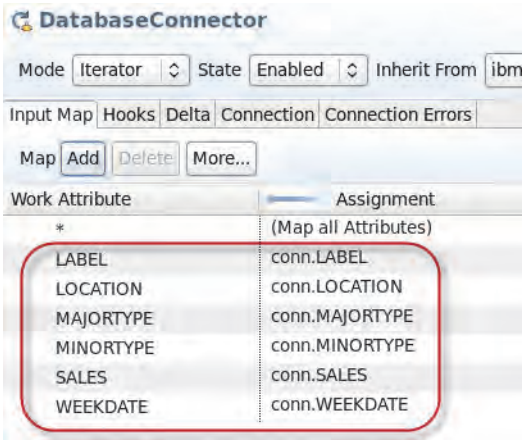
9. Map the database columns to the Tivoli Directory Integrator work object. Click **Add** in the Input Map tab.



10. For this exercise, map all of the table columns. Click **Select All** and click **OK**.



You see all the work object attribute assignments in the Input Map tab. Because the data types are included in the database schema, no additional conversion is required for any of the mapped attributes.



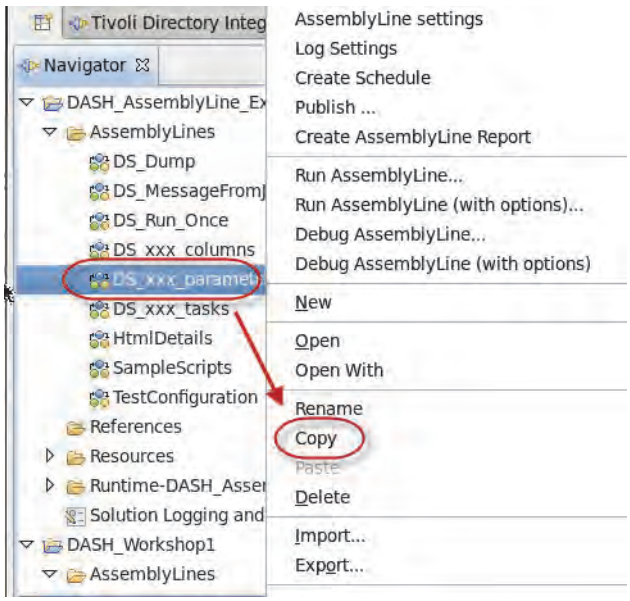
11. Save the new assembly line in the DASH_Workshop1 project. Click the **File > Save** menu at the upper left of the configuration editor.

Creating a helper assembly line to support selecting sales data by location

For this task, you create a helper assembly line that provides parameter selections for the primary assembly line.

1. Copy the DS_xxx_parameters example assembly line from the example project to the DASH_Workshop1 project. Expand **DASH_AssemblyLine_Examples > AssemblyLines** in the configuration editor Navigator.

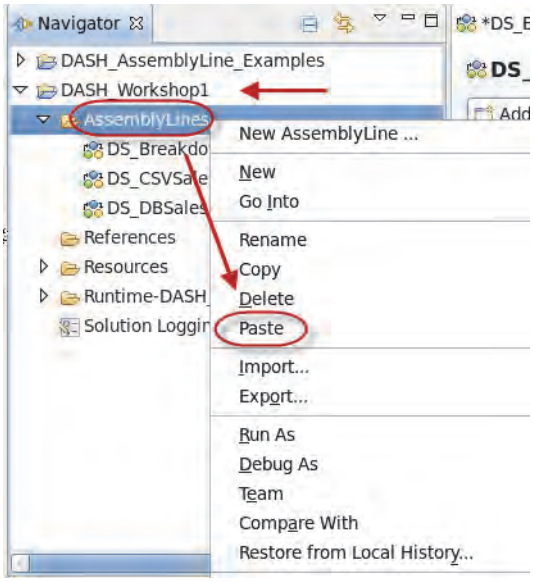
2. Right-click **DS_xxx_parameters** and select **Copy**.



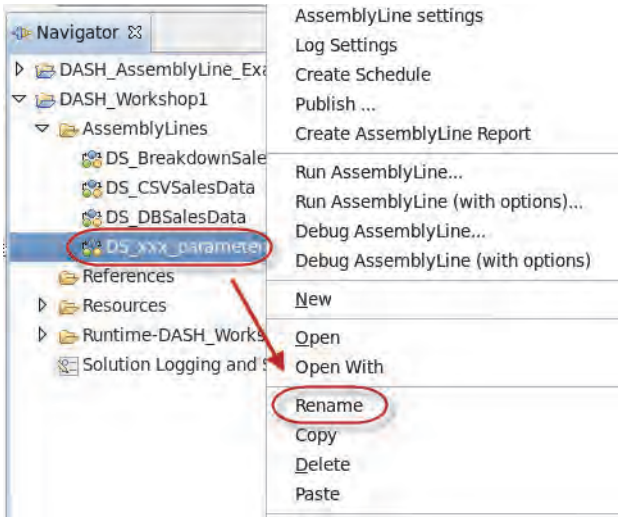
Hint: If the menu does not remain on the screen when you right-click the example assembly line, click and hold the right-click mouse button while you scroll the menu to select Copy. Release the right-click mouse button to select a menu entry.

3. Expand **DASH_Workshop1 > AssemblyLines** in the Navigator.

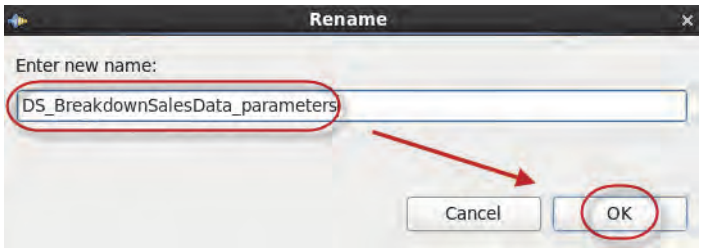
4. Right-click **AssemblyLines** and select **Paste**.



5. Rename the copied assembly line. Right-click **DS_xxx_parameters** and select **Rename**.

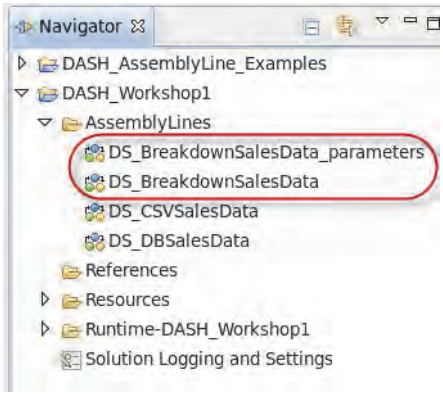


6. Enter **DS_BreakdownSalesData_parameters** and click **OK**.



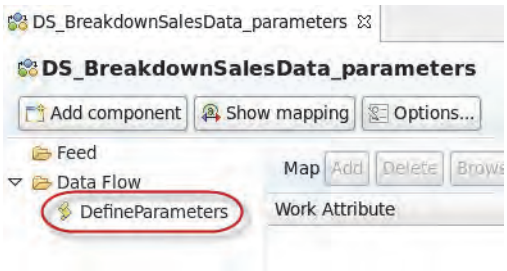
Important: The assembly line name must be in the form **<DS_Primary_ALName>_parameters**. In this exercise, the primary assembly line is named DS_BreakdownSalesData. The DASH server does not recognize the helper assembly line if the helper assembly line name is not written in this form, including letter case.

You see the two related assembly lines in the Navigator list.

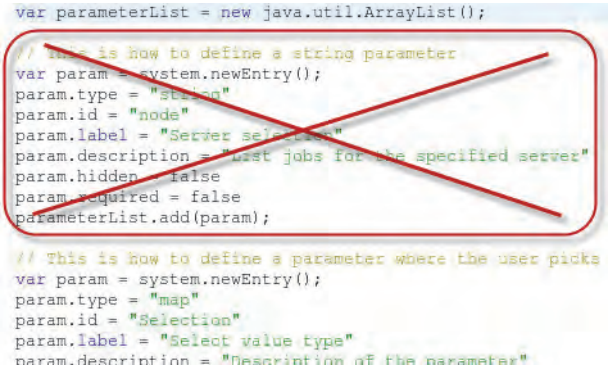


7. Open the new assembly line in the configuration editor workspace. Double-click **DS_BreakdownSalesData_parameters** in the Navigator.

8. No connector is defined in the Feed section. This assembly line is called by the primary assembly line as needed. You must modify the included DefineParameters script in the Data Flow section. Click **DefineParameters** to see the current script in the workspace.

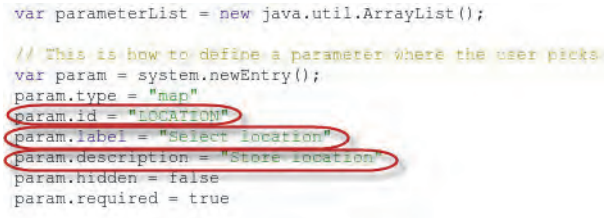


9. The included script is intended to be generic enough to be adaptable to common situations. For this exercise, delete the section below the first line.



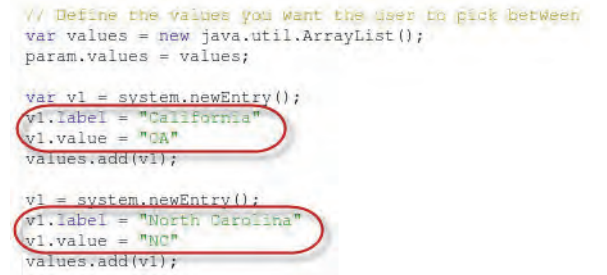
10. Modify the following variables:

    `param.id = "LOCATION"`
    `param.label = "Select location"`
    `param.description = "Store location"`

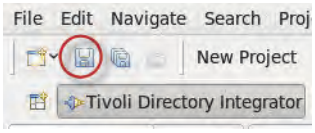


11. The script includes two v1.label and v1.value lines that define parameter names and corresponding values. Use the following values to modify the lines:

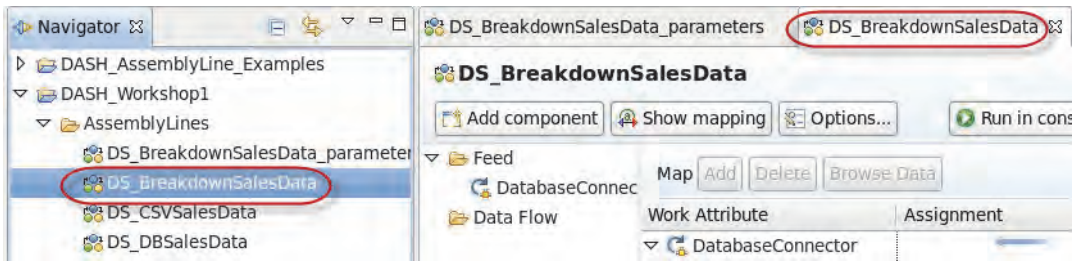
    `v1.label = "California"`
    `param.label = "CA"`
    `v1.label = "North Carolina"`
    `param.label = "NC"`



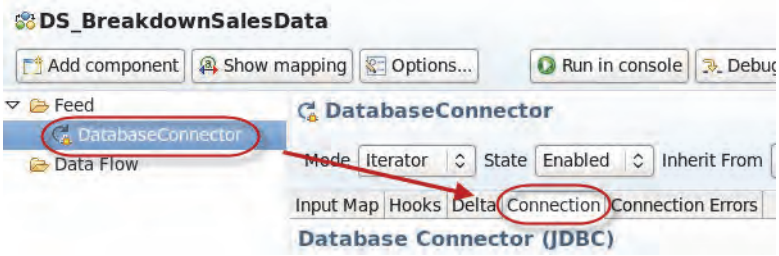
12. Save the updated assembly line and project. Click the **Save** icon at the upper left of the configuration editor.



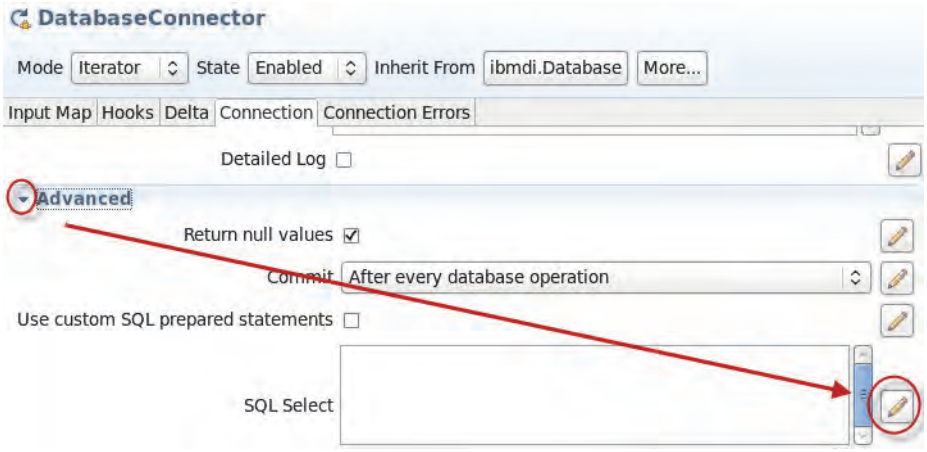
13. Now that the helper assembly line is configured, modify the primary assembly line to use the parameter selection when retrieving a data set from the database. Double-click **DS_BreakdownSalesData** in the Navigator. The assembly line opens in a second tab in the configuration editor workspace.



14. Modify the database connector connection configuration. Click **DatabaseConnector** and click the **Connection** tab.

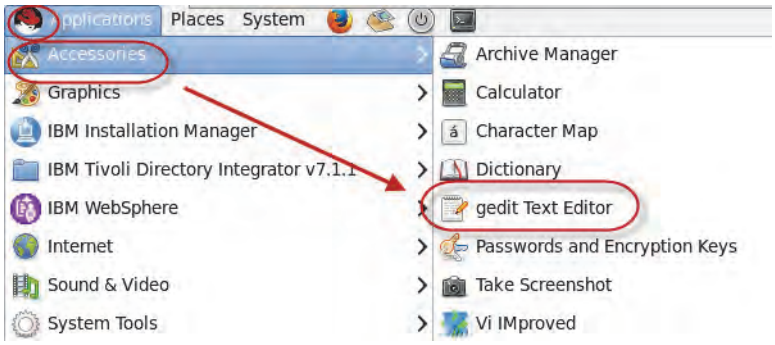


15. Expand the **Advanced** section and click the **Edit** icon to the right of the **SQL Select** field.



You use a JavaScript file to dynamically manage the SQL command. A copy of the file is stored in **/opt/labfiles/tdi/parametersHandsOn/parameter_select.js**.

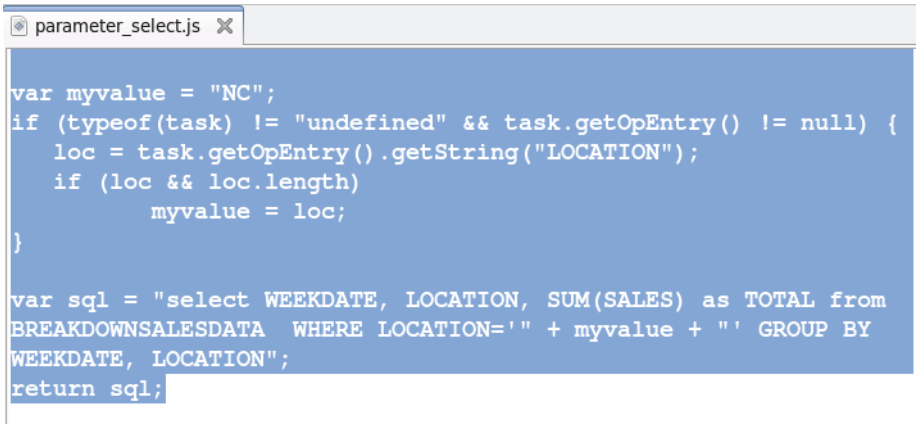
16. Select the **Applications > Accessories > gedit Text Editor** menu at the top of the image desktop.



17. Select the **File > Open** menu.

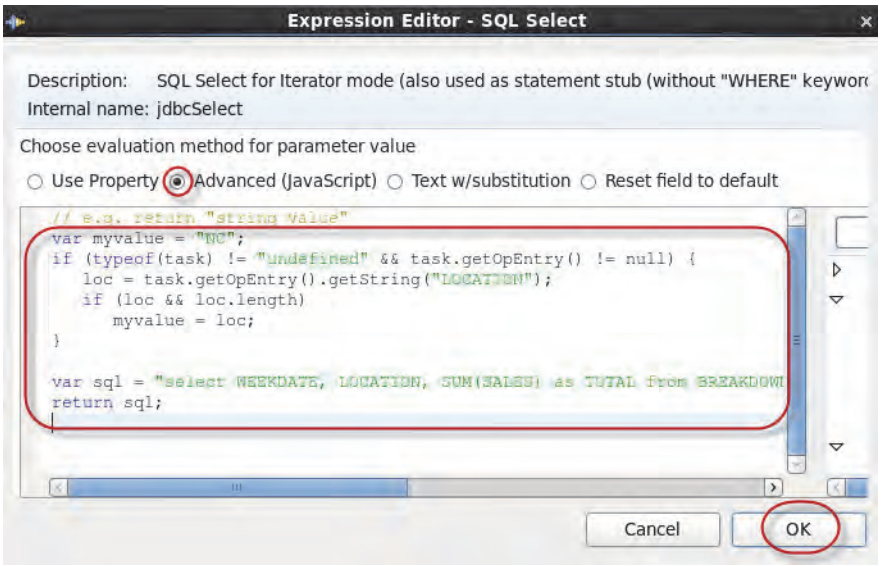
18. Browse to and select the file **/opt/labfiles/tdi/parameters_HandsOn/parameter_select.js**.

19. Copy the entire file. Select the **Edit > Select All** menu and then select the **Edit > Copy** menu.

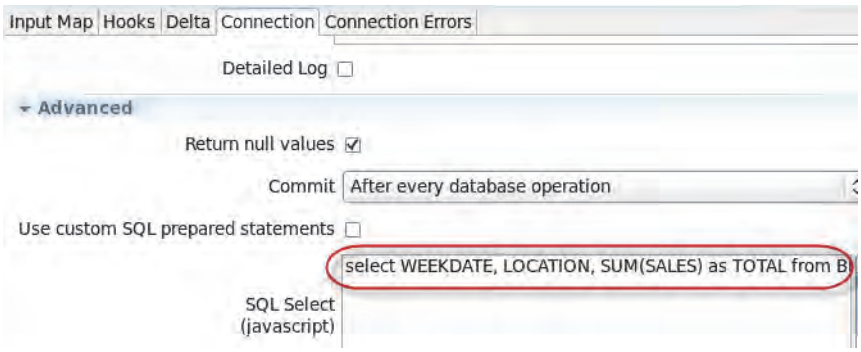


20. Switch to the configuration editor, and select **Advanced (JavaScript)**.

21. Place the cursor in the **SQL Select** field, enter the **Ctrl + v** key sequence to paste the file, and click **OK**.

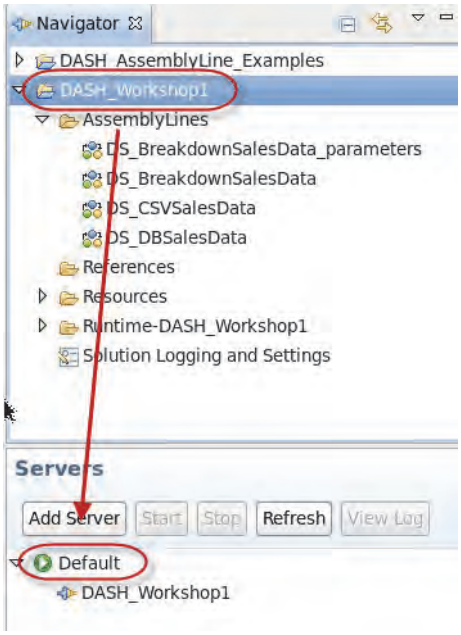


You see the final SQL query string in the SQL Select field.



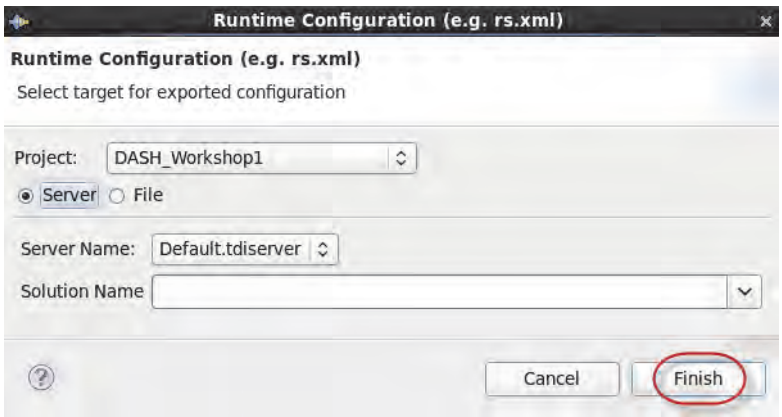
22. Save the updated assembly lines and project. Select the **File > Save All** menu at the upper left of the configuration editor.

23. Update the Tivoli Directory Integrator runtime server with the updated project file. Drag **DASH_Workshop1** from the **Navigator** onto the **Default** entry in the **Servers** section.

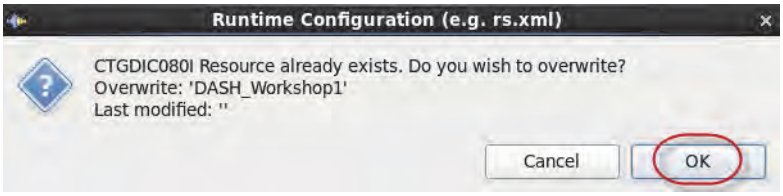


Important: If you do not see a Runtime Configuration window pop-up, the update was not successful. In that case, drag the project onto the Default entry again or use the **File > Export** menu process that is described in the hint on page 2-21.

24. Click **Finish** to use the default settings for the runtime configuration.

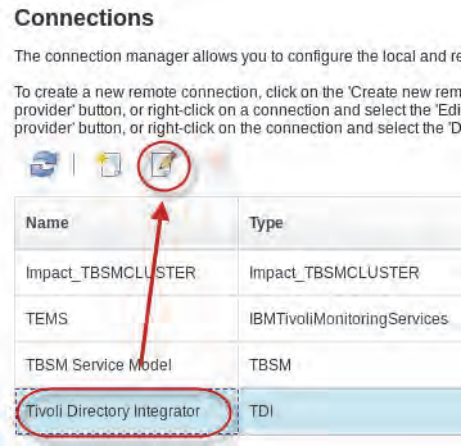


25. Click **OK** to confirm the replacement of the current runtime configuration.

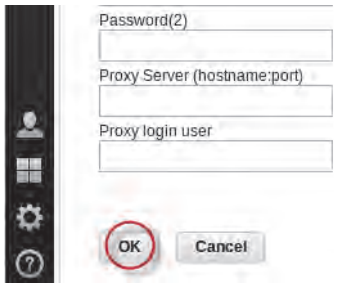


26. Recycle the cached connection between the Tivoli Directory Integrator server and the DASH server. Login to the DASH console and select **Console Settings > General > Connections**.

27. Select the Tivoli Directory Integrator entry in the connections list and click the **Edit** icon to open the connection document.



28. Do not edit the connection document. Click **OK** to close the document.



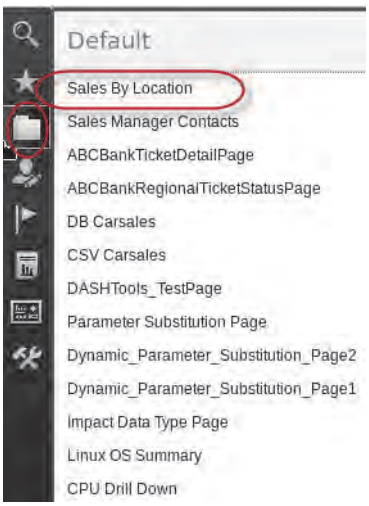
29. Log off the console. Select the person icon and click **Logout**.

30. Log in to the DASH console with the user ID **smadmin** and the password **object00**.

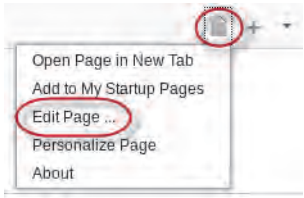
Testing the assembly lines with widgets in a dashboard page

In this task, you modify a dashboard page to use the parameter assembly line. The unconfigured page was created during the laboratory setup. The page consists of a configured image widget, two unconfigured hotspot widgets, and an unconfigured table widget.

1. Open the test page. Click the Default folder in the console taskbar and select **Sales By Location**.



2. Edit the page. Click the **Page Actions** icon at the upper right of the console and select **Edit Page**.

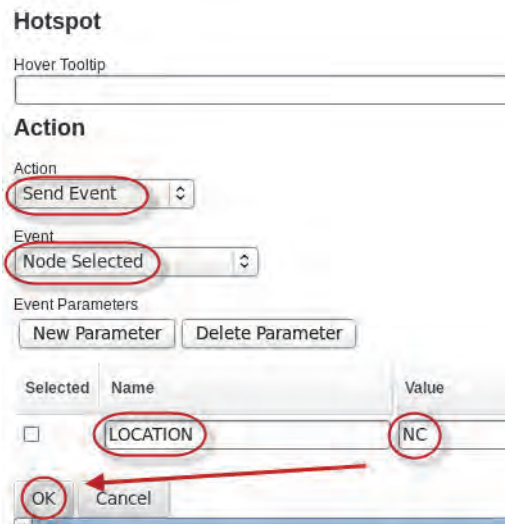


3. Edit the first hotspot widget. Click the widget on the right side of the map image and select the **Widget > Edit** menu above the widget palette.

4. Scroll to the bottom of the configuration window and make the following modifications:

```
Action: Send Event
Event: Node Selected
Name: LOCATION
Value: NC
```

5. Click **OK** to save the configuration.

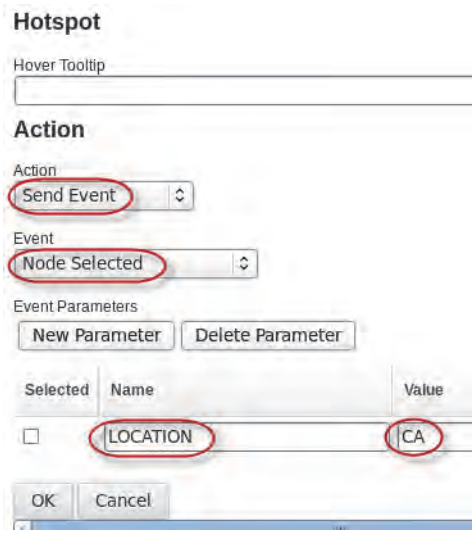


6. Edit the second hotspot widget. Click the widget on the left side of the map image and select the **Widget > Edit** menu above the widget palette.

7. Scroll to the bottom of the configuration window and make the following modifications:

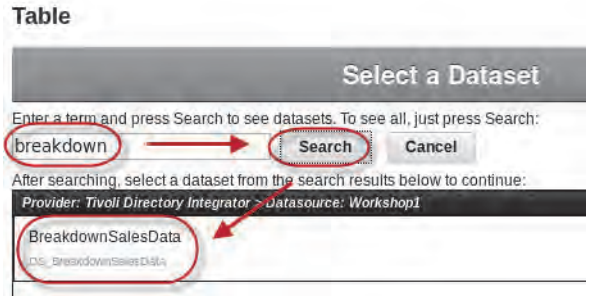
```
Action: Send Event
Event: Node Selected
Name: LOCATION
Value: CA
```

8. Click **OK** to save the configuration.

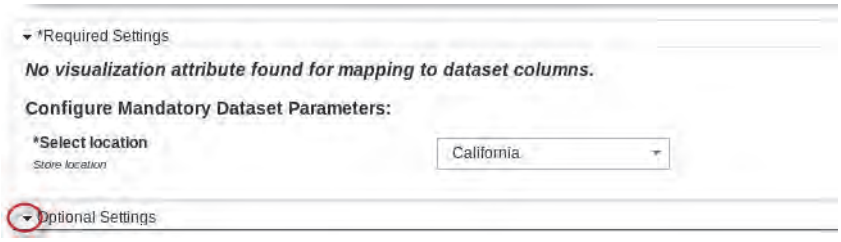


9. Edit the table widget properties. Click the widget and select the **Widget > Edit** menu above the widget palette.

10. Enter **breakdown** in the search field, click **Search**, and select **BreakdownSalesData**.



11. Use the default location selection and edit the **Optional Settings** section.

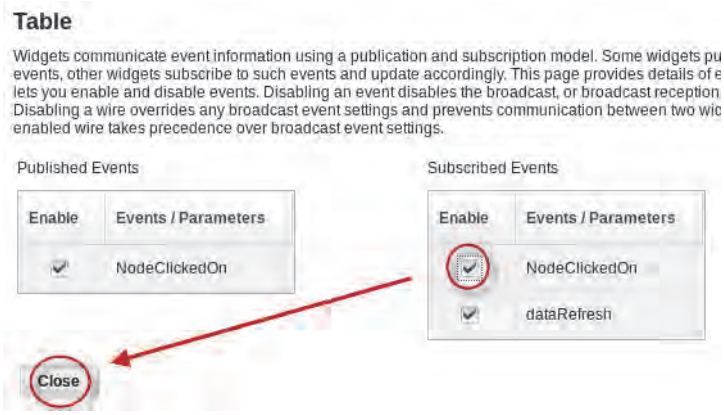


12. Change the column order so that LOCATION is at the top of the list. Click **LOCATION**, click the **up-arrow icon** until LOCATION is at the top of the list, and click **OK**.

13. By default, the table widget does not subscribe to NodeClickedOn events. Change the default behavior. Click the table widget and select the **Widget > Events** menu above the widget palette.

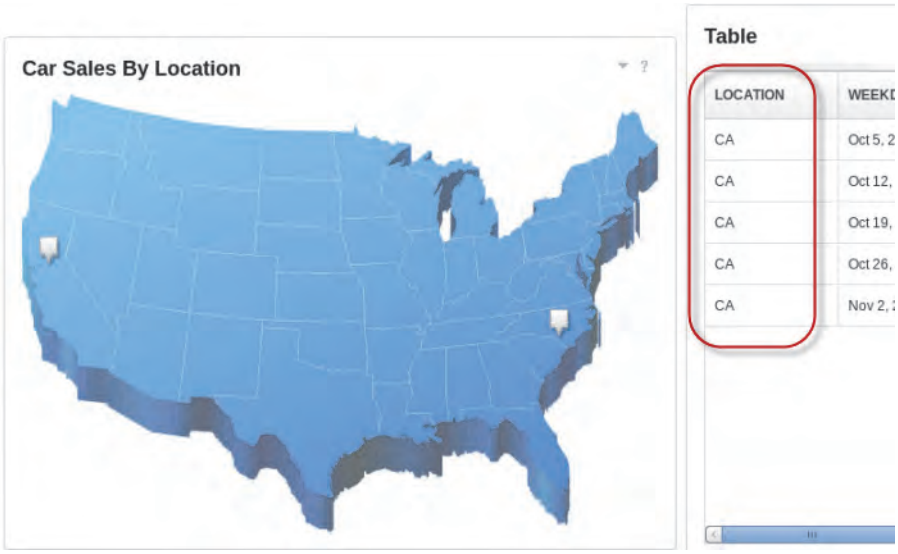
Important: Do not select **Widget > Edit** to configure the widget event publish and subscribe settings. You must select **Widget > Events**.

14. Select **NodeClickedOn** in the **Subscribed Events** column and click **Close**.

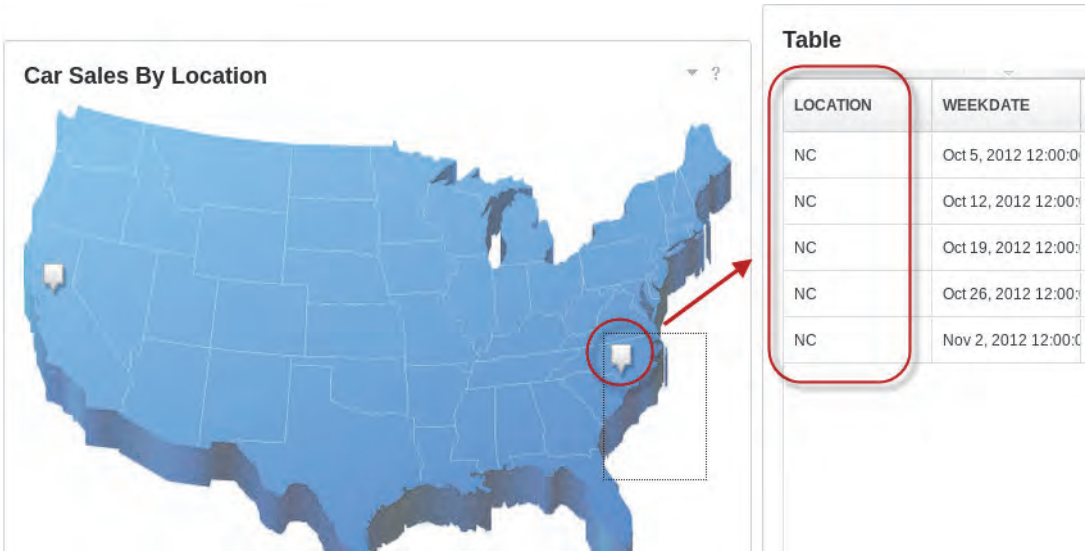


15. Save the page. Click **Save and Exit** above the widget palette.

You see the page with the default California sales information shown in the table widget.



16. Click the hotspot widget on the right side of the map image. You see the table widget update with North Carolina data. Click the hotspot widget on the left side of the map image to change the list back to California data.



Exercise 2 Using a helper assembly line to add status values to a data set

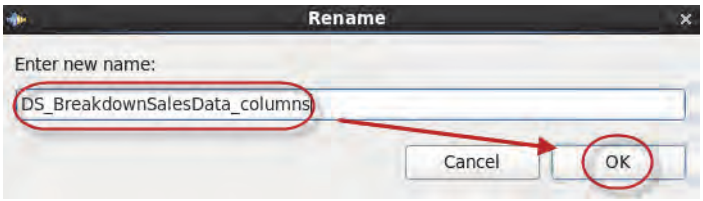
A customer might have data that contains values that can be used to assign a status value, but the values are not in a format that is used by a widget. In this example, you learn how to create a helper assembly line to supplement a primary assembly line with corresponding status values. The helper assembly line creates and populates a data set column that contains the corresponding status data.

You use the **DS_xxx_columns** helper assembly line template, which includes a set of template scripts that define data set columns.

You must complete the following tasks to complete this exercise:

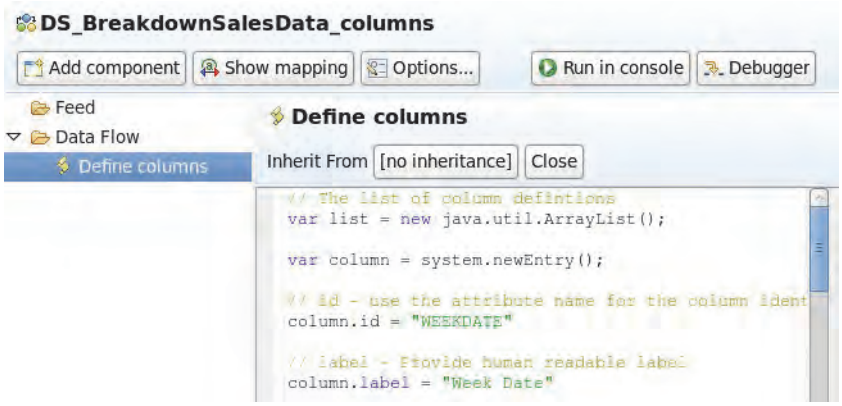
- **Import example assembly line project file**: You import a Tivoli Directory Integrator project file into the configuration editor.
- **Create a helper assembly line to create the data set columns**: This assembly line creates output columns for the data set and populates the columns with data.
- **Add a status threshold evaluation script to the primary assembly line**: You add a script to the data flow section of the assembly line that evaluates threshold information from the retrieved data and populates the Status column.
- **Test assembly lines with widgets in a dashboard page**: You update the table widget in the page that you modified in Unit 3, Exercise 1 on page 3- 1. The update picks up the new status information. You then test the page to make sure that the parameter configuration still works properly.

1. Close any open assembly lines in the configuration editor. Click the **X** symbol in the tabs at the top of the editor workspace.
2. Copy the DS_xxx_columns assembly line from the DASH_AssemblyLine_Examples project into the DASH_Workshop1 project. Expand **DASH_AssemblyLine_Examples > AssemblyLines** in the **Navigator** section.
3. Right-click **DS_xxx_columns** and select **Copy**.
4. Expand **DASH_Workshop1**, right-click AssemblyLines, and select **Paste**.
5. Rename the copied assembly line. Right-click **DASH_Workshop1 > AssemblyLines > DS_xxx_columns** and select **Rename**.
6. Enter **DS_BreakdownSalesData_columns** in the name field and click **OK**.



7. Open the new assembly line in the configuration editor. Double-click **DASH_Workshop1 > AssemblyLines > DS_BreakdownSalesData_columns**.

8. Open the **Define columns** script in the Data Flow section. Click **Define columns**.

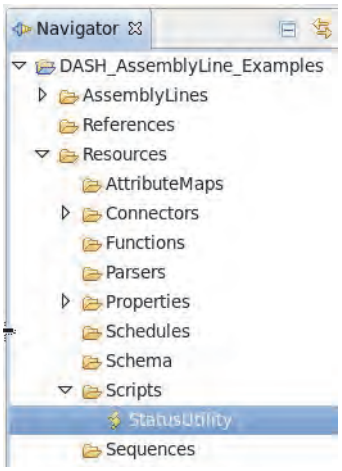


9. Replace the script contents with a copy of a script that was prepared for this exercise. Select the **Applications > Accessories > gedit Text Editor** menu in the dash151 desktop.

10. Select **File > Open** files and open the file **/opt/labfiles/tdi/status_HandsOn/BreakdownSalesData_columns.js**.

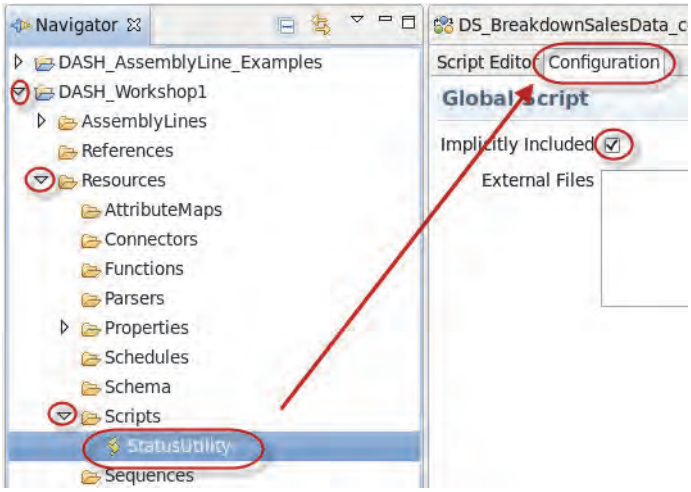
11. Use the copy and paste functions to copy the contents of this file and paste it in place of the **Define columns** script in the assembly line.

12. Add a global script to the assembly line resources. This script is called from the primary assembly line when the status value is calculated. Expand **DASH_AssemblyLine_Examples > Resources > Scripts** in the Navigator, right-click StatusUtility, and select **Copy**.



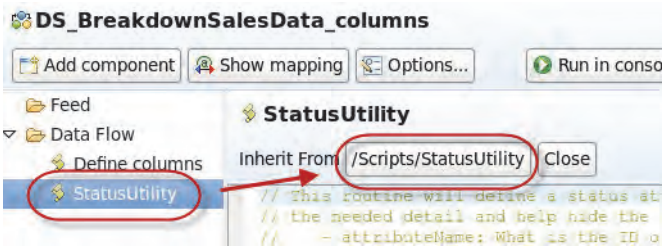
13. Expand **DASH_Workshop1 > Resources > Scripts**, right-click **Scripts**, and select **Paste**.

14. Change the script properties so that the script is globally available within the project. Double-click **DASH_Workshop1 > Resources > Scripts > StatusUtility**, click the **Configuration** tab in the workspace, and select **Implicitly Included**.



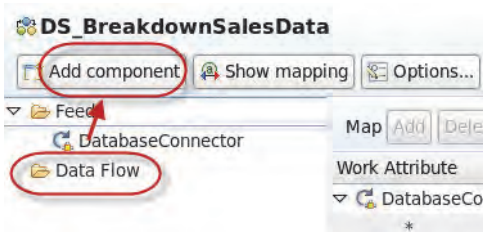
15. Copy the StatusUtility script from the DASH_Workshop1 > Resources > Scripts folder and place the copy in the DS_BreakdownSalesData_columns Data Flow section.

16. Click **StatusUtility** in the **Data Flow** section and verify that the **Inherit From** value is set to **/Scripts/StatusUtility**.

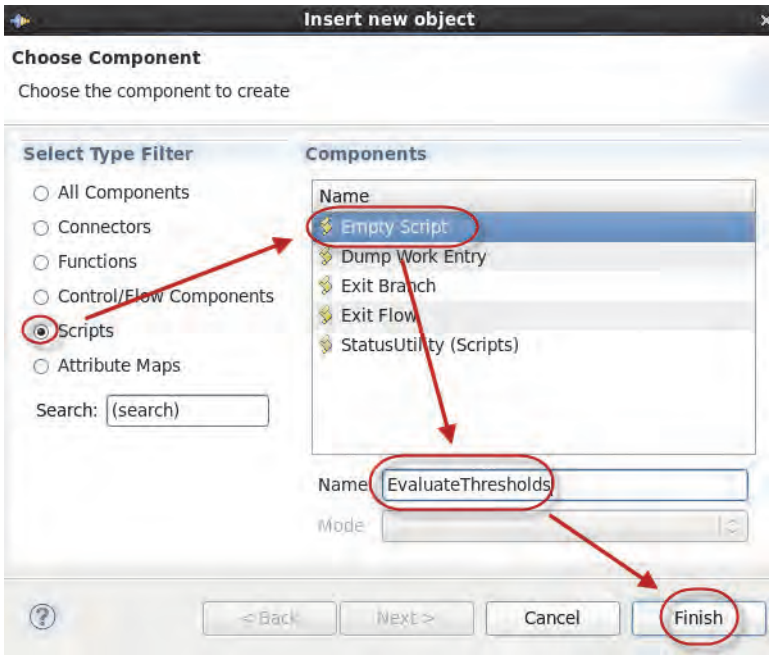


17. Add a script to the DS_BreakdownSalesData AssemblyLine that evaluates the status thresholds. Double-click **DASH_Workshop1 > AssemblyLines > DS_BreakdownSalesData** in the Navigator section to open the DS_BreakdownSalesData assembly line in the workspace.

18. Click **Data Flow** in the **DS_BreakdownSalesData** workspace and select **Add Component**.

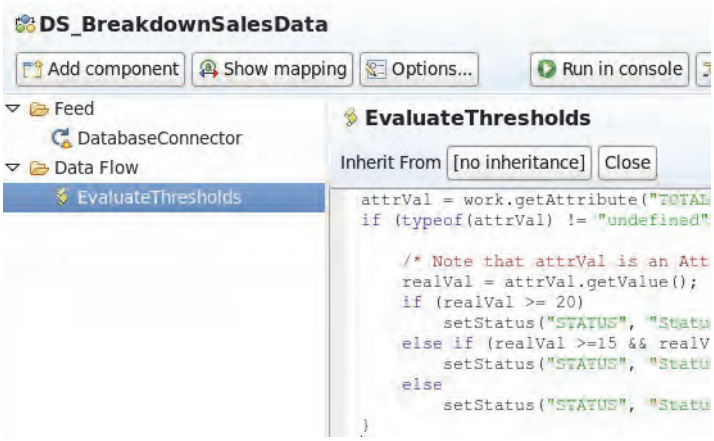


19. Select **Scripts** in the **Select Type Filter** column, select **Empty Script** in the **Name** list, enter **EvaluateThresholds** in the **Name** field, and click **Finish**.



20. Open the file **/opt/labfiles/tdi/status_HandsOn/BreakdownSalesData_status.js** file in the **Applications > Accessories > gedit Text Editor** application.

21. Copy the contents of the file to the EvaluateThresholds data flow script. Switch to the configuration editor, click **EvaluateThresholds** in the **Data Flow** section and paste the copied file into the blank script editor.



22. Save the updated project files. Click the **File > Save All** menu.

23. Add the updated assembly line to the runtime server. Drag the **DASH_Workshop1** project in the **Navigator** to the **Default** server entry in the **Servers** section.

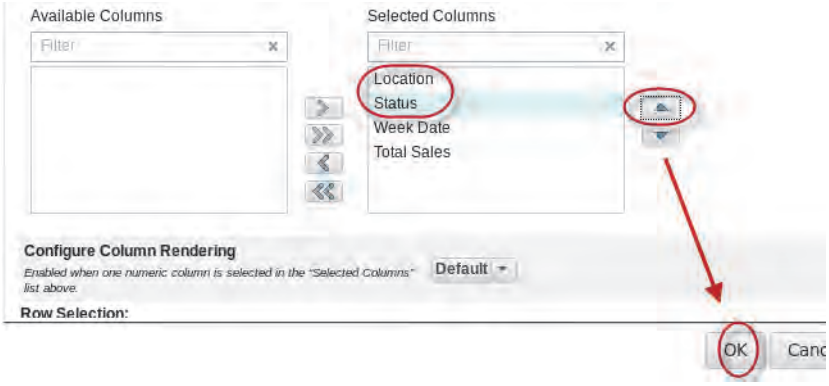
24. The Runtime Configuration window opens. Use the default values and click **Finish**.

25. Click **OK** to confirm the project overwrite.

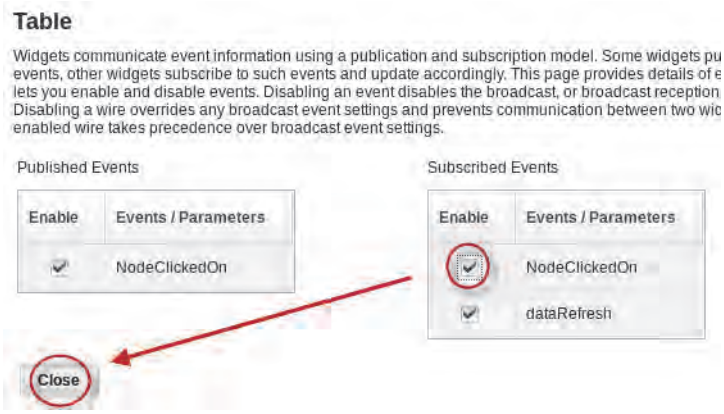
26. Refresh the Tivoli Directory Integrator cache connection with the DASH server. Open and close the Tivoli Directory Integrator connection document, log off the DASH console, and login to the DASH console with the user ID **smadmin** and the password **object00**.

Test the assembly line with widgets in a dashboard page

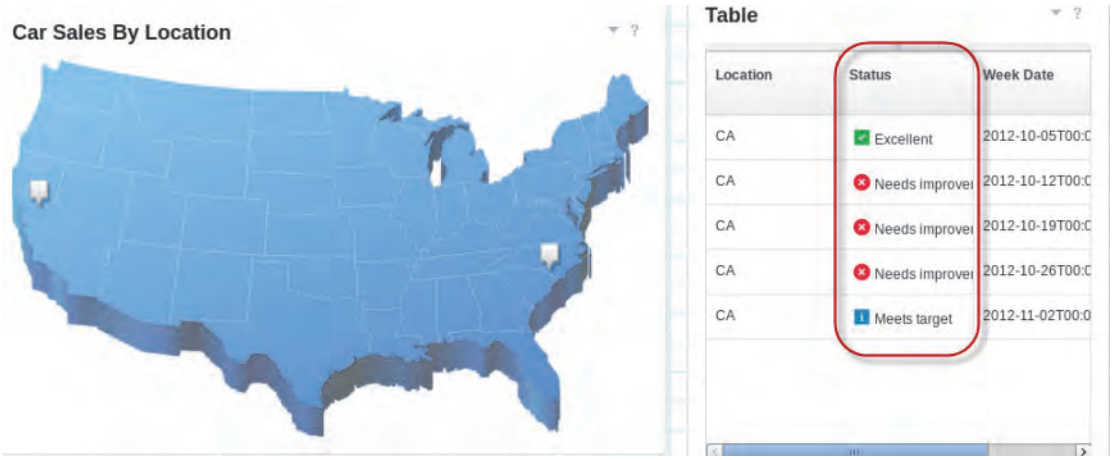
1. Edit the Sales By Location page.
2. Remove the table widget. Select **Edit options > Delete** in the table widget menu.
3. Drag a replacement table widget to the page canvas.
4. Edit the widget properties. Enter breakdown in the search field, click **Search**, and select BreakdownSalesData.
5. Edit the optional settings, change the order of the selected columns so that **Location** and **Status** are at the top of the list, and click **OK**.

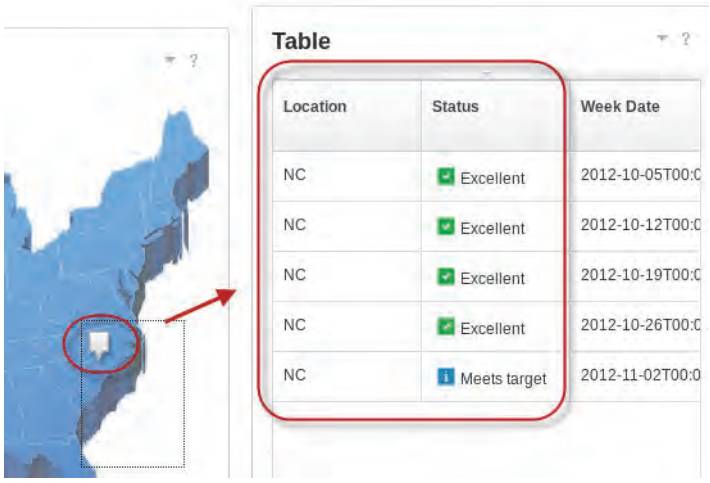


6. Configure the table widget to receive NodeClickedOn events. Select the **Edit options > Events** menu, select **NodeClickedOn** in the **Subscribed Events** table, and click **Close**.



You see the status column and corresponding status values in the table widget. Click a location hotspot and see the table values change correspondingly.





Exercise 3 Using a helper assembly line to add menu tasks to a data set

Executives must be able to easily find the sales manager with responsibility for the sales in each office. In this exercise, you need to add a widget right-click task that opens a page to show the resource owner contact information.

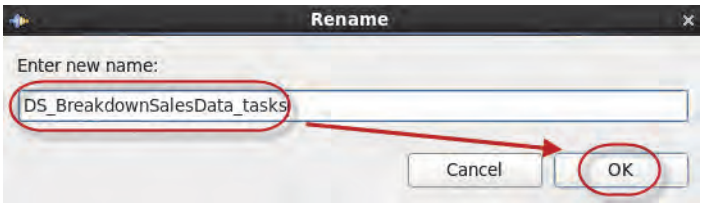
You must complete five major tasks to complete this exercise:

- **Copy the tasks assembly line template into your project**: Copy the example DS_xxx_tasks assembly line from the example project into the DASH_Workshop1 project.
- **Modify the tasks script in the Data Flow section of the new assembly line**: Modify the example script to open a page in your DASH console.

- **Update the project in the runtime server**: Add the new assembly line to the Tivoli Directory Integrator runtime server.
- **Test the task in a dashboard page**: Open a page that contains a widget that uses the configured assembly line. You test the right-click function that you configured in the assembly line.

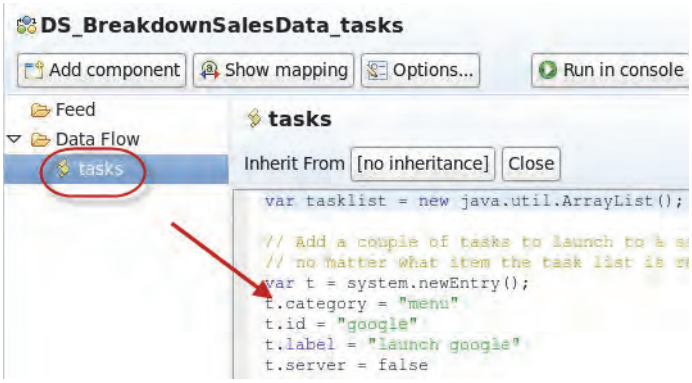
Copying the tasks assembly line template into your project

1. Close all open assembly lines in the configuration editor workspace. Click the **X** symbol in the tabs at the top of the workspace.
2. Copy the DS_xxx_tasks assembly line from the DASH_AssemblyLine_Examples project into the DASH_Workshop1 project. Expand **DASH_AssemblyLine_Examples > AssemblyLines** in the **Navigator** section.
3. Right-click **DS_xxx_tasks** and select **Copy**.
4. Expand **DASH_Workshop1**, right-click AssemblyLines, and select **Paste**.
5. Rename the copied assembly line. Right-click **DASH_Workshop1 > AssemblyLines > DS_xxx_tasks** and select **Rename**.
6. Enter **DS_BreakdownSalesData_tasks** in the name field and click **OK**.



Modifying the tasks script in the Data Flow section

1. Open the assembly line. Double-click **DASH_Workshop1 > AssemblyLines > DS_BreakdownSalesData_tasks**.
2. Click **tasks** in the **Data Flow** section to open the template script.



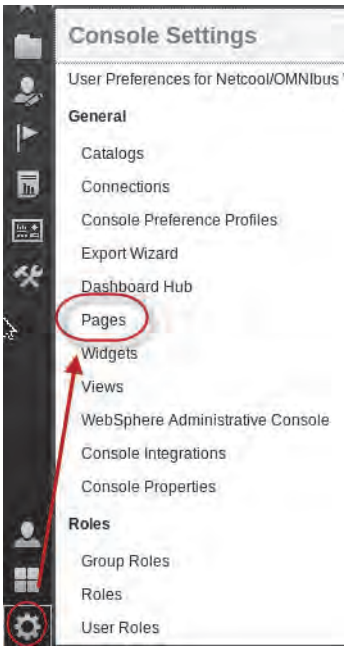
The script includes the code for several sample task entries. In a customer environment, you would modify the file as needed. For simplicity, you modify only one entry for this exercise.

3. Scroll down until you see the following comment:

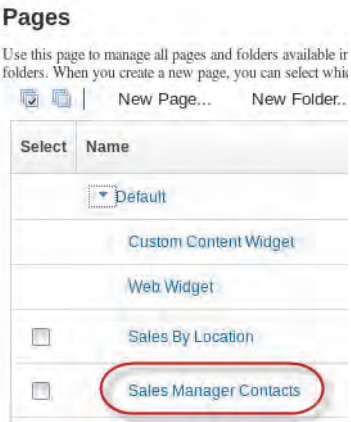
```
// This is an example task that will navigate the user to another page in DASH.
```

You modify the first entry under this comment section.

4. You need to modify the **t.pageID** value and the **t.label** value. The page ID value is unique to each DASH installation. Identify the page ID value for the Sales Manager Contacts page. This page was created during the laboratory setup.
   a. Switch to the DASH console, logged on as the user ID **smadmin** and the password **object00**.
   b. Select **Console Settings > General > Pages** in the taskbar on the left of the console.



   c. Expand the **Default** folder and click **Sales Manager Contacts**.



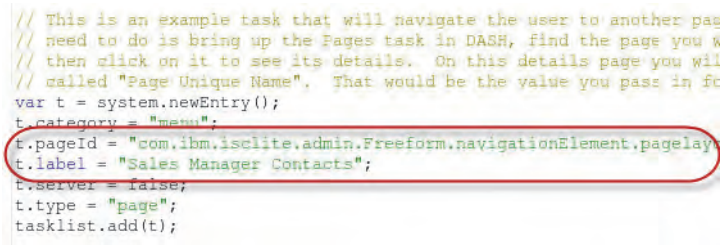
   The page properties page opens.

d. Look for the parameter name **Page unique name**. Copy the entire string, which does not show completely in the form field. Triple-click the **Page unique name** field to highlight the entire string and enter the **Ctrl + C** key combination to copy to the editor buffer.

Important: The unique page name value will change between DASH console deployments. If you export a custom right-click launch task from a test environment and then import the project into a production environment, you must modify the value of the task target.

5. Find the t.pageID entry in the tasks script under the previously referenced comment line. Highlight everything between the two quotation marks and enter the key sequence **Ctrl + V** to paste the copied string.

6. Modify the **t.label** parameter value to **“Sales Manager Contacts”**.



7. Save the updated assembly line. Click the **File > Save All** menu in the upper left of the configuration editor.

Update the project in the runtime server

1. Add the updated assembly line to the runtime server. Drag the **DASH_Workshop1** project in the **Navigator** to the **Default** server entry in the **Servers** section.

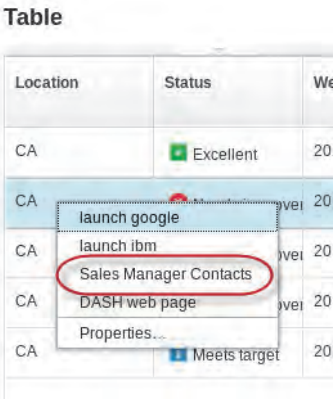
2. The Runtime Configuration window opens. Use the default values and click **Finish**.

3. Click **OK** to confirm the project overwrite.

4. Refresh the Tivoli Directory Integrator cache connection with the DASH server. Open and close the Tivoli Directory Integrator connection document, log off the DASH console, and login to the DASH console with the user ID **smadmin** and the password **object00**.

Testing the task in a dashboard page

1. Open the **Default > Sales By Location** page in the DASH console.
2. Right-click any row in the table widget to show the task menu and select **Sales Manager Contacts**.



The contact information page is opened in the console.

Exercise 4 Creating a run-once assembly line to generate metric data

Run-once assembly lines are useful when you always need to return a single value, regardless of the data source. In this exercise, you create and use a run-once assembly line. A customer periodically stores metric data to a comma-delimited CSV file. When a series of zeros are placed in the file, the previous entry is used to provide a critical metric. You must create a run-once assembly line that scans the file and evaluates the most up-to-date metric.

The DS_Run_Once template consists of an empty Feed section and several components in the Data Flow section. There is no connector in the Feed section because the assembly line iterates through a set of data only one time. The Data Flow section consists of the following components:

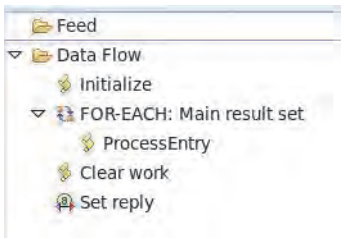
Initialize: This script consists of a code snippet that prevents the assembly line from entering an infinite loop when it evaluates a data set. You also create variables that are used later to summarize values from the iteration of the complete data set.

FOR-EACH: Main result set: This object is similar to a Feed connector. It connects to a data source and reads configured data.

ProcessEntry: This script iterates and evaluates data records, one record at a time.

Clear work: Because of the script logic, the final work object entry is always zero. This script clears the work object so that it can be assigned the variables from the ProcessEntry script.

Set reply: This section is used to assign the final work object output variables.



To complete this exercise, you must complete the following high-level tasks:

Create a run-once assembly line from a template: Copy and configure an assembly line from the DS_Run_Once example.

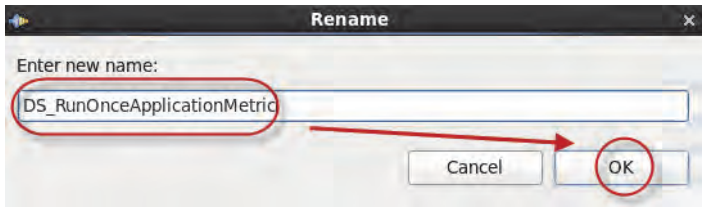
Creating the Process Entry script: The script evaluates a row of data. If the values are all zero, the script is ended. If the values are not all zero, store the current row for use in the Set reply section. When the processing loop ends, the stored values are the last nonzero values in the file.

Configure the assembly line output work object: Configure the Set reply section to use the last nonzero values as the assembly line metrics.

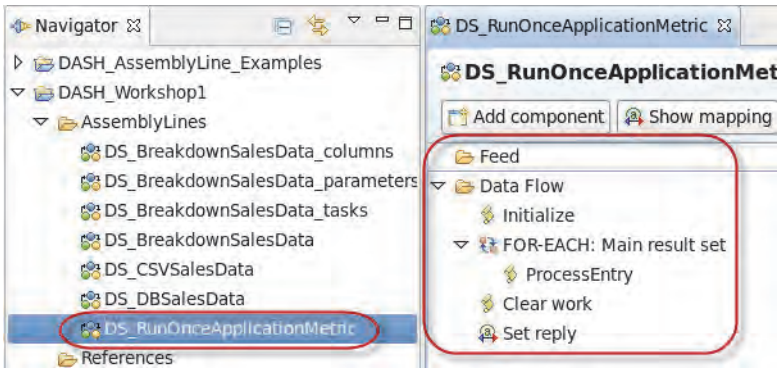
Test the completed assembly line with in a dashboard page: Create a page that shows the calculated metric in a table and status gauge.

Creating a run-once assembly line from a template

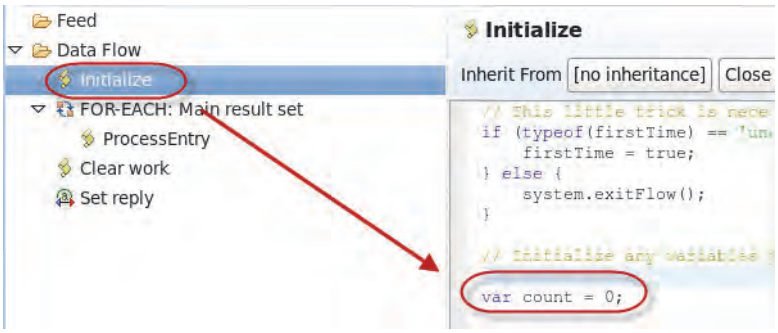
1. Close all open assembly lines in the configuration editor workspace. Click the **X** symbol in the tabs at the top of the workspace.
2. Copy the DS_Run_Once assembly line from the DASH_AssemblyLine_Examples project into the DASH_Workshop1 project. Expand **DASH_AssemblyLine_Examples > AssemblyLines** in the **Navigator** section.
3. Right-click **DS_Run_Once** and select **Copy**.
4. Expand **DASH_Workshop1**, right-click AssemblyLines, and select **Paste**.
5. Rename the copied assembly line. Right-click **DASH_Workshop1 > AssemblyLines > DS_Run_Once** and select **Rename**.
6. Enter **DS_RunOnceApplicationMetric** in the name field and click **OK**.



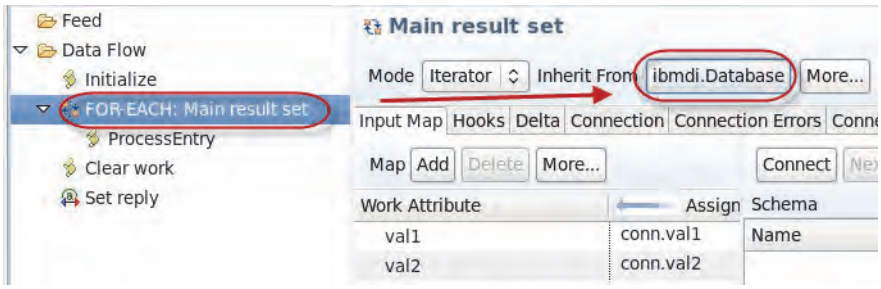
7. Open the new assembly line. Double-click **DS_RunOnceApplicationMetric** in the Navigator.



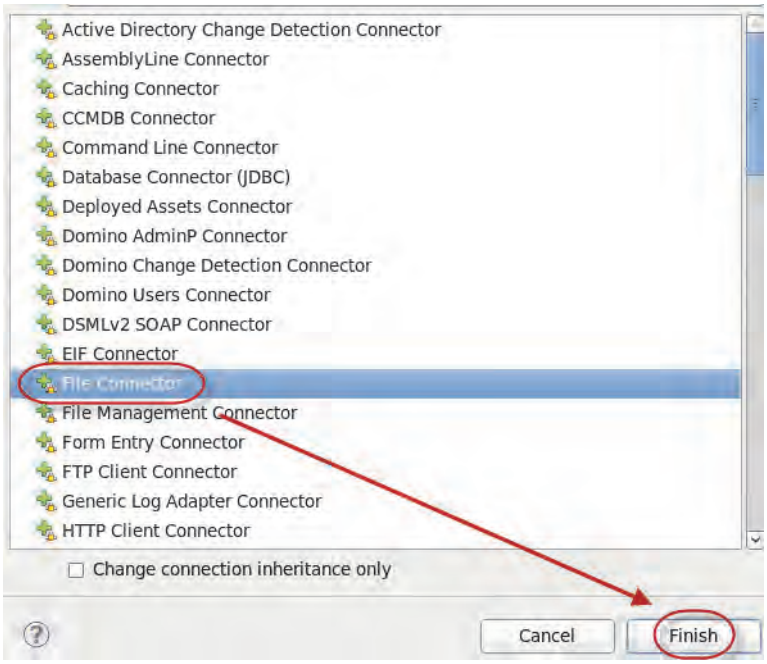
8. Modify the Initialize script to create a variable to count the number of rows that are read in the CSV file. Click **Initialize** in the **Data Flow** section and replace **var sum = 0;** with **var count = 0;**.



9. Modify the **FOR-EACH: Main result set** connector loop in the Data Flow section. By default, the connector is configured as a database connector. Click **FOR-EACH: Main result set** and click the icon to the right of **Inherit From**.

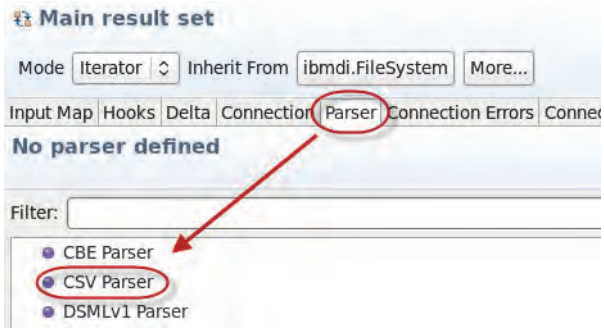


10. Click **File Connector** in the list and click **Finish**.

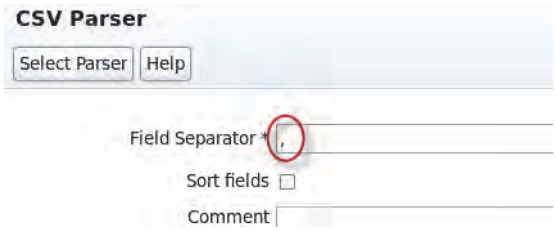


11. Configure the file connector. Click **Connection** and enter **/opt/labfiles/tdi/runonce_HandsOn/test.fil** in the **File Path** field.

12. Configure how the CSV file is parsed. Click the **Parser** tab and select **CSV Parser**.

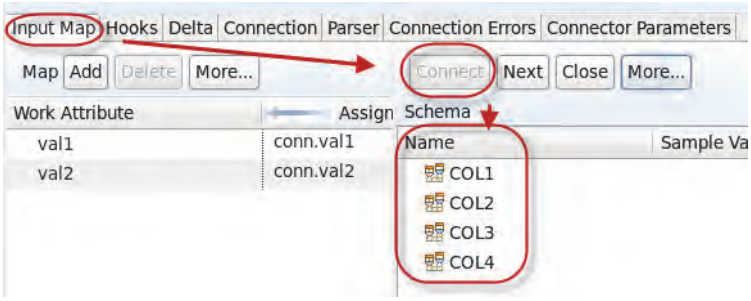


13. Enter the comma character (**,**) in the **Field Separator** field.

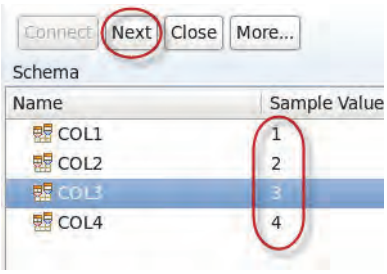


14. Add column schema information for the file. Click to expand the Advanced section and enter COL1,COL2,COL3,COL4 in the Field Names field. Do not put spaces in the text string.

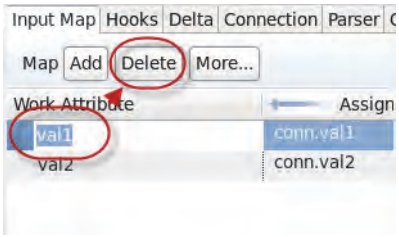
15. Test the connector. Click the **Input Map** tab and click **Connect**. You see the column headers in the Schema section.



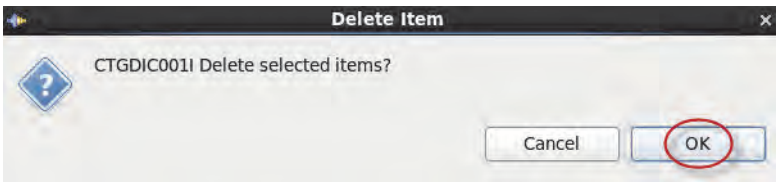
16. Test iteration through the target file. Click **Next** to see data in the **Sample Value** column.



17. Map the work attributes to the file data. Remove the work attribute values that are included withe the template file. Click **val1** in the **Work Attribute** column and click **Delete**.



18. Click **OK** to close the confirmation window.



19. Repeat the previous two steps to delete the **val2** entry in the **Work Attribute** column.

20. Create an input map. Click **Add** in the **Input Map** tab, select **COL1**, select **COL2**, and click **OK**.

Note: Technically, for this scenario, all columns should be mapped and tested for a zero value in the file. Two columns are used in this exercise to simplify the process.

Creating the Process Entry script

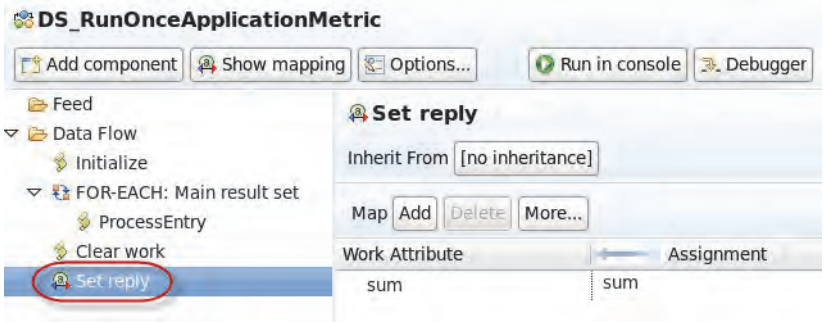
1. Replace the template ProcessEntry script. Click **ProcessEntry** in the **Data Flow** section.
2. Replace the template code in the Process Entry script with the following lines of code:

```
// Read in attributes from the current row in the CSV file
var1 = work.getAttribute("COL1");
var2 = work.getAttribute("COL2");
// If we have data
if (typeof(var1) != 'undefined' && typeof(var2) != 'undefined') {
// If we see a value of 0 and 0 we will consider this a marker to quit processing
   if (var1.getValue() == 0 && var2.getValue() == 0) {
// Remember to use two equal symbols (==)in the above line.
task.logmsg("found the end");
   system.exitBranch();
} else {
// Sum the data, and increment the row count
   count++;
   task.logmsg("Processing row " + count);
// Save the values for this record, so that when we hit the "0,0,0,0" marker
// we'll have the values from the row that precedes 0,0,0,0
   lastvar1 = java.lang.Integer.parseInt(var1.getValue());
   task.logmsg("COL1 last value: " + lastvar1);
   lastvar2 = java.lang.Integer.parseInt(var2.getValue());
   task.logmsg("COL2 last value: " + lastvar2);
   }
}
```

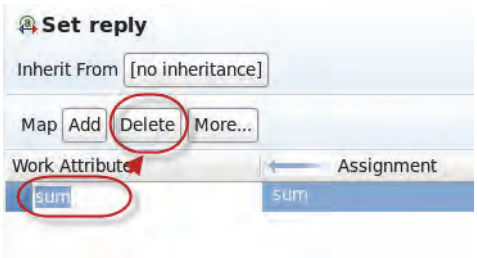
Note: A working version of the Process Entry script is available for copying and pasting. The script is found in **/opt/labfiles/tdi/runonce_HandsOn/ProcessEntry.js.**

Configure the assembly line output work object

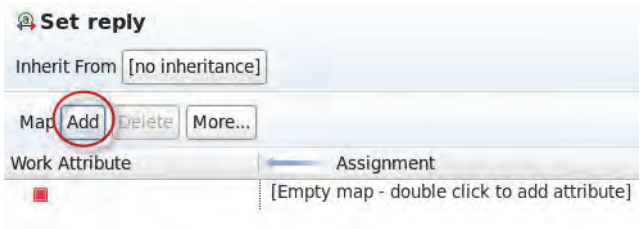
1. Click **Set reply** in the **Data Flow** section.



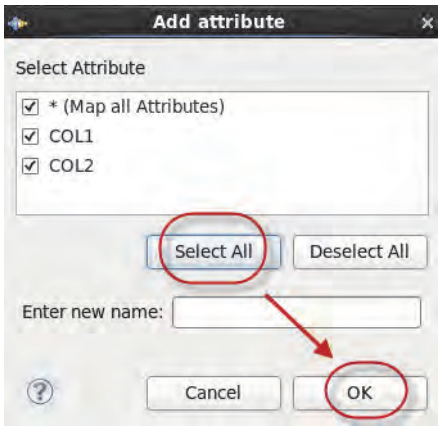
2. Remove the work attributes from the template copy. Click **sum** in the **Work Attributes** column and click **Delete**.



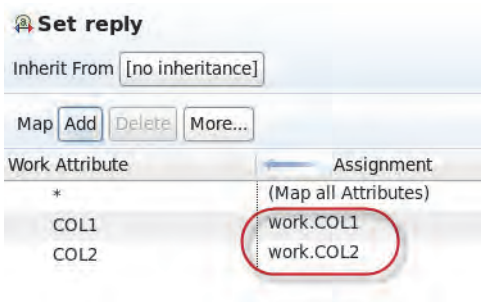
3. Add the final metric work attributes. Click **Add**.



4. Click **Select All** and click **OK**.

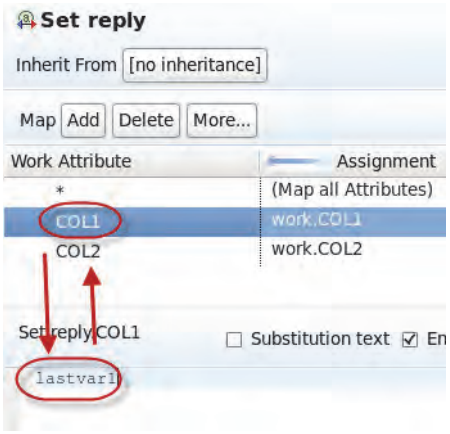


You see **COL1** mapped to **work.COL1** and **COL2** mapped to **work.COL2**.

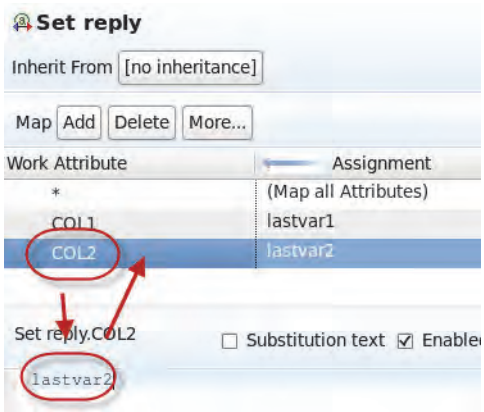


5. Change the mapping to the corresponding **lastvar** variables that were calculated in the **Process Entry** script. Double-click COL1 in the Work Attribute column and set the value to lastvar1.

6. Click **COL2** to see the COL1 update in the **Assignment** column.



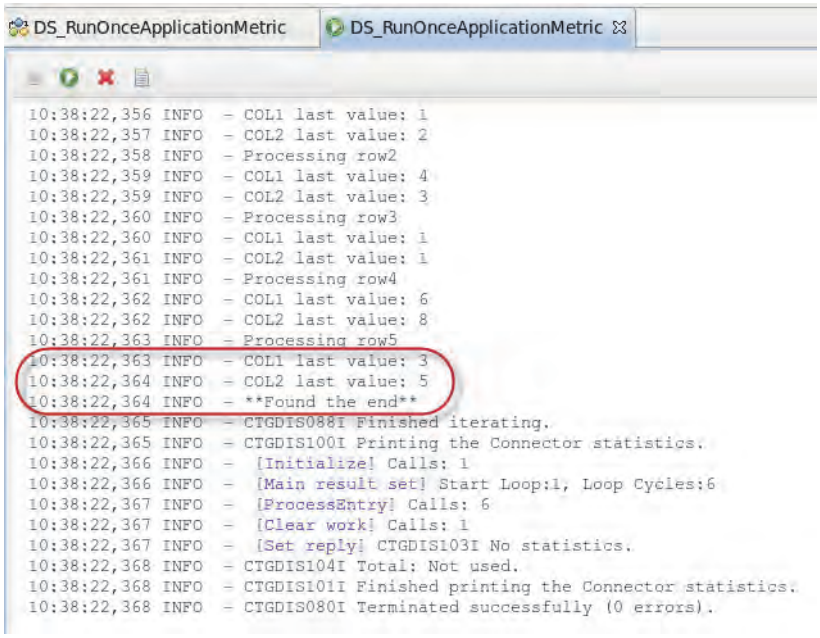
7. Change the **COL2** assignment to **lastvar2** and click the **COL2** row to see the update in the **Assignment** column.



8. Save the updated assembly lines. Select the **File > Save All** menu.

9. Test the assembly line. Click **Run in console**.

10. You see the processing output in a console window tab. Verify that the COL1 last value is 3 and the COL2 last value is 5.



11. Close the DS_RunOnceApplicationMetric tab that opened to show the console output. Do not close the **DS_RunOnceApplicationMetric** assembly line tab.

12. Select **File > Save All** to save all of the assembly lines.

13. Add the updated assembly line to the runtime server. Drag **DASH_Workshop1** to the **Default** entry in the **Servers** section.

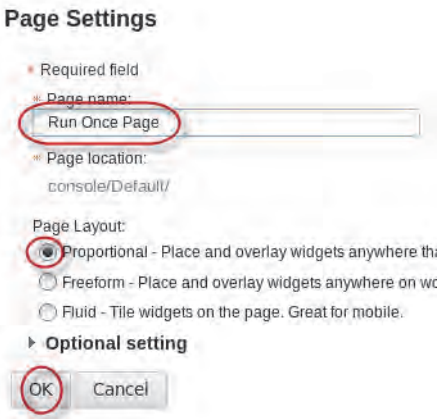
14. The Runtime Configuration window opens. Use the default values and click **Finish**.

15. Click **OK** to confirm the project overwrite.

16. Refresh the Tivoli Directory Integrator cache connection with the DASH server. Open and close the Tivoli Directory Integrator connection document, log off the DASH console, and login to the DASH console with the user ID **smadmin** and the password **object00**.

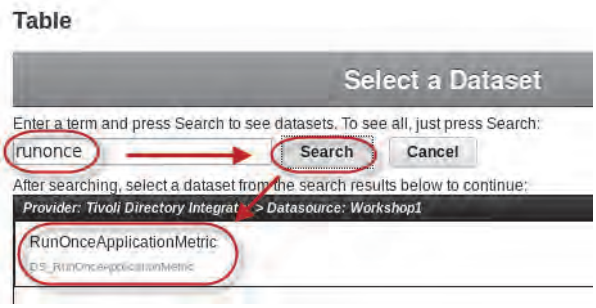
Test the completed assembly line with in a dashboard page

1. Create a page in the DASH server.
2. Enter **Run Once Page** in the **Page** name field, **Proportional** in the **Page Layout** section, and click **OK**.

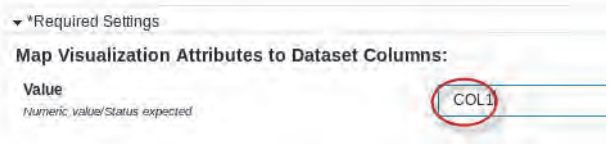


3. Drag a **Status Gauge** widget to the page canvas and move it to the upper left corner.
4. Drag a **Table** widget to the page canvas and move it to the upper right corner.

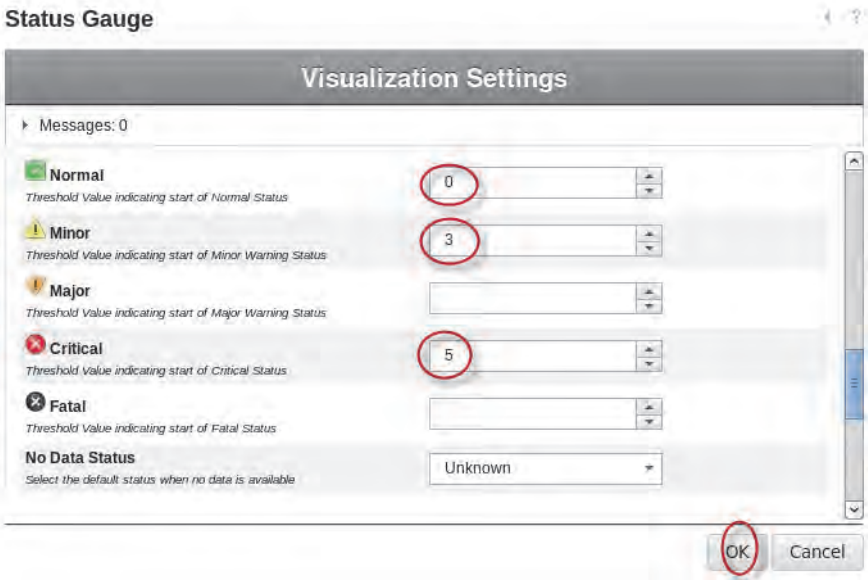
5. Edit the table widget properties. Enter **runonce** in the search field, click **Search**, and select **RunOnceApplicationMetric** in the list.



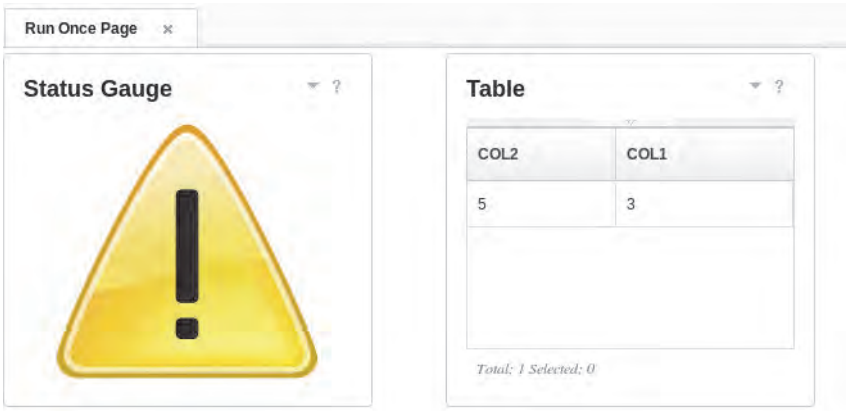
6. Use the default values for the remaining part of the widget configuration. Click **OK** to save the widget.
7. Edit the Status Gauge widget properties. Enter **runonce** in the search field, click **Search**, and select **RunOnceApplicationMetric** in the list.
8. Select **COL1** for the **Value** field.



9. Edit the **Optional settings** section, set the **Normal** threshold to **0**, the **Minor** threshold to **3**, the **Critical** threshold to **5**, and click **OK**.



10. You see the metric values represented in both widgets on the page.



Unit 4 DASH and Netcool/Impact integration exercises

The exercises in this unit provide hands-on demonstrations of how to use Netcool/Impact tools to serve data to DASH dashboard widgets with custom UI data provider data sets.

Exercise 1 Creating a Netcool/Impact data type data set

A Netcool/Impact data model consists of a *data source* and *data type*. A data source defines the connection parameters and target of a connection to a data source, such as a database or file. A data type defines the data that is available through the data source. For example, a data type might define a database table and how the columns in the table are queried. In this exercise, you create a Netcool/Impact data source that connects to a car sales database and you create a data type that queries the data source and returns a set of data. You enable the data type to function as a UI data provider to a DASH server.

You complete the following high-level tasks to complete this exercise:

Create a Netcool/Impact data source: You configure a connection to a car sales database.

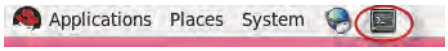
Create a Netcool/Impact data type: You configure a database query and enable the data type as a UI data provider.

Create a DASH connection document to the Netcool/Impact UI data provider: You configure a DASH connection document to connect to the Netcool/Impact data provider on the taddm153 virtual image.

Test the data type UI data provider: You create a dashboard page and use the data type UI data provider as the data set for a chart widget.

Starting the Netcool/Impact server

1. Switch to the **taddm153** virtual image desktop.
2. Click the **Terminal** icon at the top of the desktop to open a command window.



3. Enter the following command to start the Netcool/Impact server:

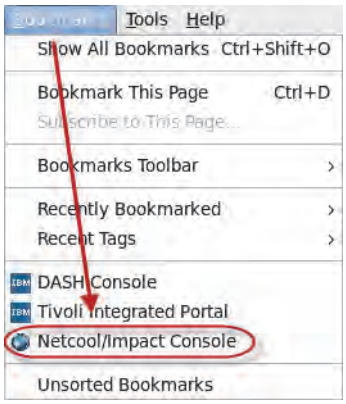
   `/opt/IBM/tivoli/impact/bin/startImpactServer.sh`

4. When the Netcool/Impact server is started, start the Netcool/Impact GUI server with the following command:

   `/opt/IBM/tivoli/impact/bin/startGUIServer.sh`

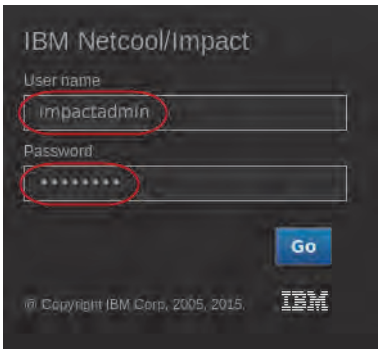
Creating a Netcool/Impact data source

1. Switch to the **dash151** virtual image desktop.
2. Open a second browser tab. Select **File > New Tab**.
3. The Netcool/Impact instance that you use in this exercise is the embedded instance on the taddm153 virtual image. There is a browser bookmark configured to connect to the application console. Select **Bookmarks > Netcool/Impact Console**.

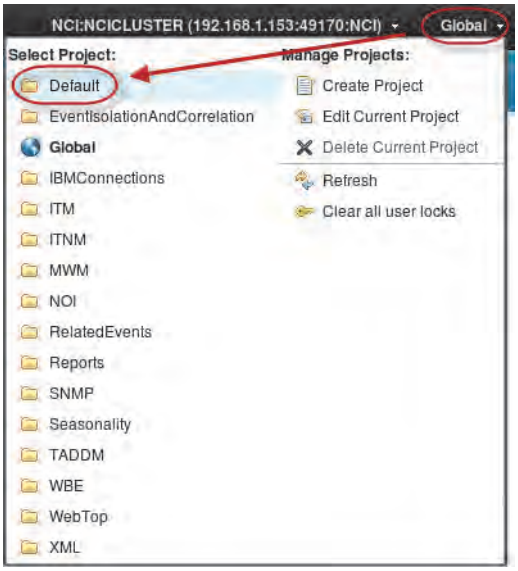


4. If you see a warning about an untrusted connection, click **I Understand the Risks**.
5. Click **Add Exception**.
6. Click **Confirm Security Exception**.

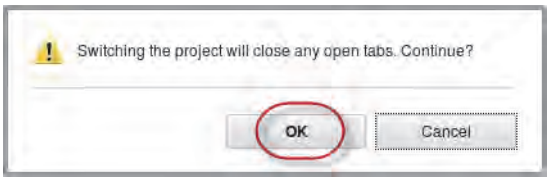
7. Log in to the Netcool/Impact console with the user ID **impactadmin** and the password **object00**.



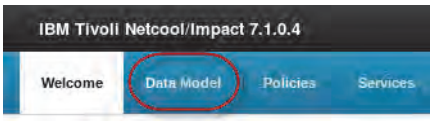
8. Switch to the Default project view. Click **Global** at the top of the console and select **Default** in the **Select Project** section.



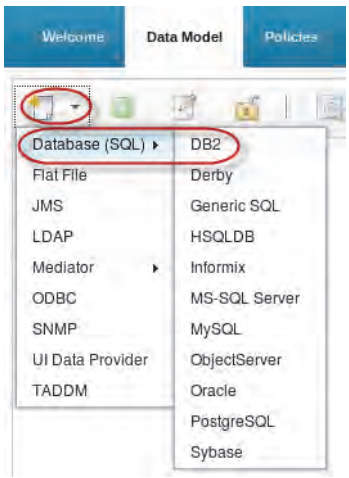
9. Click **OK** to confirm the project view switch.



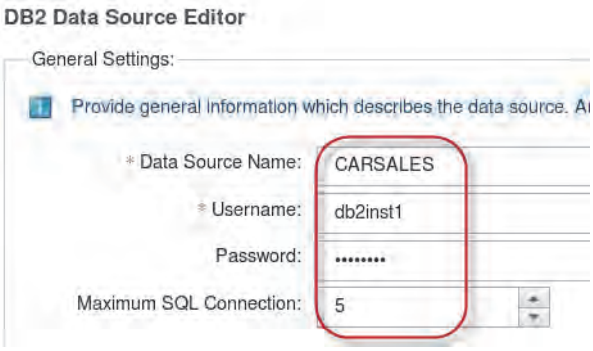
10. Click the **Data Model** tab at the upper left of the console.



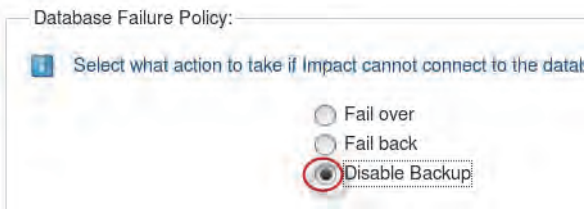
11. Click the **New Data Source** icon and select **Database (SQL) > DB2**.



12. Enter the following values in the **General Settings** section:
    a. Data Source Name: CARSALES
    b. Username: db2inst1
    c. Password: object00
    d. Maximum SQL Connection: 5



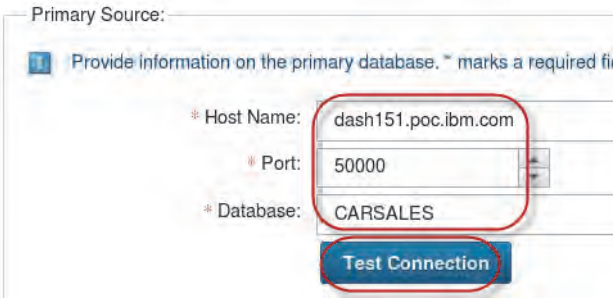
13. Select **Disable Backup** in the **Database Failure Policy** section.



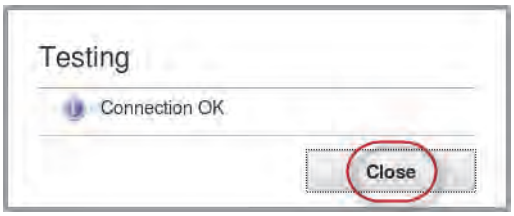
14. Enter the following values in the **General Settings** section:
    a. Host Name: dash151.poc.ibm.com
    b. Port: 50000

c. Database: CARSALES

15. Click **Test Connection** to verify that the properties specified are correct and that the database connection is working.

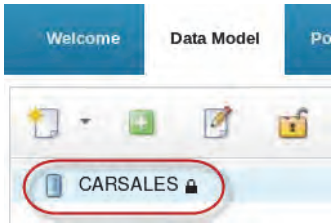


16. Click **Close** to close the Testing window.



17. Click the **Save** icon at the top of the form to save the data model configuration.

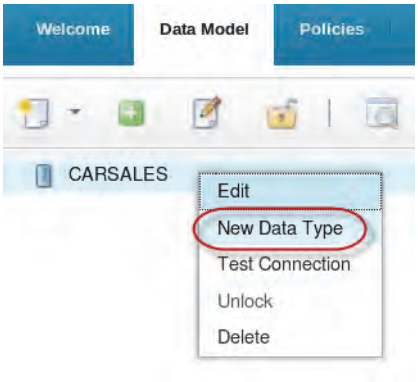
You see the new data source listed in the navigation section of the console. The lock icon to the right of the CARSALES entry is normal. The icon indicates that the configuration form is still open.



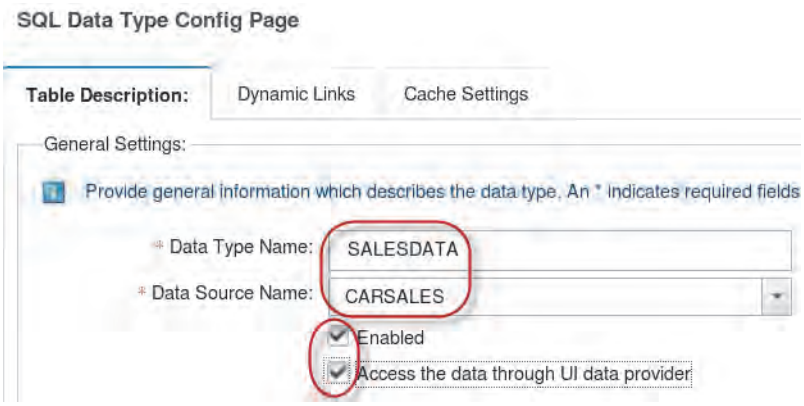
18. Close the data source configuration page. Click the **X** icon in the **CARSALES** tab in the console workspace.

Creating a Netcool/Impact data type

1. Right-click the **CARSALES** data source in the navigator list and select **New Data Type**.

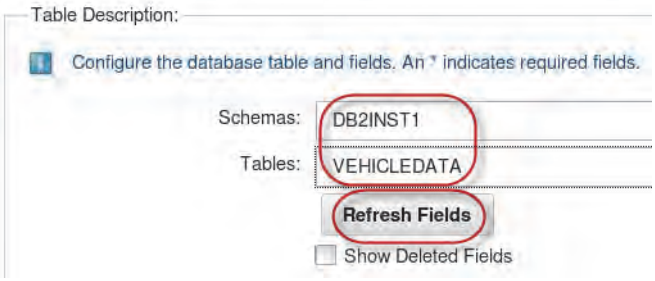


2. Enter the following values in the General Settings section:
   a. Data Type Name: SALESDATA
   b. Data Source Name: CARSALES
   c. Enabled: Selected
   d. Access the data through UI data provider: Selected



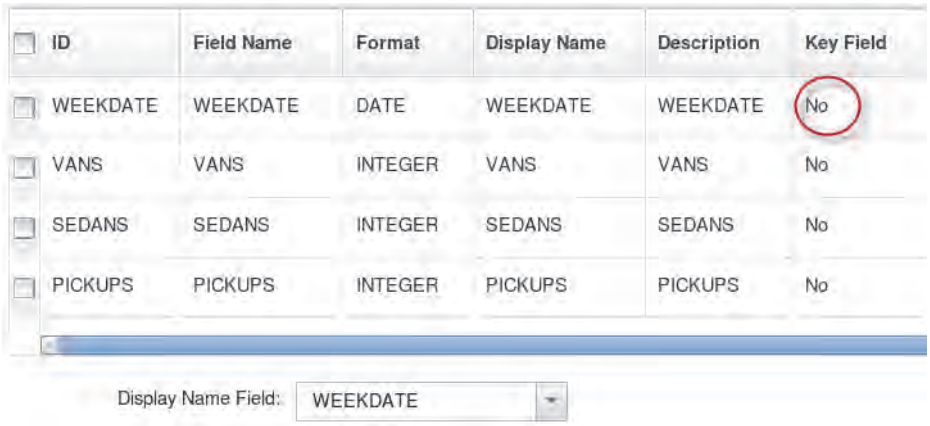
3. Enter the following values in the Table Description section:
   a. Schemas: DB2INST1
   b. Tables: VEHICLEDATA

4. Click the **Refresh Fields** to show the selected table configuration.



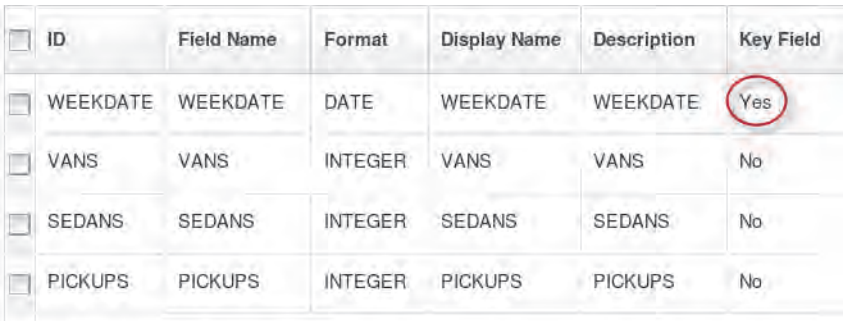
The schema information is retrieved from the data type and shown in the **Table Description** section.

5. Configure the WEEKDATE column as a database key field. Double-click **No** in the **Key Field** column in the WEEKDATE row.



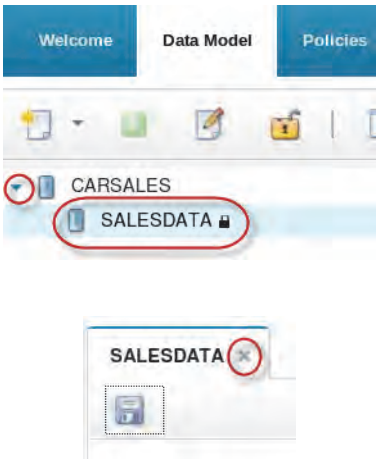
6. The table cell changes to show a selection box. Click the selection box.

7. Click the **Key Field** header and you see the WEEKDATE key field cell change to **Yes**.

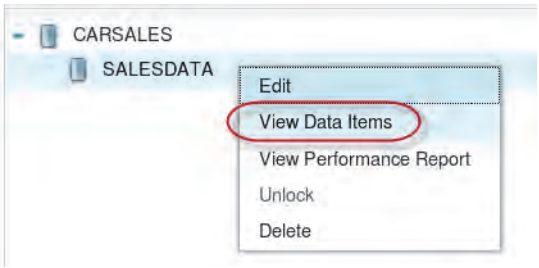


8. Save the data type configuration. Click the **Save** icon at the top of the form.

9. You see the new SALESDATA data type in the navigation section of the console. Close the SALESDATA data type form. Click the **X** icon in the **SALESDATA** tab in the console workspace.



10. Verify the data type data. Right-click the new data type and select **View Data Items**.



You see the retrieved data rows from the database table.

Data Type Name: SALESDATA

Number of Objects: 5

Select	WEEKDATE	VANS	SEDANS	PICKUPS
☐	2012-10-05	5	10	8
☐	2012-10-12	3	8	11
☐	2012-10-19	4	7	10
☐	2012-10-26	5	8	7
☐	2012-11-02	2	9	8

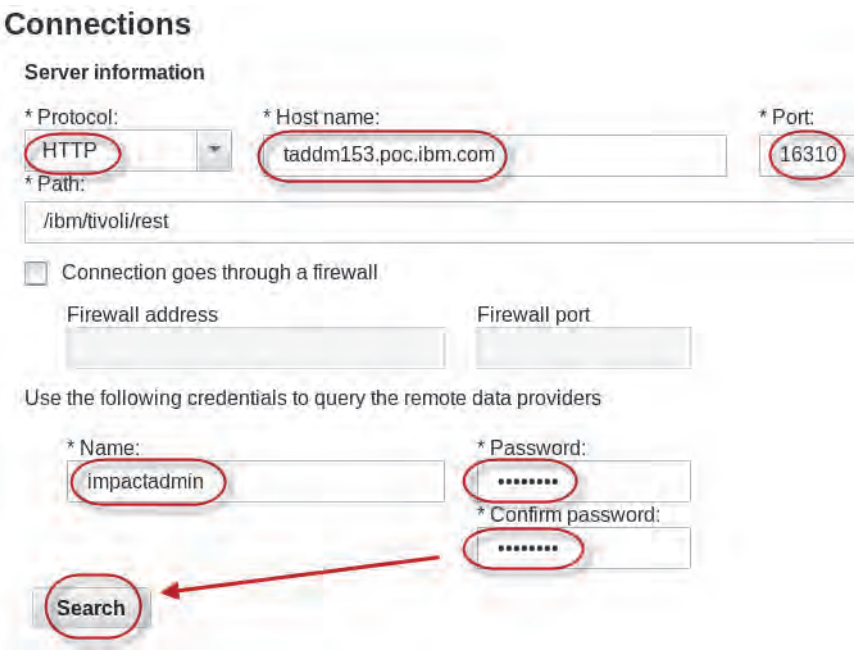
Delete

11. Click the X icon in the **Data Items: SALESDATA** tab in the Netcool/Impact console workspace to close the workspace page.

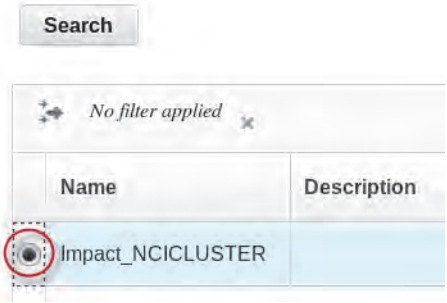
12. Do not log off the Netcool/Impact console.

Creating a DASH connection document to the Netcool/Impact UI data provider

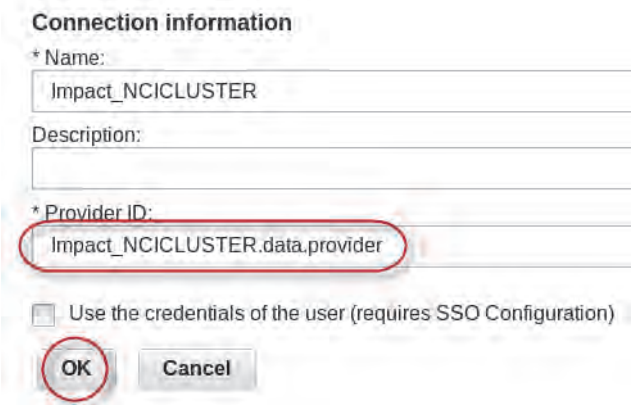
1. Switch to the DASH console tab in your browser. If you are not logged on to the console, login with the user ID **smadmin** and password **object00**.
2. Select the **Console Settings > General > Connections** task from the taskbar on the left of the console.
3. Click the **Create new remote provider** icon.
4. Enter the following parameters:
   a. Protocol: HTTP
   b. Host Name: taddm153.poc.ibm.com
   c. Port: 16310
   d. Path: /ibm/tivoli/rest
   e. User Id: impactadmin
   f. Password: object00
5. Click **Search**.



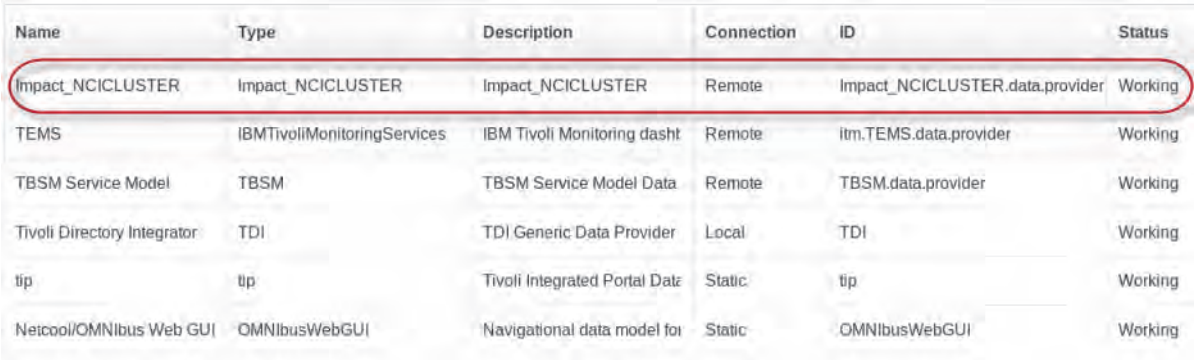
6. You see the available UI data provider on the taddm153 virtual image. Select **Impact_NCICLUSTER**.



7. Change the default Provider ID value to make it more portable. Enter **Impact_NCICLUSTER.data.provider** in the **Provider ID** field and click **OK**.



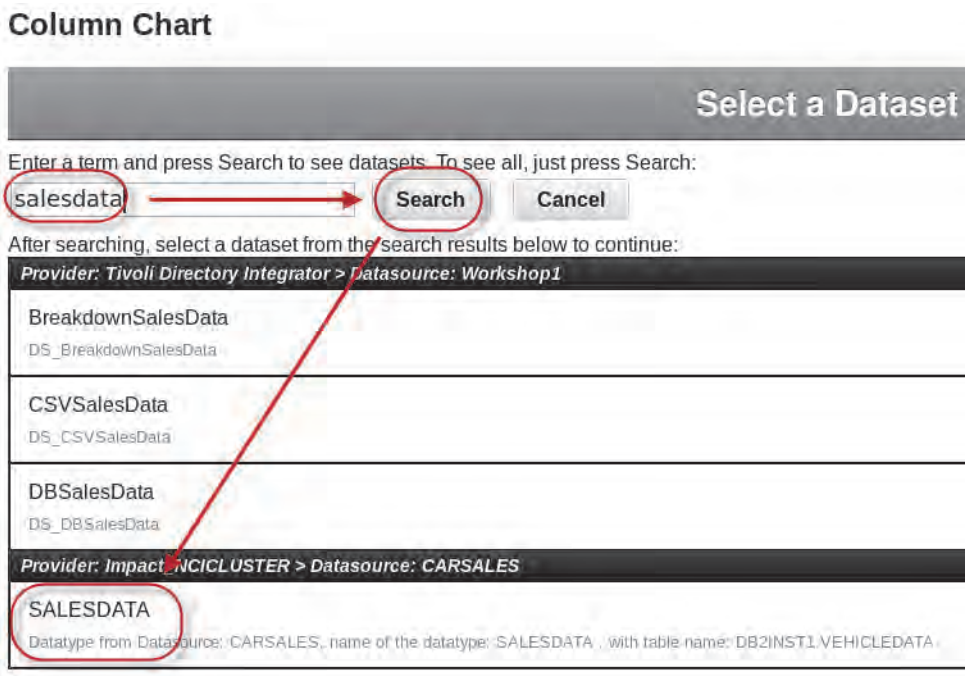
8. Verify that the Netcool/Impact connection status is **Working**.



Name	Type	Description	Connection	ID	Status
Impact_NCICLUSTER	Impact_NCICLUSTER	Impact_NCICLUSTER	Remote	Impact_NCICLUSTER.data.provider	Working
TEMS	IBMTivoliMonitoringServices	IBM Tivoli Monitoring dasht	Remote	itm.TEMS.data.provider	Working
TBSM Service Model	TBSM	TBSM Service Model Data	Remote	TBSM.data.provider	Working
Tivoli Directory Integrator	TDI	TDI Generic Data Provider	Local	TDI	Working
tip	tip	Tivoli Integrated Portal Data	Static	tip	Working
Netcool/OMNIbus Web GUI	OMNIbusWebGUI	Navigational data model for	Static	OMNIbusWebGUI	Working

Create the dashboard page with the Netcool/Impact Data Type data set

1. Create a page. Click the new page plus symbol (**+**) in the upper right of the console.
2. Enter **Impact Data Type Page** in the **Page Name** field, use **console/Default** for the **Page Location**, select **Proportional** for the **Page Layout** mode, and click **OK**.
3. Drag a column chart widget object to the page canvas.
4. Edit the widget properties. Select **Edit options > Edit** in the upper right of the widget.
5. Enter **salesdata** in the search field and select search. SALESDATA is the name of the data type that you created in the task Creating a Netcool/Impact data type.
6. Click **SALESDATA** below the **Impact_NCICLUSTER** provider.

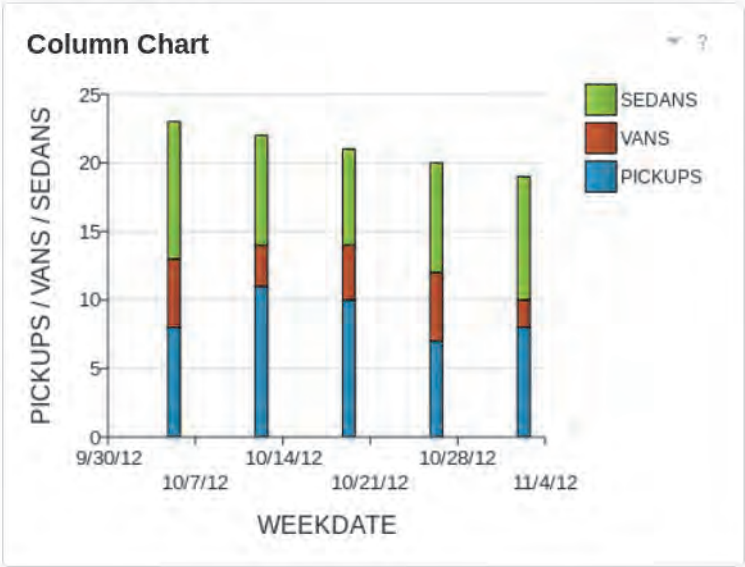


7. Configure the chart. Select WEEKDATE in the **X axis field** values.
8. Select PICKUPS, SEDANS, and VANS for the **Y axis value field** values.

9. Scroll down in the **Optional Settings** section and select **Side** in the **Legend** field and click **OK**.

10. Click **Save and Exit** to save the page.

    The database column chart is now shown in the widget.



11. Do not log off the DASH console.

Exercise 2 Adding status information to a data type UI data provider

In this exercise, you modify the SALESDATA data type to evaluate the sales data and add a corresponding status column to the data set.

1. Switch to the Netcool/Impact console tab in your browser. If you are not logged on to the console, select **Bookmarks > Netcool/Impact Console** in the browser menus and login with the user ID **impactadmin** and the password **object00**.

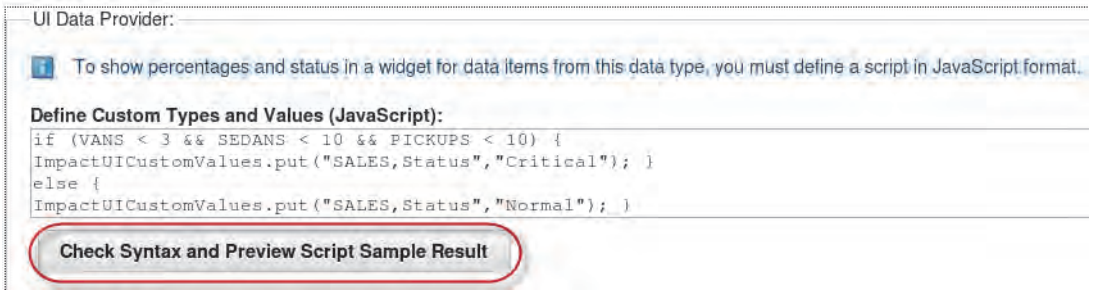
2. Open the SALESDATA data type.
   a. Click the **Data Model** tab at the top of the console.
   b. Expand the CARSALES data source and double-click **SALESDATA**.

3. Scroll to the bottom of the SALESDATA configuration page and enter the following lines of JavaScript code in the **Define Custom Types and Values** field in the **UI Data Provider** section:

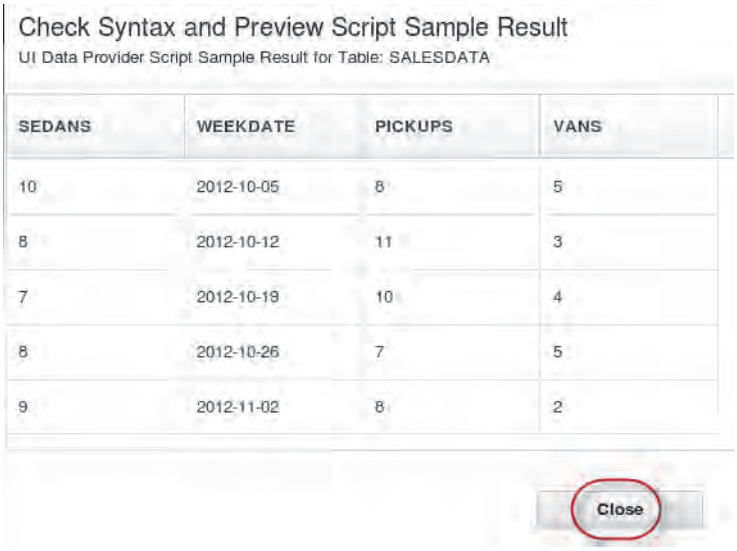
```
if (VANS < 3 && SEDANS < 10 && PICKUPS < 10) {
ImpactUICustomValues.put("SALES,Status","Critical"); }
else {
ImpactUICustomValues.put("SALES,Status","Normal"); }
```

Important: The status value must be capitalized.

4. Click the **Check Syntax and Preview Script Sample Result** icon to verify that the script was created correctly.



5. You see the data type columns in a pop-up window. Click **Close** to close the window.



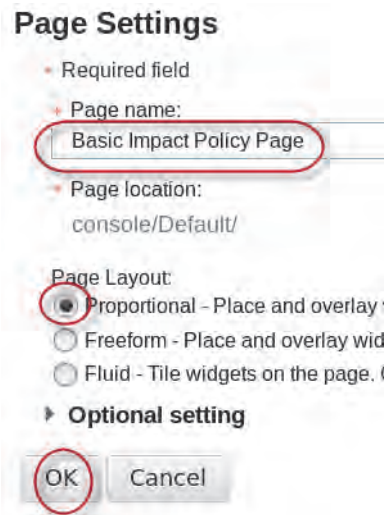
Important: There is a bug in the version of Netcool/Impact software that is installed on the taddm153 image. When you preview the data at this point in the exercise, you should also see the SALES column with the calculated status values. However, the SALES column data is being properly created and is available in the Netcool/Impact data set. You may proceed with the exercise and successfully use the status values in a DASH widget.

6. Save the updated data type configuration. Click the **Save** icon at the top of the page.
7. Close the SALESDATA data type configuration form. Click the **X** icon in the SALESDATA tab in the console workspace.

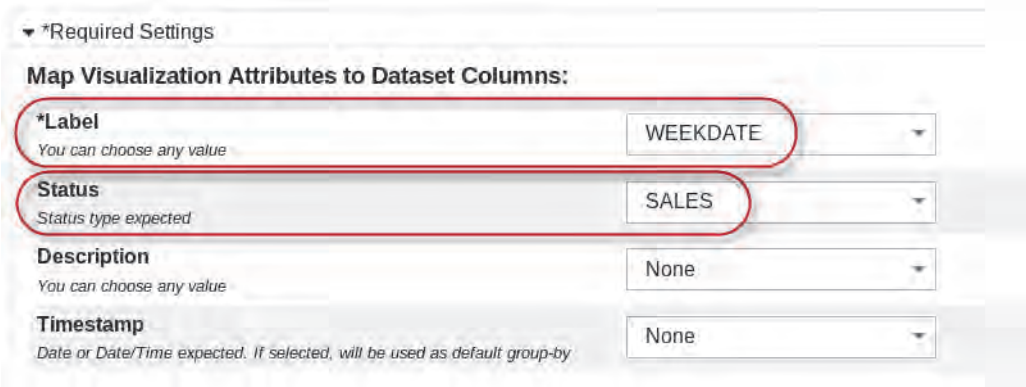
Using custom output parameters with a list widget

1. Switch to the DASH console tab in your browser. If you are not logged in to the console, login with the user ID **smadmin** and the password **object00**.
2. Create a page. Click the plus symbol (**+**) at the upper right of the console.

3. Enter **Custom Impact Output Parameters Page** in the **Page Name** field, use console/Default for the Page Location, select **Proportional** for the page layout mode, and click **OK**.

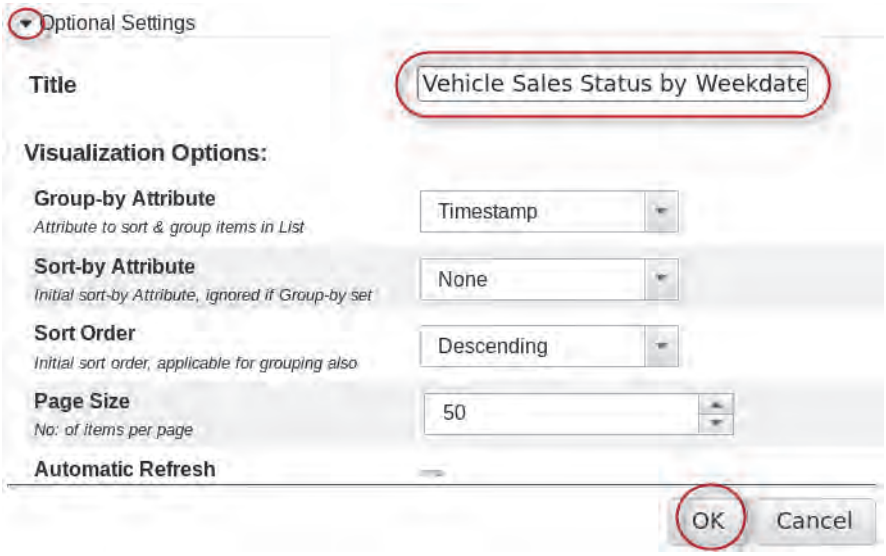


4. Drag a list widget to the page canvas.
5. Edit the list widget properties.
6. Enter **sales** in the search field and click Search.
7. Scroll down in the results list and click **SALESDATA** below the **Impact_NCICLUSTER** entry.
8. Select the **WEEKDATE** column for the label.
9. Select the **SALES** column for the Status value.



10. Click to expand the **Optional Settings** section.

11. Enter **Vehicle Sales Status by Weekdate** in the **Title** field, leave all other values at the default settings and click **OK**.



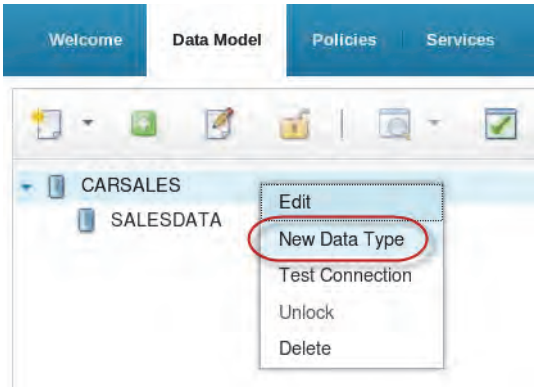
12. Click **Save and Exit** above the widget palette to save the page configuration.

Exercise 3 Using a Netcool/Impact policy to create a data set

In this exercise, you will create data set with a Netcool/Impact policy script. The script will query a data type and the result of the data type query is used as the data set. You must complete the following tasks to create a policy-based data set:

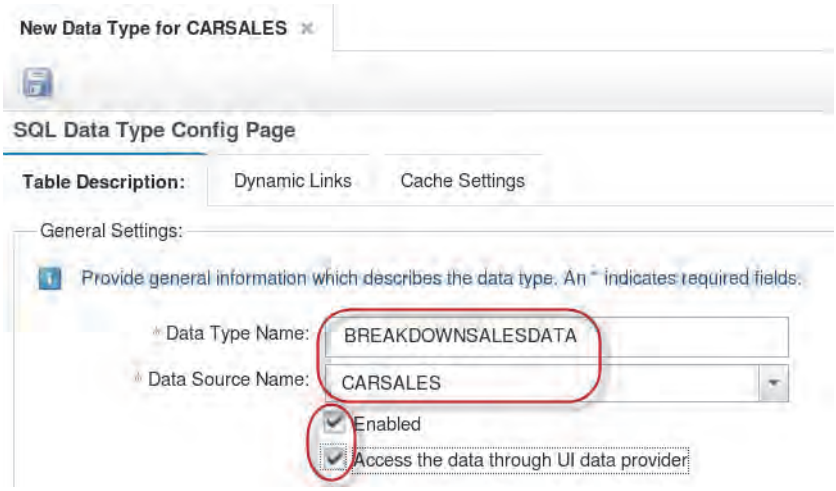
- Create a data type
- Create a policy script
- Create the policy output parameter

1. Create a Netcool/Impact data type.
   a. Right-click **CARSALES** and select **New Data Type**.



   b. Enter the following values in the General Settings section:
      i. Data Type Name: BREAKDOWNSALESDATA
      ii. Data Source Name: CARSALES
      iii. Enabled: Selected

iv. Access the data through UI data provider: Selected



c. Enter the following values in the Table Description section:

i. Schemas: DB2INST1

ii. Tables: BREAKDOWNSALESDATA

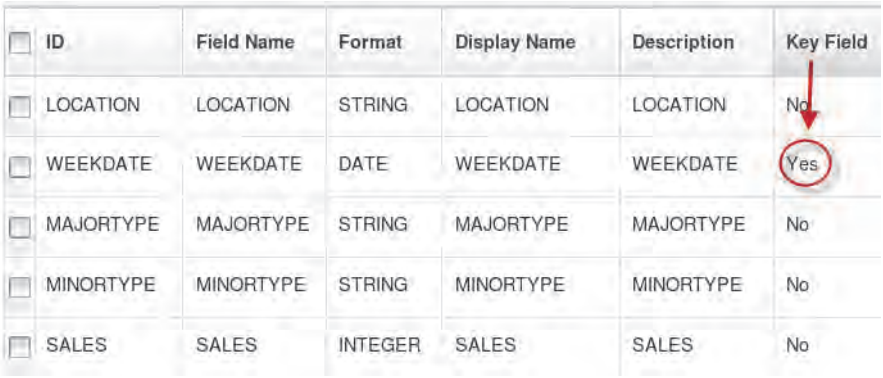
d. Click **Refresh Fields** to see the table configuration.

e. You must designate one of the table columns as a key field. Configure the WEEKDATE column as a key field. Double-click the **Key Field** cell in the **WEEKDATE** row.

ID	Field Name	Format	Display Name	Description	Key Field
LOCATION	LOCATION	STRING	LOCATION	LOCATION	No
WEEKDATE	WEEKDATE	DATE	WEEKDATE	WEEKDATE	No
MAJORTYPE	MAJORTYPE	STRING	MAJORTYPE	MAJORTYPE	No
MINORTYPE	MINORTYPE	STRING	MINORTYPE	MINORTYPE	No
SALES	SALES	INTEGER	SALES	SALES	No

f. Select the check box in the cell.

g. Click the **Key Field** column header to update the form and see the WEEKDATE key field cell value changed to Yes.

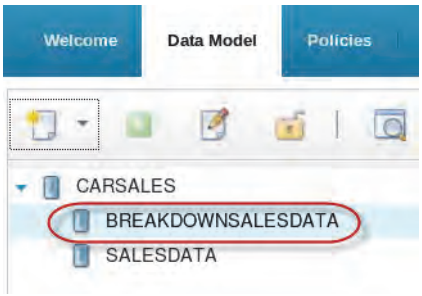


ID	Field Name	Format	Display Name	Description	Key Field
LOCATION	LOCATION	STRING	LOCATION	LOCATION	No
WEEKDATE	WEEKDATE	DATE	WEEKDATE	WEEKDATE	Yes
MAJORTYPE	MAJORTYPE	STRING	MAJORTYPE	MAJORTYPE	No
MINORTYPE	MINORTYPE	STRING	MINORTYPE	MINORTYPE	No
SALES	SALES	INTEGER	SALES	SALES	No

h. Save data type definition. Click the **Save** icon at the top of the workspace.

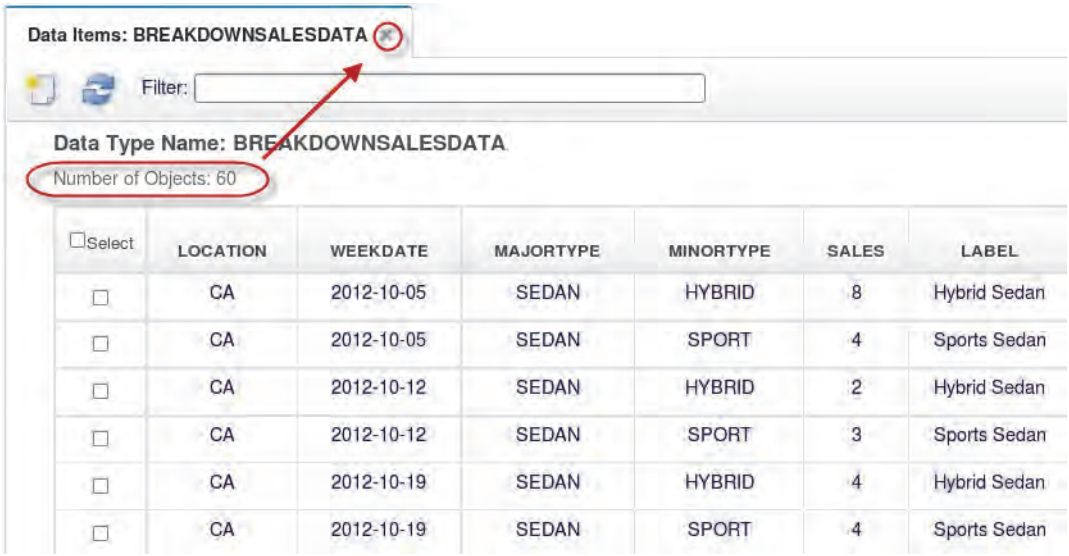
2. Close the BREAKDOWNSALESDATA configuration form. Click the **X** icon in the BREAKDOWNSALES data tab in the workspace.

   You see the new data type listed under the CARSALES data source.



3. Verify that the data type returns data. Right-click **BREAKDOWNSALESDATA** and select **View Data Items**.

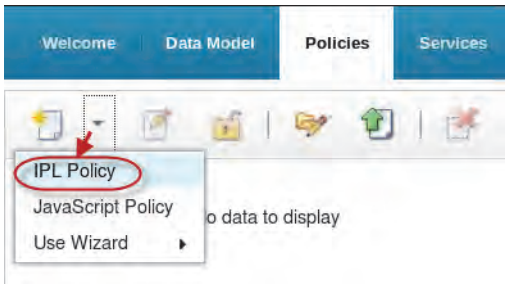
4. Verify that the output page shows 60 retrieved objects and then close the **Data Items: BREAKDOWNSALESDATA** tab. Click the **X** icon in the tab in the console workspace.



Select	LOCATION	WEEKDATE	MAJORTYPE	MINORTYPE	SALES	LABEL
☐	CA	2012-10-05	SEDAN	HYBRID	8	Hybrid Sedan
☐	CA	2012-10-05	SEDAN	SPORT	4	Sports Sedan
☐	CA	2012-10-12	SEDAN	HYBRID	2	Hybrid Sedan
☐	CA	2012-10-12	SEDAN	SPORT	3	Sports Sedan
☐	CA	2012-10-19	SEDAN	HYBRID	4	Hybrid Sedan
☐	CA	2012-10-19	SEDAN	SPORT	4	Sports Sedan

5. Create a Netcool/Impact Policy that retrieves the data from the BREAKDOWNSALESDATA data type.
   a. Click the Policies tab at the top of the console.

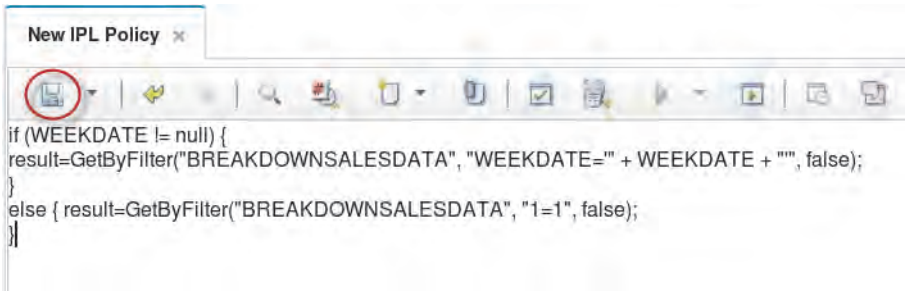
   b. Click the **New Policy** icon and select **IPL Policy**.



   c. The policy editor opens in the console workspace. Enter the following script in the policy editor workspace:

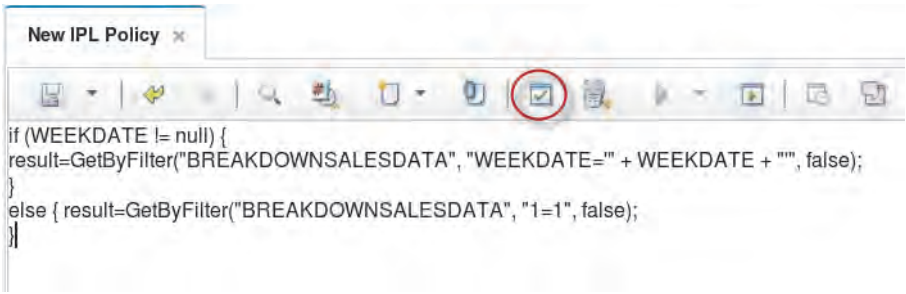
```
if (WEEKDATE != null) {
result=GetByFilter("BREAKDOWNSALESDATA", "WEEKDATE='" + WEEKDATE + "'", false);
}
else { result=GetByFilter("BREAKDOWNSALESDATA", "1=1", false);
}
```

d. Save the policy. Click the **Save** icon in the upper right of the workspace.

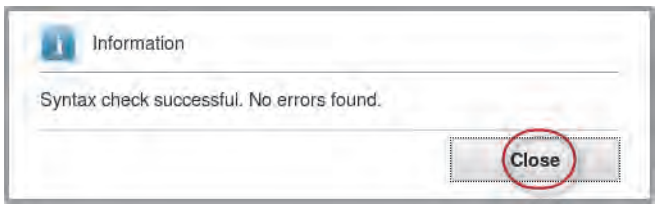


e. Enter **Impact_BREAKDOWNSALESDATA** in the **Policy Name** field and click **Save**.

f. Check the syntax of the policy script. Click the **Check Syntax** icon in the policy editor toolbar.

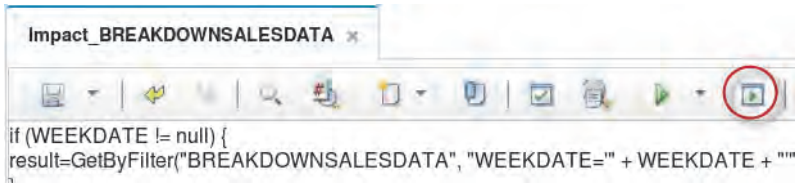


g. If there are no errors, you see a successful check message. Click Close.



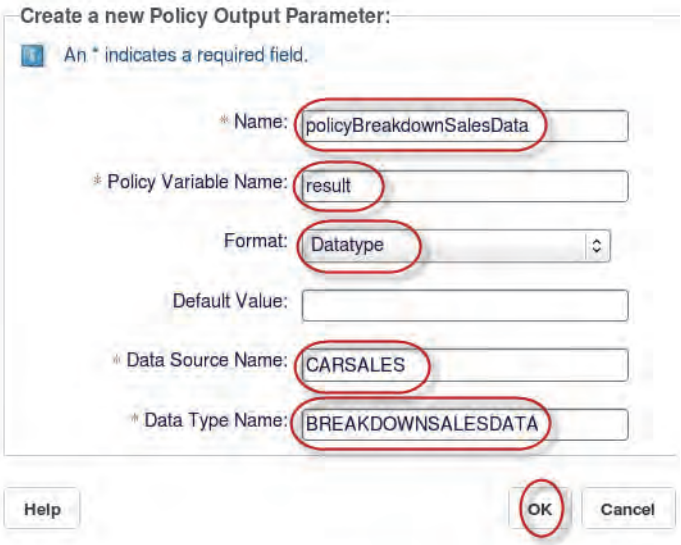
Note: If syntax errors are reported, check the policy against the exercise guide. Repeat the syntax check until there are no reported errors.

6. You now configure an *output policy parameter* that publishes the **result** variable from the policy as the data set.
   a. Click the **Configure Policy Settings** icon in the toolbar.

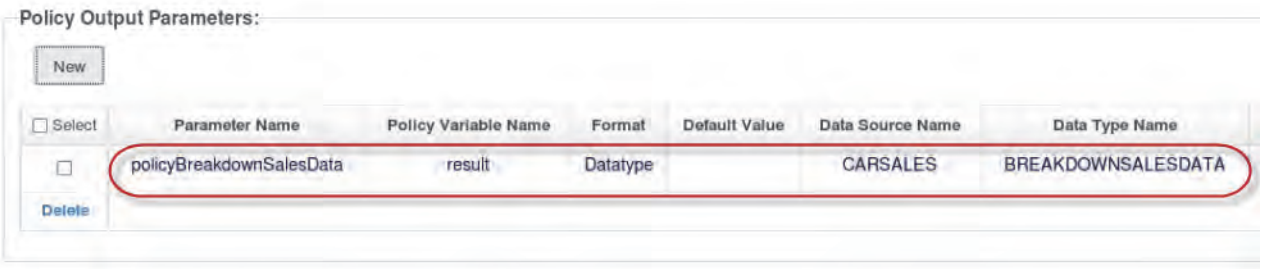


   b. Click **New** in the **Policy Output Parameters** section.

   c. Enter the following values:
      i. Name: policyBreakdownSalesData
      ii. Policy Variable Name: result
      iii. Format: Datatype
      iv. Data Source Name: CARSALES
      v. Data Type Name: BREAKDOWNSALESDATA
   d. Click **OK**.



You see the output policy listed in the **Policy Output Parameters** section.



e. Save the configuration. Scroll to the bottom of the page and click **OK**.

7. Close the policy page. Click the **X** icon in the Impact_BREAKDOWNSALESDATA tab in the console workspace.

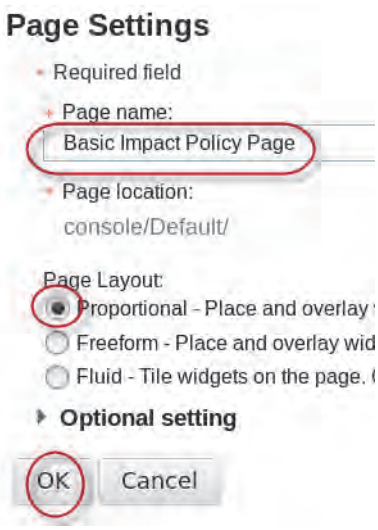
Using a Netcool/Impact policy data set with a table widget

1. Switch to the DASH console tab in your browser. If you are not logged in to the console, login with the user ID **smadmin** and the password **object00**.

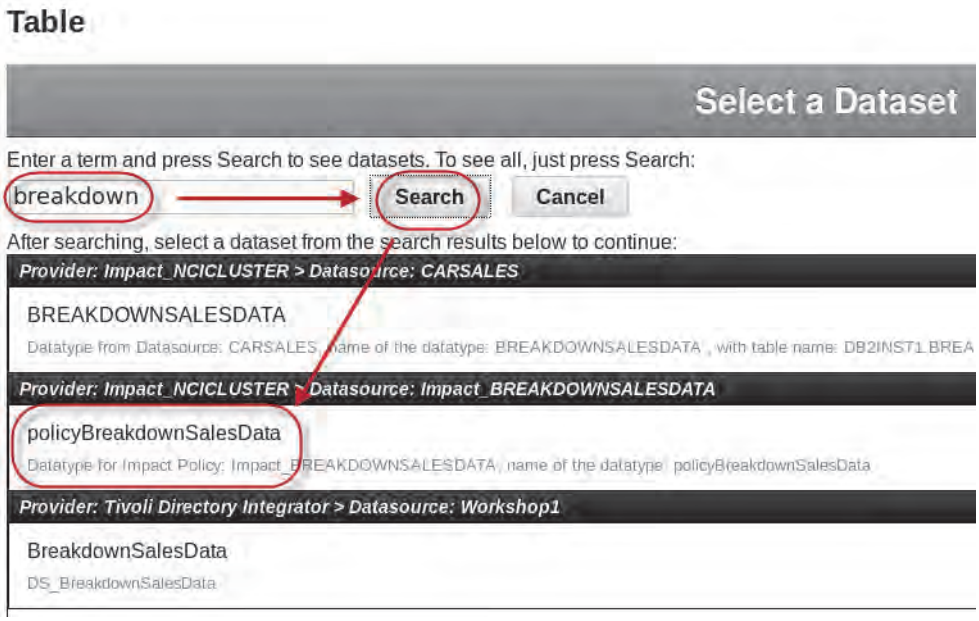
Important: By default, the Netcool/Impact UI data provider cache is updated every 5 minutes. To force the cache to update, open and save the Netcool/Impact connection document. The procedure is similar to the update cache procedure that you used with Tivoli Directory Integrator.

2. Create a page. Click the plus symbol (**+**) at the upper right of the console.

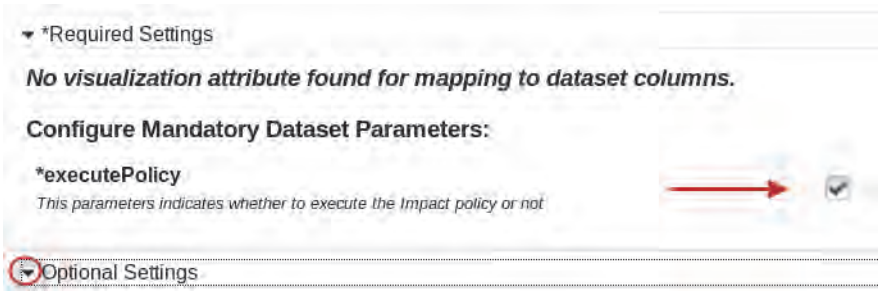
3. Enter **Basic Impact Policy Page** in the **Page Name** field, use console/Default for the Page Location, select **Proportional** for the page layout mode, and click **OK**.



4. Drag a table widget to the page canvas.
5. Edit the table widget properties.
6. Enter **breakdown** in the search field and click **Search**.
7. Scroll down in the results list and click **policyBreakdownSalesData** below the **Impact_NCICLUSTER** entry.

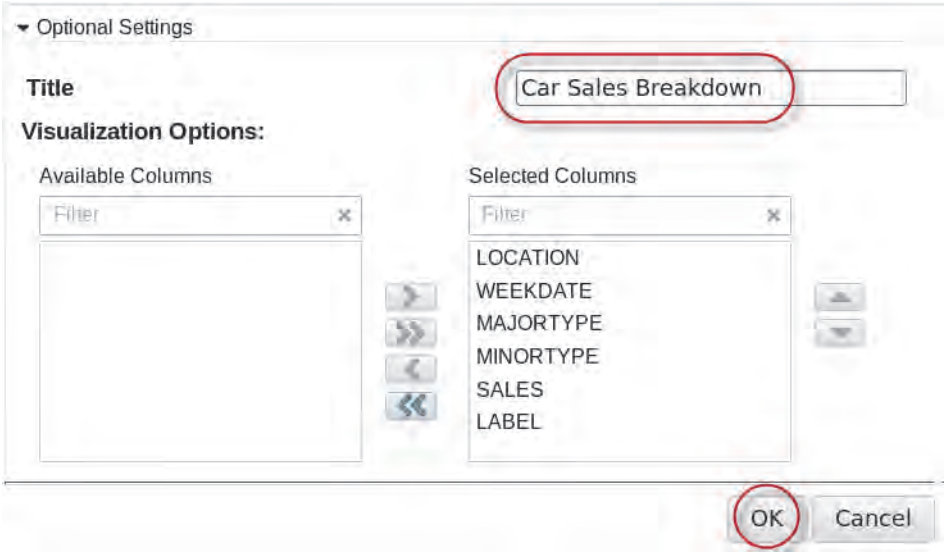


8. Verify that the **executePolicy** is selected in the **Required Settings** section.



9. Click to expand the **Optional Settings** section.

10. Enter **Car Sales Breakdown** in the **Title** field, leave all other default values, and click **OK**.



11. Click **Save and Exit** to save the page.

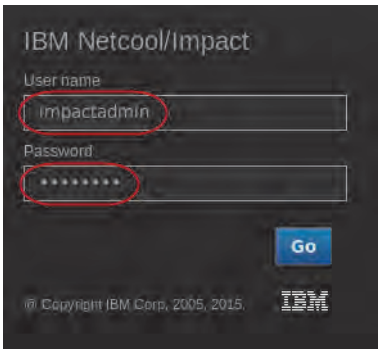
Exercise 4 Using policy variables as UI data provider output parameters

In this exercise, you use Netcool/Impact policy variables to create dashboard data set values. You use a Netcool/Impact policy to calculate the number of open Critical severity entries in a trouble ticket database. To complete this exercise, you must complete the following high-level tasks:

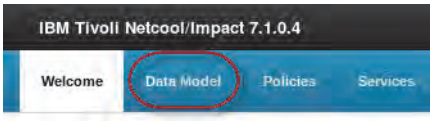
- Create a data source that connects to a trouble ticket database and a data type that contains all customer trouble ticket records.
- Create a policy that counts the number of open Critical tickets for a specified customer.
- Configure a Netcool/Impact output parameter to use the calculated open ticket variable as the data set value.
- Test the policy variable data set in a dashboard widget.

Creating the open tickets data source and data type

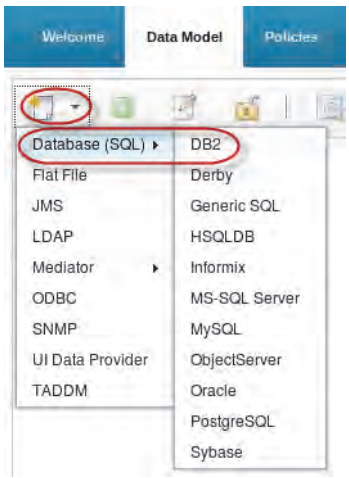
1. If you are not currently logged into the Netcool/Impact console, log in to the Netcool/Impact console with the user ID **impactadmin** and the password **object00**.



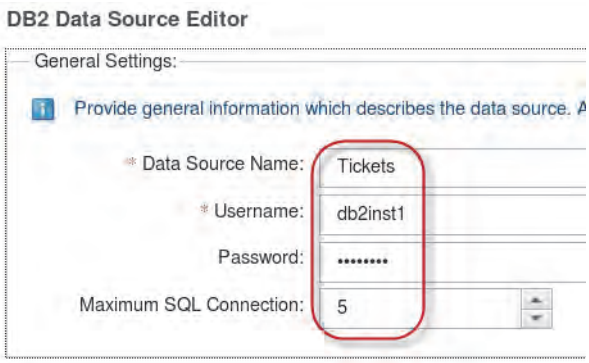
2. Click the **Data Model** tab at the upper left of the console.



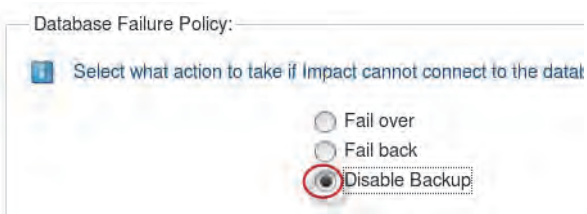
3. Click the **New Data Source** icon and select **Database (SQL) > DB2**.



4. Enter the following values in the **General Settings** section:
   a. Data Source Name: Tickets
   b. Username: db2inst1
   c. Password: object00
   d. Maximum SQL Connection: 5



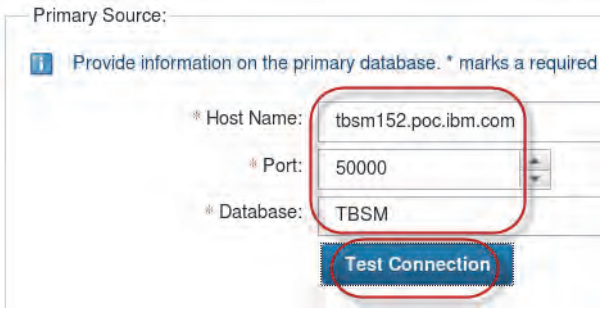
5. Select **Disable Backup** in the **Database Failure Policy** section.



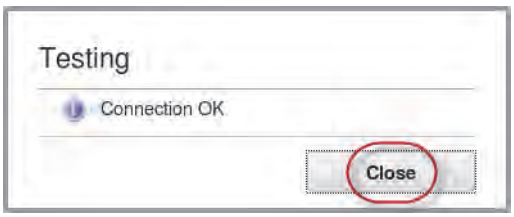
6. Enter the following values in the **Primary Source** section:
   a. Host Name: tbsm152.poc.ibm.com
   b. Port: 50000

c. Database: TBSM

7. Click **Test Connection** to verify that the properties specified are correct and that the database connection is working.

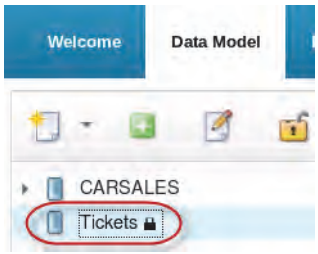


8. Click **Close** to close the Testing window.

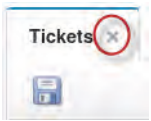


9. Click the **Save** icon at the top of the form to save the data model configuration.

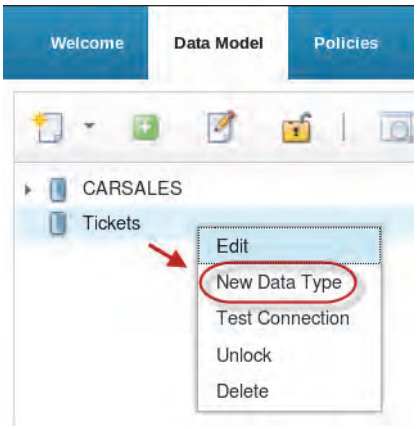
You see the new data source listed in the navigation section of the console.



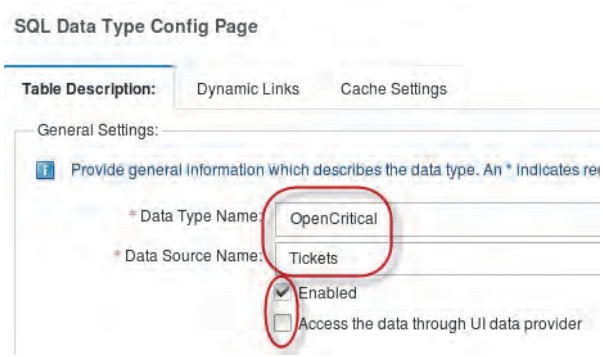
10. Close the data source configuration page. Click the **X** icon in the **Tickets** tab in the console workspace.



11. Create the trouble ticket data type. Right-click the **Tickets** data source in the navigator list and select **New Data Type**.

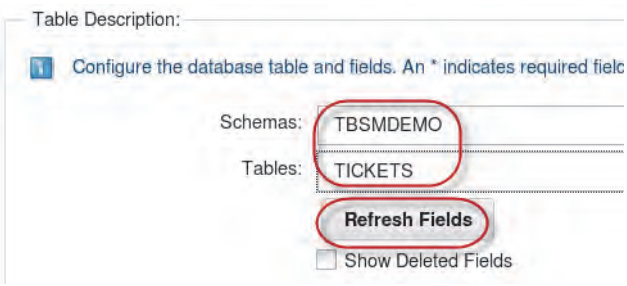


12. Enter the following values in the General Settings section:
    a. Data Type Name: OpenCritical
    b. Data Source Name: Tickets
    c. Enabled: Selected
    d. Access the data through UI data provider: Not selected



13. Enter the following values in the Table Description section:
    a. Schemas: TBSMDEMO
    b. Tables: TICKETS

14. Click the **Refresh Fields** to show the selected table configuration.



The schema information is retrieved and shown in the **Table Description** section.

15. Configure the TICKETID column as a database key field. Double-click **No** in the **Key Field** column in the TICKETID row.

ID	Field Name	Format	Display Name	Description	Key Field
TICKETID	TICKETID	INTEGER	TICKETID	TICKETID	No
HIGHLEVELSERVICEID	HIGHLEVELSERVICEID	STRING	HIGHLEVELSERVICEID	HIGHLEVELSERVICEID	No
LOWLEVELSERVICEID	LOWLEVELSERVICEID	STRING	LOWLEVELSERVICEID	LOWLEVELSERVICEID	No
CUSTOMER	CUSTOMER	STRING	CUSTOMER	CUSTOMER	No
TICKETTYPE	TICKETTYPE	STRING	TICKETTYPE	TICKETTYPE	No

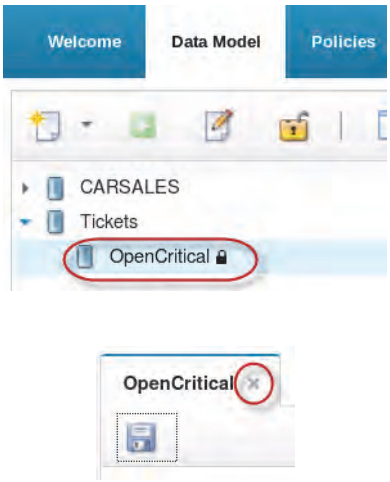
16. The table cell changes to show a selection box. Click the selection box.

17. Click the **Key Field** header and you see the TICKETID key field cell change to **Yes**.

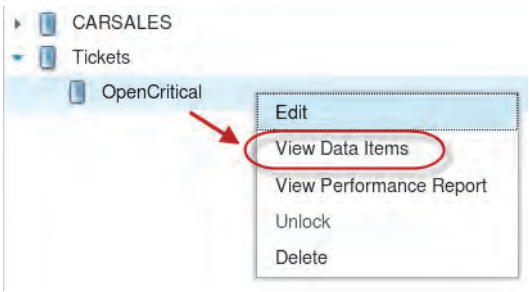
Description	Key Field
TICKETID	Yes
HIGHLEVELSERVICEID	No

18. Save the data type configuration. Click the **Save** icon at the top of the form.

19. You see the new **OpenCritical** data type in the navigation section of the console. Close the **OpenCritical** data type form. Click the **X** icon in the **OpenCritical** tab in the console workspace.



20. Verify the data type data. Right-click the new data type and select **View Data Items**.



You see the retrieved data rows from the database table.

Data Type Name: OpenCritical

Number of Objects: 2165

Select	TICKETID	HIGHLEVELSERVICEID	LOWLEVELSERVICEID	CUSTOMER	TICKETTYPE	REGION	SEVERITY	STATUS	DATEOPENED	SUMMARYTEXT
☐	1	BigBuxOnlineBanking	DBFarm	BigBux	Problem	US	3	Open	1065809715	Database responding slowly
☐	2	BigBuxOnlineBanking	DBFarm	BigBux	Problem	US	5	Open	1065809615	Database not responding
☐	3	BigBuxOnlineBanking	DBFarm	BigBux	Problem	US	3	Open	1065709615	Database not being replicated
☐	4	BigBuxOnlineBanking	DBFarm	BigBux	Problem	US	5	Open	1065709615	Not saving new accounts

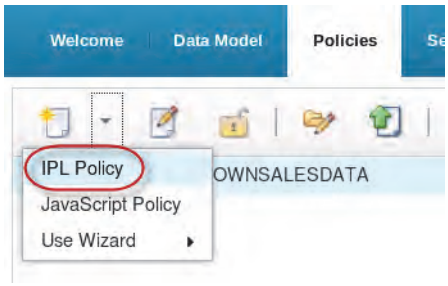
21. Close the page with the retrieved data items. Click the **X** icon in the **Data Items: OpenCritical** tab in the Netcool/Impact console workspace to close the workspace page.

22. Do not log off the Netcool/Impact console.

Creating the policy that counts the number of open Critical tickets for a specified customer

1. Create a Netcool/Impact Policy that retrieves the data from the **OpenCritical** data type.
   a. Click the **Policies** tab at the top of the console.

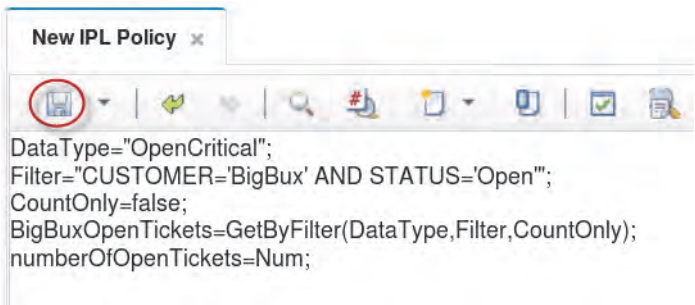
   b. Click the **New Policy** icon and select **IPL Policy**.



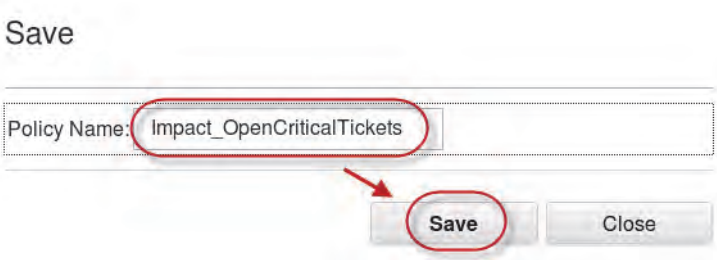
   c. The policy editor opens in the console workspace. Enter the following script in the policy editor workspace:

```
DataType="OpenCritical";
Filter="CUSTOMER='BigBux' AND STATUS='Open'";
CountOnly=false;
BigBuxOpenTickets=GetByFilter(DataType,Filter,CountOnly);
numberOfOpenTickets=Num;
```

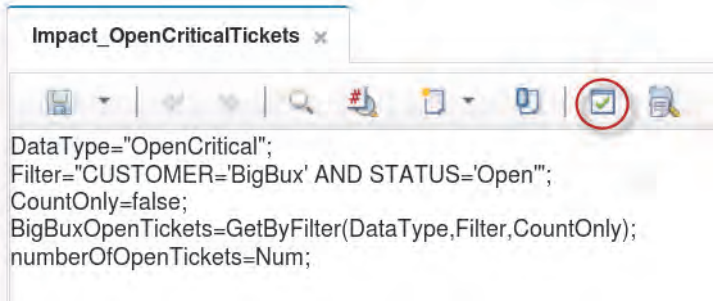
   d. Save the policy. Click the **Save** icon in the upper right of the workspace.



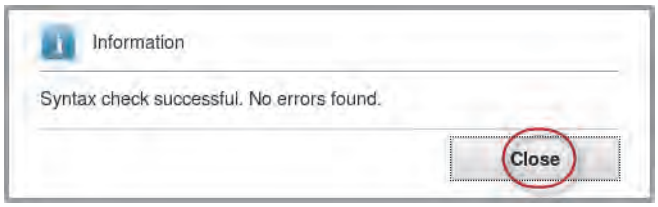
e. Enter **Impact_OpenCriticalTickets** in the **Policy Name** field and click **Save**.



f. Check the syntax of the policy script. Click the **Check Syntax** icon in the policy editor toolbar.



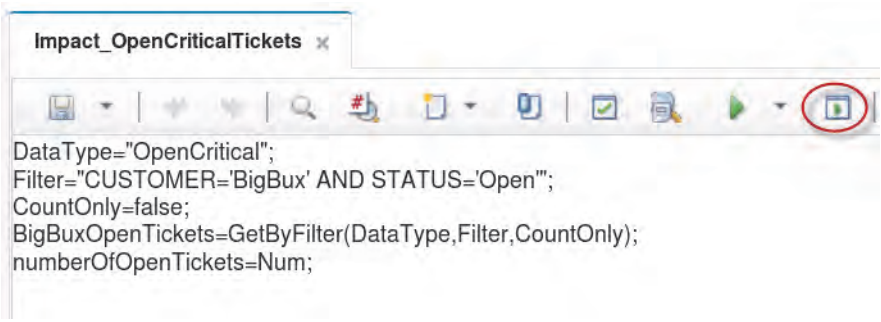
g. If there are no errors, you see a successful check message. Click **Close**.



Note: If syntax errors are reported, check the policy against the exercise guide. Repeat the syntax check until there are no reported errors.

Configuring the Netcool/Impact output parameter

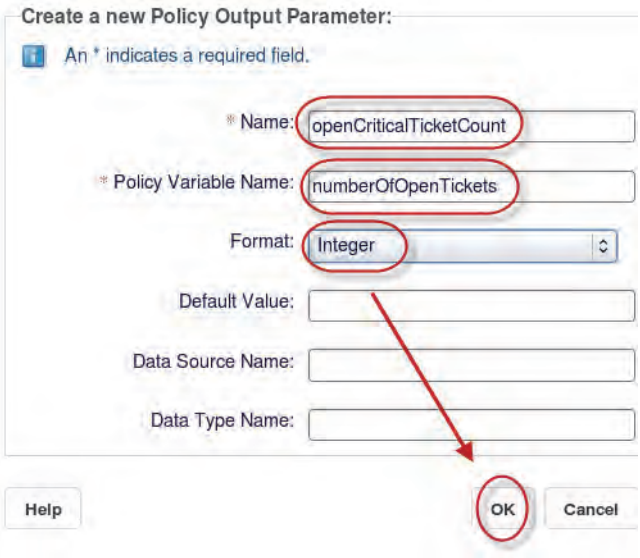
1. You now configure an *output policy parameter* that publishes the **result** variable from the policy as the data set.
   a. Click the **Configure Policy Settings** icon in the toolbar.



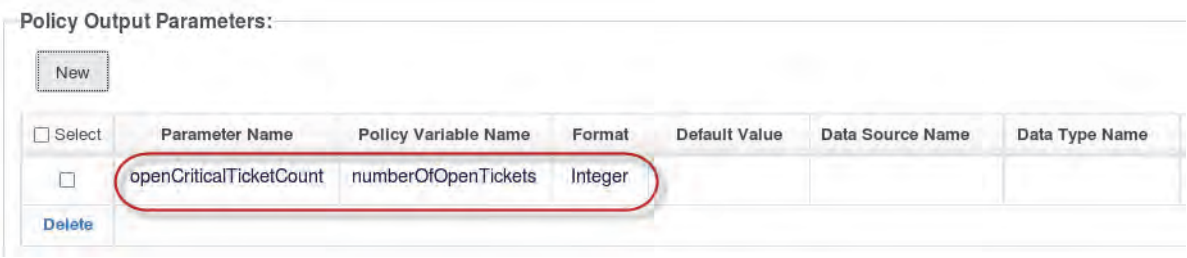
   b. Click **New** in the **Policy Output Parameters** section.

   c. Enter the following values:
      i. Name: openCriticalTicketCount
      ii. Policy Variable Name: numberOfOpenTickets
      iii. Format: Integer
      iv. Default Value: Leave blank
      v. Data Source Name: Leave blank
      vi. Data Type Name: Leave blank

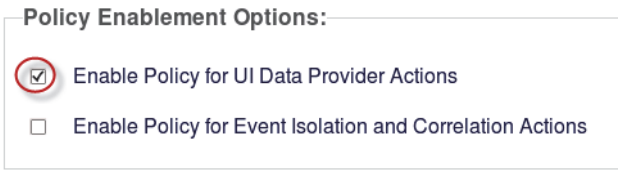
d. Click **OK**.



You see the output policy listed in the **Policy Output Parameters** section.



e. Verify that **Enable Policy for UI Data Provider Actions** is selected in the **Policy Enablement Options** section.



f. Save the configuration. Scroll to the bottom of the page and click **OK**.

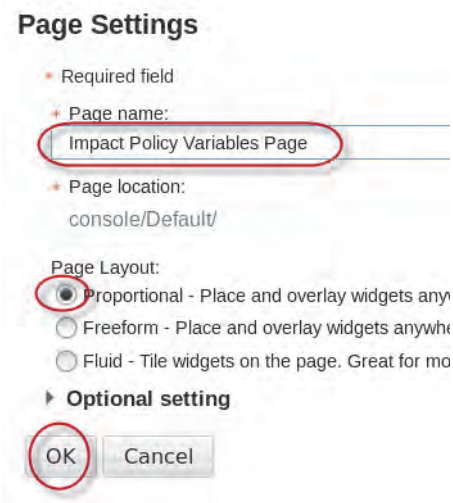
2. Close the policy page. Click the **X** icon in the **Impact_OpenCriticalTickets** tab in the console workspace.

Testing the policy variable data set in a value status gauge widget

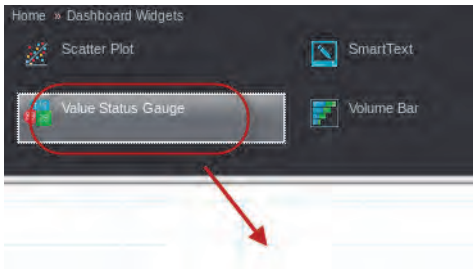
1. Switch to the DASH console tab in your browser. If you are not logged in to the console, login with the user ID **smadmin** and the password **object00**.

Important: By default, the Netcool/Impact UI data provider cache is updated every 5 minutes. To force the cache to update, open and save the Netcool/Impact connection document. The procedure is similar to the update cache procedure that you used with Tivoli Directory Integrator.

2. Create a page. Click the plus symbol (**+**) at the upper right of the console.
3. Enter **Impact Policy Variables Page** in the **Page Name** field, use console/Default for the Page Location, select **Proportional** for the page layout mode, and click **OK**.

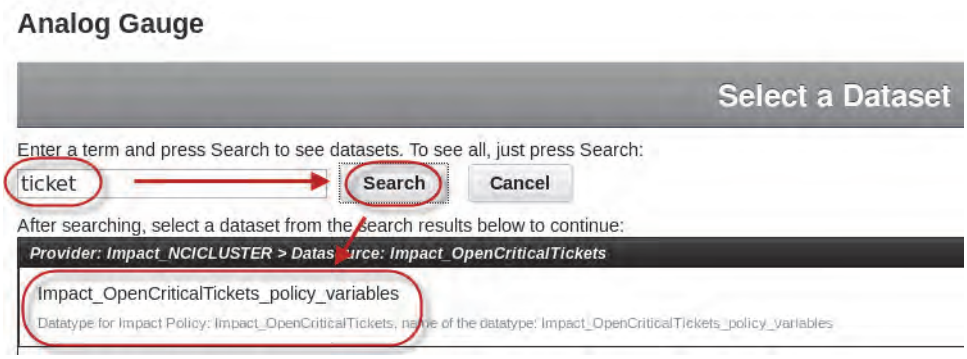


4. Drag a value status gauge widget to the page canvas.



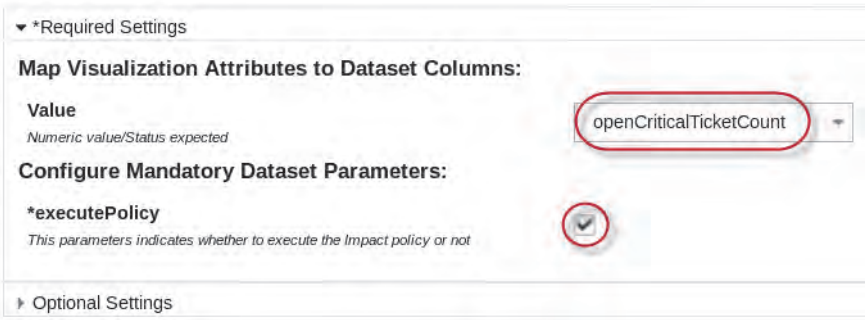
5. Edit the value status gauge widget properties.
6. Enter **ticket** in the search field and click **Search**.

7. Click **Impact_OpenCriticalTickets_policy_variables** below the **Provider: Impact_NCICLUSTER** entry.



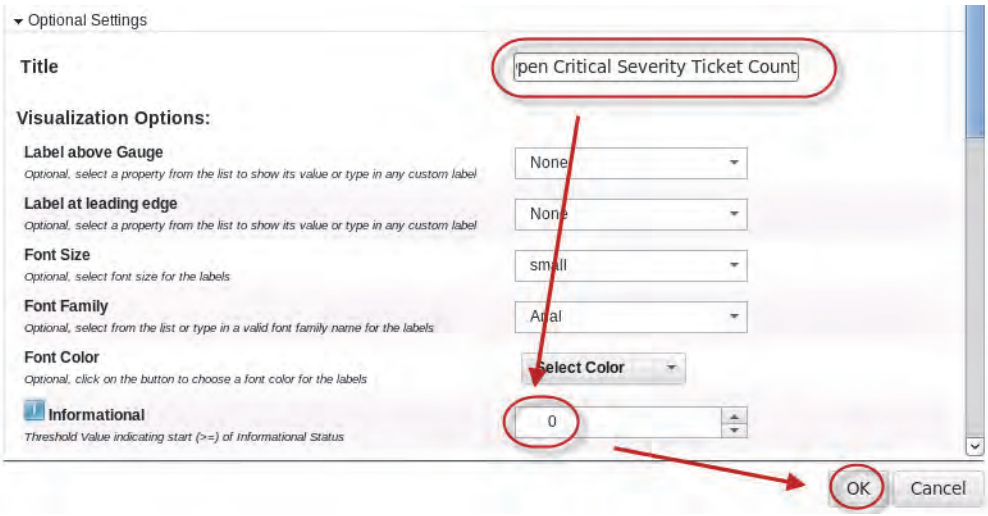
8. Select **openCriticalTicketCount** in the **Value** menu.

9. Verify that the **executePolicy** is selected.



10. Click to expand the **Optional Settings** section.

11. Enter **Open Critical Severity Ticket Count** in the **Title** field, enter **0** in the **Informational** threshold field, leave all other values at their default settings, and click **OK**.



Hint: Because this widget is being configured to show a value without indicating a particular severity, the Information threshold is set to zero so that it is always used. Using the Information threshold sets the widget icon to blue. If no value is entered for any threshold, the icon color is a light grey.

12. Click **Save and Exit** to save the page.

Unit 5 Using the Jazz for Service Management web widget with business dashboards exercises

The exercises in this unit show several techniques to use web widgets to show business dashboard data. You first learn how to use debugging tools to see the parameters that are available in published widget events. You then learn how to pass event parameters to the web widget URL address. Next, you learn how to pass multiple parameters in a web widget URL to a JavaServer Page (JSP) file. You use this technique to create a web widget that shows dynamic text. Finally, you learn how to use hotspot widgets to pass data to web widgets and create interactive dashboard pages.

Exercise 1 Examining widget event parameters

In this exercise, you use debugging tools to examine widget event data. You use the event data information to pass parameter data to Web widgets. You use the DASH Tools Event Spy widget and Firebug browser plug-in to examine the widget events.

You must complete four tasks in this exercise:

- **Start the virtual image applications**: You start the IBM Monitoring virtual image and verify that the monitoring services are started.
- **Create the UI data provider connections**: You create a connection document to the IBM Monitoring UI data provider interface. The dashboard widgets use the connection document information to retrieve data from the applications that are used in these exercises.
- **Create the dashboard page to test widget events**: You create and configure a dashboard page that you use to generate widget events. You use widget debugger tools to monitor the events.
- **View widget events**: You generate and record the NodeClickedOn event information for two table widgets. Each table widget is configured with a data set from a different virtual image application. You examine the differences in the widget events, which are based on the source data set.

Note: The DASH Tools are installed during the virtual image setup. The tool package is available for download from the IBM developerWorks Jazz for Service Management UI Services wiki: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W3d82b8c8fc4b_482e_94c3_34d88b299e0e/page/Dash%20Tools.

Perform the following steps to complete the exercise.

Starting the IBM Tivoli Monitoring server components

In this task, you start the IBM Tivoli Monitoring components on the **sysitmsles** virtual image.

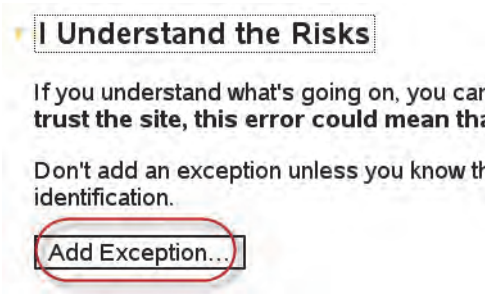
1. Switch to the **sysitmsles** virtual image desktop.
2. Right-click the desktop and select **Open in Terminal** to open a command window.
3. Enter the following command to manage the IBM Tivoli Monitoring services:

   ```
   /opt/IBM/ITM/bin/itmcmd manage
   ```

   The tool takes 15 - 20 seconds to open. Initially, none of the IBM Tivoli Monitoring services are started.
4. Start the Tivoli Enterprise Monitoring Server service. Click to select **Tivoli Enterprise Monitoring Server** in the **Service** column, right-click the entry, and select **Start Service**.

5. Repeat the previous process to start the following services, in order:
   - Tivoli Enterprise Portal Server
   - Warehouse Proxy
   - Monitoring Agent for Linux OS
   - Summarization and Pruning Agent

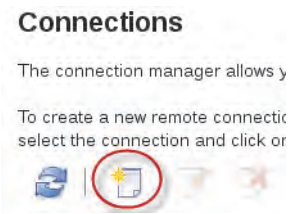
When completed, the Service column looks like the following screen image.



Note: The IBM Eclipse Help Server service is started automatically when the Tivoli Enterprise Portal Server service is started.

Creating the IBM Tivoli Monitoring UI data provider connection

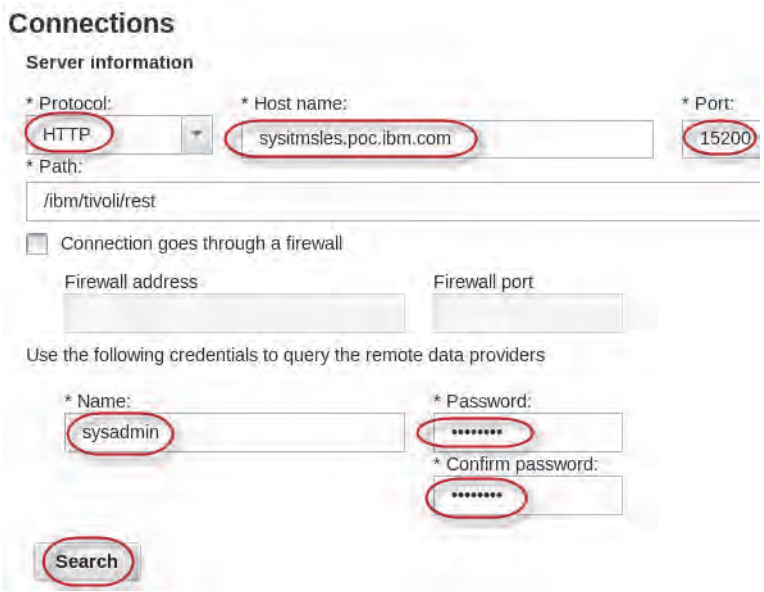
1. Switch to the **dash151** image desktop.
2. Open the DASH server console or connect to the DASH server console with the user ID **smadmin** and the password **object00**.
3. Create a UI data provider connection document to the IBM Monitoring server REST interface.
   a. Select the **Console Settings > General > Connections** task in the taskbar on the left side of the console.
   b. Click the **Create new remote provider** icon to start the connection wizard.



c. Configure the connection document to query the IBM Tivoli Monitoring UI data provider REST interface. Use the information in the following table to prepare the connection query.

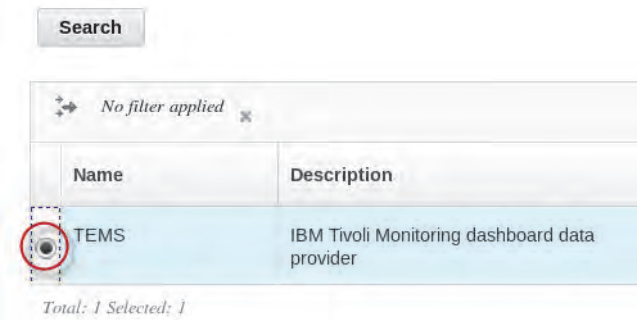
Parameter name	Value
Protocol	HTTP
Host name	sysitmsles.poc.ibm.com
Port	15200
Path	/ibm/tivoli/rest
Name	sysadmin
Password	sysadmin
Confirm Password	sysadmin

d. Query the UI data provider. Click **Search**.



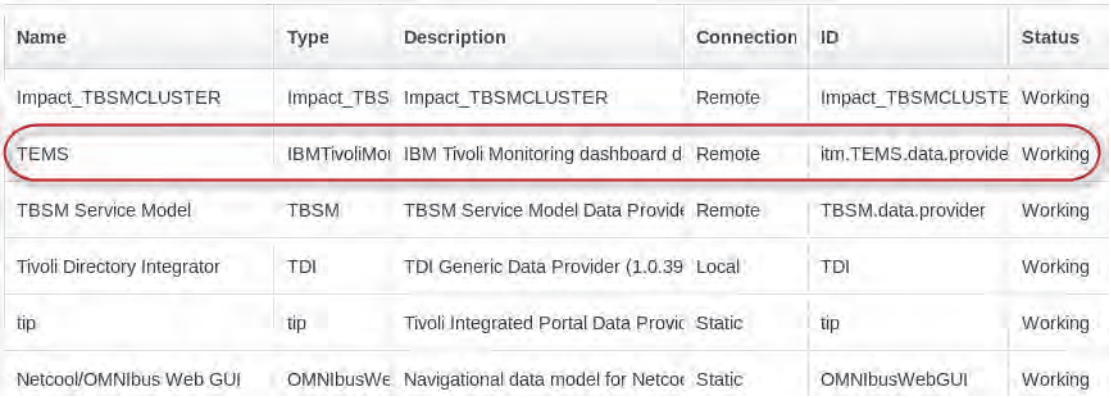
The UI data provider is shown in a list.

e. Select **TEMS**.



f. In the Connection information section, change the default Provider ID to a more portable name. Enter **itm.TEMS.data.provider** in the **Provider ID** field and click **OK**.

g. Verify that the connection document status is listed as **Working** in the **Status** column.

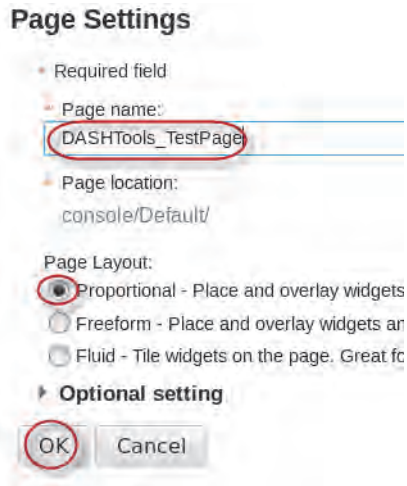


Name	Type	Description	Connection	ID	Status
Impact_TBSMCLUSTER	Impact_TBS	Impact_TBSMCLUSTER	Remote	Impact_TBSMCLUSTE	Working
TEMS	IBMTivoliMo	IBM Tivoli Monitoring dashboard d	Remote	itm.TEMS.data.provide	Working
TBSM Service Model	TBSM	TBSM Service Model Data Provide	Remote	TBSM.data.provider	Working
Tivoli Directory Integrator	TDI	TDI Generic Data Provider (1.0.39	Local	TDI	Working
tip	tip	Tivoli Integrated Portal Data Provic	Static	tip	Working
Netcool/OMNIbus Web GUI	OMNIbusWe	Navigational data model for Netco	Static	OMNIbusWebGUI	Working

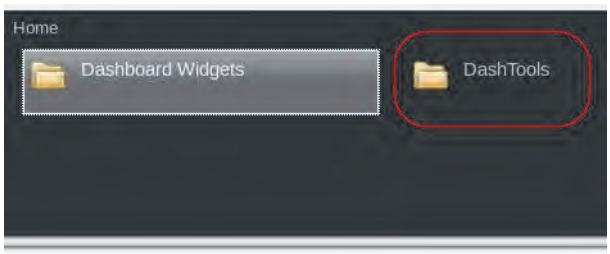
Creating the dashboard page to test widget events

1. Create a dashboard page. The finished page consists of two table widgets and one instance of the **Node Clicked On** widget. You use the page to generate and examine widget events with a browser debugging tool.
   a. Click the **Create New Page** icon (**+**) in the upper right of the console.

b. Complete the page settings form. Enter **DASHTools_TestPage** in the **Page name** field, leave the default **Page location** setting, select **Proportional** in the **Page Layout** list, and click **OK** to save the page settings.

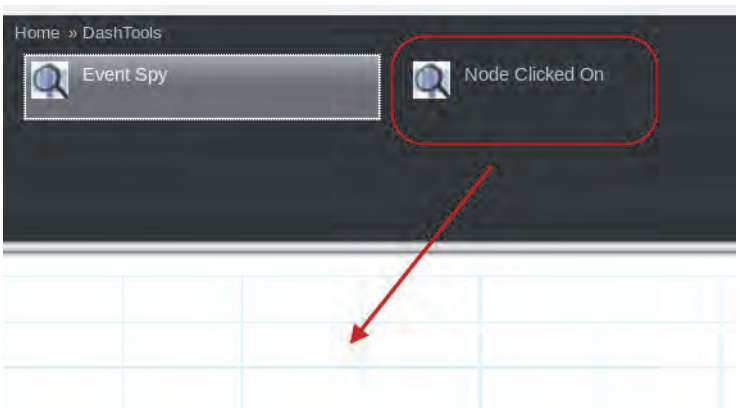


c. Click the **All** widget catalog icon in the widget palette section of the page editor.

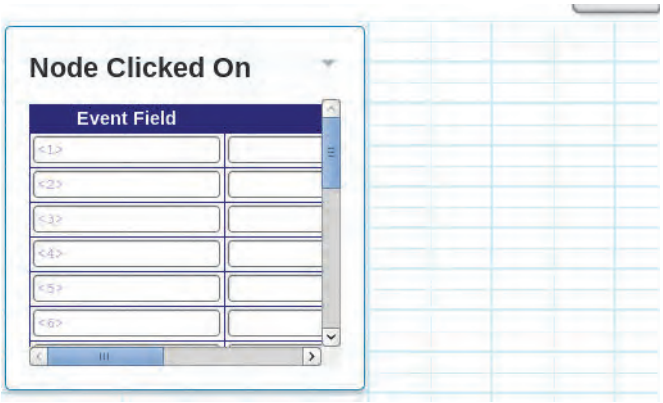


All of the available dashboard widgets are shown in the widget palette.

d. The Node Clicked On widget shows a table that contains any generated NodeClickedOn event parameter names and values on a page. Click and drag an instance of the **Node Clicked On** widget from the widget palette to the page canvas.

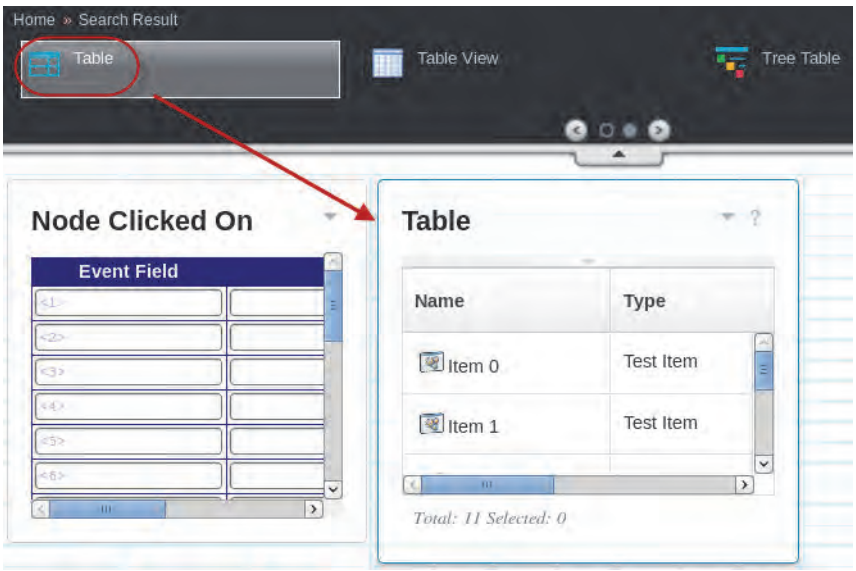


You do not configure the widget. Click the widget title bar and drag the widget to the upper left of the page canvas. Expand the widget so that the two columns are visible.

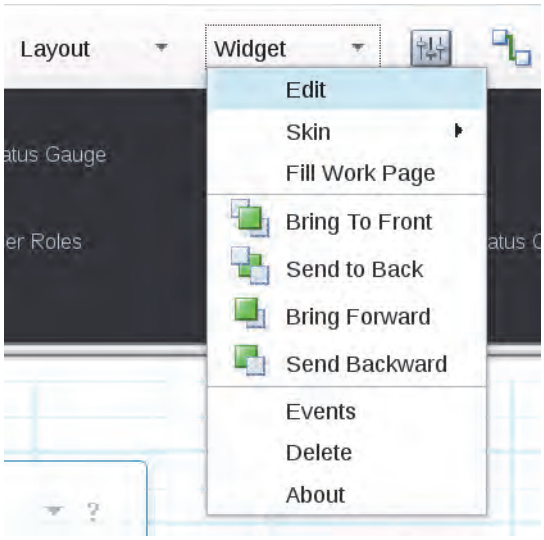


e. Click and drag an instance of a table widget to the page canvas.

f. Click an edge of the widget to activate the widget anchor points. Resize the widget to use one-third of the page canvas.

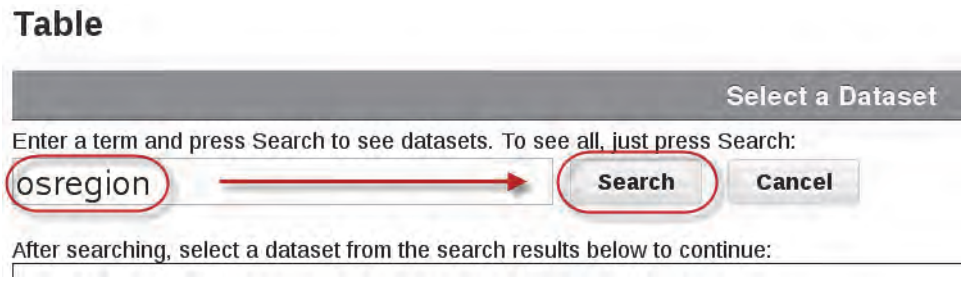


g. Edit the table widget configuration properties. Click the **Edit options** icon in the upper right of the widget and select **Edit** or click the widget and select the **Widget > Edit** menu above the widget palette.



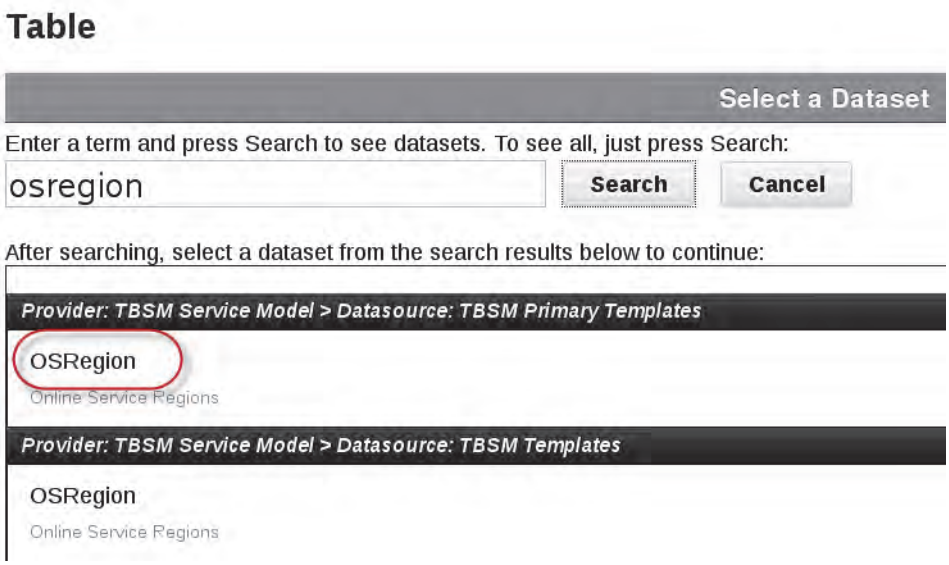
For this widget, you show data with a Tivoli Business Service Manager data set. The data set corresponds to an online services business service model. The service model calculates the number of open high severity and critical severity trouble tickets for a global trouble ticket system.

h. Enter **osregion** in the search field and click **Search**.



Two data sources are shown in the filtered list.

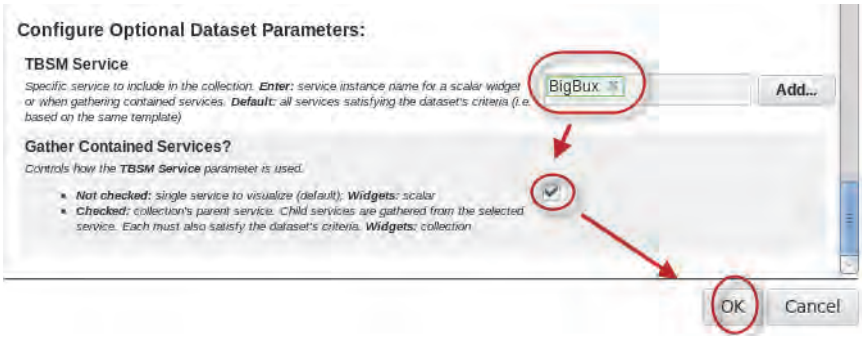
i. Select **OSRegion** in the **TBSM Primary Templates** data source listing.



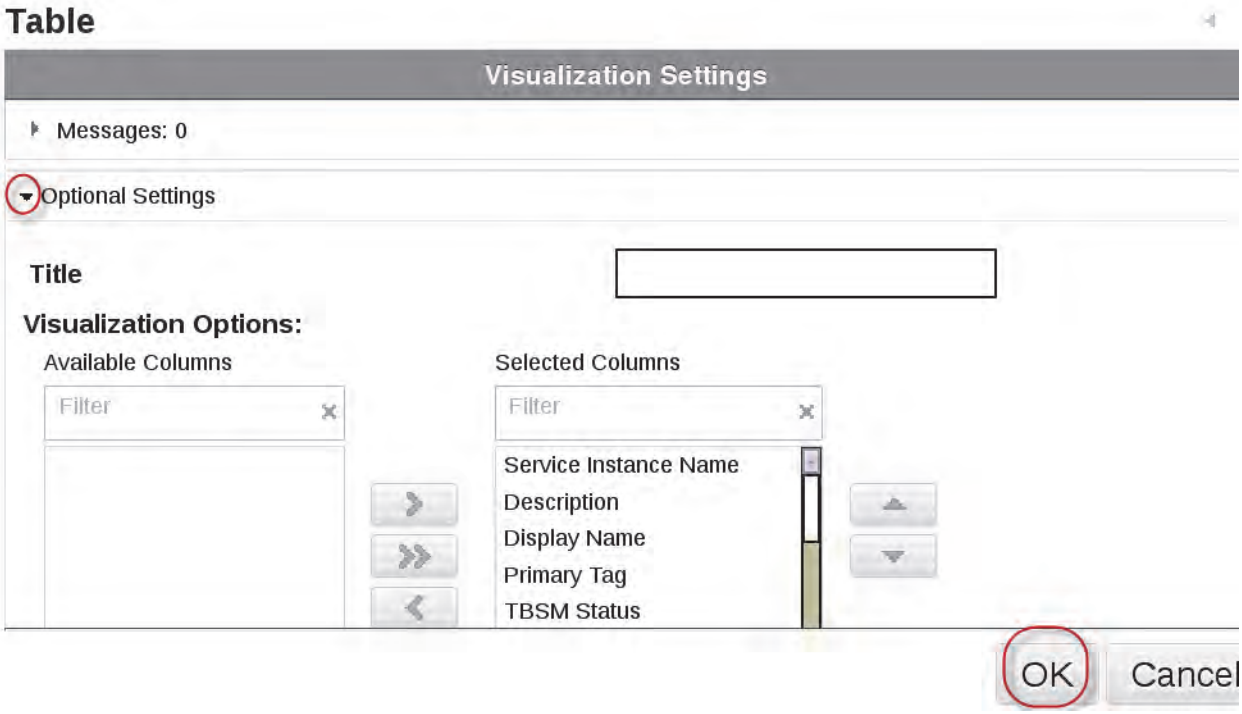
j. Click to expand the **Optional Settings** section.

k. Scroll to the bottom of the optional settings section and enter **BigBux** in the **TBSM Service** field.

l. Select **Gather Contained Services** and click **OK**.



m. Leave the default settings for all other configuration options and click **OK** to save the widget configuration.



n. Click and drag a second instance of a table widget to the page canvas.

o. Click an edge of the widget to activate the widget anchor points. Resize the widget to use one-third of the page canvas.

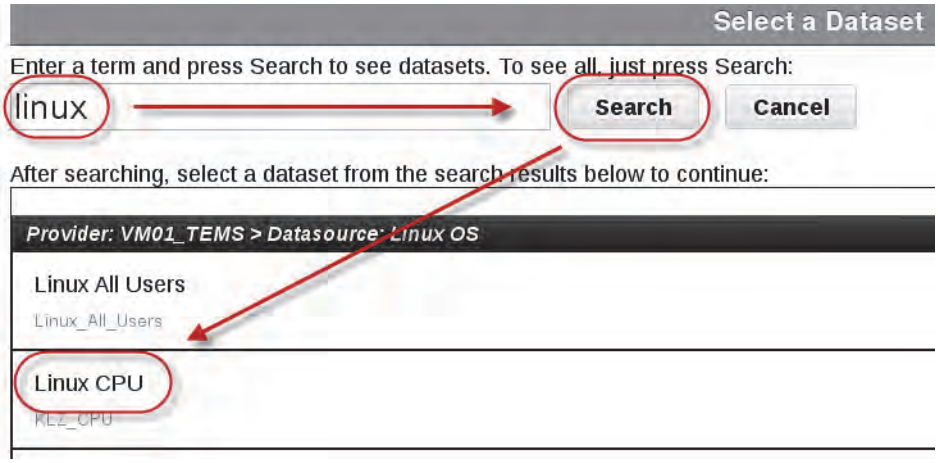
p. Click the widget title bar to grab and move the second widget so that both table widgets are not overlapping on the page canvas.

q. Edit the table widget configuration properties. Click the **Edit options** icon in the upper right of the widget and select **Edit** or click the widget and select the **Widget > Edit** menu above the widget palette.

   For this widget, you show data with an IBM Tivoli Monitoring data set. The data set shows Linux operating system agent monitoring statistics.

r. Enter **linux** in the search field and click **Search**.

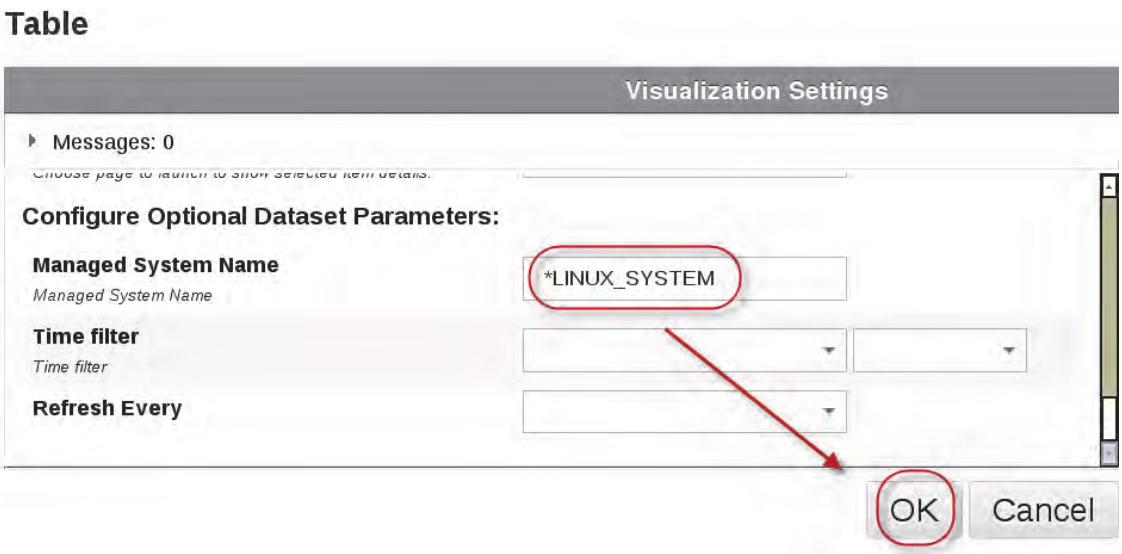
s. Select **Linux CPU** in the filtered **Linux OS** data source list.



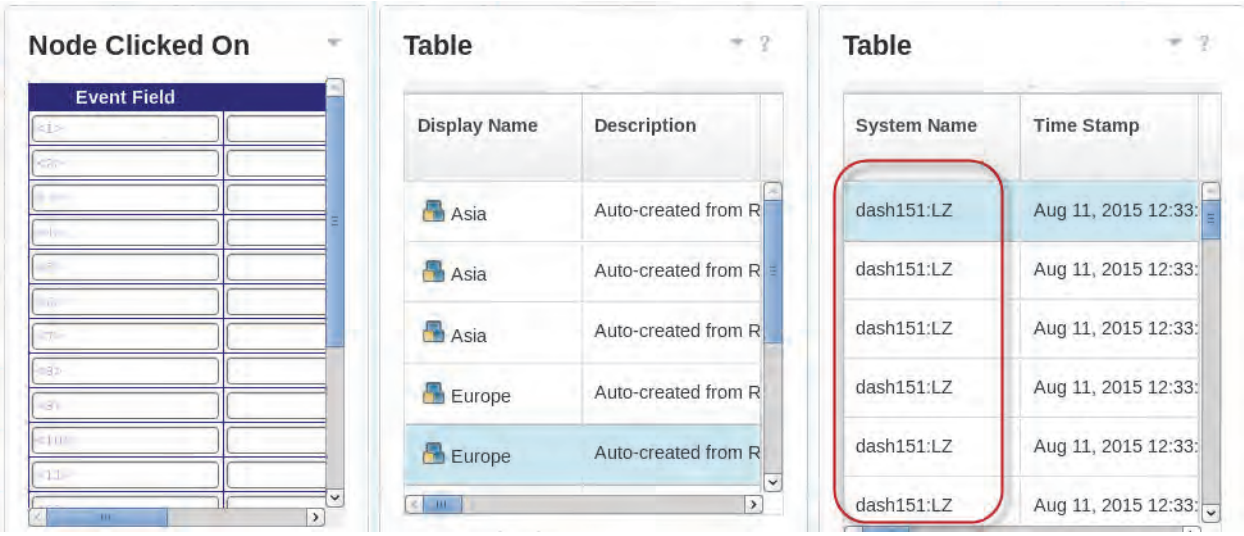
t. Click to expand the Optional Settings section.

u. Leave the default values for all configuration parameters except the Managed System Name in the Configure Optional Dataset Parameters section. Enter ***LINUX_SYSTEM** in the Managed System Name field and click **OK**.

Note: Use the ***LINUX_SYSTEM** letter case as written.



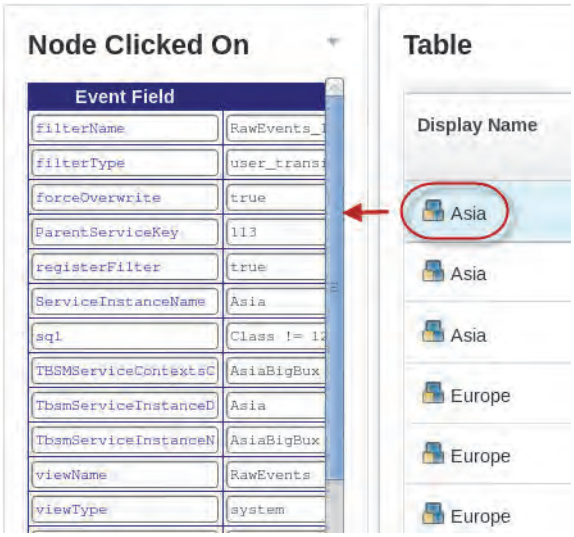
A list of all currently monitored Linux agents is shown in the table.



v. Click **Save and Exit** in the upper right of the console to save the page.

Viewing widget events

1. Generate a NodeClickedOn widget event from the first table widget. Click the first row entry in the Display Name column of the table.



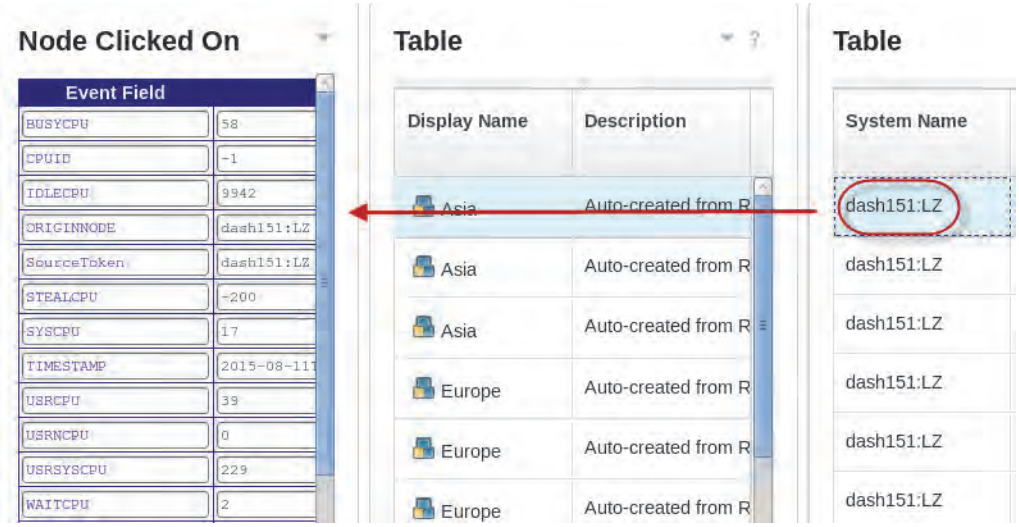
The NodeClickedOn widget event field and value columns show the event generated by the table widget selection.

Table 1 IBM Tivoli Business Service Manager NodeClickedOn parameters and values

NodeClickedOn parameter name	Parameter value
ParentServiceKey	113
ServiceInstanceName	Asia
TBSMServiceContextsCollectionSelectedServices	AsiaBigBux
TbsmServiceInstanceDisplayName	Asia
TbsmServiceInstanceName	AsiaBigBux
filterName	RawEvents_113
filterType	user_transient
forceOverwrite	true
registerFilter	true
sql	Class != 12000 and RAD_FunctionType= ‘Raw’ AND (RAD_UserFunctionName in (select EventKey from alerts.service_deps where Service = ‘1:113’))
viewName	RawEvents
viewType	system

Note: The actual values vary in your laboratory environment.

2. Test the other table widget in the page. Click the first **System Name** entry in the second table widget.



The NodeClickedOn widget event field and value columns show the event generated by the table widget selection.

Table 2 IBM Tivoli Monitoring NodeClickedOn parameters and values

NodeClickedOn parameter name	Parameter value
BUSYCPU	58
CPUID	-1
IDLECPU	9942
ORIGINNODE	dash151:LZ
STEALCPU	-200
SYSCPU	17
SourceToken	dash151:LZ
TIMESTAMP	2015-08-11T12:35:56
USRCPU	39
USRNCPU	0
USRSYSCPU	229
WAITCPU	2

Note: The actual values vary in your laboratory environment.

Important: Many, but not all, of the NodeClickedOn parameter names correspond to the data set column names. The values that are published in a NodeClickedOn event vary, based on the data source and data set provider. If you must pass data set parameters to other widgets and are unsure of the NodeClickedOn parameter values that are sent with a data set, you should configure a test that is similar to this exercise.

3. Close the page by clicking the **X** icon in the console page tab. Do not log off the DASH console.

Exercise 2 Using parameter substitution with web widgets

In this exercise, you learn how to pass widget event parameters to dashboard web widgets. You use the event parameters to pass context information and dynamically change the target web address for a web widget. You configure a web widget to show a corresponding HTML run book page when you select an item in a list widget. The web files that you use in these exercises are served with a ContentBox web server object. The ContentBox object, called **myBox**, is installed automatically with Jazz for Service Management 1.1 Fix Pack 3. You add HTML, image, and JSP pages to the myBox web server container and update the DASH server configuration.

You complete two tasks in this exercise:

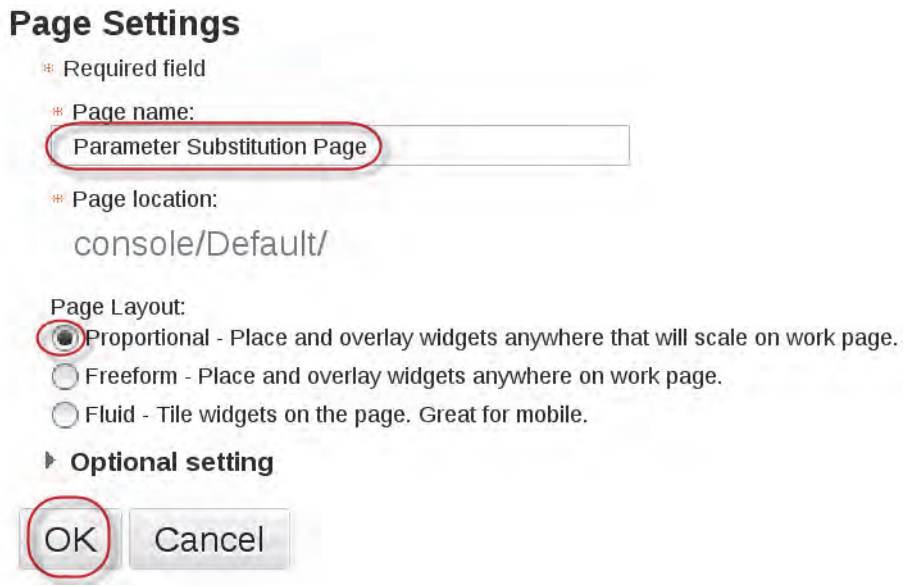
- **Create the web widget substitution dashboard page**: You configure the dashboard page with a table widget and web widget. The table widget shows data from an IBM Tivoli Monitoring data set. You configure the web widget to substitute a NodeClickedOn event parameter
- **Test the parameter substitution**: You click items in the table widget and examine the changes in the web address and data that is shown in the web widget frame.

Perform the following steps to complete the exercise.

Creating the web widget substitution dashboard page

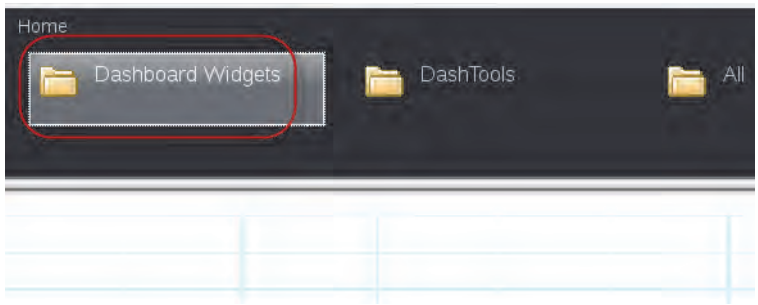
1. Create a dashboard page.
   a. Click the **Create New Page** icon (**+**) in the upper right of the console.

b. Complete the page settings form. Enter **ParameterSubstitutionPage** in the **Page name** field, leave the default **Page location** setting, select **Proportional** in the **Page Layout** list, and click **OK**.



2. Configure the dashboard page elements

a. Click the **Dashboard Widgets** catalog icon in the widget palette section.



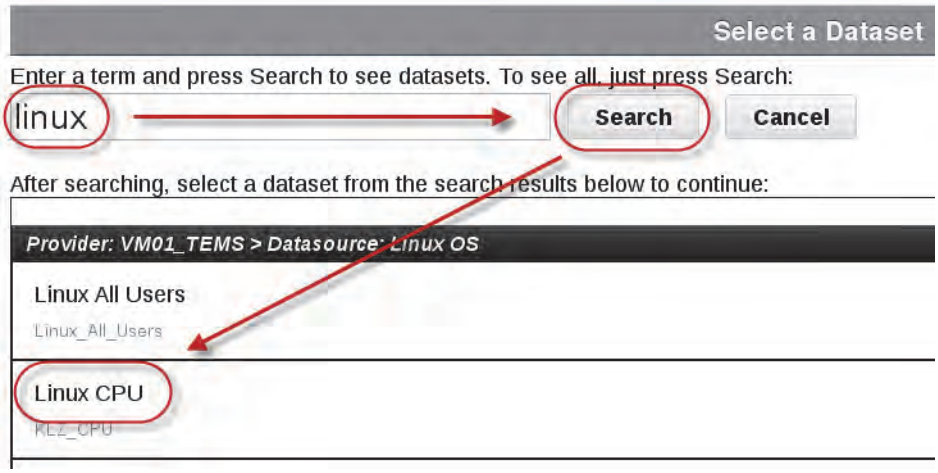
b. Click and drag an instance of a table widget to the page canvas.

c. Edit the table widget configuration properties. Click the **Edit options** icon in the upper right of the widget and select **Edit** or click the widget and select the **Widget > Edit** menu above the widget palette.

For this widget, you use an IBM Tivoli Monitoring data set. The data set shows Linux operating system agent monitoring statistics.

d. Enter **linux** in the search field and click **Search**.

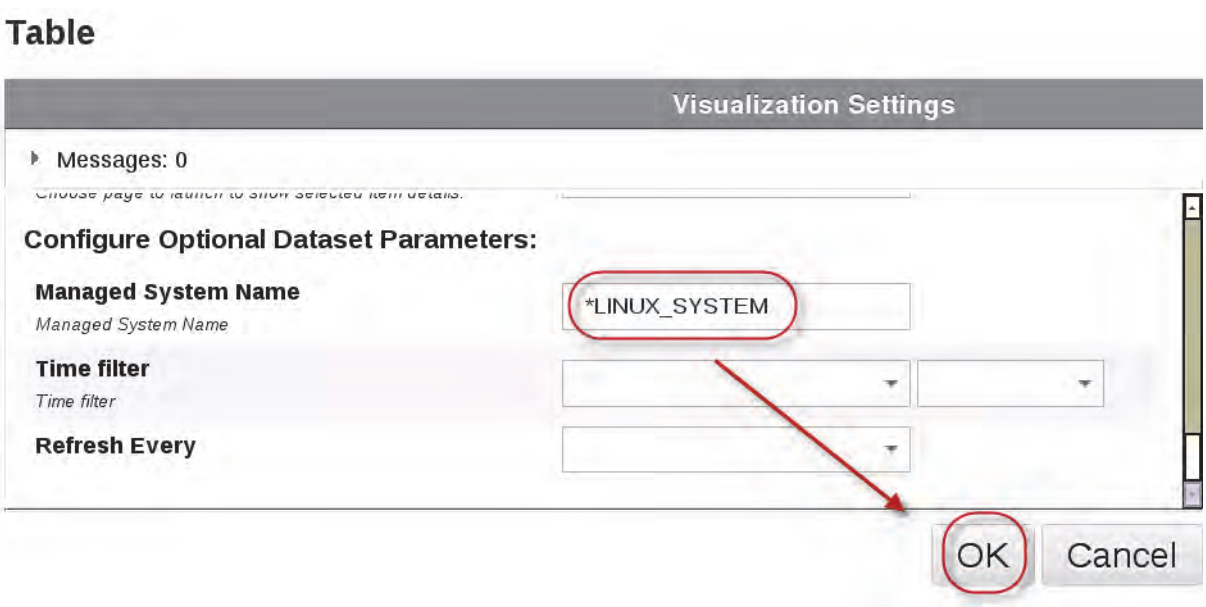
e. Select **Linux CPU** in the filtered **Linux OS** data source list.



f. Click to expand the **Optional Settings** section.

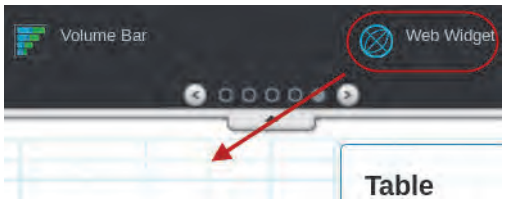
g. Leave the default values for all configuration parameters except the Managed System Name in the Configure Optional Dataset Parameters section. Enter ***LINUX_SYSTEM** in the Managed System Name field and click **OK**.

Note: Use the ***LINUX_SYSTEM** letter case as written.



A list of all currently monitored Linux agents is shown in the table.

3. Add a web widget to the page and configure it.
   a. Drag the web widget icon from the widget palette to the page canvas.

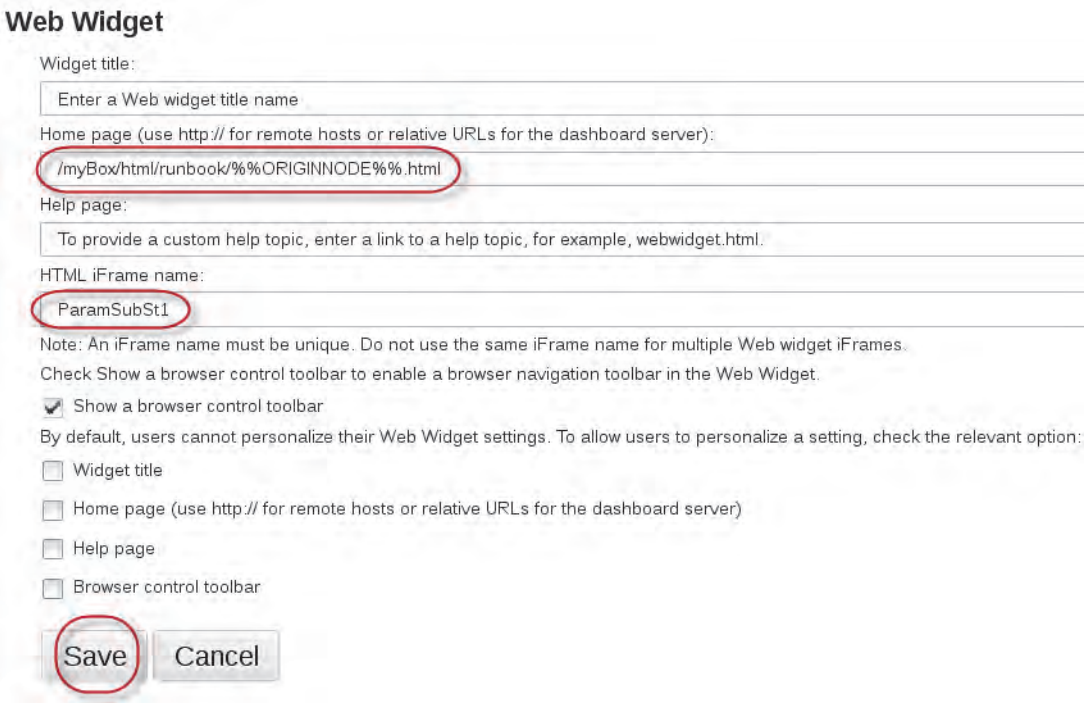


   b. Edit the web widget configuration properties. Click the **Edit options** icon in the upper right of the widget and select **Edit** or click the widget and select the **Widget > Edit** menu above the widget palette.

   c. Use the IBM Monitoring ORIGINNODE value that is generated with the NodeClickedOn event in the web widget home page address. The value is listed in Figure 2, “IBM Tivoli Monitoring NodeClickedOn parameters and values,” on page 5-14. Enter **/myBox/html/runbook/%%ORIGINNODE%%.html** in the **Home page** field.

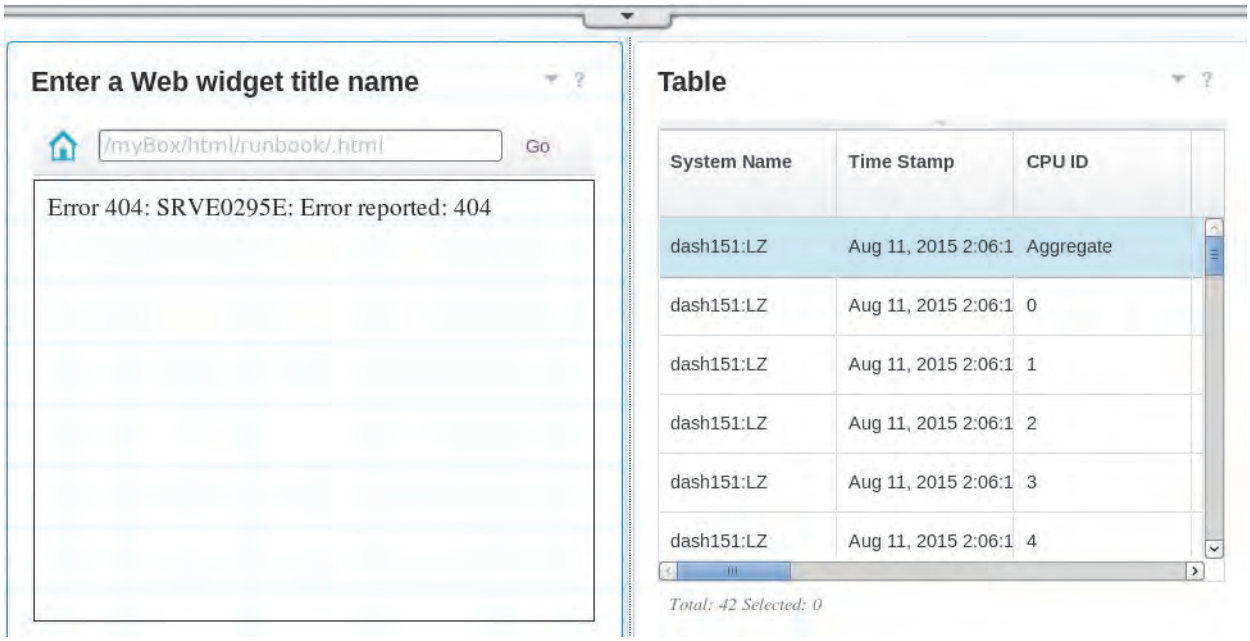
Note: The value of a NodeClickedOn event that contains the parameter name ORIGINNODE is substituted in the corresponding name that is defined in the Home page field. Enclose the substituted parameter name with two per cent sign pairs (**%%**ParameterName**%%**). You can pass multiple parameter names to a URL with a NodeClickedOn event. You separate multiple parameter value with the ampersand (&) character.

   d. Enter **ParamSubSt1** in the **HTML iFrame name** field and click **Save**.



The web widget initially shows an Error 404 message. This message is expected because no NodeClickedOn event is generated without selecting an item in the table widget.

4. Enable the web widget anchor points by clicking the web widget border.
5. Drag the widget anchor points to resize the widget to use half of the page canvas.
6. Position the mouse cursor over the widget title bar until the cursor icon changes to a closed hand. Drag the web widget title bar to position the widget on the left half of the page canvas.
7. Resize the table widget to use half of the page canvas.

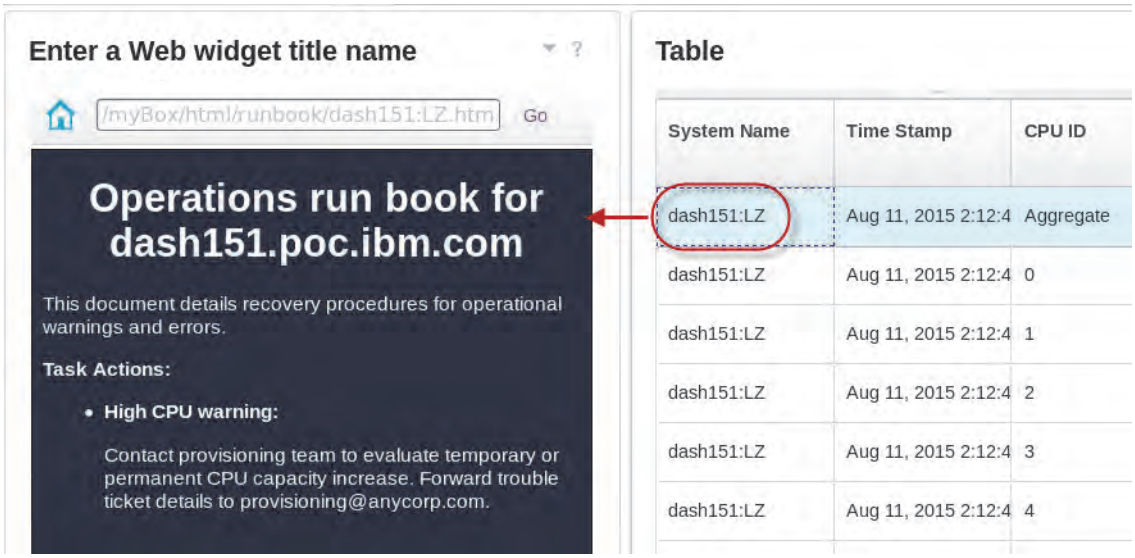


8. Save the page and close the page editor. Click **Save and Exit** in the upper right of the console.

Testing the parameter substitution

1. Select the first dash151:LZ entry in the System Name column.

   The ORIGINNODE parameter and value is passed to the web widget and substituted in the widget web address, which corresponds to a sample html file.



2. Select the additional rows of the table widget and notice how the web widget address and content changes.

Note: The laboratory environment contains runbook html files for only the dash151, tbsm152, and sysitmsles systems. Selecting other table entries generates a 404 error.

3. Close the **Parameter Substitution Page** page when you finish examining the page behavior. Do not log off the console.

Exercise 3 Using parameter substitution to create and show dynamic text

In this exercise, you configure a web widget to use a JavaServer Page (JSP) file to dynamically create HTML text. The HTML text is created based on NodeClickedOn parameters that are passed from a source widget. You create two pages in this exercise. The top-level page, **Dynamic_Parameter_Substitution_Page1**, contains a list widget. When an item in the list widget is selected, a NodeClickedOn event is generated and a second-level page, **Dynamic_Parameter_Substitution_Page2**, is opened. The second-level page contains a web widget and an analog gauge widget. The input parameters to the JSP page are defined in the web widget home address and parameter substitution with the top-level page NodeClickedOn event.

The JSP file that is used in this exercise contains the following lines:

```
Line 01:<%@ page import="java.util.*" %>
Line 02:<html>
Line 03:<head>
Line 04:<title>Dynamic Text Example</title>
Line 05:<link href="dynamicTextExample.css" rel="stylesheet" type="text/css">
Line 06:</head>
Line 07:<body>
Line 08:<h1> <%=request.getParameter("TbsmServiceInstanceName")%> Open Critical
Trouble Ticket Count </h1>
Line 09:<h1> <%=request.getParameter("TbsmServiceInstanceDisplayName")%> Region
</h1>
Line 10:</body>
Line 11:</html>
```

The following list shows a line-by-line explanation of the JSP file:

- **Line 01** imports the **java.util** libraries to make them available to the Java code.
- **Line 02** starts the main <html> tag.
- **Line 03** starts the <head> tag.
- **Line 04** defines the HTML page title.
- **Line 05** links to the cascading style sheet (CSS) file that is used in this exercise. The CSS file contains a single **h1** tag format definition, **h1 {color:black; text-align: center; font-family: Arial, Verdana;}**. The definition configures the h1 tag to use black as the text color, center the text, and use one of two default font types.
- **Line 06** closes the head tag.
- **Line 07** starts the <body> tag. The body tag configuration generates the data that is shown in the web widget.
- **Line 08** builds the first <h1> tag. A JSP expression returns the value of the TbsmServiceInstanceName input parameter. The parameter name and value is provided in the

web widget URL. The table widget NodeClickedOn event parameter value was substituted in the corresponding part of the web widget home page (**%%**TbsmServiceInstanceName**%%**). The **request.getParameter()** method returns a string that is used with the static text to create the first line of text that is shown in the web widget.

- **Line 09** builds the second <h1> tag content. A JSP expression returns the value of the TbsmServiceInstanceName input parameter. The parameter name and value is provided in the web widget URL. The table widget NodeClickedOn event parameter value was substituted in the corresponding part of the web widget home page (**%%**TbsmServiceInstanceDisplayName**%%**). The **request.getParameter()** method returns a string that is used with the static text to create the second line of text that is shown in the web widget.
- **Line 10** closes the <body> tag.
- **Line 11** closes the main <html> tag.

You must complete the following tasks in this exercise:

- **Create the second-level page**: This page contains an analog gauge widget and a web widget. Both widgets use context information that is supplied to the page with a source event wire.
- **Create the top-level page**: This page contains a list widget that shows data from a Tivoli Business Service Manager data set. You click an item in the list widget to send a NodeClickedOn event to the second-level page.
- **Create an event wire that connects the top-level page to the second-level page**: The event wire sends a NodeClickedOn event to the second-level page when an item in the top-level page is clicked. The NodeClickedOn event parameters and values are listed in Figure 1, “IBM Tivoli Business Service Manager NodeClickedOn parameters and values,” on page 5-13.
- **Test the dynamic parameter substitution**: You click items in the top-level page and verify the changes in the data that is shown in the second-level page.

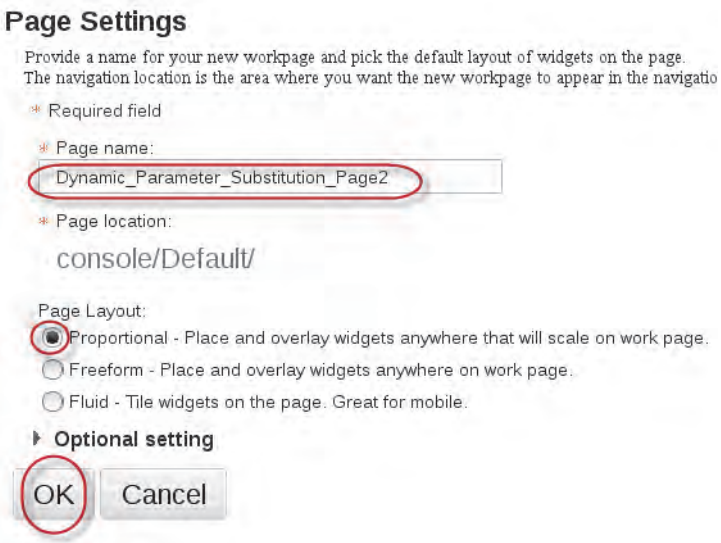
Perform the following steps to complete the exercise.

Creating the second-level page

1. To create and configure the two pages with a minimum amount of page editing, create the second-level page first, the target page.
   a. Click the **Create New Page** icon (**+**) in the upper right of the console.

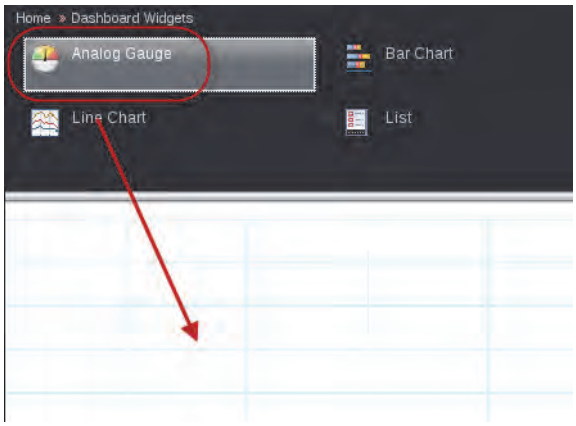
   b. Complete the page settings form. Enter **Dynamic_Parameter_Substitution_Page2** in the **Page name** field. Leave the default **Page location** setting and select **Proportional** in the **Page Layout** list.

c. Click **OK**.



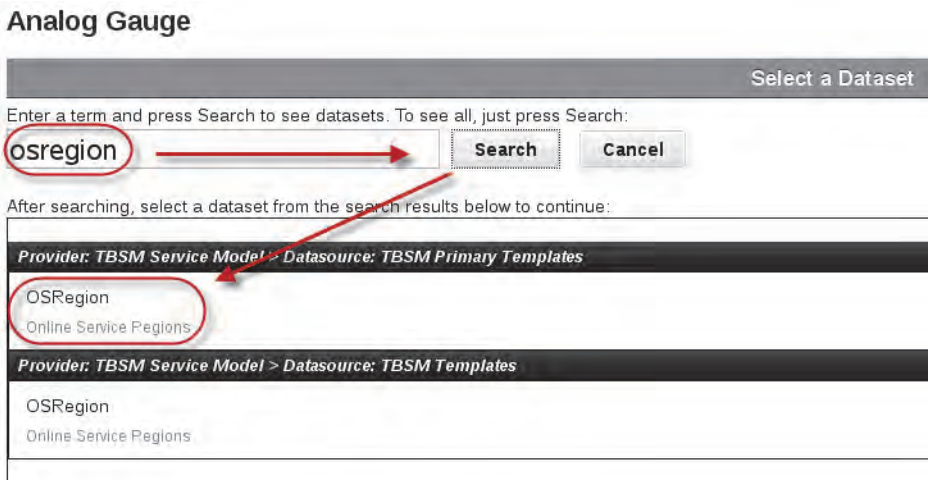
d. Click the **Dashboard Widgets** catalog icon in the widget palette section.

e. Drag an analog gauge instance from the widget palette to the page canvas.



f. Configure this gauge to show data from a Tivoli Business Service Manager service model. Enter **osregion** in the data set search field and click **Search**.

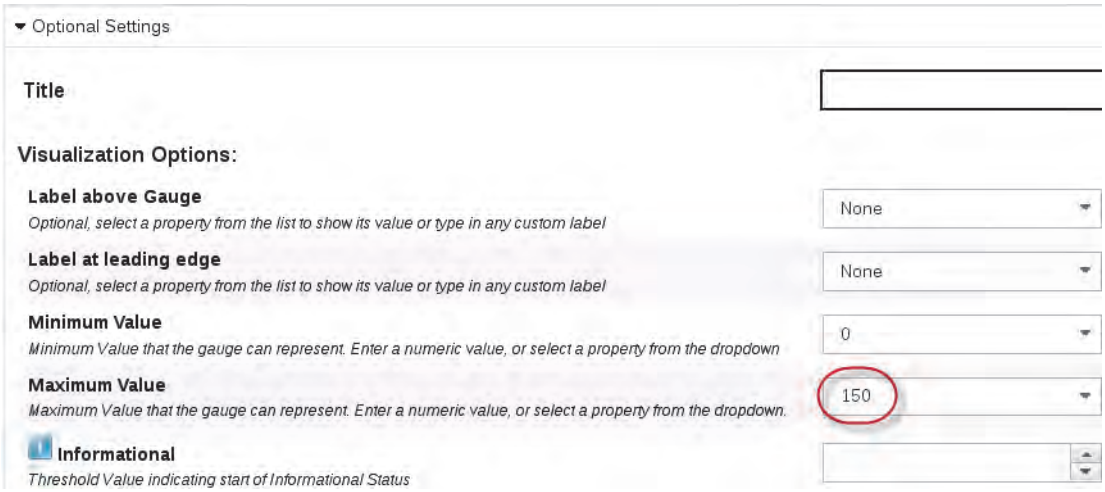
g. Select **OSRegion** in the **TBSM Primary Templates** data source entry that is shown in the filtered list.



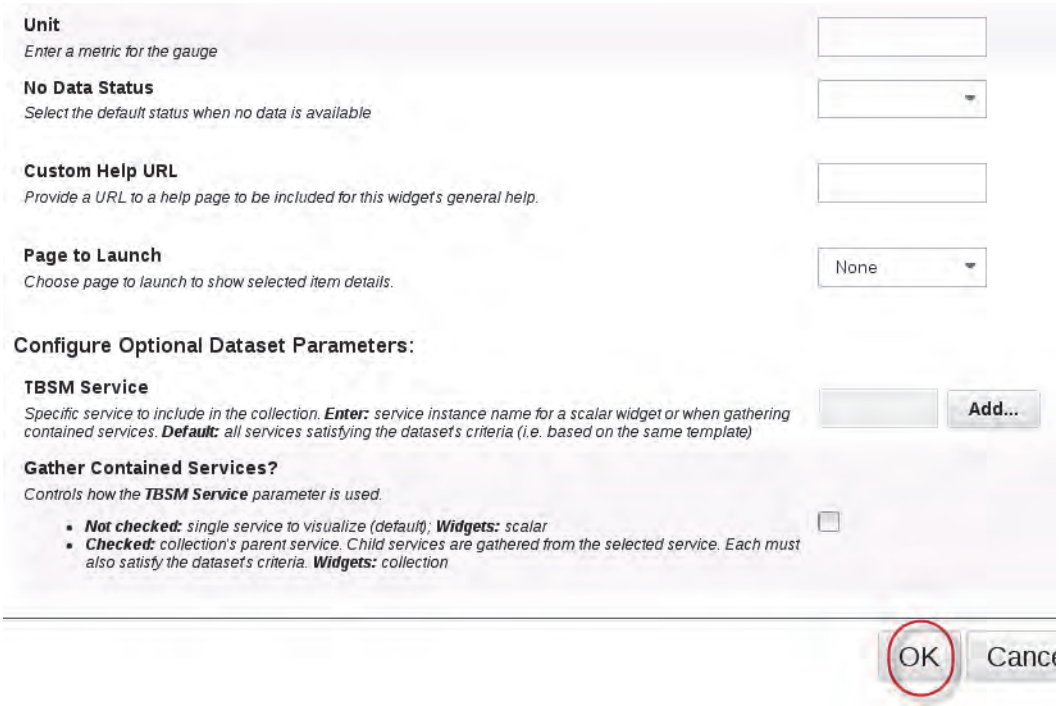
h. Configure the data set column visualization parameter for the gauge. This gauge is configured to show the total number of open critical severity trouble tickets for a selected region of a company. Select **RegSumCritical** in the **Value** menu.

i. Click the icon to the left of **Optional Settings**.

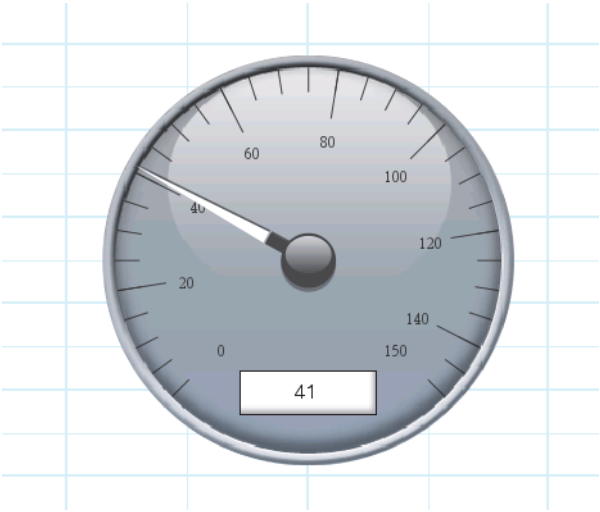
j. Leave all default values, except the Maximum Value. Enter **150** in the **Maximum Value** field.



k. You do not specify a business service in the **Configure Optional Dataset Parameters** section. The business service information is sent with a widget event wire that is connected to the top-level page that you create in this exercise. Click **OK** to save the widget configuration.



Note: The value that is initially shown in the analog gauge is based on the first matching business service in the configured data set. In operation, the page is seen only when a business service is selected in the top-level page.



l. Click the gauge widget title bar and drag the widget so that it is centered at the bottom of the page canvas.

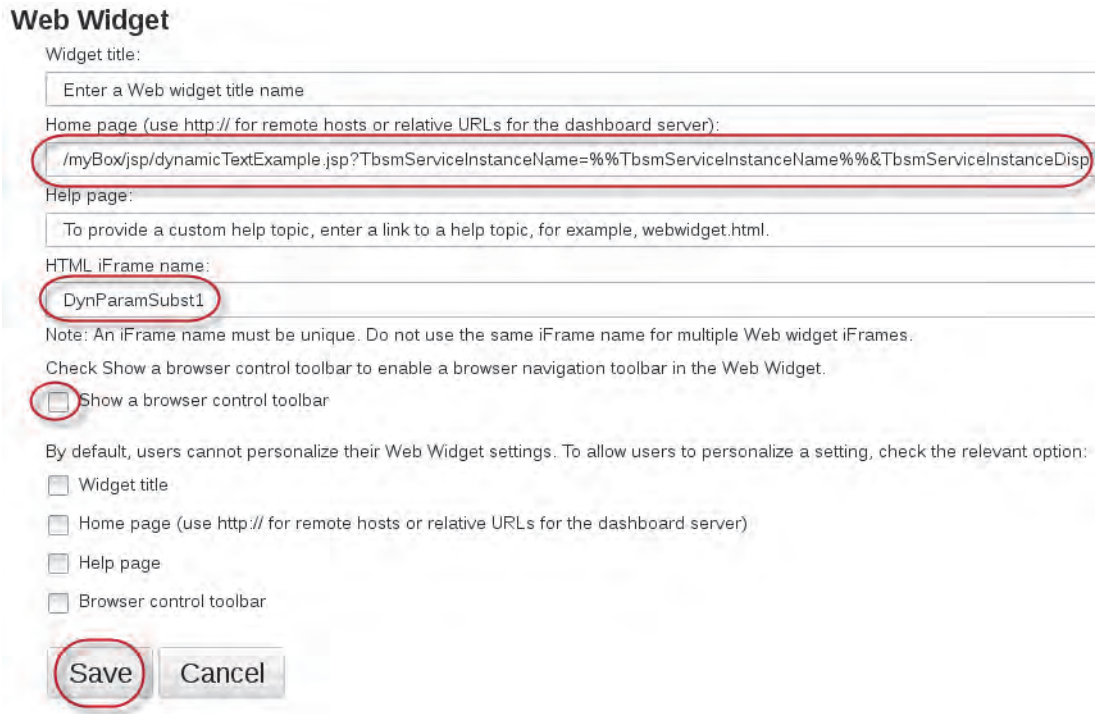
m. Add a web widget to the page. Click the web widget icon in the widget palette and drag an instance to the page canvas.

n. Configure the web widget properties. Click the **Edit options** icon in the upper right of the widget and select **Edit**.

o. Leave the default value in the **Widget title** field.

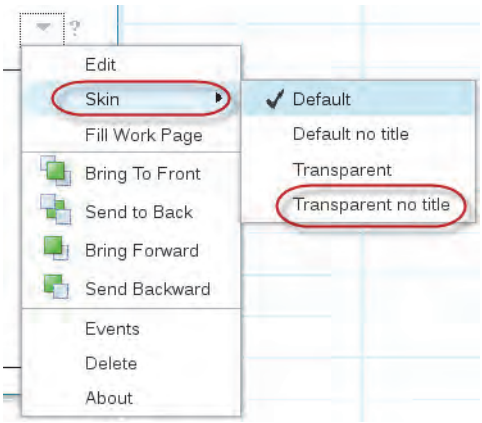
p. Enter the following value for the **Home page** field:

```
/myBox/jsp/dynamicTextExample.jsp?TbsmServiceInstanceName=%%TbsmServiceInstanceName%%&TbsmServiceInstanceDisplayName=%%TbsmServiceInstanceDisplayName%%
```

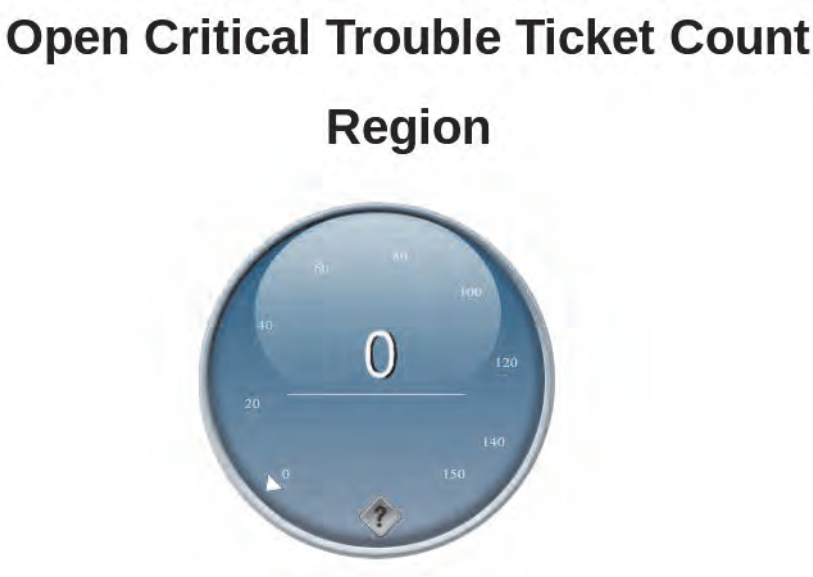


Note: The URL passes two parameters in the query string portion of the web address. The general form for a URL that is used with a web program is http://<*server*>/<*web_program_file*>?<*query_string*>. The question mark is the separator character between the program file and the query string. Query string items are written in the form *parameter_name=parameter_value*. Multiple input values are separated with the ampersand (&) character. Spaces are not allowed in the URL. If a parameter value requires a space, the space character is substituted with the hexadecimal notation %20. The parameter names, TbsmServiceInstanceName and TbsmServiceInstanceDisplayName, are used in the JSP page. The parameter values, %%TbsmServiceInstanceName%% and %%TbsmServiceInstanceDisplayName%%, are substituted with corresponding NodeClickedOn event values. The NodeClickedOn event is generated when a widget item in the top-level page is selected.

2. Enter **DynParamSubst1** in the **HTML iFrame name** field. The iFrame name is used by some programs to specify which parts of a web page should be updated with new data.
3. Remove the selection for **Show a browser control toolbar**.
4. Leave all other options at their default values and click **Save** to save the widget configuration.
5. Resize the web widget so that each line of text is shown on a single separate line. Move the widget to the top center of the page canvas.
6. Set the web widget skin configuration. Click the **Edit options** icon in the upper right of the widget and select **Skin > Transparent no title**.



The page canvas looks like the following screen image.



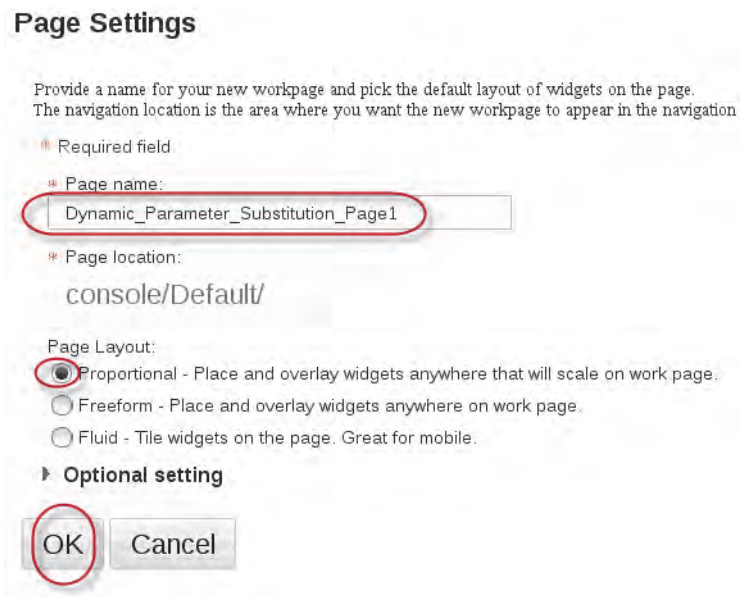
7. Save the page configuration. Click **Save and Exit** in the upper right of the console.

Creating the top-level page

1. Create the top-level page, Dynamic_Parameter_Substitution_Page1.
   a. Click the **Create New Page** icon (**+**) in the upper right of the console.

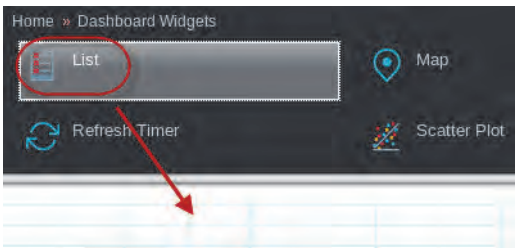
2. Complete the page settings form. Enter **Dynamic_Parameter_Substitution_Page1** in the **Page name** field. Leave the default **Page location** setting and select **Proportional** in the **Page Layout** list.

3. Click **OK**.



4. Click **Dashboard Widgets** in the widget palette.

5. Click and drag a list widget instance from the widget palette to the page canvas.

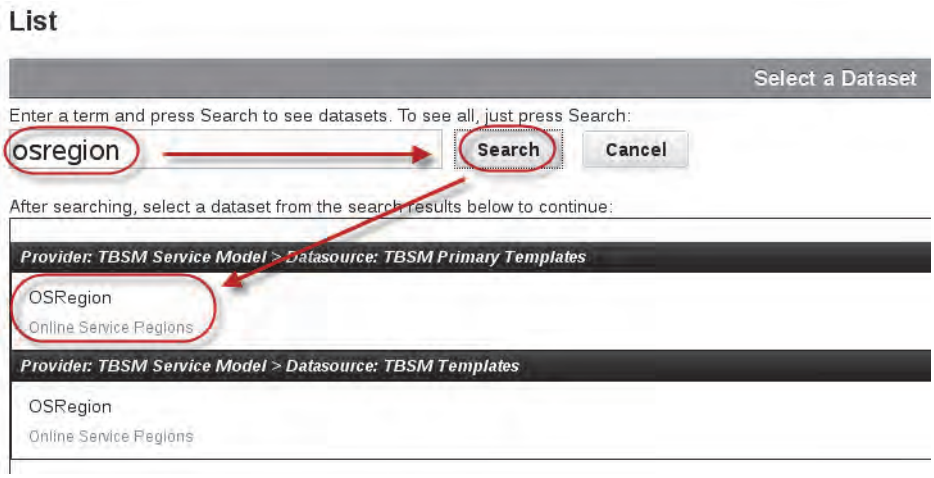


6. Edit the list widget properties. Click the **Edit options** icon in the upper right of the widget and select **Edit**.

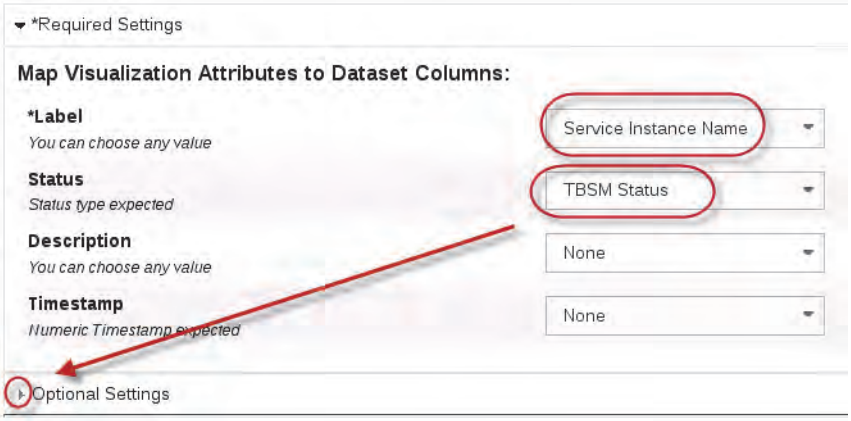
   You configure this widget to show a list of regional business services for the AnyBank business service model. The AnyBank service model is defined in the Tivoli Business Service Manager server.

7. Enter **osregion** in the data set search field and click **Search**.

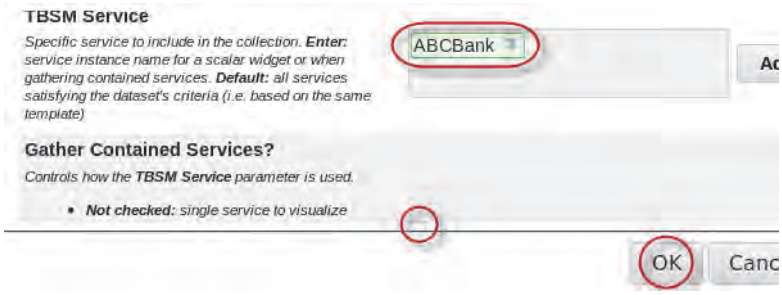
8. Click **OSRegion** in the **TBSM Primary Templates** data source listing.



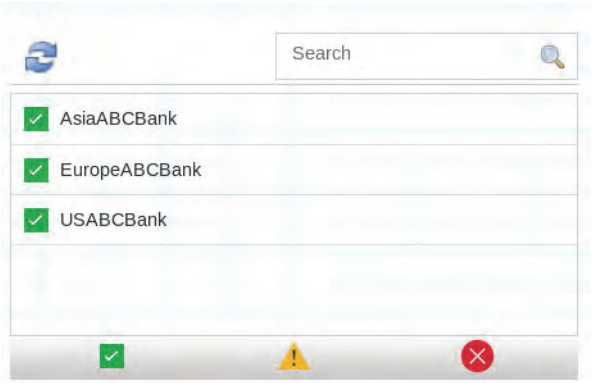
9. Configure the Required Settings section. Select **Service Instance Name** in the **Label** menu.
10. Select **TBSM Status** in the **Status** menu.
11. Leave the default values for the **Description** and **Timestamp** menus.
12. Configure the widget optional settings. Click the icon to the left of **Optional Settings**.



13. Configure the TBSM Service value in the Configure Optional Dataset Parameters section. Enter **ABCBank** in the **TBSM Service** field.
14. Select the **Gather Contained Services** option and click **OK**.



The list widget shows a list and status of three regional services in the ABCBank service model.



Save the page, but do not exit the page editor until you complete the next task.

Creating an event wire that connects the top-level page to the second-level page

1. Create an event wire from the list widget in the top-level page:
   a. Click the **Show Wires** icon in the upper right of the console.

   A Summary of wires table opens above the page canvas.

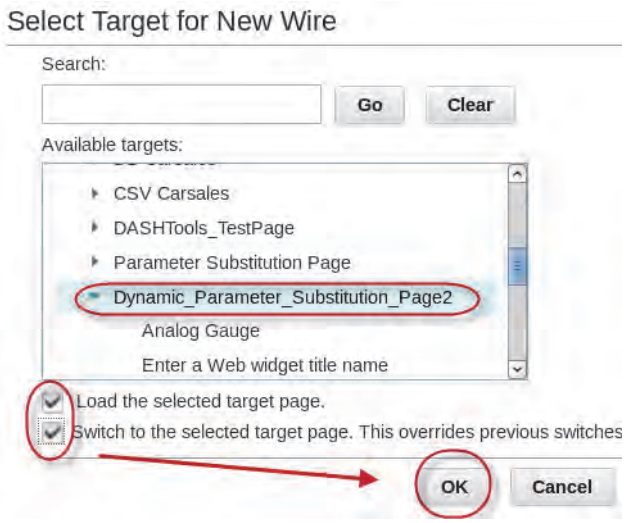
   b. Click **New Wire** in the upper left of the console.

   c. Select the wire source. Click the icon to the left of **List** in the **Available source events** section.

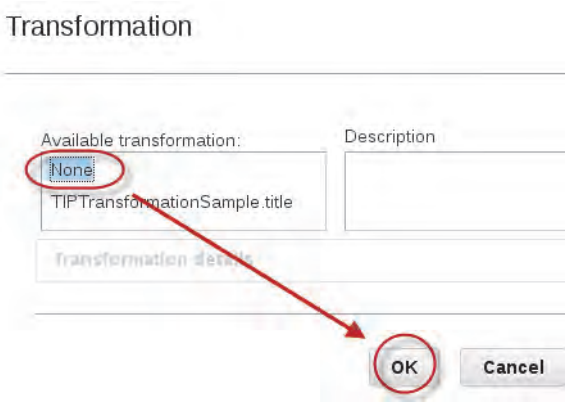
d. Click **NodeClickedOn** and click **OK**.

2. Select the event wire target. Click **Dynamic_Parameter_Substitution_Page2** and click **OK**. Do not select an individual page widget. Selecting the page sends the NodeClickedOn event to all widgets on the page.

3. Select the **Load the selected target page** and **Switch to the selected target page** options.

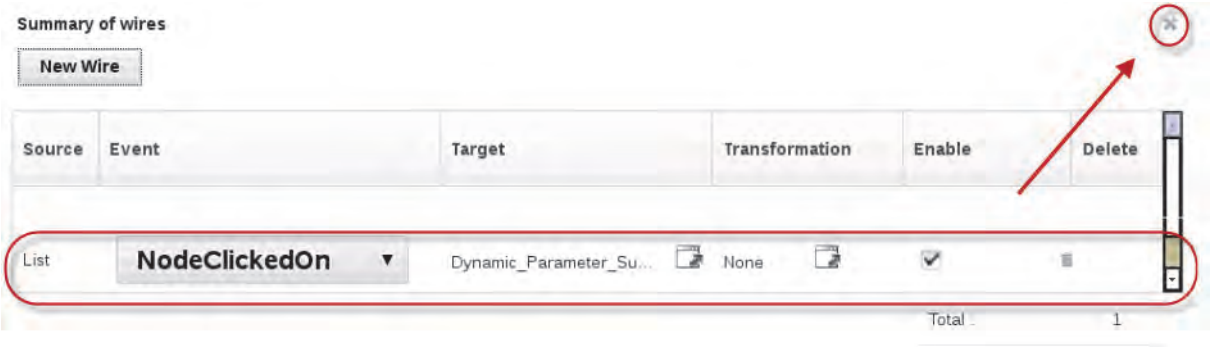


4. Do not select a transformation for the event. Click **None** and **OK** in the Transformation section.



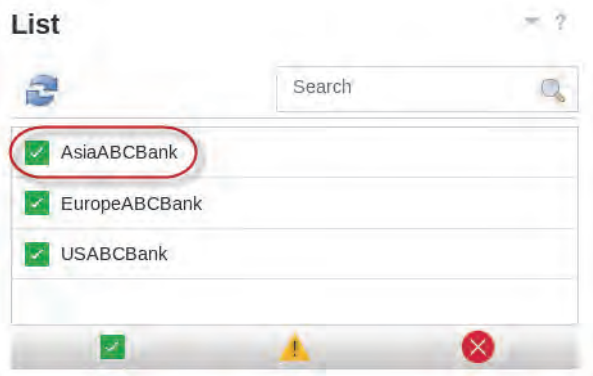
You see the wire event summarized in the table.

5. Close the Summary of wires table. Click the **X** symbol in the upper right of the table. Do not click the X icon to close the browser application.



6. Save the page configuration. Click **Save and Exit** in the upper right of the console.

7. To test the event wire, web substitution, and JSP file, click **AsiaABCBank** in the list widget in the top-level page.



The second-level page is opened and the configured NodeClickedOn event parameter values are shown in the analog gauge and web widgets.

8. Click other entries in the top-level page list widget and verify that the information that is shown in the second-level page shows the data for the new selection.

Exercise 4 Using hotspot widgets with web widgets

In this exercise, you learn how to configure hotspot widgets to change the data that is shown in a web widget in a dashboard page. You configure the hotspot widgets to send displayURL and NodeClickedOn events to web widgets. The hotspot widgets are arranged on a map image, as an example of the use of static graphics and hotspots to create dashboard navigation visualizations.

Three hotspot widgets on the page are configured to send displayURL events to a specific web widget. The event target for each displayURL event is based on the unique iFrame name in each web widget. One hotspot widget is configured to send a NodeClickedOn event to a specific web widget, which is based on the widget iFrame name. The NodeClickedOn event is configured to send a parameter name and value that is substituted in the web widget URL.

You complete the following tasks:

- **Create the dashboard page and add the web and image widgets**: You create the dashboard page and add the web widgets that are changed by clicking a corresponding hotspot widget.
- **Add and configure the hotspot widgets**: You add hotspot widgets to the page and configure each hotspot widget to change a web widget. The target of each hotspot widget is specified with the iFrame name that you used to uniquely identify each web widget on the page.
- **Test the hotspot widgets**: You click each hotspot widget and verify that the data that is shown in each corresponding web widget is changed.

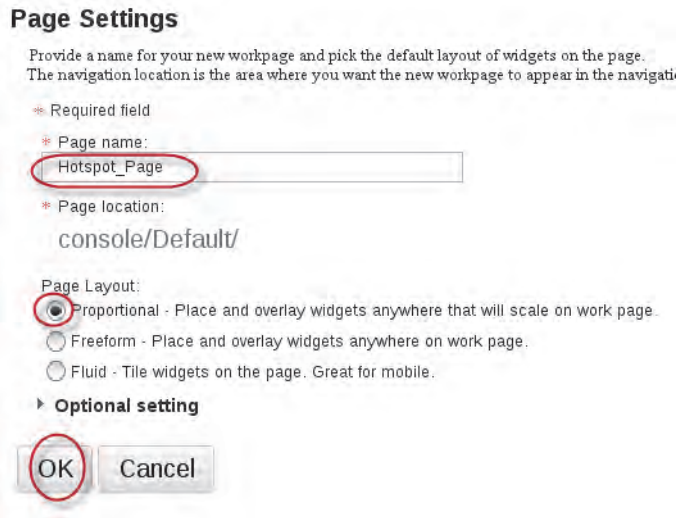
Perform the following steps to complete the exercise.

Creating the dashboard page and adding the web and image widgets

1. Create a dashboard page.
   a. Click the **Create New Page** icon (**+**) in the upper right of the console.

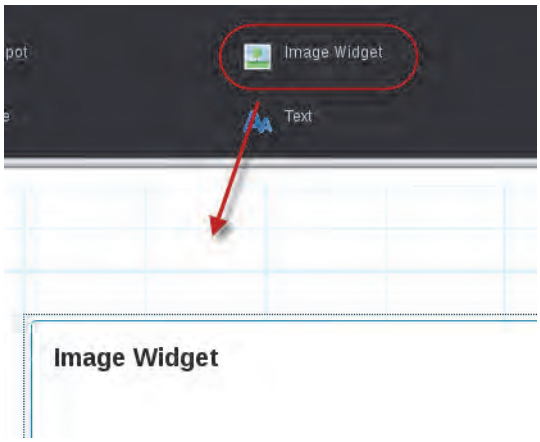
   b. Complete the page settings form. Enter **Hotspot_Page** in the **Page name** field. Leave the default **Page location** setting and select **Proportional** in the **Page Layout** list.

   c. Click **OK**.



2. Click the **Dashboard Widgets** in the widget palette section of the page editor.

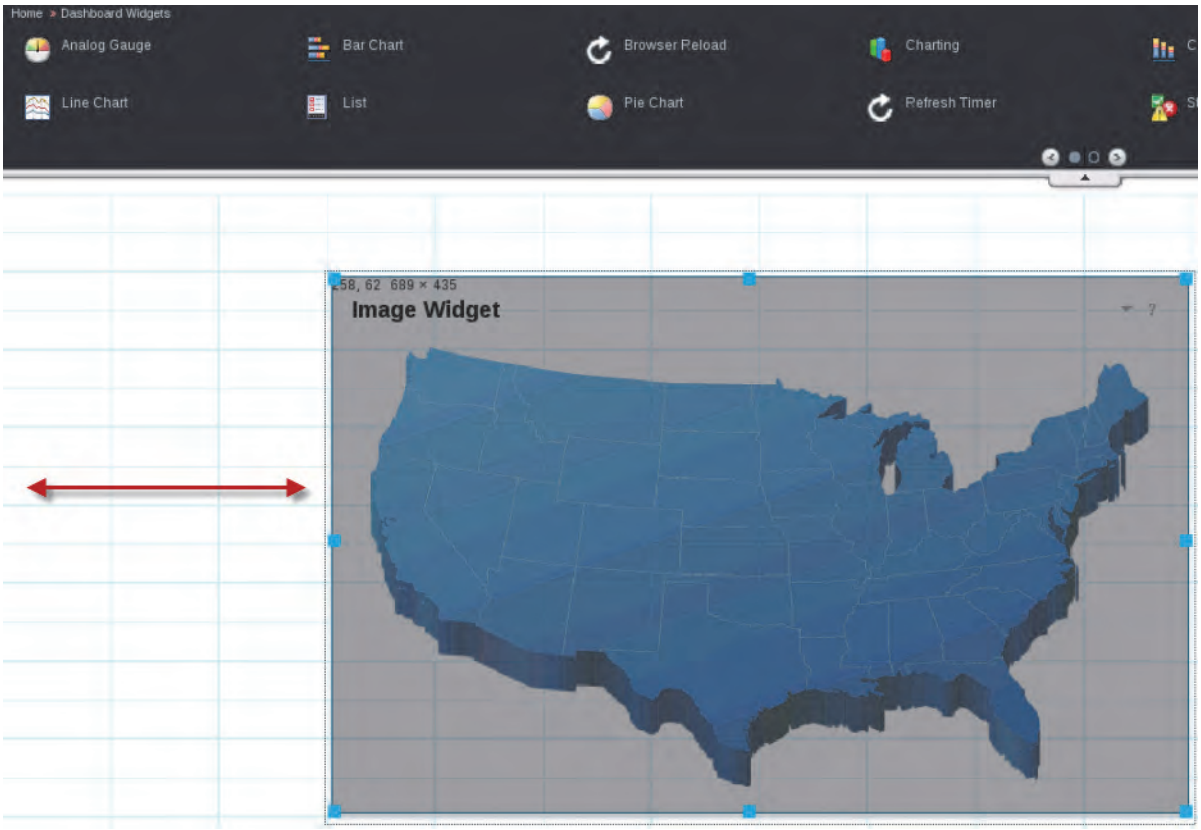
3. Add an image widget instance to the page. Click and drag the **Image Widget** icon to the page canvas.



4. Click **Edit options** and select **Edit**.
5. Configure the widget to show a map of the United States. Enter **/myBox/maps/us_map.png** in the **Image URL** field.
6. Select **Do not scale** and use the default image alignment.
7. Click **Save**.

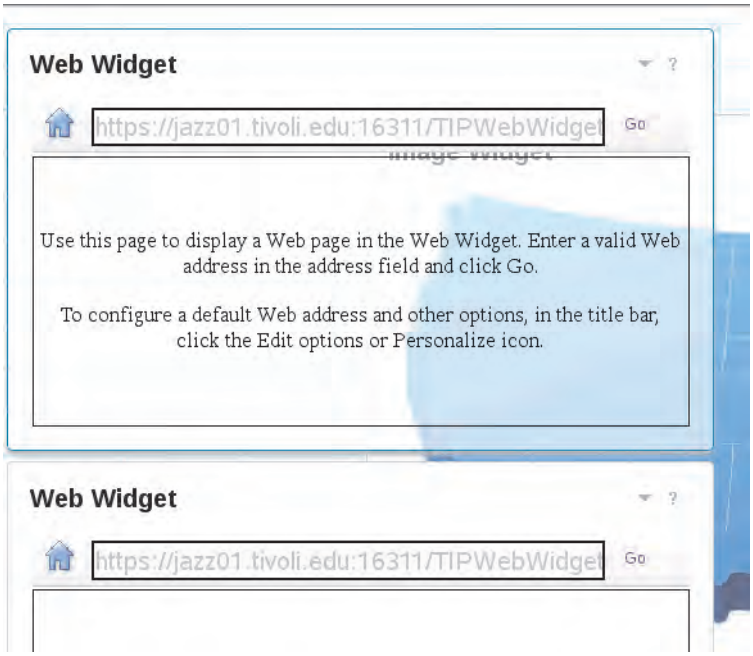
8. Click the widget edge to enable the widget anchor points. Resize the widget so that the map image is shown in the widget frame.

9. Move the widget to the left side of the page canvas, approximately 2 - 3 canvas grid blocks from the left edge.



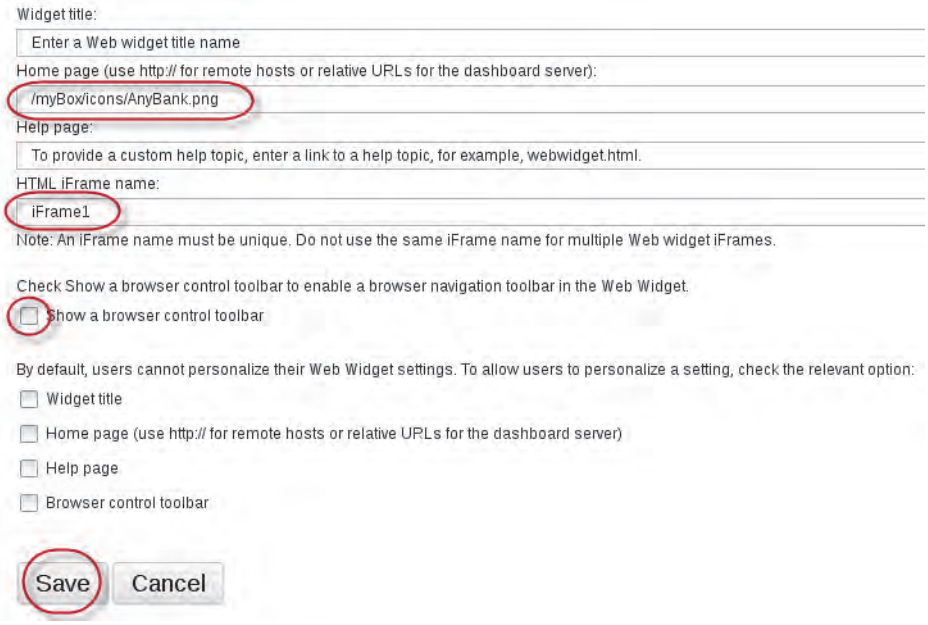
10. Drag two web widget instances to the page canvas, arranging them in a vertical stack to the left of the image widget.

Note: The size of each widget is adjusted after the widget configuration so that they do not obscure the image widget.

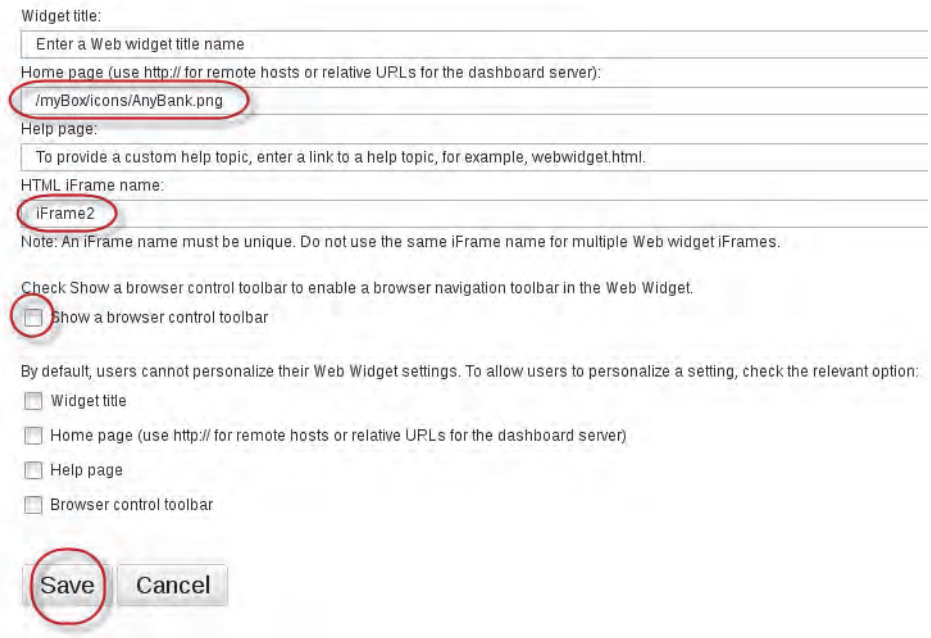


11. Configure the upper left web widget.
    a. Click **Edit options** and select **Edit** in the widget frame.
    b. Leave the default widget title value.
    c. Enter **/myBox/icons/AnyBank.png** in the **Home page** field.
    d. Each hotspot widget on the page is configured to send NodeClickedOn events to a specific web widget with the unique iFrame name. Enter **iFrame1** in the **HTML iFrame name** field.
    e. Remove the **Show a browser control toolbar** selection.

f. Leave all other default values and click **Save**.

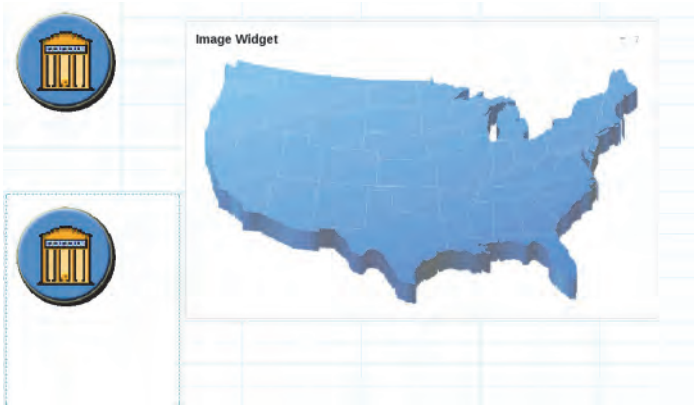


12. Configure the lower left web widget.
    a. Click **Edit options** and select **Edit**.
    b. Leave the default widget title value.
    c. Enter **/myBox/icons/AnyBank.png** in the **Home page** field.
    d. Enter **iFrame2** in the **HTML iFrame name** field.
    e. Remove the **Show a browser control toolbar** selection.
    f. Leave all other default values and click **Save**.



13. Resize the web widgets on the left size of the page to a width of three canvas grid blocks and a height of six canvas grid blocks.
14. Set the skin property for each widget to **Skin > Transparent no title**.

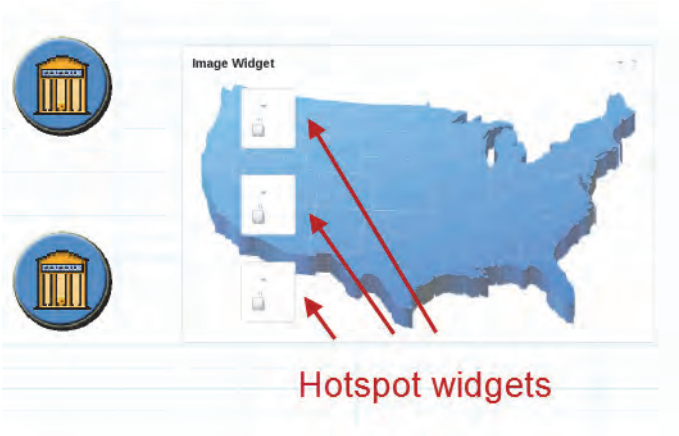
    The page layout should look like the following screen capture.



Note: The screen resolution affects the proportions of the widgets on the page. If the widgets show scrollbars, increase the widget size by one canvas grid.

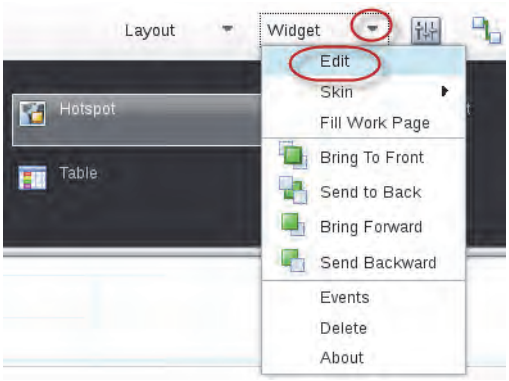
Adding and configuring the hotspot widgets

1. Add three hotspot widgets to the page canvas. As you place each hotspot widget instance on the canvas, resize the widget so that it is one canvas grid block wide by three canvas grid blocks high. Place three of the hotspot widgets on the left side of the map image and one instance on the right side of the map image. The following screen capture shows the hotspot widget arrangement.



You configure the three hotspot widgets on the left of the image widget to change the image that is shown in one of the web widgets on the left. The target web widget image is changed by sending a **displayURL** event as each hotspot widget is selected.

2. Configure the upper left hotspot widget properties.
   a. Click the widget **Edit options** icon and select **Edit** or click the widget and select **Widget > Edit** in the menus above the widget palette section of the page.

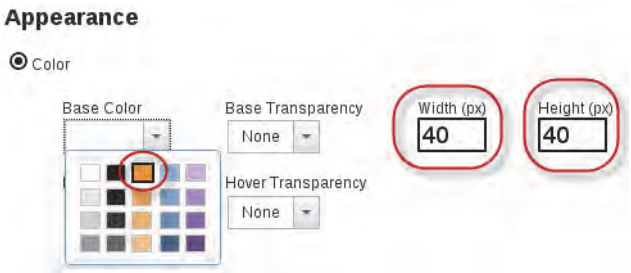


   b. Configure the widget to show a 40x40 pixel icon. Select **Color** in the **Appearance** section.

   c. Click the **Base Color** menu and select the dark orange color in the center of the top row of the color palette.

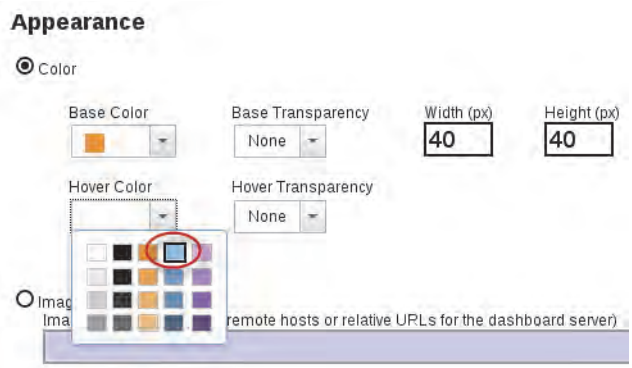
   d. Leave the default value for **Base Transparency**.

   e. Enter **40** in the **Width (px)** and **Height (px)** fields.



   f. Click the **Hover Color** menu and select the dark blue color that is the second from the right of the top row of the color palette.

   g. Leave the default value for **Hover Transparency**.



h. Configure the Hover Tooltip value so that you can quickly see the hotspot target during testing. This widget is configured to change the image file that is used by the upper left web widget (iFrame1). Enter **iFrame1 - AnyInsurance.png** in the **Hover Tooltip** field.

i. Select **Send Event** in the **Action** menu.

j. Select **Web Widget Display URL** in the **Event** menu.

k. Enter **/myBox/icons/AnyInsurance.png** in the **Target URL** field.

l. Enter **iFrame1** in the **Target Frame** field and click **OK**.

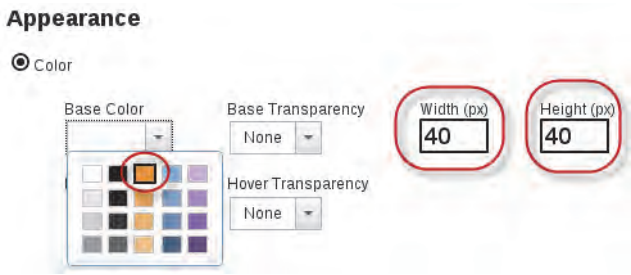
3. Edit the middle left hotspot widget properties.

a. Configure the widget to show a 40x40 pixel icon. Select **Color** in the **Appearance** section.

b. Click the **Base Color** menu and select the dark orange color in the center of the top row of the color palette.

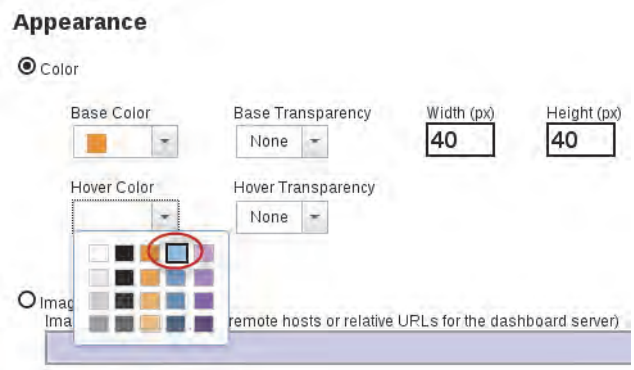
c. Leave the default value for **Base Transparency**.

d. Enter **40** in the **Width (px)** and **Height (px)** fields.



e. Click the **Hover Color** menu and select the dark blue color that is the second from the right of the top row of the color palette.

f. Leave the default value for **Hover Transparency**.



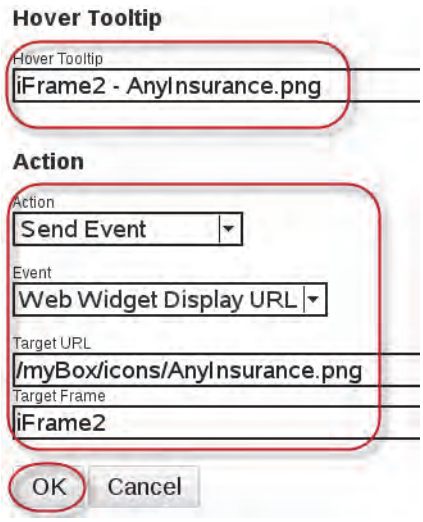
g. Configure the Hover Tooltip value so that you can quickly see the hotspot target during testing. This widget is configured to change the image file that is used by the lower left web widget (iFrame2). Enter **iFrame2 - AnyInsurance.png** in the **Hover Tooltip** field.

h. Select **Send Event** in the **Action** menu.

i. Select **Web Widget Display URL** in the **Event** menu.

j. Enter **/myBox/icons/AnyInsurance.png** in the **Target URL** field.

k. Enter **iFrame2** in the **Target Frame** field and click **OK**.



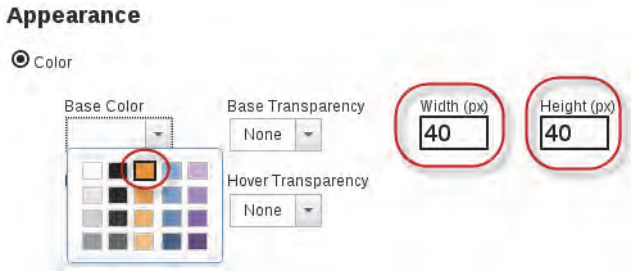
4. Edit the lower left hotspot widget properties.

a. Configure the widget to show a 40x40 pixel icon. Select **Color** in the **Appearance** section.

b. Click the **Base Color** menu and select the dark orange color in the center of the top row of the color palette.

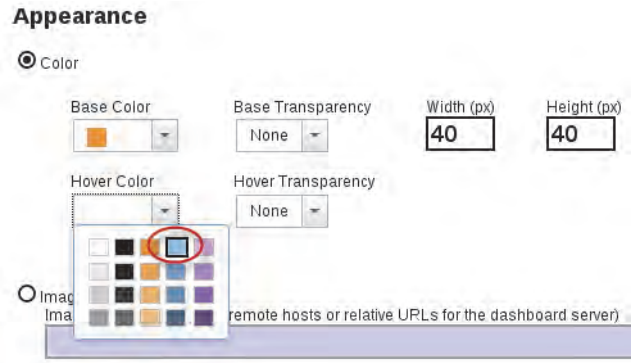
c. Leave the default value for **Base Transparency**.

d. Enter **40** in the **Width (px)** and **Height (px)** fields.



e. Click the **Hover Color** menu and select the dark blue color that is the second from the right of the top row of the color palette.

f. Leave the default value for **Hover Transparency**.



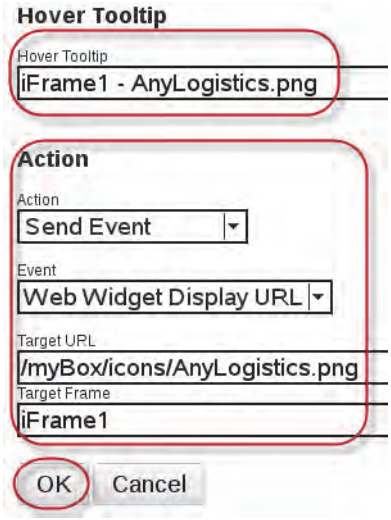
g. Configure the Hover Tooltip value so that you can quickly see the hotspot target during testing. This widget is configured to change the image file that is used by the upper left web widget (iFrame1). This hotspot widget uses a different image file than you configured for the upper left hotspot widget. Enter **iFrame1 - AnyLogistics.png** in the **Hover Tooltip** field.

h. Select **Send Event** in the **Action** menu.

i. Select **Web Widget Display URL** in the **Event** menu.

j. Enter **/myBox/icons/AnyLogistics.png** in the **Target URL** field.

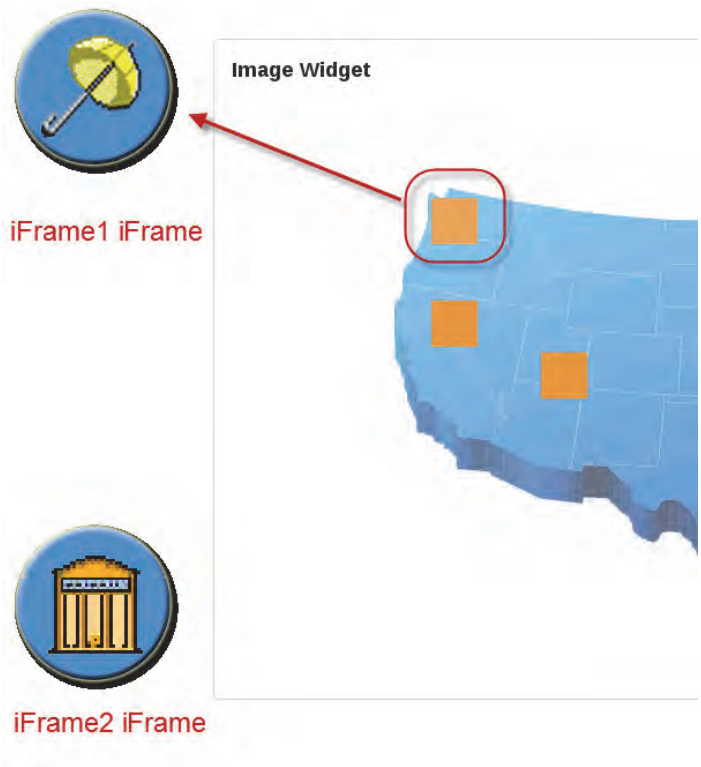
k. Enter **iFrame1** in the **Target Frame** field and click **OK**.



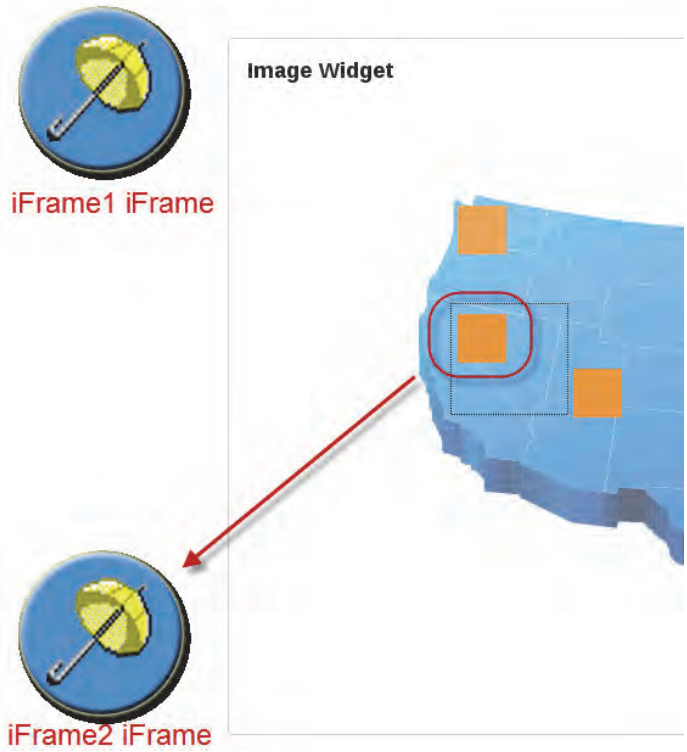
5. Edit the skin properties for each hotspot widget and select **Skin > Transparent no title**.
6. Save the page configuration. Click **Save and exit** in the upper right of the console.

Testing the hotspot widgets

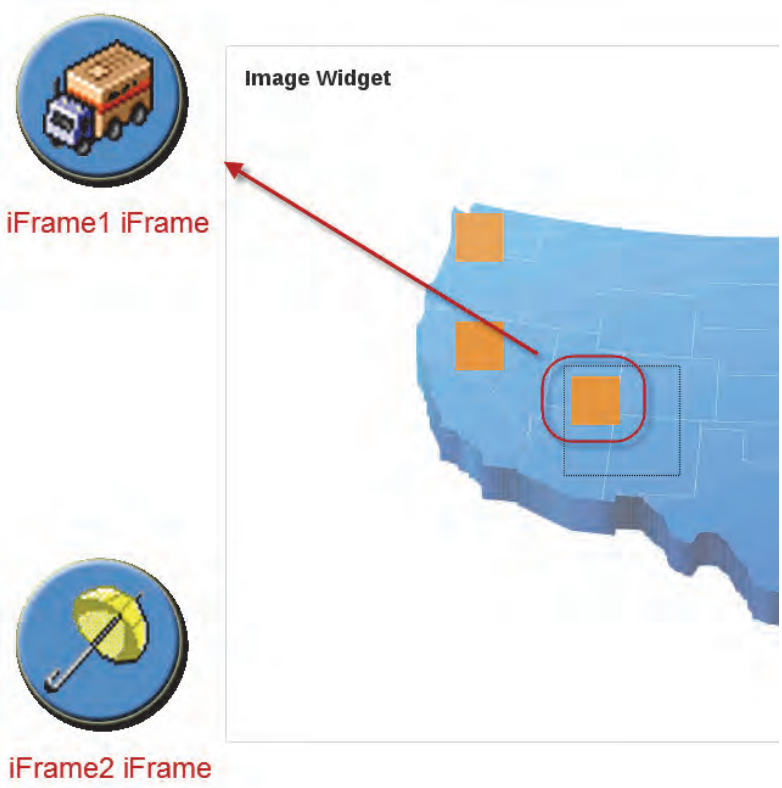
1. Test the operation of each hotspot widget.
   a. Click the upper left hotspot widget. The upper left web widget icon changes to show the AnyInsurance icon.



b. Click the middle left hotspot icon. The lower left web widget icon changes to show the AnyInsurance icon.



c. Click the lower left hotspot icon. The upper left web widget icon changes to show the AnyLogistics icon.



Unit 6 Integrating Netcool/OMNIbus with DASH exercises

Netcool/OMNIbus provides data feeds of event-based and metric-based data from the Web GUI server. In these exercises, you use several DASH widgets to add Netcool/OMNIbus event data to a DASH dashboard.

You configure a column chart widget to show a summary of the current total number of events, by severity.

You want a gauge that shows the total event records in the ObjectServer.

Exercise 1 Adding event data to a dashboard page

In this exercise, you create two dashboard pages that contain several examples of widgets that show event data. The widgets are configured to retrieve Netcool/OMNIbus event records with through the connection document that was created when Web GUI was installed as part of the laboratory setup.

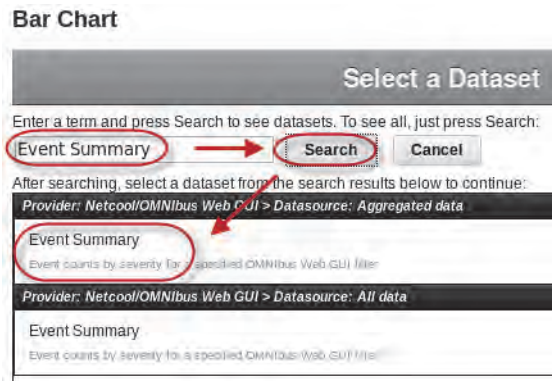
Perform the following steps:

1. Log in to the DASH console with the user ID **smadmin** and the password **object00**.
2. Create a page. Click the plus symbol (**+**) in the upper right of the console.
3. Enter **Event Data Page** in the name field, select **Proportional** for the page layout, and click **OK**.

Adding a column chart widget

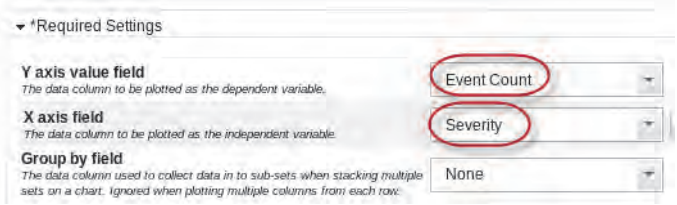
1. Drag a column chart widget to the page canvas.
2. Each of the Web GUI data sets provides event records in various forms. Some of the data sets summarize the event records, as aggregated. Some data sets provide the option of applying a Web GUI filter to the data. For this bar chart, you use aggregated data. Edit the column chart widget properties, enter **Event Summary** in the search field and click **Search**.

3. Select **Event Summary** below the **Aggregated data** entry.



This data source provides event data in a summarized form. The three data types are Event Grouping, Event Summary, and Filter Summary. Based on the data type that is selected, the event data is provided in a different summarized form. For this task, you create a column chart that shows the total number of event records for each value of the Severity column.

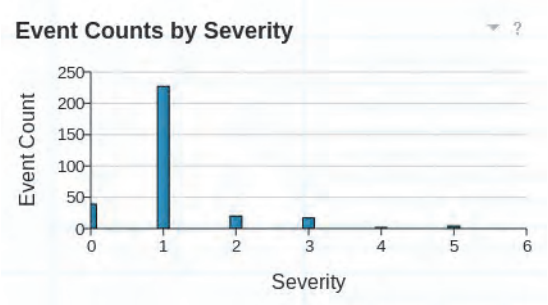
4. Select **Event Count** in the **Y axis value field** menu and select **Severity** in the **X axis field** menu.



5. Click **Optional Settings**.

6. Enter **Event Count by Severity** in the **Title** field and click **OK**.

   The total event records for each value of Severity are computed before the data is delivered to the widget. The computation is done by the UI data provider application in the Web GUI server.

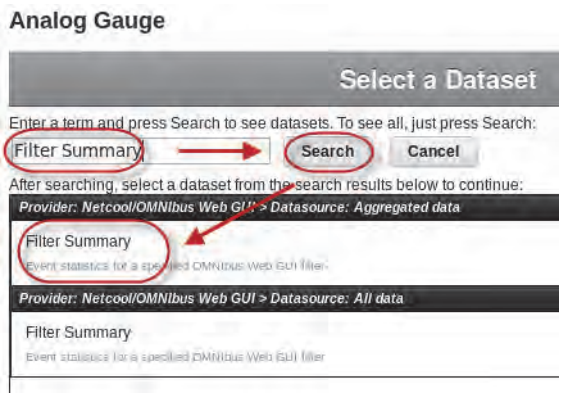


7. Save your work, but do not close the page editor. Click **Save** above the widget palette.

Hint: If the bar chart is empty, it might mean that you did not select a filter. It might also mean that the filter that you selected did not match any event records. You control which event records are used in the chart when you select the filter in the data set configuration. The filter does not control how the data is aggregated. The data set that is used in this example aggregates the events by severity.

Showing event data with a gauge widget

1. Drag an **Analog Gauge** widget to the page canvas.
2. Edit the widget properties.
3. Enter **Filter Summary** in the search field and click **Search** to generate the list of data sets.
4. The Event Summary data set that you used previously aggregates the event data by Severity. The Filter Summary data set also aggregates the event data. This data set aggregates the data based on a filter. Click the **Filter Summary** data set below the **Aggregated data** entry.



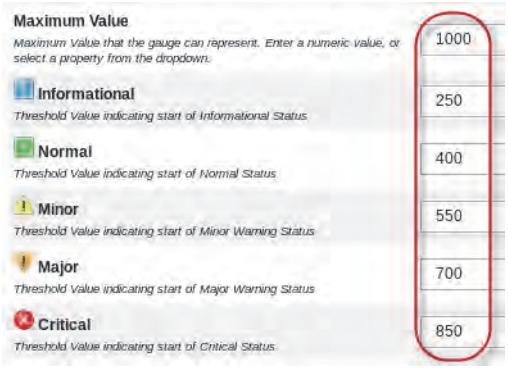
5. Select Total Event Count in the Value menu and select **AllEvents (global group)** in the **Filter to apply to the ObjectServer** menu.

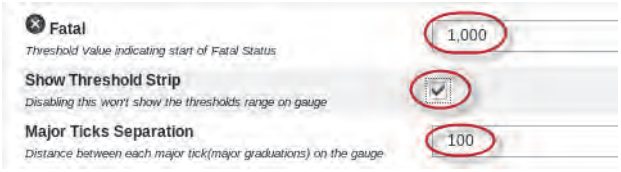
6. Click the arrow to expand **Optional Settings**.
7. Enter **Total Events** in the **Title** field.

8. Enter threshold values for the gauge. Use the values in the following table to complete the threshold configuration:

Table 1

Parameter Name	Value
Maximum Value	1000
Informational	250
Normal	400
Minor	550
Major	700
Critical	850





9. Select **Show Threshold Strip**, enter **100** in the **Major Ticks Separation** field, and click **OK**.

Hint: When the **Show Threshold Strip** option is selected, the gauge displays a colored band around its edge. If this feature is not selected, the entire gauge changes color to represent the threshold value as indicated by the settings that are configured.

10. You see the current total event count.

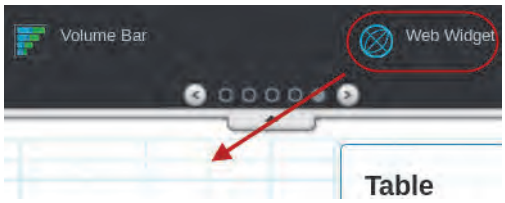
11. Save your work. Click **Save and Exit** above the widget palette.

Exercise 2 Creating dynamic OMNIbus event lists with DASH context events

In this exercise, you configure a web widget to show Netcool/OMNIbus events in an event viewer based on a transient filter. The transient filter definition uses parameter substitution with NodeClickedOn events to dynamically generate a list of events. You then configure a table widget

to show a list of Tivoli Business Service Manager services. Finally, you configure an event wire to direct NodeClickedOn events from the table widget to the event viewer in the web widget.

1. Create a dashboard page.
   a. Click the **Create New Page** icon (**+**) in the upper right of the console.
   b. Complete the page settings form. Enter **OMNIbus Dynamic Filter Page** in the **Page name** field, leave the default **Page location** setting, select **Proportional** in the **Page Layout** list, and click **OK**.
2. Add a web widget to the page and configure it.
   a. Drag the web widget icon from the widget palette to the page canvas.



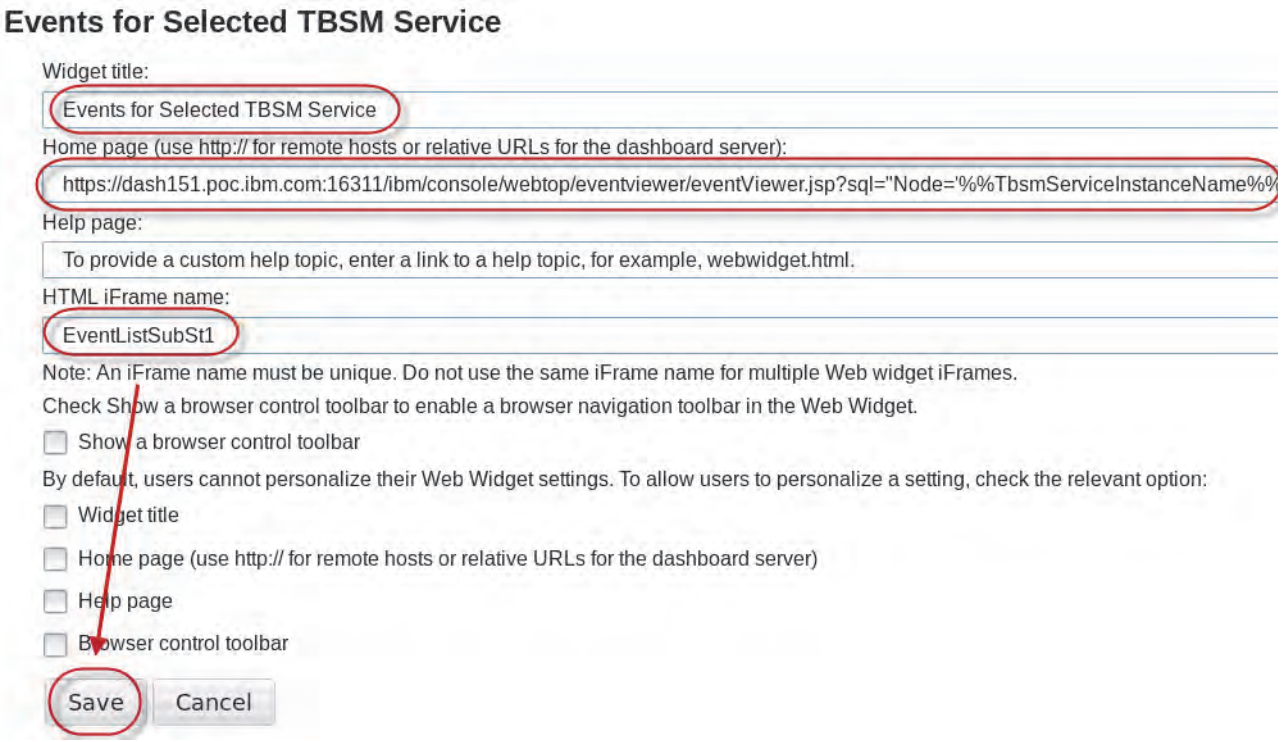
   b. Edit the web widget configuration properties. Click the **Edit options** icon in the upper right of the widget and select **Edit** or click the widget and select the **Widget > Edit** menu above the widget palette.

Configuring the web widget and transient filter

   c. Use the Tivoli Business Service Manager Node value that is generated with the NodeClickedOn event in the transient filter in the web widget home page address. Enter the following URL in the **Home Page** field:

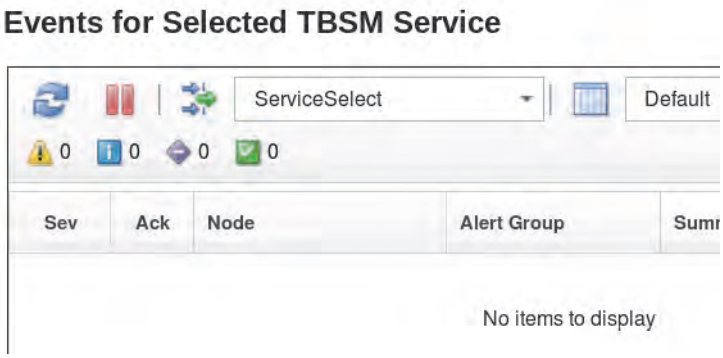
```
https://dash151.poc.ibm.com:16311/ibm/console/webtop/eventviewer/eventViewer.jsp?sql="Node='%%TbsmServiceInstanceName%%'"&transientname=ServiceSelect&viewname=Default&viewtype=global&datasource=OMNIBUS
```

d. Enter **EventListSubSt1** in the **HTML iFrame name** field and click **Save**.



The web widget initially shows an empty event viewer list. The empty viewer list is expected because initially there is no substitution value from a NodeClickedOn event. Initially, the SQL definition for the filter is **sql=”Node=””**, which returns an empty list.

Note: In the string **sql=”Node=””**, the characters immediately after **Node=** are two single quotation symbols, not a double quotation symbol.



3. Enable the web widget anchor points by clicking the web widget border.
4. Drag the widget anchor points to resize the widget to 20 grid blocks in height and 10 grid blocks in width.

5. Position the mouse cursor over the widget title bar until the cursor icon changes to a closed hand. Drag the web widget title bar to position the widget on the right half of the page canvas.

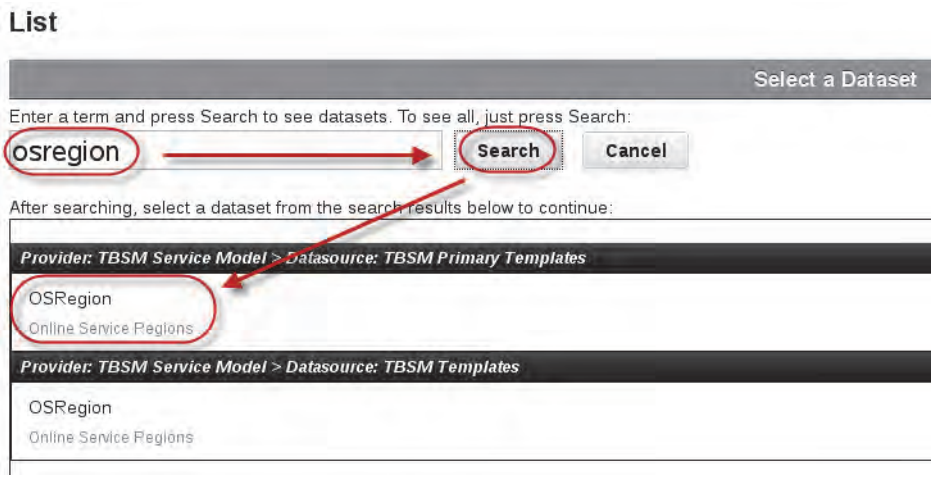
6. Click and drag a **table** widget instance from the widget palette to the page canvas.

7. Edit the table widget properties. Click the **Edit options** icon in the upper right of the widget and select **Edit**.

   You configure this widget to show a list of regional business services for the AnyBank business service model. The AnyBank service model is defined in the Tivoli Business Service Manager server.

8. Enter **osregion** in the data set search field and click **Search**.

9. Click **OSRegion** in the **TBSM Primary Templates** data source listing.



10. Configure the widget optional settings. Click the icon to the left of **Optional Settings**.

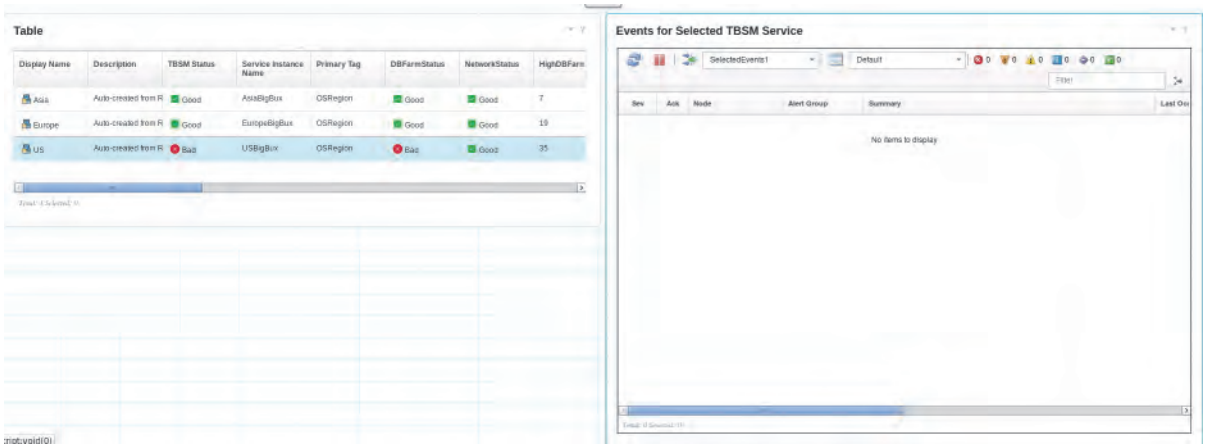
11. Enter **Events for Selected TBSM Service** in the **Widget Title** field.

12. Configure the TBSM Service value in the **Configure Optional Dataset Parameters** section. Enter **BigBux** in the **TBSM Service** field, select the **Gather Contained Services** check box, and click **OK**.

    The table widget shows the service information for three regional services in the BigBux service model.

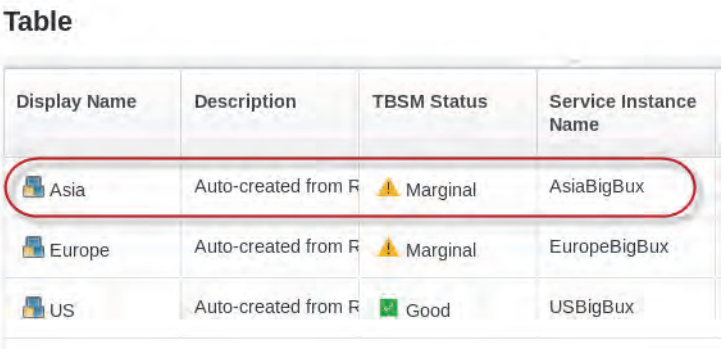
13. Resize the table widget so that it is 10 grid blocks in height and 10 grid blocks in width.

14. Move the table widget so that it is in the upper left corner of the page canvas.

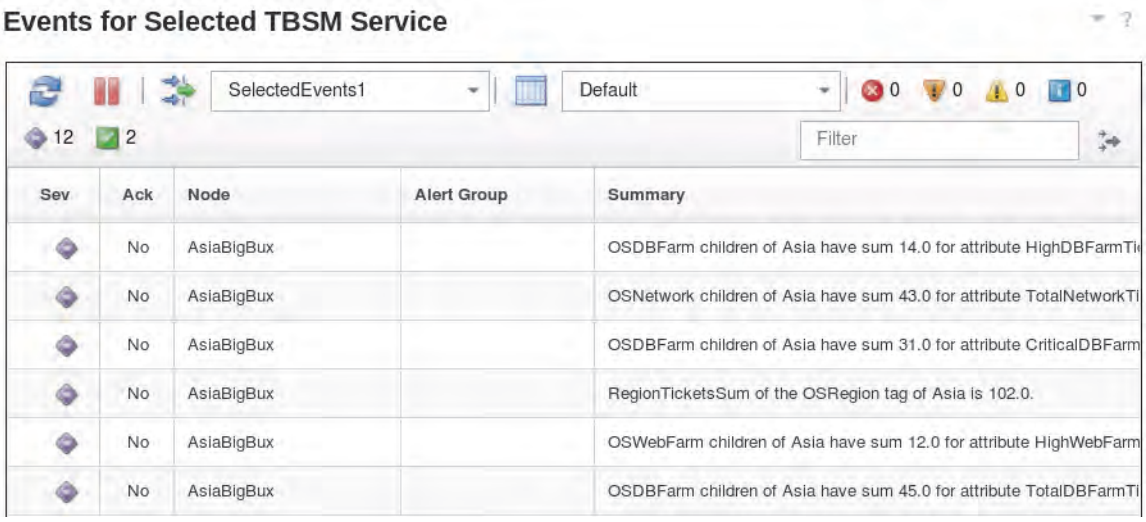


15. Save the page configuration. Click **Save and Exit** in the upper right of the console.

16. To test the web substitution and event viewer, click any cell in the **Asia** row in the table widget.



You see all events that are related to the AsiaBigBux service instance in the event viewer.



17. Click other entries in the table widget and verify that the filtered list of related events is shown in the event viewer.

18. Close the **OMNIbus Dynamic Filter Page** page when you finish examining the page behavior. Do not log off the console.

Unit 7 Integrating IBM Tivoli Monitoring and DASH exercises

This lab exercise demonstrates how to create IBM Tivoli Monitoring dashboards in the DASH server. The custom dashboard that you create uses the Tivoli Enterprise Portal UI data provider. The data provider retrieves IBM Tivoli Monitoring agent data and display the selected attributes in a dashboard page.

Exercise 1

In this exercise, you complete the following high-level tasks:

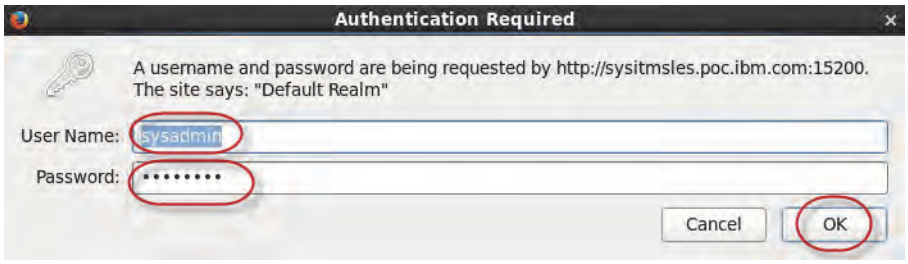
- Verify that the IBM Tivoli Monitoring UI data provider is available
- Create a top-level dashboard page to show key operating system metrics with a table, line graph, and bar graph widget
- Create a drill-down dashboard page to show context-driven metrics, based on an item selection in the top-level dashboard page
- Create an event wire to another dashboard page to drill down for more metric detail about a selected item

Checking for the IBM Tivoli Monitoring UI data provider Availability

Before you create the connection to the IBM Tivoli Monitoring UI data provider, you need to ensure that it is working. You connect to the test URL to confirm that the data provider is functioning.

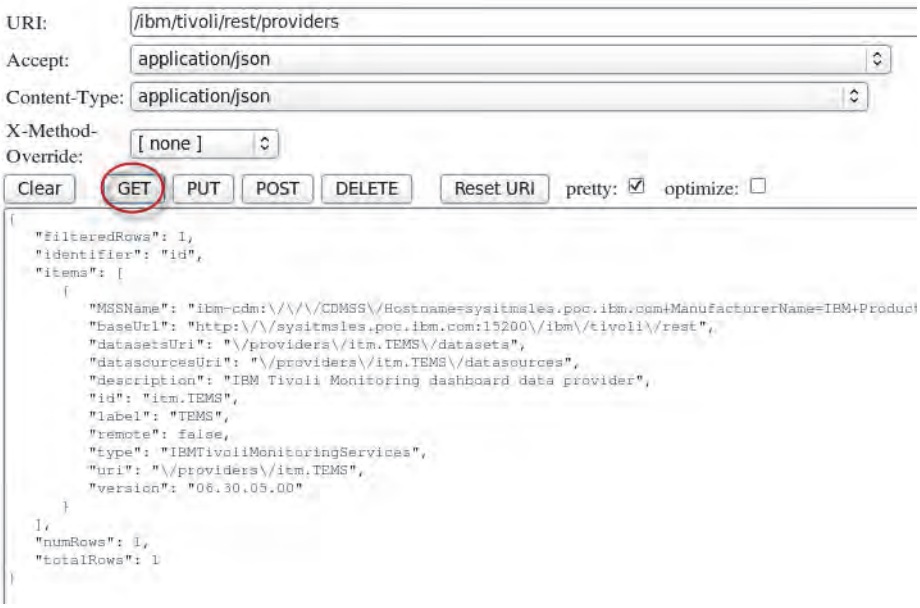
1. Switch to the **dash151** desktop.
2. Open a new page tab in the dash151 web browser and enter the following URL:
   `http://sysitmsles.poc.ibm.com:15210/ITMRESTProvider/test.html`

3. You are challenged to enter authentication credentials. Enter the user name **sysadmin** and the password **sysadmin** and click **OK**.



You see a web page that includes icons to GET, PUT, POST, and DELETE data to the interface.

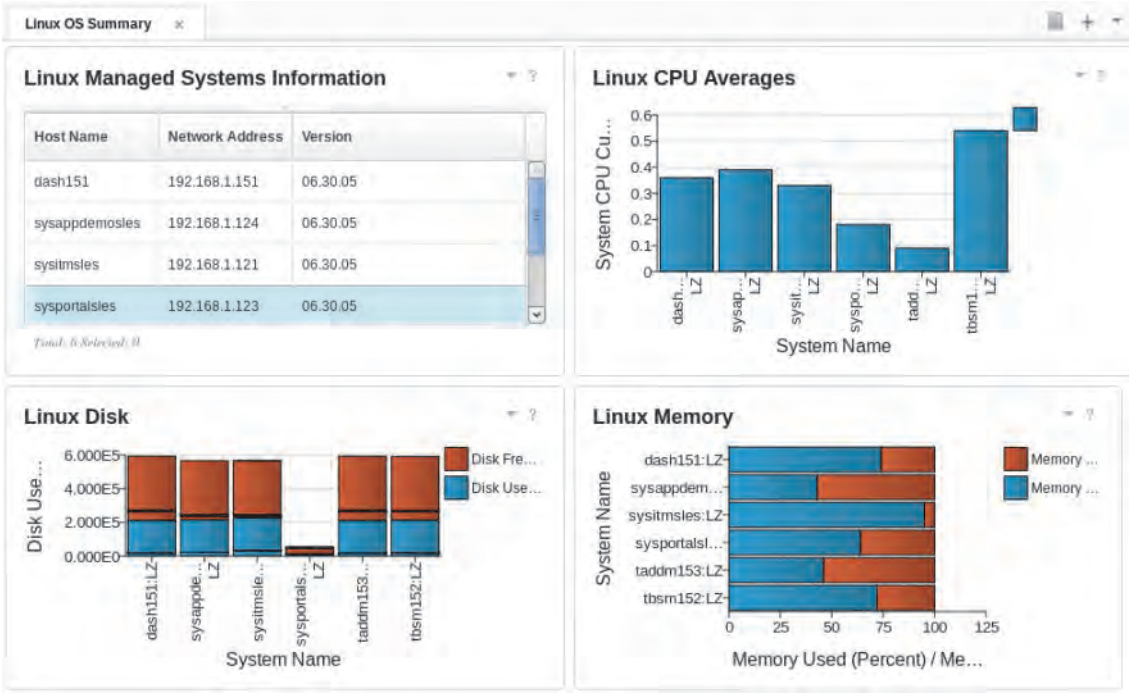
4. Click GET to verify that the interface is responding to data requests.



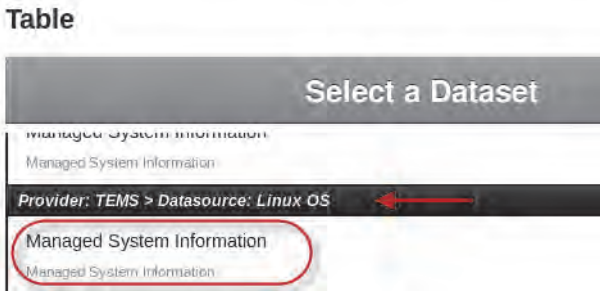
5. Close the browser tab.

Creating a high-level metric dashboard page

The UI data provider connection document to the IBM Monitoring server was created in Unit 5, Exercise 1 on page 5-1. Start with something simple, such as a Linux Systems Overview dashboard that shows a high-level view of the performance of all the Linux Systems in the laboratory environment. The final page contains a table widget, two column chart widgets, and a bar chart widget.



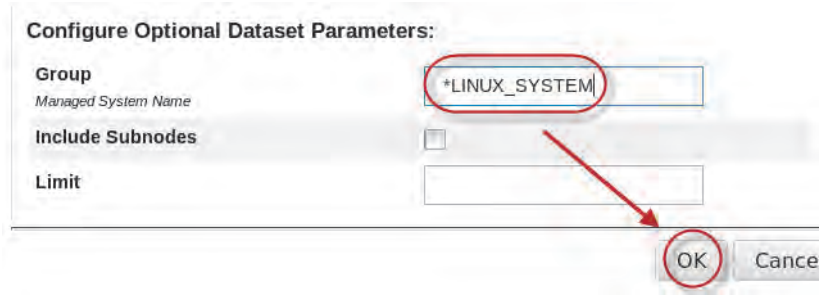
1. Open the DASH server console or log in to the DASH server console with the user ID **smadmin** and the password **object00**.
2. Create a page by clicking the plus (**+**) symbol in the upper right of the console.
3. Enter **Linux OS Summary** in the Page Name field, select **Proportional** as the page layout, and click **OK**.
4. First, create a table that shows the name of all the Linux managed systems. Select **Dashboard Widgets** and drag the **Table** widget onto the page canvas.
5. Edit the widget properties. The Linux managed systems (agents) are available in a data set called **Managed System Information**. Enter **Managed System Information** into the data set search field and select the entry below the **Linux OS** data source.



Note: You see a Managed System Information data set for every agent that is configured in the Tivoli Enterprise Monitoring Server database. Scroll down the list and select the entry below the **Datasource: Linux OS** entry.

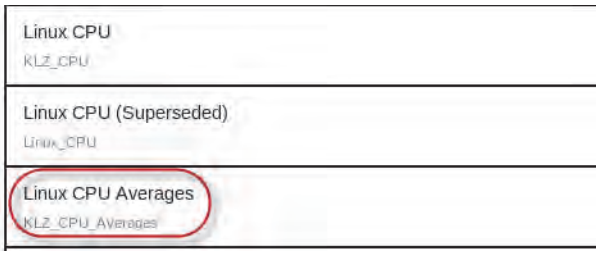
6. Edit the widget optional settings. By default, every available column in the data set is shown in the table widget. Filter the information and order the table to show only the **Host Name**, **Network Address**, and **Version** columns.

7. Retrieve data for all the known Linux managed systems. Scroll down the Optional Settings form, enter *LINUX_SYSTEM in the Group field, and click **OK**.

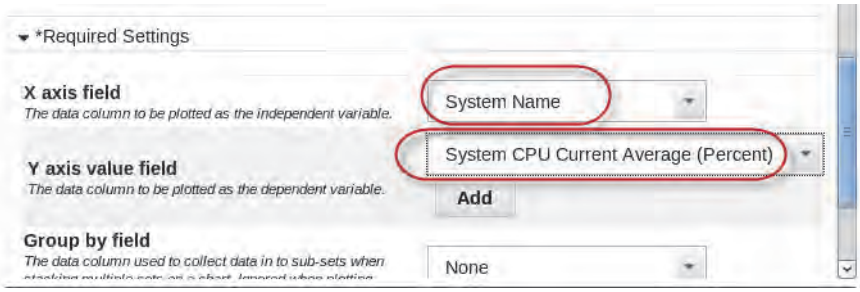


8. Create a CPU usage chart for the managed systems. Drag a **Column Chart** widget to the page canvas.

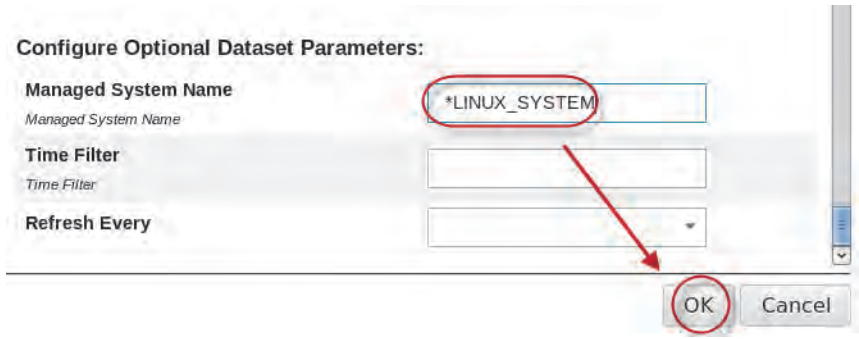
9. Edit the chart widget properties and enter **Linux** in the data set search field. Select the **Linux CPU Averages** data set.



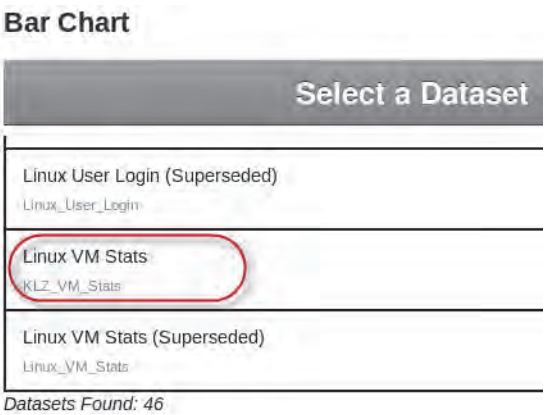
10. Select **System Name** for the **X axis field** value and **System CPU Current Average (Percent)** for the **Y axis value field** value.



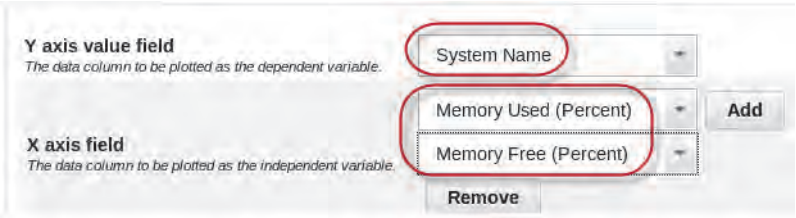
11. Select **Side** in the **Legend** field, ***LINUX_SYSTEM** in the **Managed System Name** field, and click **OK**.



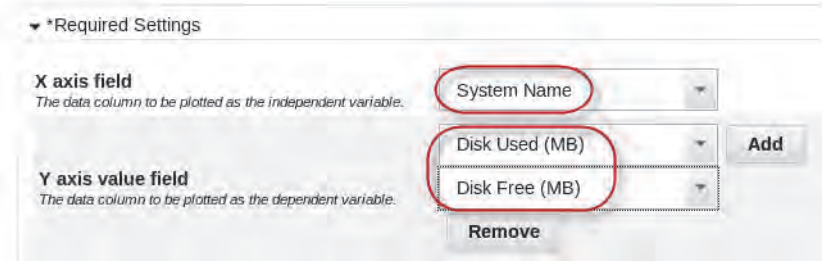
12. Click **Save** in the menu bar above the widget palette, but do not close the page editor.
13. Add a bar chart to the page. Drag a **Bar Chart** widget from the widget palette to the page canvas.
14. Edit the chart properties and enter for **Linux** in the data set search field.
15. Scroll down to the bottom of the list of data sets and select **Linux VM Stats**.



16. Select **System Name** for the Y-Axis and **Memory Used (Percent)** and **Memory Free (Percent)** for the X-Axis. Click **Add** to add a second Y-axis entry field.

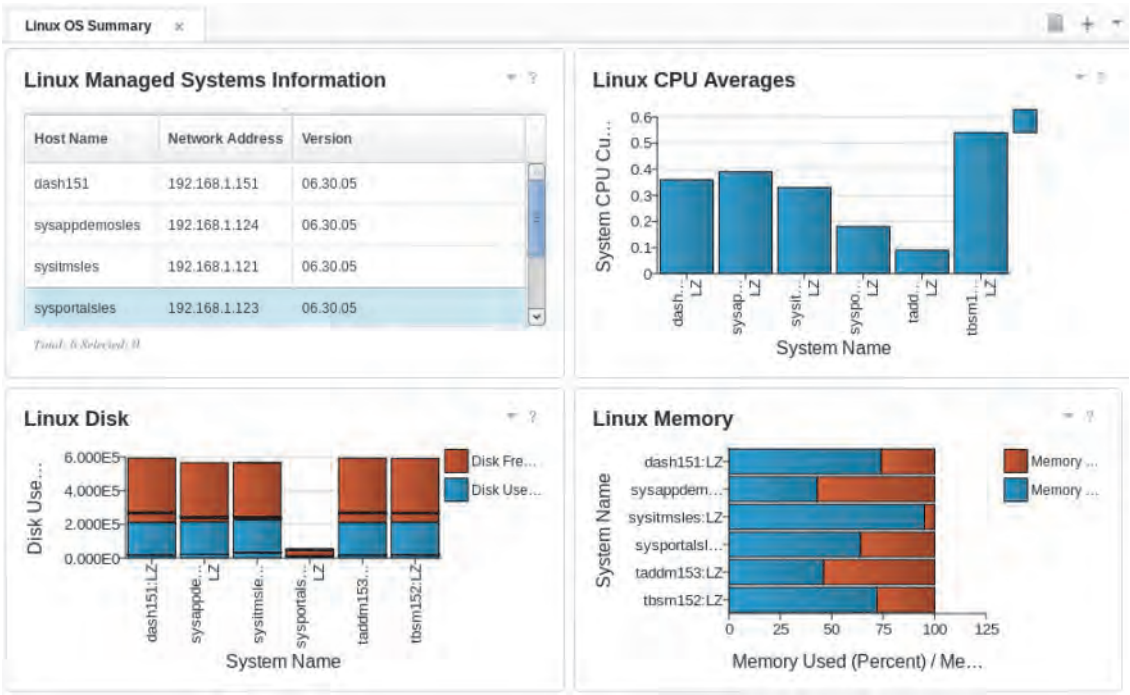


17. Edit the widget optional settings. Enter **Linux Memory** in the **Title** field, select **Side** in the **Legend** field and ***LINUX_SYSTEM** for the managed system name.
18. Click Save in the menu bar above the widget palette, but do not close the page editor.
19. Next, add a chart that summarizes disk information. Drag a column chart widget to the page canvas.
20. Edit the widget properties and enter **Linux** in the search properties field.
21. Select **Linux Disk** in the data set list.
22. Select **System Name** for the X-Axis and **Disk Used (MB) and Disk Free (MB)** for the Y-Axis. Click **Add** to add a second Y-axis entry field.



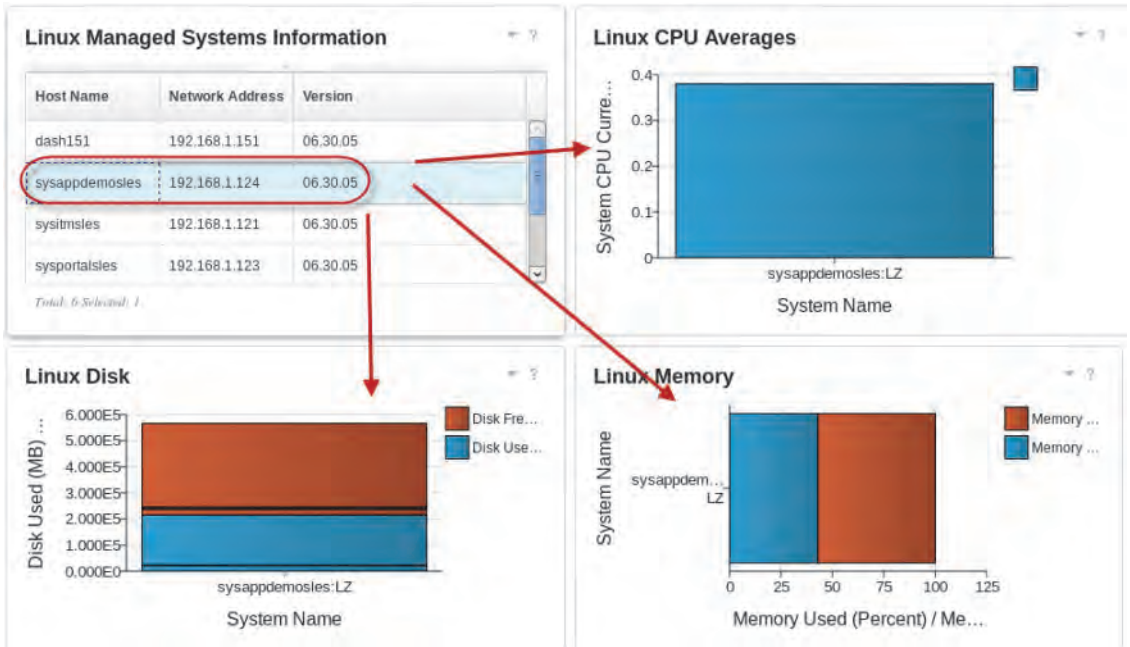
23. Edit the optional settings. Enter **Linux Disk** in the **Title** field, select **Side** in the **Legend** field, enter ***LINUX_SYSTEM** in the **Managed System** name field, and click **OK**.
24. Click **Save** in the menu bar above the widget palette, but do not close the page editor.

25. Resize the widgets so that each widget uses one quarter of the page canvas and arrange the widgets to match the following screen image.



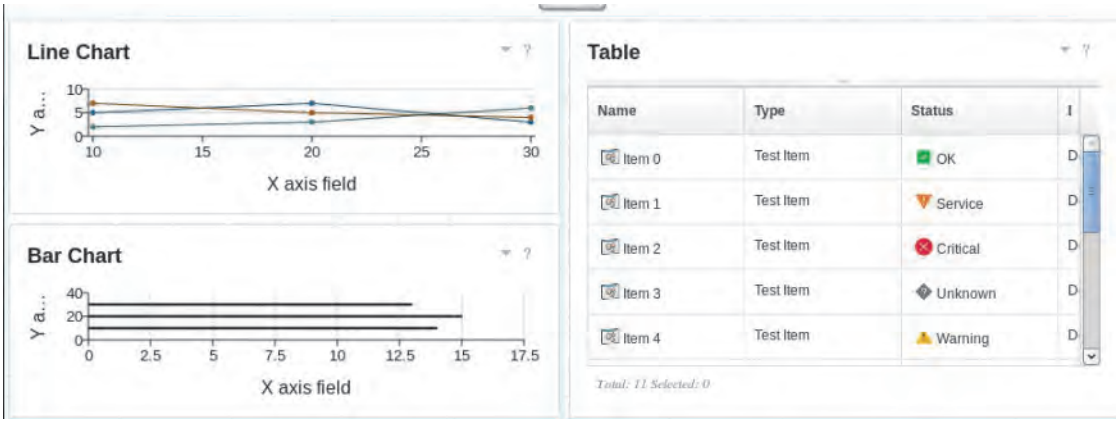
26. Save the page and close the page editor. Click **Save and Exit** above the widget palette.

27. Click any of the host name entries in the **Linux Managed Systems Information** widget to see the other three widgets automatically update and refresh their data with contextual data.

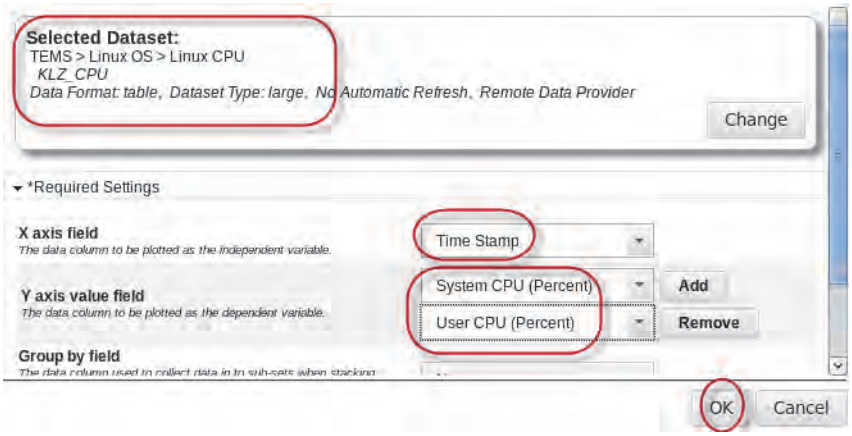


Creating a Drill-Down dashboard page

Now create a detail page that is used when an entry in the top-level page is selected. The page shows memory usage detail and contain three widgets: a line chart widget, a bar chart widget, and a table widget.

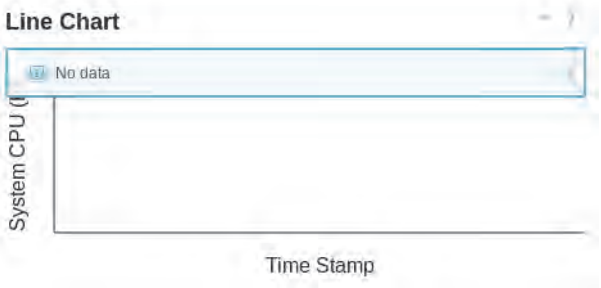


1. Click '**+**' to create a new page.
2. Enter **CPU Drill Down** in the **Name** field, use default values for all other selections, and click **OK**.
3. Drag a Line Chart widget to the upper left of the page canvas.
4. Edit the widget properties. Enter **Linux CPU** in the search field and select the **Linux CPU** data set.
5. Select **Time Stamp** as the X-Axis.
6. Select **System CPU (Percent)** and **User CPU (Percent)** for the Y-Axis.

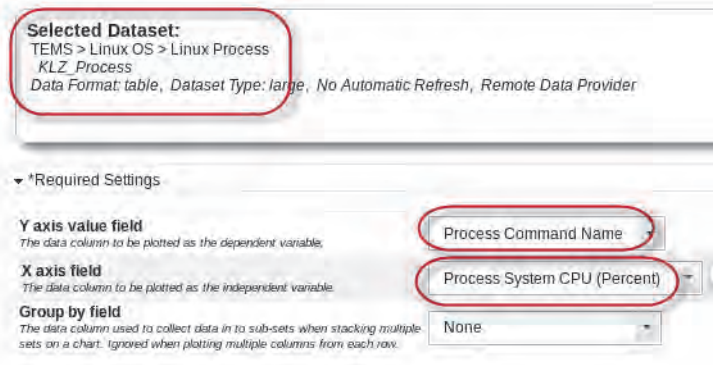


7. Edit the optional settings. Select **Side** in the **Legend** field, leave the Managed System Name field blank, and click **OK**.

Note: The target widget uses the Managed System Name value when a table entry is selected in the top-level page. When the widget properties are saved, the widget shows a message that indicates that no data available. This message is normal.

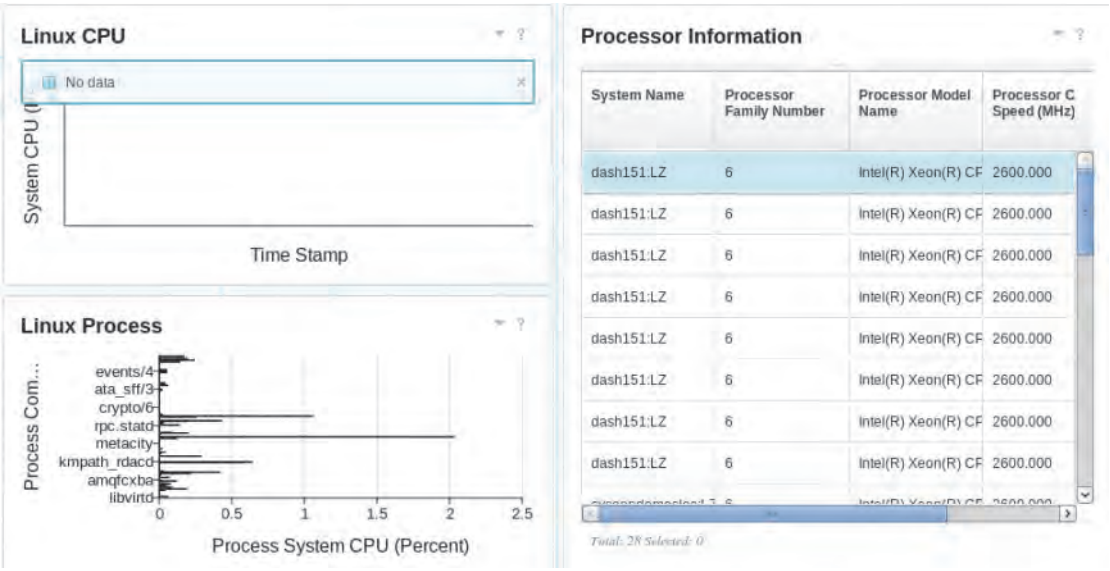


8. Click **Save** in the menu bar above the widget palette, but do not close the page editor.
9. Drag a bar chart widget to the lower left of the page canvas.
10. Enter **Linux Process** in the search field and select the **Linux Process** data set.
11. Select **Process Command Name** for the Y-Axis and select **Process System CPU (Percent)** for the X-Axis.



12. Edit the optional settings. Enter **Linux Process** in the **Title** field, enter *LINUX_SYSTEM in the Managed System Name field, and click **OK**.
13. Drag a **Table Widget** to the upper right of the page canvas.
14. Edit the widget properties. Enter **Linux CPU Config** in the search field and select the **Linux CPU Config** data set.
15. Select the columns **System Name, Processor Family Number, Processor Model Name, Processor Clock Speed (MHz), and Processor Cache Size (KB)**. Remove all other column names from the selected column.

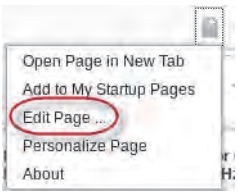
16. Enter **Processor Information** in the **Title** field, enter *LINUX_SYSTEM in the Managed System Name field, and click **OK**.
17. Click **Save** in the menu bar above the widget palette, but do not close the page editor.
18. Modify the size and arrangement of the widgets to match the following screen image:



19. Save the page. Click **Save and Exit** above the widget palette.

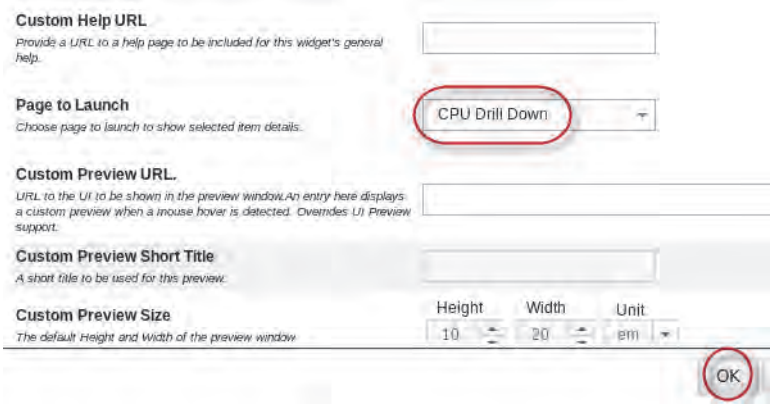
Hint: You use one of two methods to link to a dashboard page with context information. The first method is to set the **Page to Launch** option in the top-level table widget configuration. The second method is to create an event wire from the top-level chart widget to the detail page.

20. Modify the top-level table widget configuration. Switch to the **Linux OS Summary** page.
21. Open the page editor. Select **Edit Page** in the **Page Actions** icon in the upper right of the console.



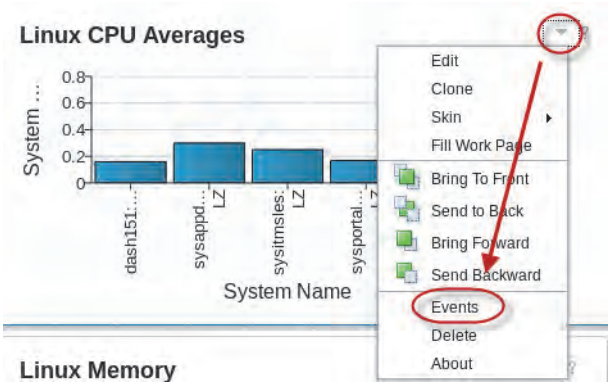
22. Edit the table widget properties.

23. Scroll down in the optional settings section, select **CPU Drill Down** in the **Page to Launch** menu, and click **OK**.



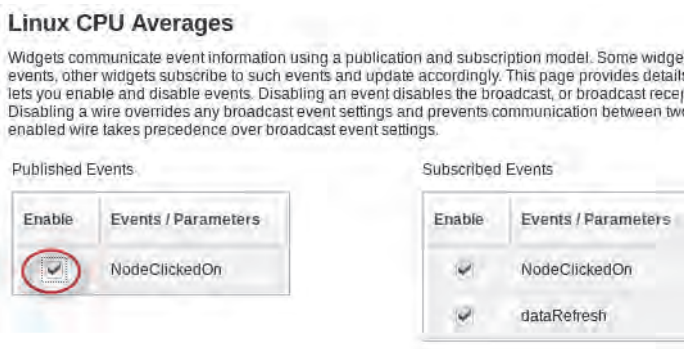
24. Save the page configuration but do not close the page editor. Click **Save** above the widget palette.

25. Create an event wire from the Linux CPU Averages chart widget to the CPU Drill Down page. By default, a chart widget does not publish NodeClickedOn events. Change the default behavior. Click the **Edit options** icon in the upper right of the **Linux CPU Averages** widget and select **Events**.



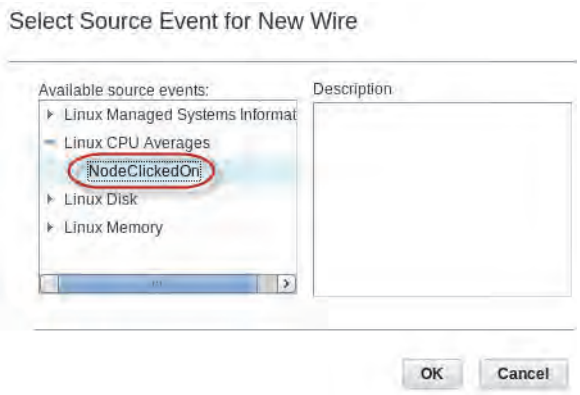
You see the events to which the widget is subscribed and the events that are published by the widget. To send context data to pages or widgets, you must enable **NodeClickedOn** in the **Published Events** section.

26. Select **NodeClickedOn** in the **Published Events** section and click **Close**.

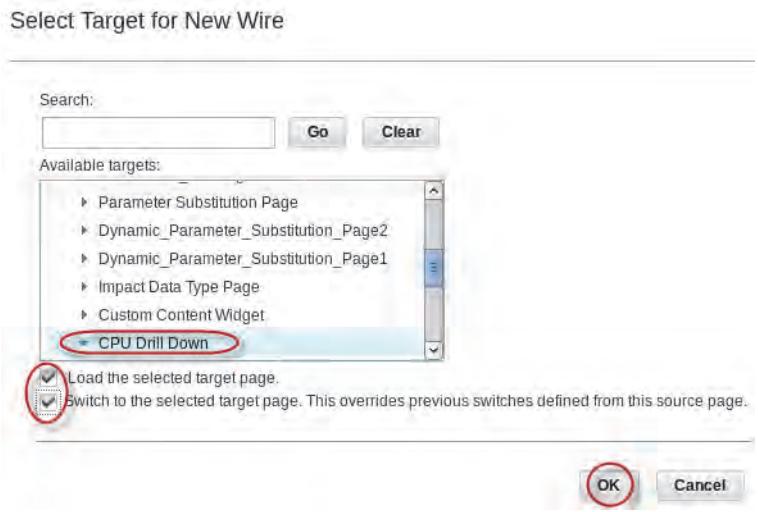


27. Now that the published events are enabled, create an event wire to another page. Click the **Show Wires** icon above the widget palette.

28. Select the source widget for the wire. Click **Linux CPU Averages**, click **NodeClickedOn**, and click **OK**.



29. Select the target for the new wire. You want the context event to be sent to all the widgets on the CPU Drill Down page. Select **CPU Drill Down**, select **Load the selected target page,** select **Switch to the selected target page**, and click **OK**.



30. Select **None** for transformation and click **OK**.

The wire is shown in the wires table above the page canvas.

31. Close the wire table by clicking the small x at the upper right of the console. Do not click the X that closes the browser.

32. Save the updated page configuration. Click **Save and Exit** above the widget palette.

33. Click a column in the **Linux CPU Averages** chart widget and it opens the **CPU Drill Down** page with the context information.

34. Switch back to the Linux OS Summary page and click an entry in the **Linux Managed Systems Information** table widget and it also opens the **CPU Drill Down** page.

Unit 8 Integrating with Tivoli Business Service Manager exercises

In these exercises, you use the DASH server tools to show Tivoli Business Service Manager business service data in two connected business dashboard pages. You also learn how to add and use widget context tasks with DASH dashboard widgets.

You complete Exercises 1 - 5 to create and test a high-level dashboard for a line-of-business owner that visualizes the status of the dayTrader application. The dayTrader service model represents data from a trading application that is hosted at five data centers across the United States: New York, Chicago, Dallas, Seattle, and Los Angeles. You build a multi-page dashboard that gives the line-of-business owner a quick view of the overall revenue status and detailed revenue key performance indicators (KPIs) for each data center.

dayTrader service model review

The dayTrader service model tracks several KPIs that indicate the status of the stock trading service. The five templates and their functions are as follows:

- **AEDayTrader_AppPerf**: This template tracks the availability of the Buy, Sell, Quote, Homepage, Login, and Portfolio subcomponents of the dayTrader web application. Availability is defined as the percentage of successful transactions versus attempted transactions for the previous one-hour window. The availability of these services is collectively evaluated to determine the overall status of the application.
- **AEDayTrader_ActiveUsersAndQuotes**: This template shows the number of currently active stock quotes. The KPIs are tracked for a selected list of high-volume users (VIPs) and for all users.
- **AEDayTrader_BuyAndSell**: This template tracks the current number of buy and sell orders for VIPs and all users.
- **AEDayTrader_Revenue**: This template tracks the total revenue volume change and lost trade revenue for VIPs and all users.
- **AEDayTrader_TradePlatformPerf**: This template tracks the latency of trades, in seconds, on the buy and sell queues for VIPs and all users.

Each template calculates one or more KPIs, independent of the other templates. No dependency rules are used in any template.

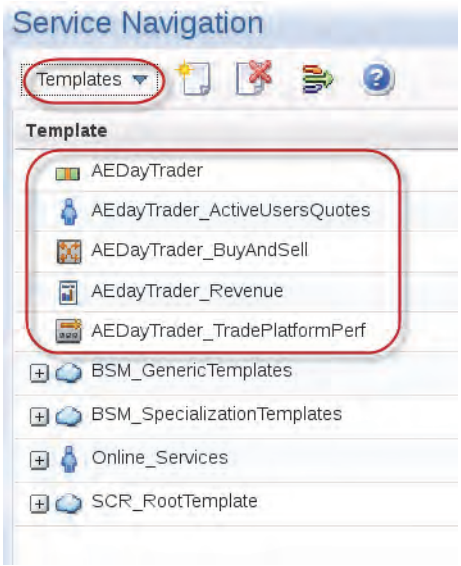
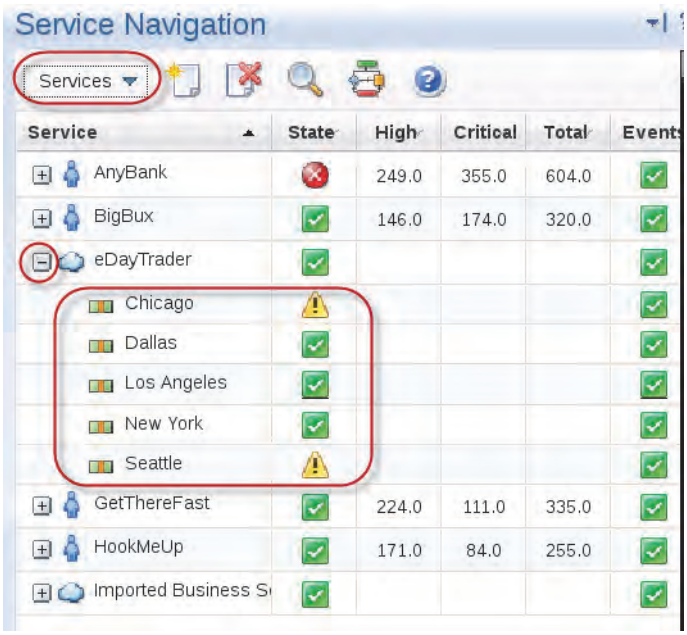


Figure 1 dayTrader templates

The following screen image shows the dayTrader services in the service navigation portlet. The eDayTrader service is not tagged with a template and is used to group the KPIs for the five data centers as dependent services. The default service tree does not show the KPIs that are tracked in the service templates.



During the laboratory configuration, five tree templates were configured that show the dayTrader KPI metrics. The following screen image shows the template KPIs in five service tree portlets. Each portlet is configured to use a corresponding tree template. Tivoli Business Service Manager 6.1.1 fix pack1 added a tree template data set that is available through the UI data provider. All service

tree templates that are defined in the data server are available for use with DASH server data widgets.

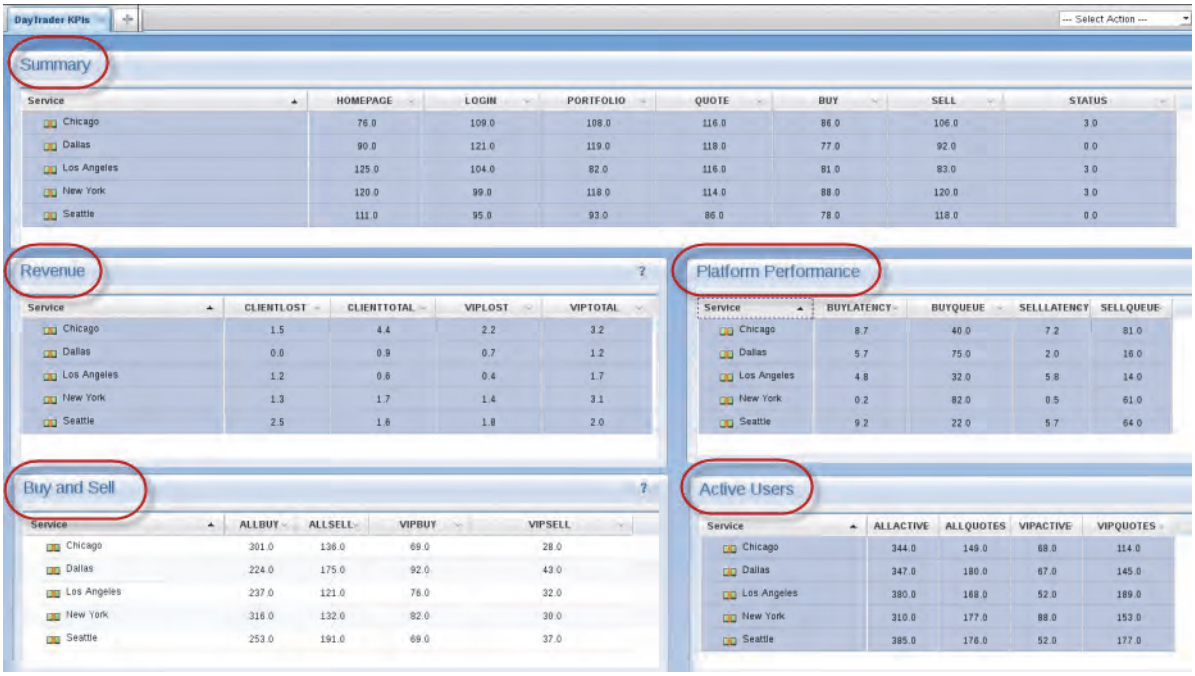


Figure 2 dayTrader KPIs filtered by tree template

The exercises in this unit are summarized in the following list:

- Exercise 1, “Creating the dayTrader logo custom image widget”: You create a custom image widget that shows the dayTrader application logo. The custom widget is added to the dashboard pages that you create in exercises 2 and 3.
- Exercise 2, “Creating the dayTrader Revenue Detail page”: You create a dashboard page that shows detailed dayTrader revenue KPIs for a selected data center. This page corresponds to the second-level dashboard page that is shown in the Figure 3 logical diagram:

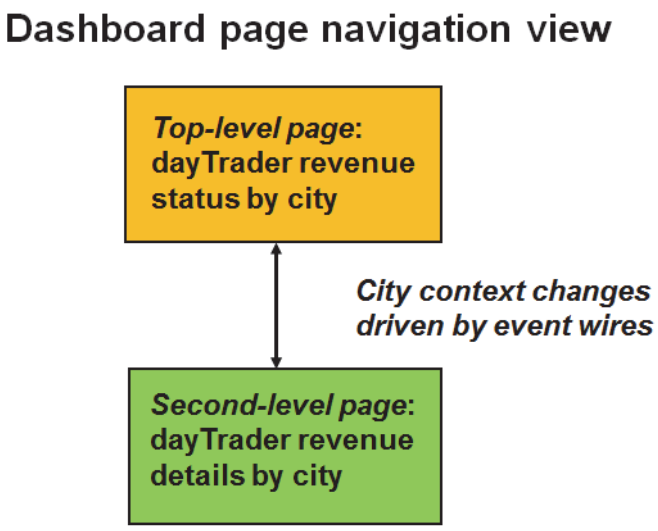


Figure 3 dayTrader dashboard page navigation view

  The second-level page is configured to show data with the context information that is sent with a widget selection in the top-level page.
- Exercise 3, “Creating the dayTrader Revenue Status page”: You configure the top-level page that is shown in the Figure 3 logical diagram. This page uses two status gauge widgets to show the high-level status of the dayTrader revenue KPIs for each data center. You click a status gauge to see the corresponding revenue KPI details in the second-level page.
- Exercise 4, “Linking pages with wires”: You configure event wires for each status gauge in the top-level page to send contextual event data to the second-level page.
- Exercise 5, “Testing the dashboard design”: You add the pages to a view and test the dashboard page function and navigational configuration.
- Exercise c, “Click the status gauge for the other data centers and verify that the dayTrader Revenue Detail page shows the correct contextual data.”: You configure task menus for business services that are shown in DASH widgets on a dashboard page.

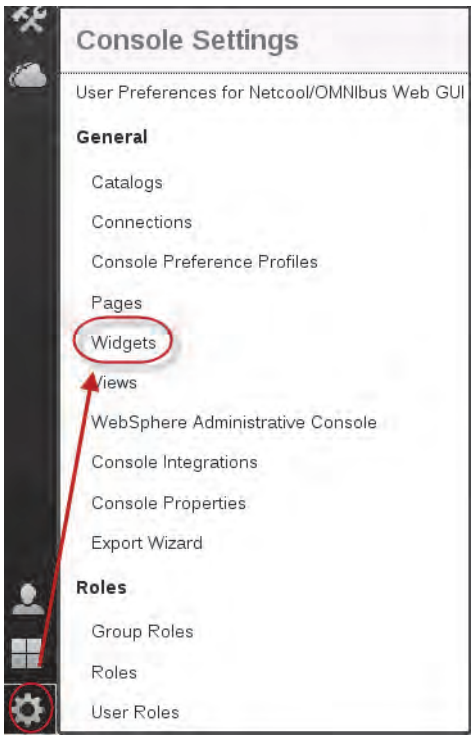
Exercise 1 Creating the dayTrader logo custom image widget

In this exercise, you create and configure a custom image widget with a locally served image file. The web application server is provided with the DASH **myBox** web application. You add the custom widget to dashboard pages that you create in this exercise.

Note: The myBox web application was added in the Jazz for Service Management 1.1 fix pack 3 installation. The image file that is used in this task was added to the myBox web application during the virtual image configuration.

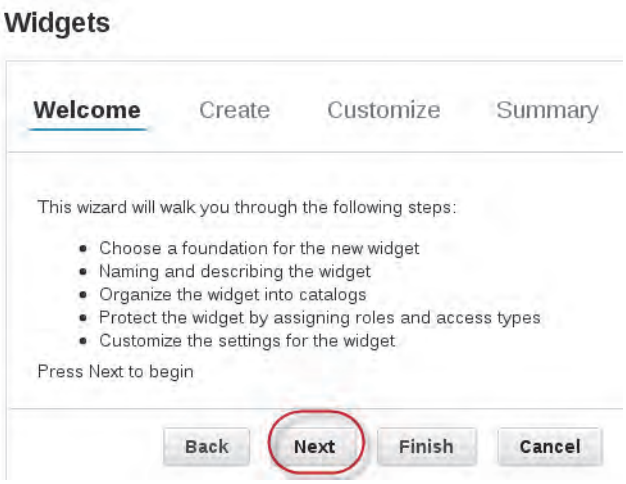
Log in to the DASH console with the user ID **smadmin** and the password **object00** and complete the following steps to create the custom widget:

1. Open the widget management page by selecting the **Console Settings > General > Widgets** task.

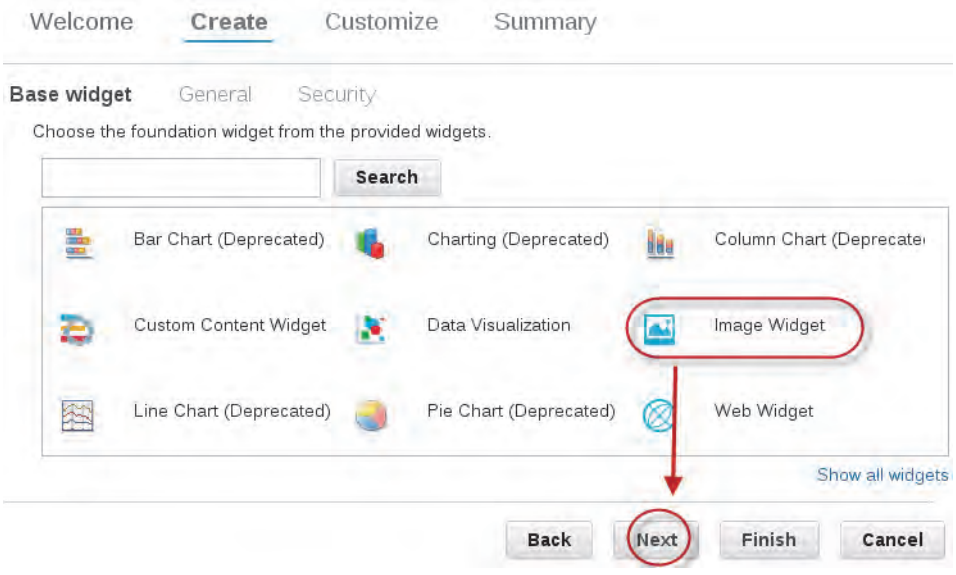


2. Click the **New** icon to start the widget creation process.

3. At the Welcome page, click **Next**.



4. Select **Image Widget** in the **Base widget** section and click **Next**.

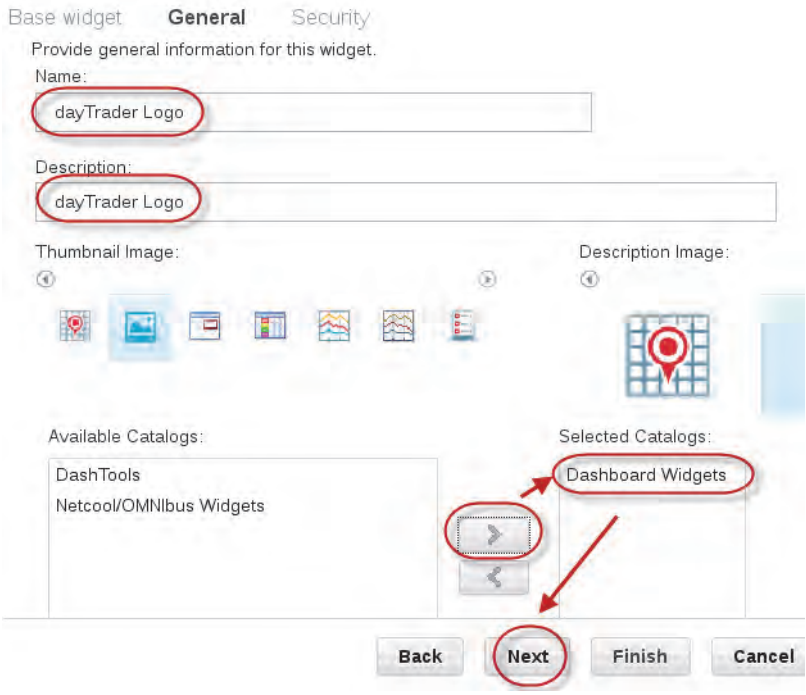


5. Enter **dayTrader Logo** in the **Name** field.

6. Enter **dayTrader Logo** in the **Description** field.

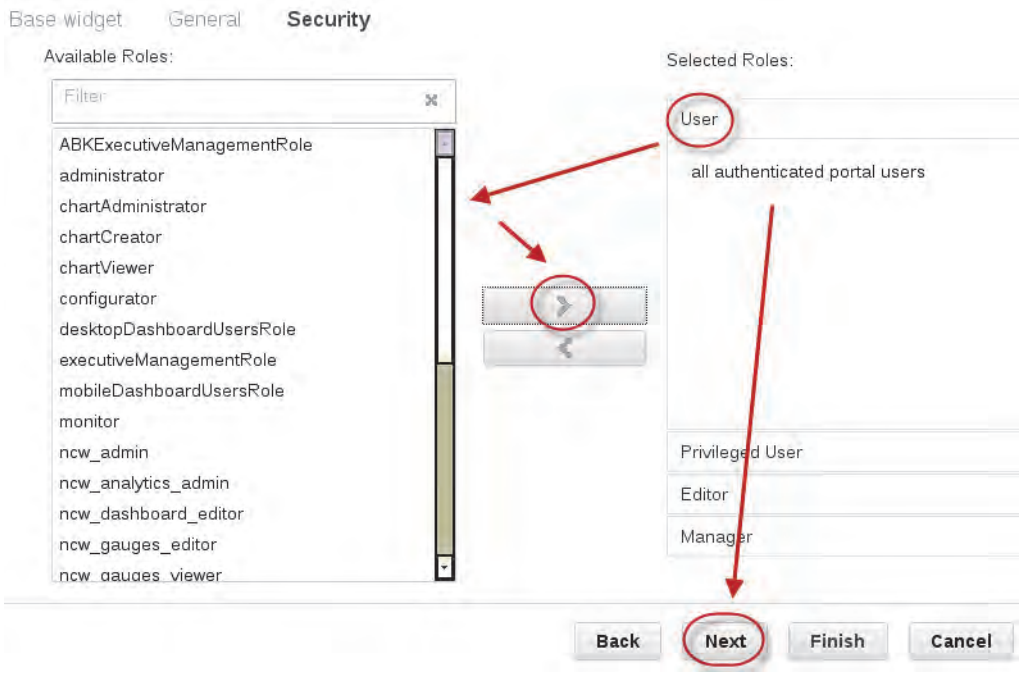
7. Leave the default values for the **Description**, **Thumbnail Image**, and **Description Image** selections.

8. Assign the widget to a widget *catalog*. A catalog contains groupings of widgets. Click **Dashboard Widgets** in the **Available Catalogs** list and click the right arrow to move the selection to the **Selected Catalogs** list and click **Next**.



9. Assign an authorization role to the widget so that all authenticated users of the DASH console can see the widget.
   a. Click **User** in the **Selected Roles** section.
   b. Click **all authenticated portal users** in the **Available Roles** list and click the right arrow to move the role into the **Selected Roles** list.

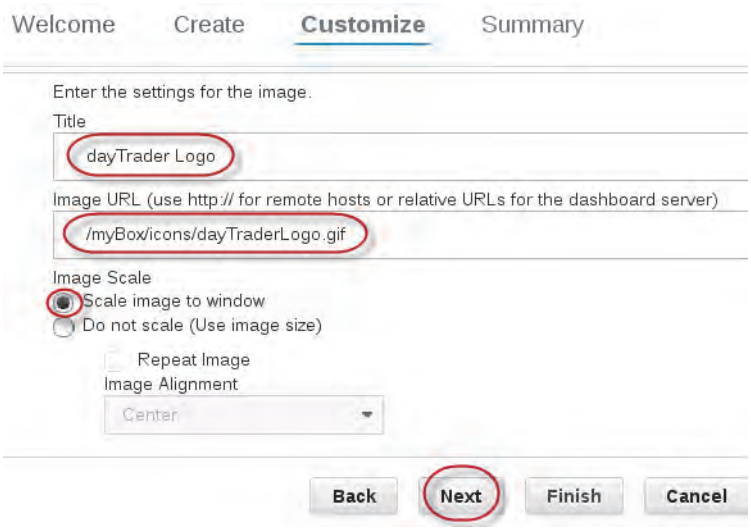
10. Click **Next**.



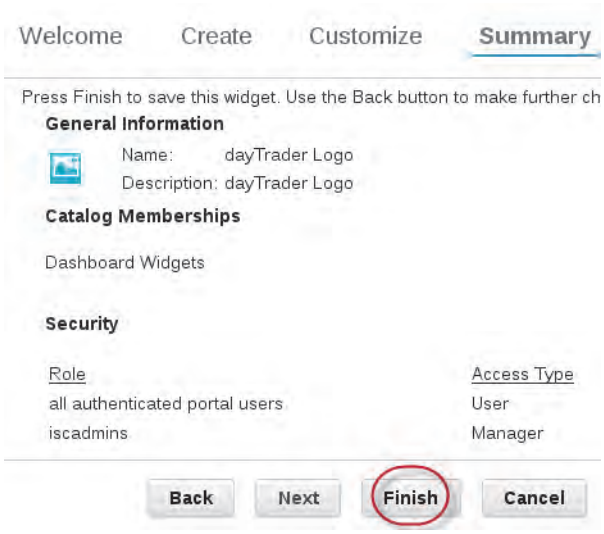
11. Configure the custom widget to show the dayTrader logo image. Enter **/myBox/icons/dayTraderLogo.png** in the **Image URL** field.

Note: The directory, **/myBox**, is the logical *context root* of the web application, relative to the DASH console server. The context root was configured when the web application was installed. Because the files are local to the server, you do not have to use HTTP or HTTPS in the web address. The physical directory that contains the web application file is **/opt/IBM/JazzSM/ui/myBox/web_files/icons**.

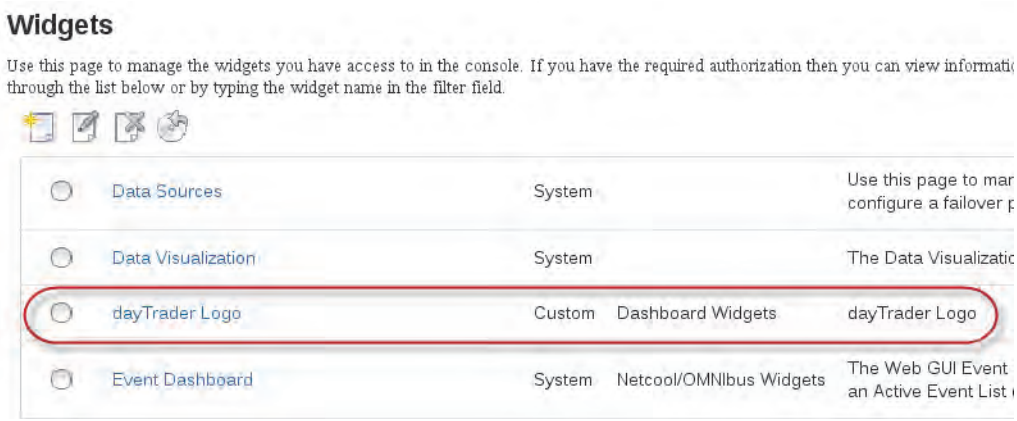
12. This widget is used in mobile and desktop dashboard pages. To maintain the image proportions for both client types, select **Scale image to window** and click **Next**.



13. Review the Summary page information and click **Finish**.



You see the widget in the **Widgets** list.



14. Click the **X** symbol in the Widgets tab to close the Widgets page.

Exercise 2 Creating the dayTrader Revenue Detail page

For this exercise, you create a dashboard page that shows detailed dayTrader revenue data for a selected city. This page is shown only when a status gauge widget is clicked in the top-level **dayTrader Revenue Status** page. You send context information to this page with event wires that you configure in Unit 8, Exercise 4 on page 8-38. The page in this exercise consists of a custom image widget, a web widget, four analog gauge widgets, and two volume bar widgets. Figure 4, “dayTrader Revenue Detail page logical layout diagram,” on page 8-11 shows the logical and general layout for the page.

Note: Because the event wires are created in the top-level page, you create the second-level page first. Creating the event wire target page first reduces the number of steps that are required to complete the pages.

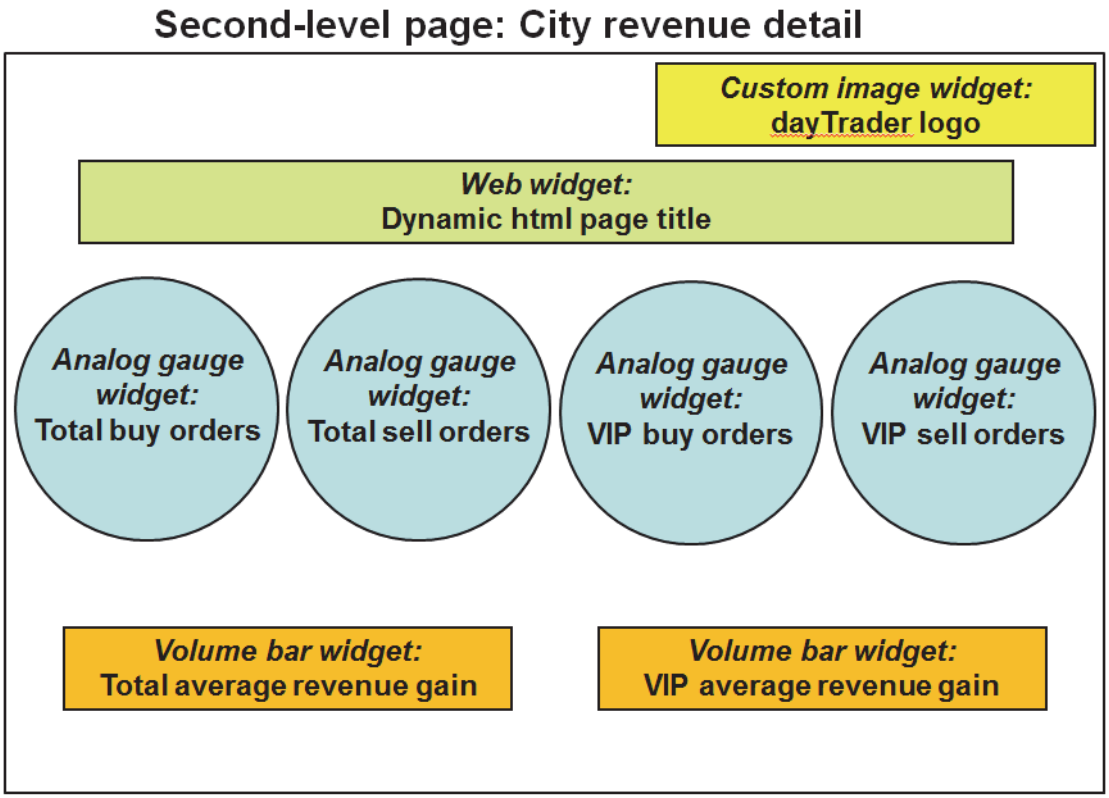


Figure 4 dayTrader Revenue Detail page logical layout diagram

You must complete the following high-level tasks to create the page:

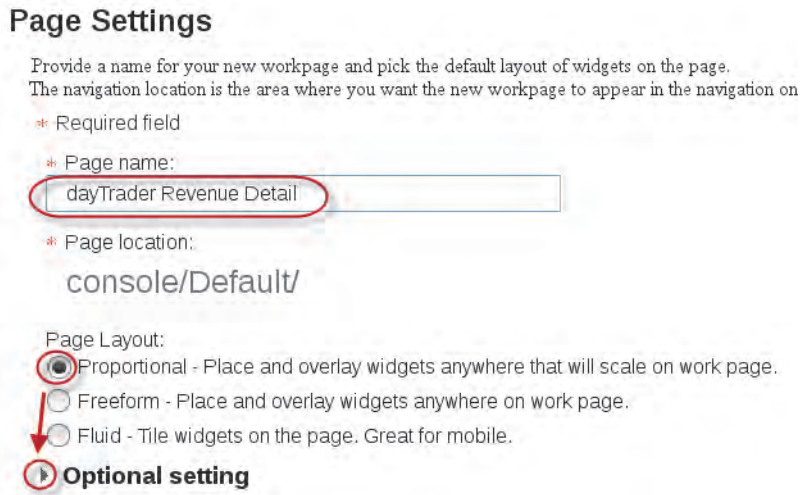
- Create and configure the page properties
- Add and configure the page widgets

Creating and configuring the page properties

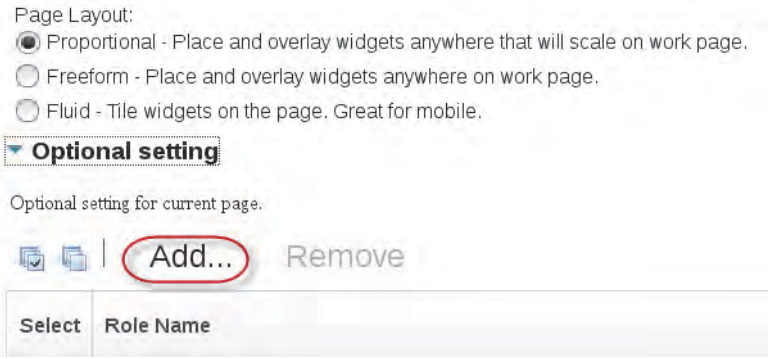
1. Create a page. Click the plus symbol (+) in the upper right of the console.

2. Enter **dayTrader Revenue Detail** in the **Page name** field.
3. Leave the **Page location** value at **console/Default**.

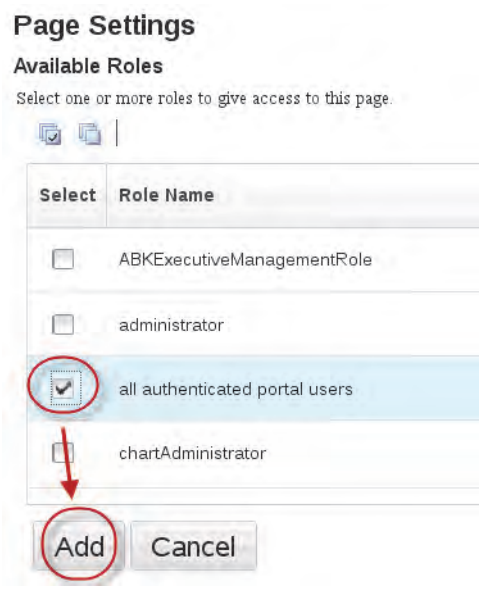
4. Select **Proportional** for the **Page Layout** type.
5. Configure authorization roles for the page, by clicking to expand the **Optional setting** section.



6. Click **Add**.

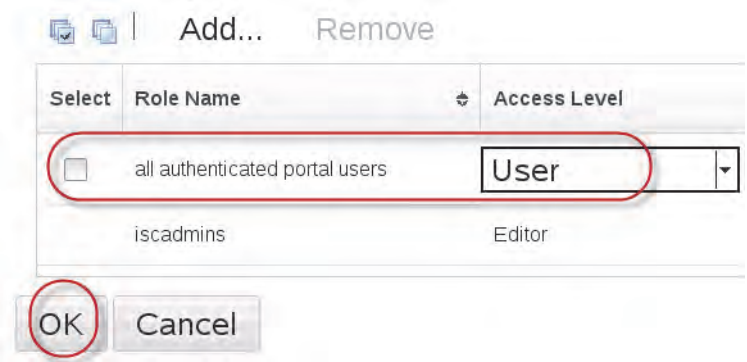


7. Select **all authenticated portal users** in the list of roles and click **Add**.



The role selection is shown the role assignment list.

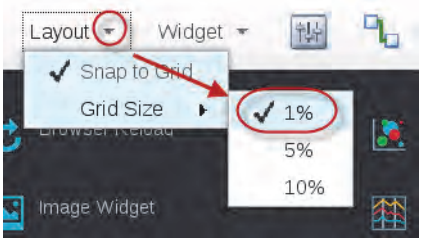
8. Leave the default access level and click **OK**.



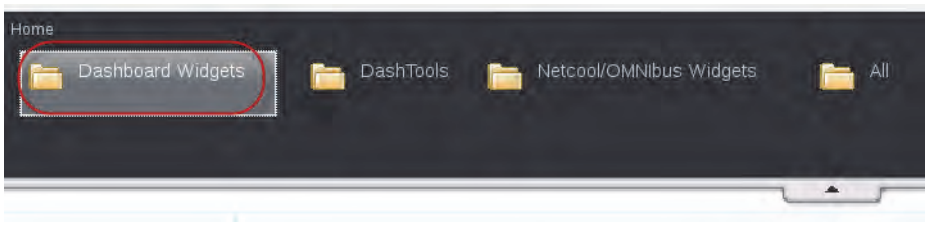
The page properties are replaced with the page editor.

Adding and configuring the page widgets

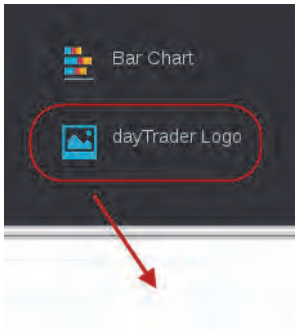
1. Set the object tracking granularity to support finer control of the widget placement in the page canvas. Click the **Layout > Grid Size > 1%** menu above the widget palette section.



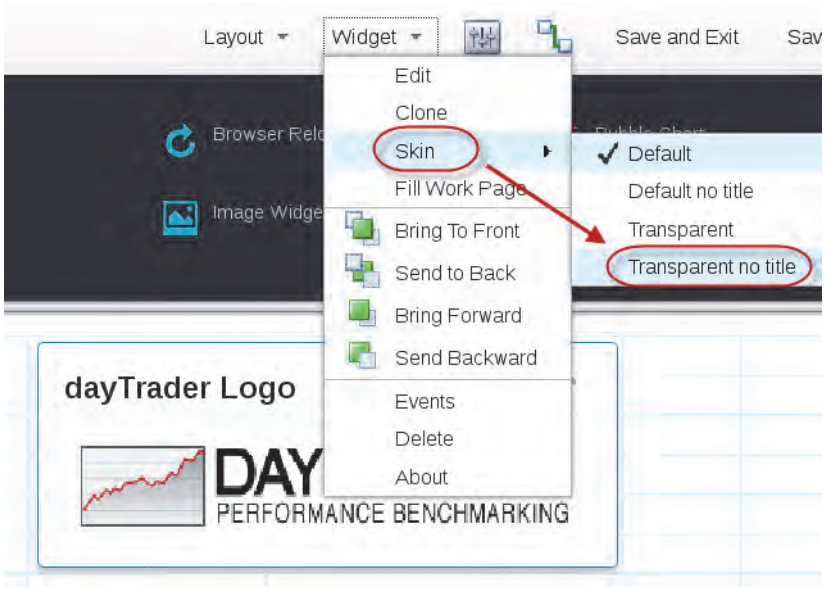
2. Click **Dashboard Widgets** in the widget palette section.



3. Add and configure the dayTrader Logo widget in the canvas. You created the dayTrader Logo widget in Unit 8, Exercise 1 on page 8-5.
   a. Click and drag **dayTrader Logo** from the widget palette to the page canvas.



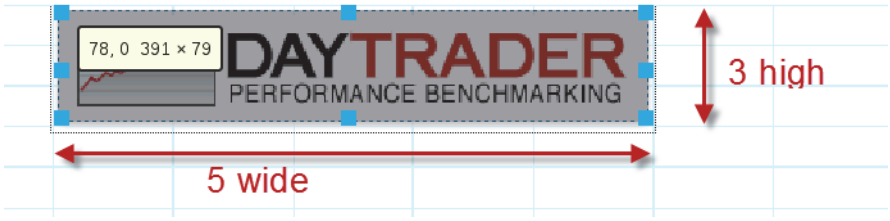
   b. Remove the widget frame and title. Click the widget to set the focus and select the **Widget > Skin > Transparent no title** menu above the widget palette.



   c. Set the widget size. Move the cursor to an edge of the widget until the cursor shape changes to a small arrow that points to a line, and click the mouse.

      The widget frame is shown with three anchor points on each side of the object.

d. Click and drag the frame anchor points to set the widget size to three canvas blocks high and five canvas blocks wide.



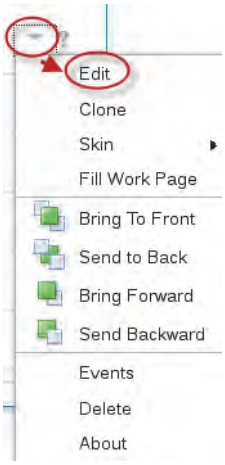
e. Move the mouse cursor below the top edge of the widget until the cursor shape changes to a closed hand image, then click and drag the widget to the upper right of the page canvas.

4. Add a web widget to the page canvas. You configure this widget to use a jsp page to generate a page banner. The page script uses context data to change the banner content.

Note: A copy of the jsp file that is used with this widget is listed in Appendix A on page A-1.

a. Click and drag **Web Widget** from the widget palette to the page canvas.

b. Edit the widget properties. Click the **Edit options** icon in the upper right of the widget or click the widget and select the **Widget > Edit** menu above the widget palette.



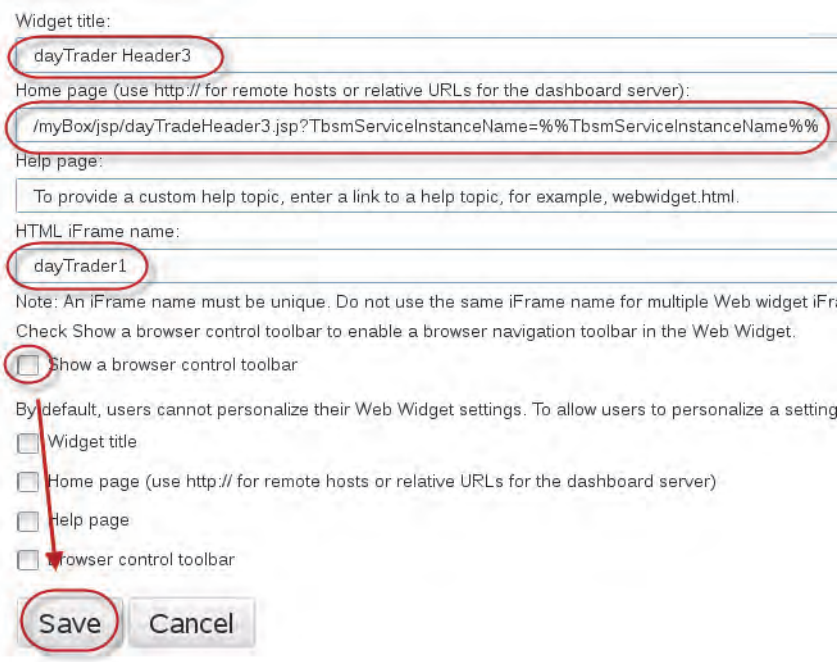
c. Enter **dayTrader Header3** in the **Widget title** field.

d. Enter **/myBox/jsp/dayTradeHeader3.jsp?TbsmServiceInstanceName=%%TbsmServiceInstanceName%%** in the **Home page** field.

Important: You must use the letter case as written.

e. Enter **dayTrader1** in the **HTML iFrame name** field.

f. Remove the **Show a browser control toolbar** selection and click **Save**.



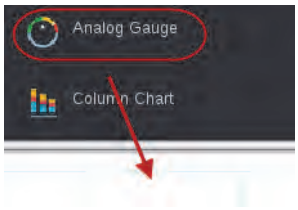
g. Remove the widget frame and title. Click the widget to set the focus and select the **Widget > Skin > Transparent no title** menu above the widget palette.

h. Resize the widget to three canvas grid blocks high and the full width of the page canvas.

i. Position the widget so that it is centered lower than, but not overlapping, the dayTrader Logo widget.

5. Save your work but do not close the page editor. Click **Save** above the widget palette section.

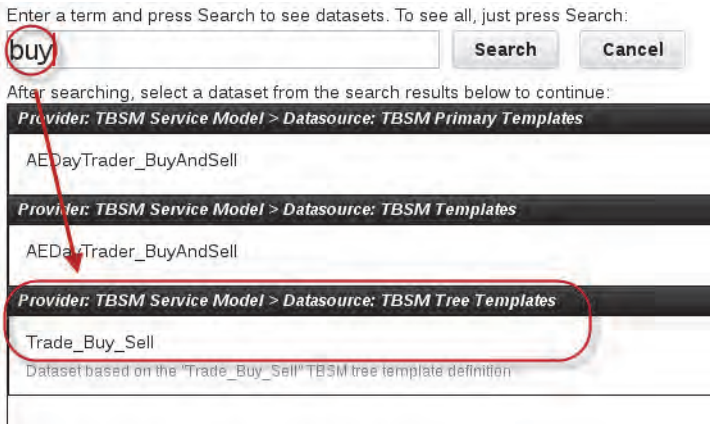
6. Add an analog gauge widget to the page. This widget is configured to show the current number of pending buy orders for all customers in the selected city.
   a. Click and drag the **Analog Gauge** object from the widget palette to the page canvas.



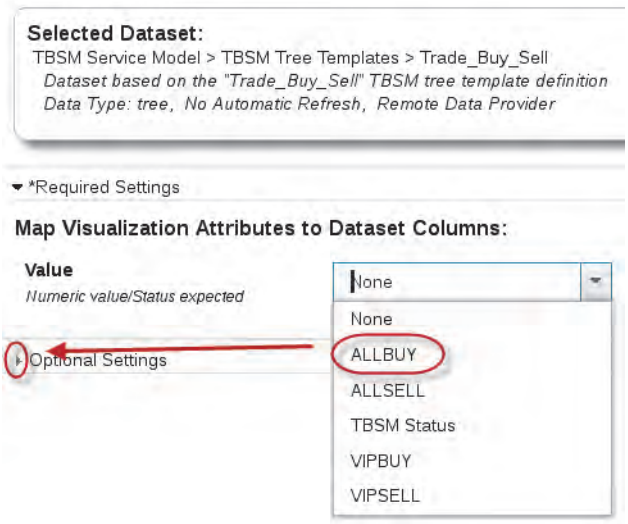
   b. Edit the widget properties. Click the **Edit options** icon in the upper right of the widget or click the widget and select the **Widget > Edit** menu above the widget palette.

   c. Select the data set that is used for this widget. This widget uses values that are filtered with the Tivoli Business Service Manager tree template Trade_Buy_Sell. The template was defined in the Tivoli Business Service Manager dashboard server and is shown in Figure 2, “dayTrader KPIs filtered by tree template,” on page 8-3. Enter **buy** in the search field and click **Search**.

   d. Multiple matching data sets are associated with the service models. Select the **Trade_Buy_Sell** entry that is listed with **Provider: TBSM Service Model > Datasource: TBSM Tree Templates**.

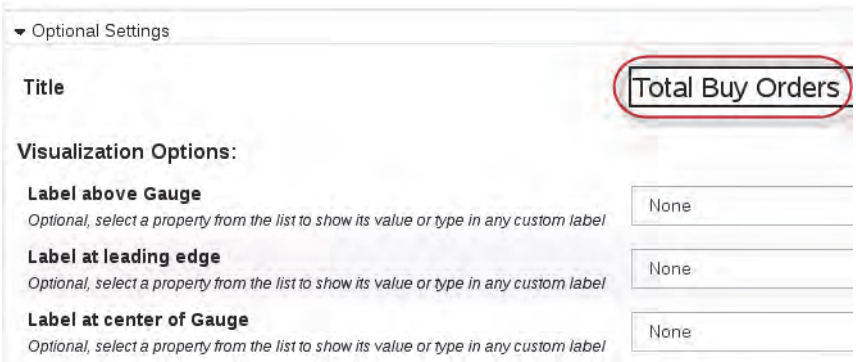


e. Select **ALLBUY** in the **Value** menu in the **Required Settings** section.



f. Expand the Optional Settings section.

g. Enter **Total Buy Orders** in the **Title** field.



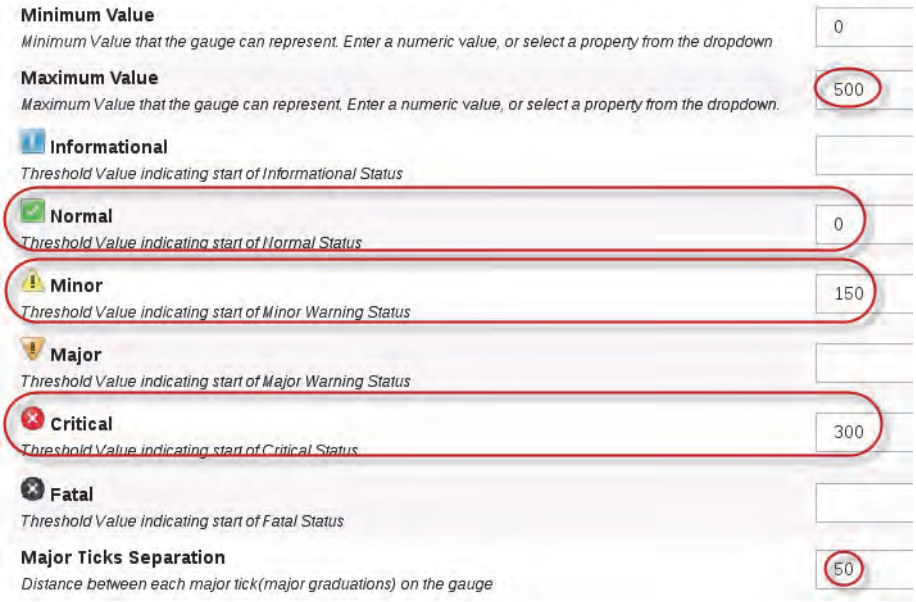
h. Change the **Maximum Value** field to **500**.

i. Set the **Normal** threshold to **0**.

j. Set the **Minor** threshold to **150**.

k. Set the **Critical** threshold to **300**.

l. Change the **Major Ticks Separation** value to **50**.

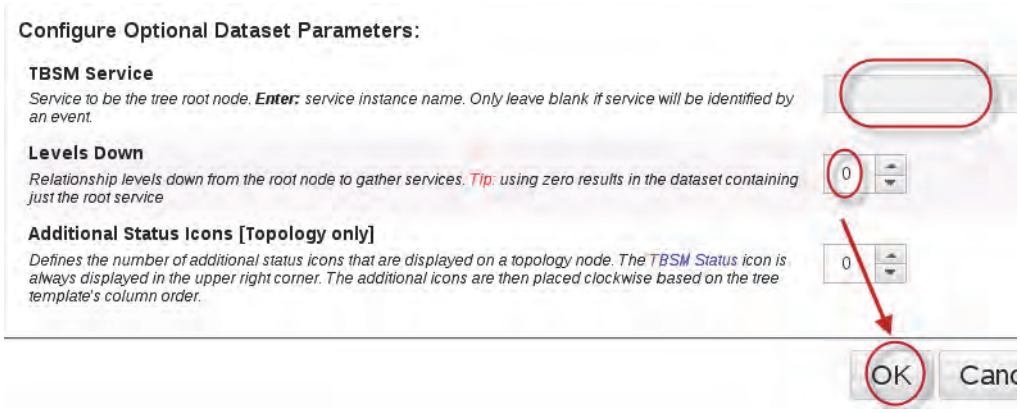


m. Leave all other values at the default settings and scroll down to the **Configure Optional Dataset Parameters** section.

n. The service data target for this widget is specified with context information that is sent when you select a status gauge in the top-level page that you create in Unit 8, Exercise 3 on page 8-26. Leave the TBSM Service field blank.

The tree template data set includes hierarchical information that is associated with the tree template. The nodes of the tree are the service models for the five data centers. This widget is configured to show data for only one selected city.

o. Change the **Levels Down** value to **0** and click **OK**.

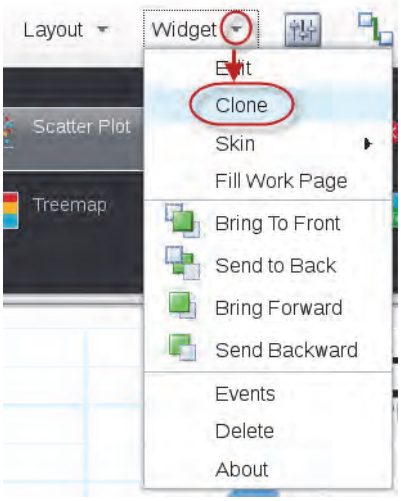


It is normal for the gauge widget to indicate that the available data is unknown. The gauge value is not specified until a context event is received.

7. Do not change the default widget size.
8. Move the widget to the left side of the page canvas, below the web widget.
9. Save your work, but do not close the page editor. Click **Save** above the widget palette section.

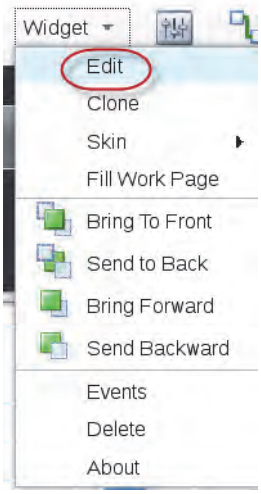
Note: The page layout specifies four analog gauge widgets. To reduce the configuration time, you *clone* three copies of the first widget. When you clone the widget, you create a copy of the source widget with the same size and configuration. The only difference between the four widgets is the widget title and the selected tree template column in the Value menu.

10. Create a clone of the Analog Gauge widget.
    a. Click the widget and select the **Widget > Clone** menu above the widget palette.

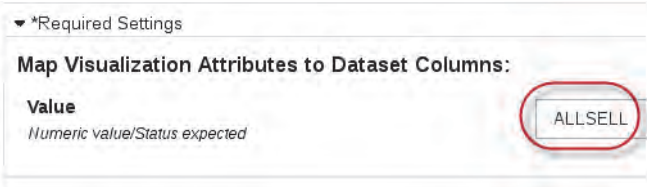


Note: Sometimes the clone widget is created so that it exactly overlaps the source widget. If you select Clone and do not see a copy on the page, move the widget that you see to verify that the source widget is not hidden.

b. Edit the widget copy properties. Click the widget copy and select the **Widget > Edit** menu above the widget palette section.



c. Change the Value menu selection to **ALLSELL**.



d. Expand the **Optional Settings** section and change the Title field value to **Total Sell Orders**.

e. Leave all other properties values and click **Save** to save the widget copy configuration.

f. Do not change the default widget size and align the widget to the right of, but not overlapping, the first analog gauge widget.

11. Create a third copy of the analog gauge widget.
    a. Click the second analog gauge widget and select the **Widget > Clone** menu above the widget palette.
    b. Edit the widget copy properties. Click the third widget copy and select the **Widget > Edit** menu above the widget palette section.
    c. Change the Value menu selection to **VIPBUY**.
    d. Expand the **Optional Settings** section and change the **Title** field value to **VIP Buy Orders**.

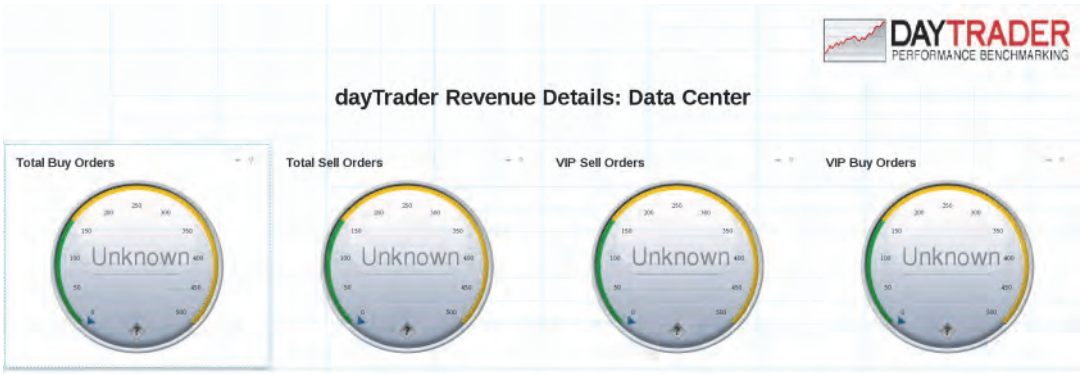
    e. Leave all other properties values and click **Save** to save the widget copy configuration.
    f. Do not change the default widget size and align the widget to the right of, but not overlapping, the second analog gauge widget.

12. Create a fourth copy of the analog gauge widget.
    a. Click the third analog gauge widget and select the **Widget > Clone** menu above the widget palette.
    b. Edit the widget copy properties. Click the third widget copy and select the **Widget > Edit** menu above the widget palette section.
    c. Change the **Value** menu selection to **VIPSELL**.
    d. Expand the **Optional Settings** section and change the **Title** field value to **VIP Sell Orders**.

    e. Leave all other properties values and click **Save** to save the widget copy configuration.
    f. Do not change the default widget size and align the widget to the right of, but not overlapping, the third analog gauge widget.
    g. Remove the frame, but not the title, from each of the analog gauge widgets. For each widget, click the **Edit options** icon in the upper right of the widget and select the **Skin > Transparent** menu.

h. Save your work, but do not close the page editor. Click **Save** above the widget palette section.

i. The page should look similar to the following screen image:



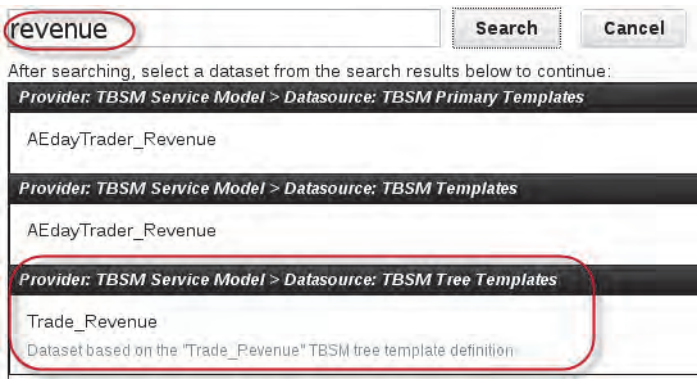
13. Add a volume bar widget to the page canvas.

a. Click and drag the **Volume Bar** object from the widget palette to the bottom of the page canvas.

b. Edit the widget properties. Click the **Edit options** icon in the upper right of the widget or click the widget and select the **Widget > Edit** menu above the widget palette.

c. Select the data set that is used for this widget. This widget uses values that are filtered with the Tivoli Business Service Manager Trade_Revenue tree template. Enter **revenue** in the search field and click **Search**.

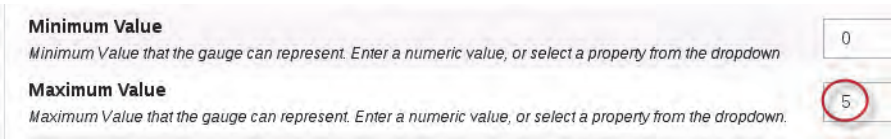
d. Multiple matching data sets are associated with the service models. Select the **Trade_Revenue** entry that follows **Provider: TBSM Service Model > Datasource: TBSM Tree Templates**.



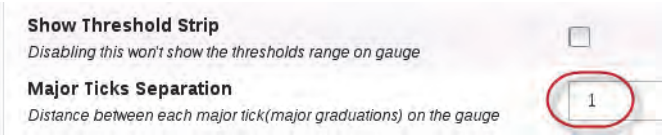
e. Select **CLIENTTOTAL** in the **Value** menu.

f. Expand the **Optional Settings** section and enter **Avg. Total Revenue Change (%)** in the **Title** field.

g. Change the **Maximum Value** field to **5**.



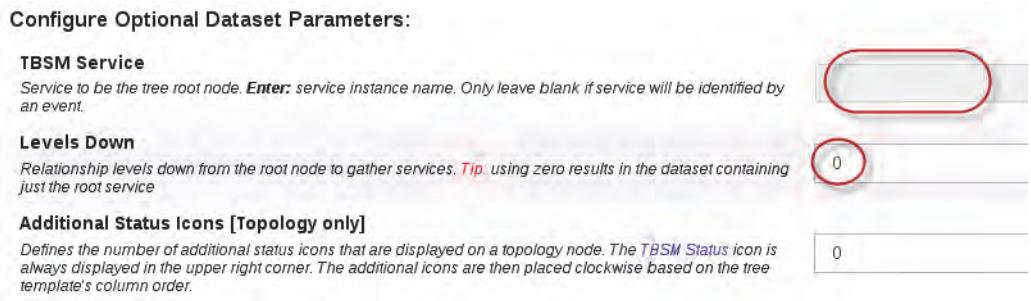
h. Change the **Major Ticks Separation** field to **1**.



i. Scroll down to the **Configure Optional Dataset Parameters** section.

j. Leave the **TBSM Service** field blank.

k. Set the **Levels Down** field to **0** and click **Save** to save the widget configuration.

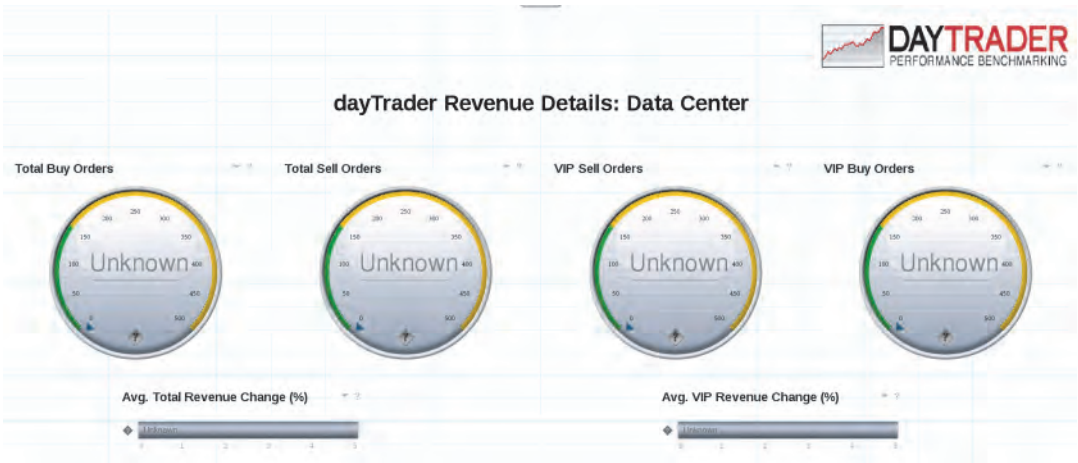


l. Set the size of the widget to **four canvas grid blocks high and four canvas grids wide** and move the widget to the lower left of the page, under the analog gauge widgets.

14. Create a clone of the volume bar widget.
    a. Click the volume bar widget and select the **Widget > Clone** menu above the widget palette section.
    b. Edit the properties of the widget copy. Click the widget copy to set the object focus and select the **Widget > Edit** menu above the widget palette.
    c. Change the **Value** menu to **VIPTOTAL**.
    d. Change the **Title** field to **Avg. VIP Revenue Change (%)**.

    e. Leave all other widget values and click **Save** to save the widget configuration.
    f. Align the widget with the first volume bar widget, but at the lower right of the page.
    g. Remove the frame, but not the title, from both of the volume bar widgets. For each widget, click the **Edit options** icon in the upper right of the widget and select the **Skin > Transparent** menu.

       The final page layout should look like the following screen image.



    h. Save the page configuration and close the page editor. Click **Save and Exit** above the widget palette section.

15. Close the page. Click the **X** symbol in the **dayTrader Revenue Detail** page tab in the console.

Exercise 3 Creating the dayTrader Revenue Status page

In this exercise, you create the top-level page for the dayTrader dashboard as shown in Figure 5. The page shows the revenue status information for the Seattle, Los Angeles, Dallas, Chicago, and New York data centers. The status information is shown on a background map image of the United States. A text widget is configured as a page title banner and the dayTrader logo widget is shown in the upper right of the page.

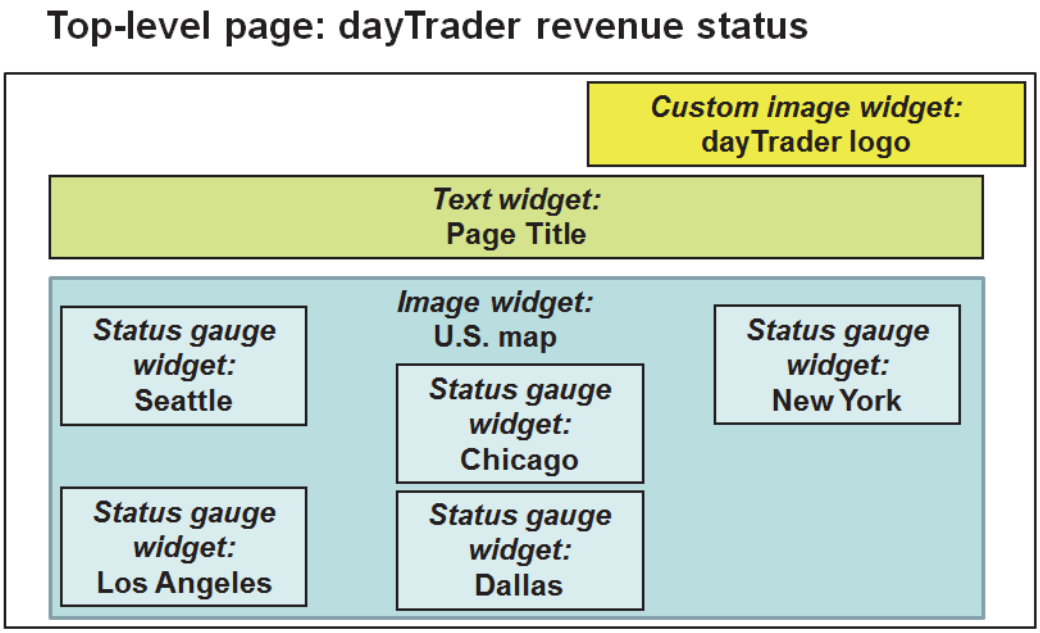


Figure 5 Top-level page diagram

You must complete the following high-level tasks to create the page:

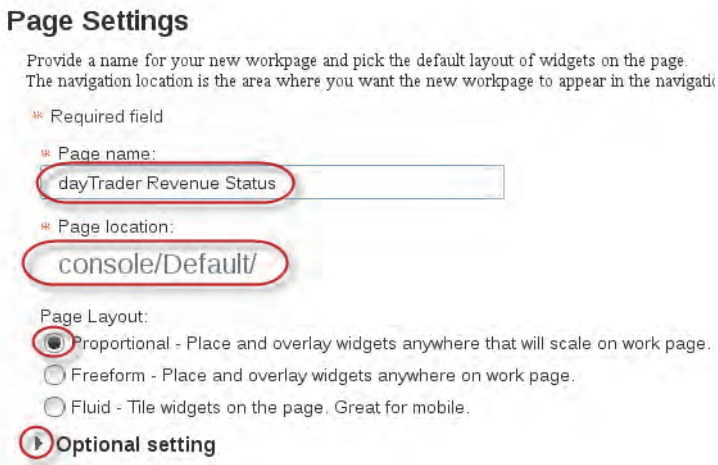
- Create and configure the page properties
- Add and configure the page widgets

Creating and configuring the page properties

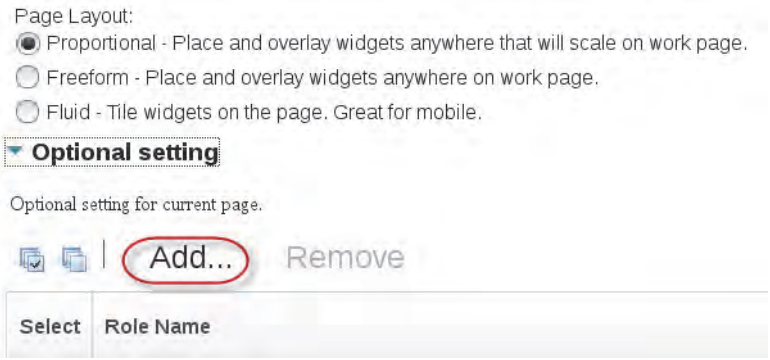
1. Create a page. Click the plus symbol (+) in the upper right of the console.

2. Enter **dayTrader Revenue Status** in the **Page name** field.
3. Leave the **Page location** value at **console/Default**.
4. Select **Proportional** for the **Page Layout** type.

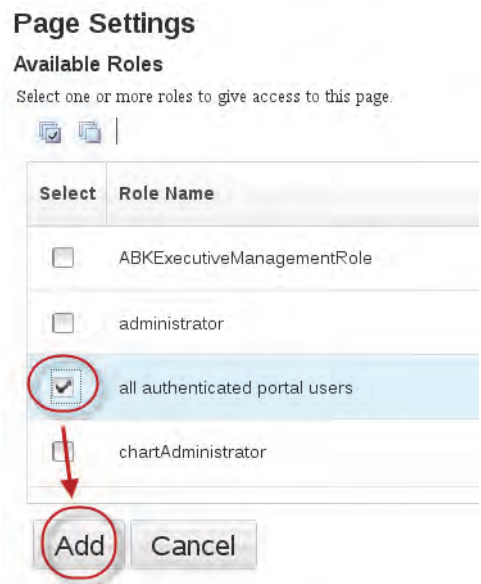
5. Configure authorization roles for the page, by clicking to expand the **Optional setting** section.



6. Click **Add**.

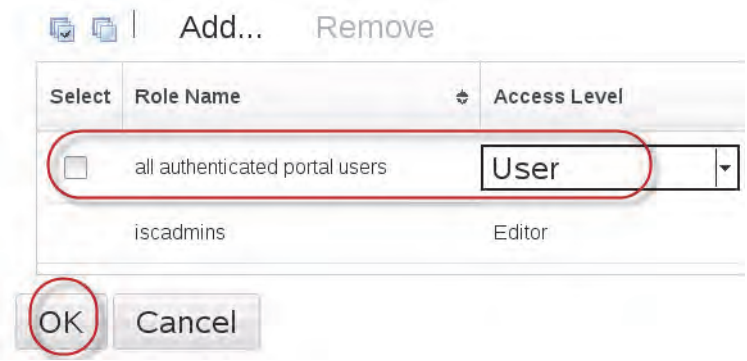


7. Select **all authenticated portal users** in the list of roles and click **Add**.



The role selection is shown the role assignment list.

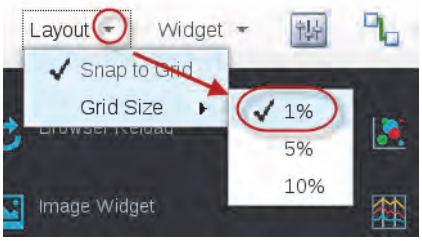
8. Leave the default access level and click **OK**.



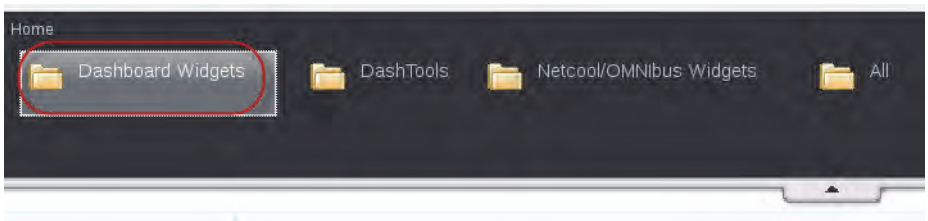
The page properties are replaced with the page editor.

Adding and configuring the page widgets

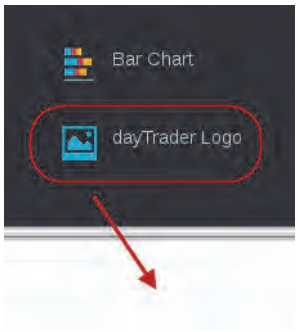
1. Set the object tracking granularity to support finer control of the widget placement in the page canvas. Click the **Layout > Grid Size > 1%** menu above the widget palette section.



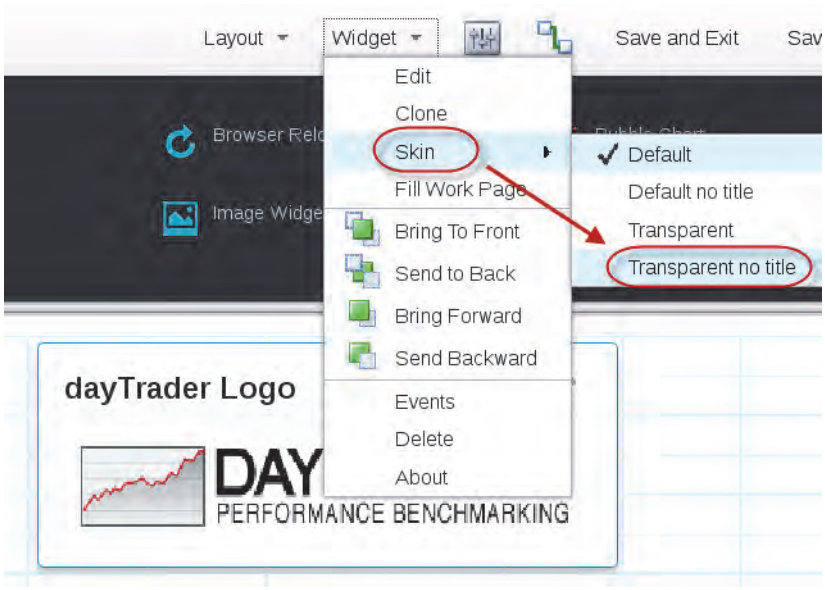
2. Click **Dashboard Widgets** in the widget palette section.



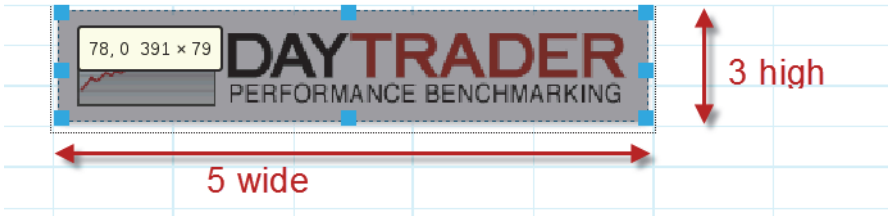
3. Add and configure the dayTrader Logo widget in the canvas.
   a. Click and drag **dayTrader Logo** from the widget palette to the page canvas.



   b. Remove the widget frame and title. Click the widget to set the focus and select the **Widget > Skin > Transparent no title** menu above the widget palette.

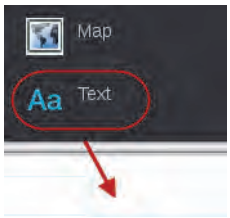


   c. Set the widget size. Click and drag the frame anchor points to set the widget size to three canvas blocks high and five canvas blocks wide.



   d. Move the dayTrader Logo widget to the upper right of the page canvas.

4. Add a text widget to the page. You configure this widget to be the page banner.
   a. Click and drag the **Text** widget object from the widget palette to the top center area of the page canvas.



   b. Edit the widget properties. Click the widget and select the **Widget > Edit** menu above the widget palette.

   c. Enter **dayTrader Revenue Status** in the **Title** field.

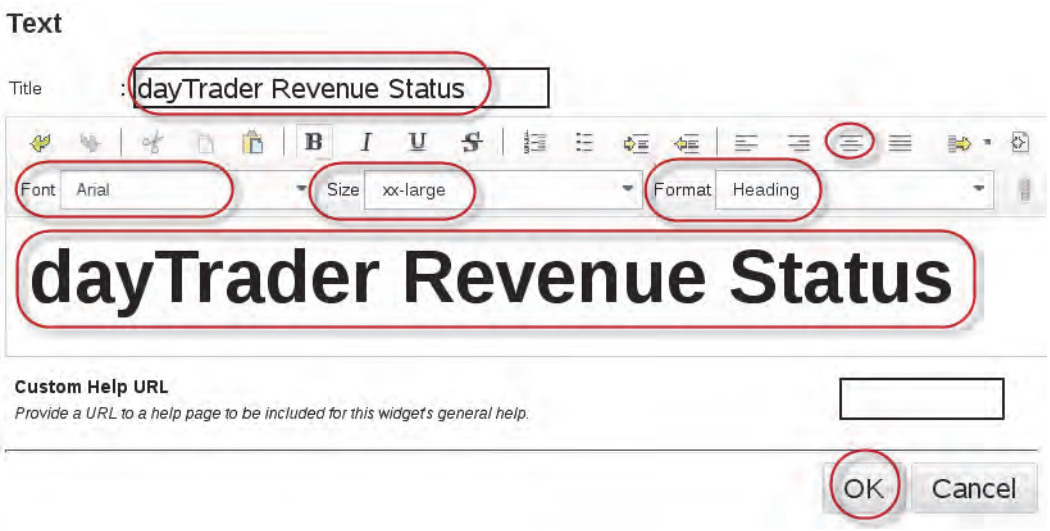
   d. Select **Arial** in the **Font** menu.

   e. Select **xx-large** in the **Size** menu.

   f. Select **Heading** in the **Format** menu.

   g. Click the **Align Center** icon at the top of the widget toolbar.

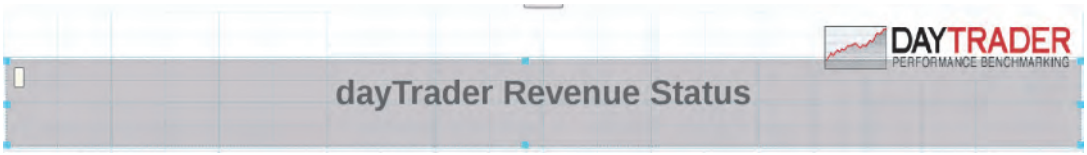
   h. Enter **dayTrader Revenue Status** in the text entry field and click **OK** to save the widget configuration.



   i. Remove the widget frame and title. Click the widget and select the **Widget > Skin > Transparent no title** menu above the widget palette.

j. Resize the widget so that it is four grid blocks high and spans the page canvas.

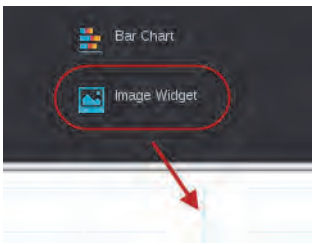
k. Move the widget so that it is aligned on the page canvas just below the dayTrader logo widget.



l. Save your work but do not close the page editor. Click **Save** above the widget palette.

5. Add an image widget icon to the page. This widget shows a background map of the United States.

a. Click the **Image Widget** icon in the palette and drag the icon to the page canvas.



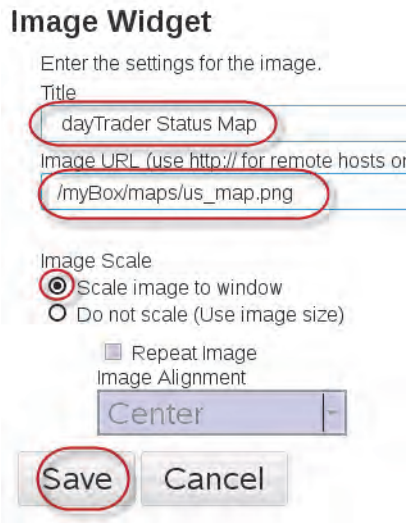
b. Remove the widget frame and title. Click the widget and select the **Widget > Skin > Transparent no title** menu above the widget palette.

c. Click the widget and select the **Widget > Edit** menu to edit the widget properties.

d. Enter **dayTrader Status Map** in the **Title** field.

e. Enter **/myBox/maps/us_map.png** in the **Image URL** field.

f. Select **Scale image to window** and click **Save** to save the widget configuration.



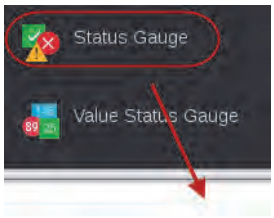
g. Resize the image to 14 grid blocks high and 12 grid blocks wide.

h. Move the widget so that it is centered at the lower edge of the page canvas.

i. Save your work but do not close the page editor. Click **Save** above the widget palette section.

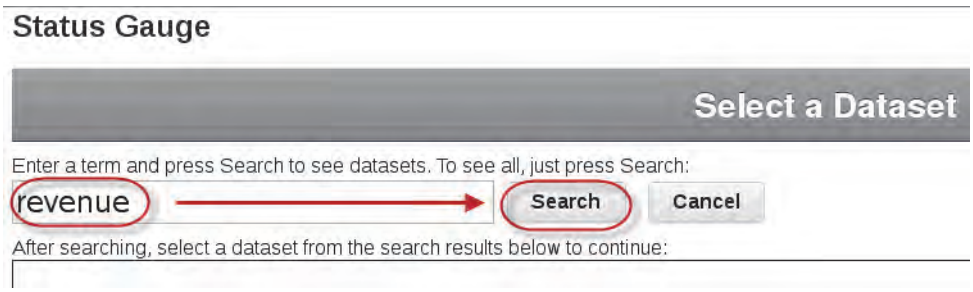
6. Add a status gauge to the page. This status gauge shows the status of the dayTrader application for the Seattle data center.

   a. Click and drag the Status Gauge object from the widget palette to the page canvas.

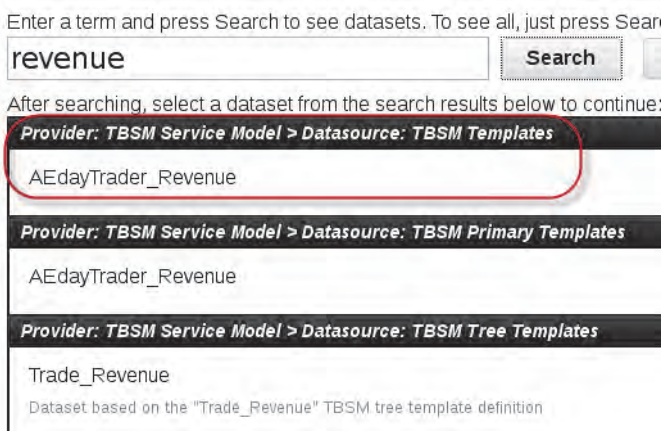


   b. Edit the widget properties with the **Widget > Edit** menu.

   c. This gauge shows the dayTrader revenue KPI status. Enter **revenue** in the search field and click **Search**.



   d. Use the status value that is calculated in the AEdayTrader_Revenue template. Select the **AEdayTrader_Revenue** data set under **Provider: TBSM Service Model > Datasource: TBSM Templates**.

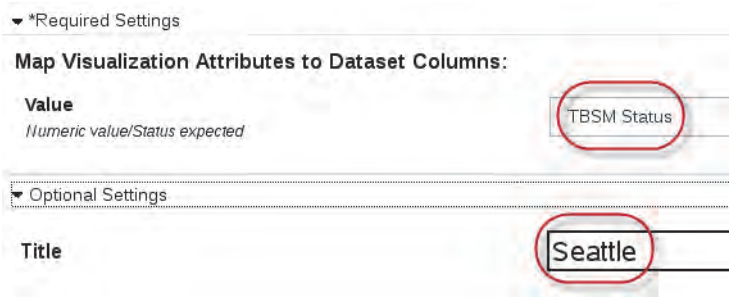


   e. Select **TBSM Status** in the **Value** menu in the **Required Settings** section.

f. Expand the **Optional Settings** section.

g. Enter **Seattle** in the **Title** field.

Important: To save time in these exercises, you do not configure a visible text label to identify each status gauge widget. In Unit 8, Exercise 4 on page 8-38, you configure event wires for the five status gauge widgets you configure on this page. If you do not enter a value in the Title field, you cannot easily distinguish the event wire source when you create the event wires.



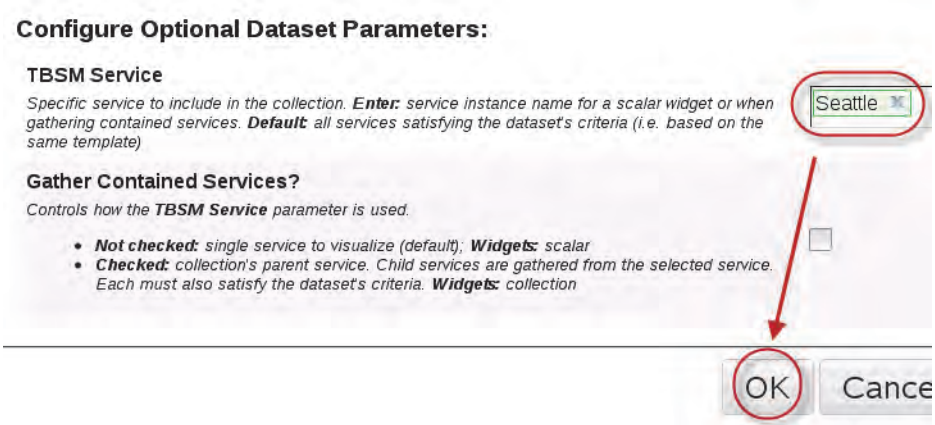
This widget is configured to show the status for the Seattle data center.

h. Enter **Seattle** in the **TBSM Service** field and press the **Return** key.

A green background encloses the input value to indicate that the input value is set.

Important: The widget properties form does not verify that the value that you entered is a valid business service name. To search and select from the list of available business services, click **Add** next to the **TBSM Service** field.

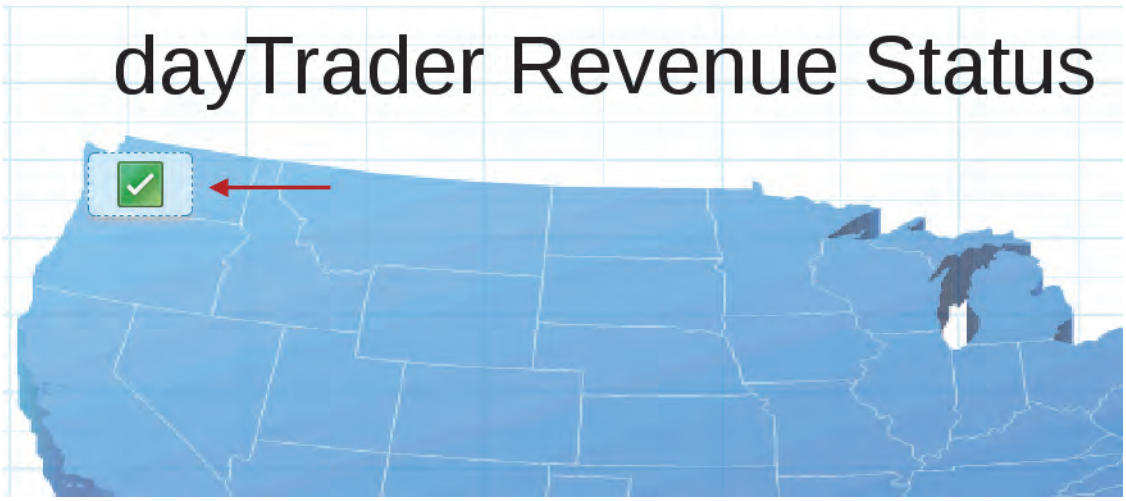
i. Click **OK** to save the widget configuration.



j. Remove the widget frame and title. Click the widget and select the **Widget > Transparent no title** menu.

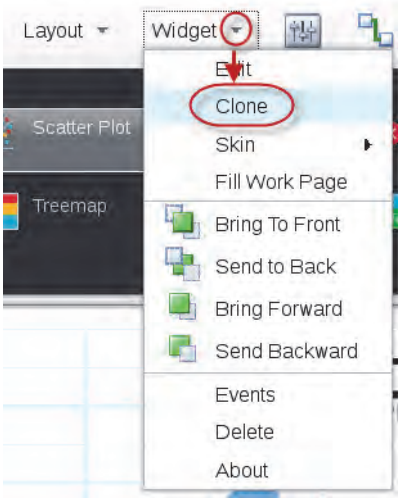
k. Set the widget size to two grid blocks high and one grid block wide.

l. Move the widget to the upper left portion of the map image.

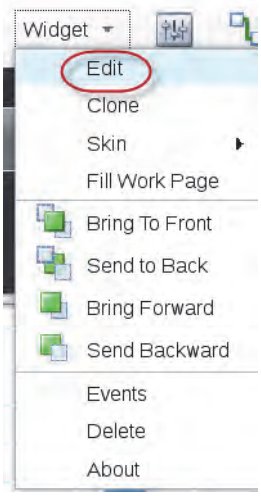


7. Create a clone of the first status gauge widget. This copy is configured to show the revenue KPI status for the Los Angeles data center. You change only the configuration of the widget title and business service.

a. Click the Seattle status gauge widget and select the **Widget > Clone** menu.



b. Edit the widget copy properties. Click the widget copy and select the **Widget > Edit** menu.

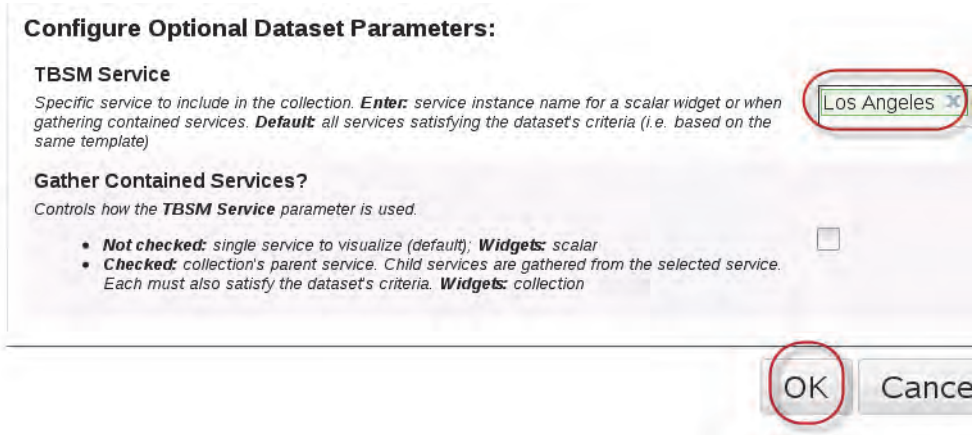


c. Leave the Value selection and expand the **Optional Settings** section.

d. Change the **Title** field to **Los Angeles**.

e. Change the **TBSM Service** value to **Los Angeles** and press the **Enter** key.

f. Click **OK** to save the configuration.

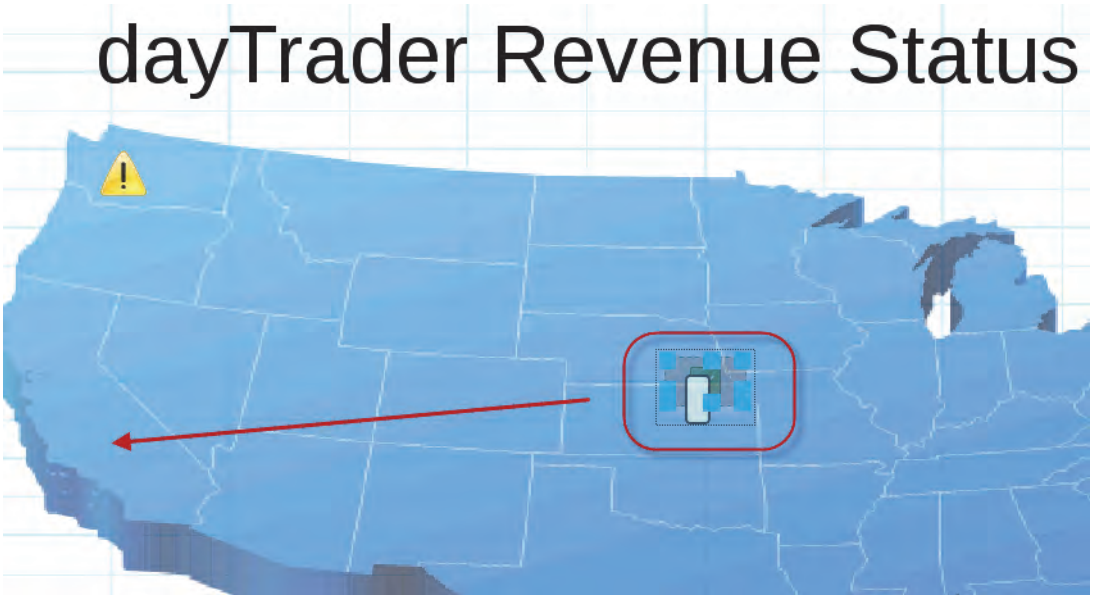


g. Click the widget, edit the widget properties, and select the **Widget > Transparent no title** menu.

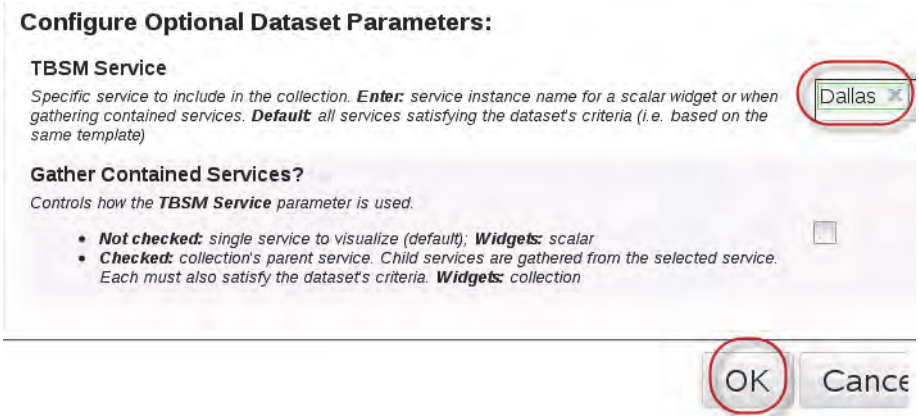
Important: A known bug, as of version 1.2, causes cloned widgets to lose the widget skin configuration. If the original widget uses any configuration other than default values, remember to set the widget skin configuration.

h. Move the widget copy to the lower left edge of the map image.

Note: If you cannot click and drag the widget with the mouse, click an edge of the widget to activate the resize points and use the keyboard cursor keys to move the widget.

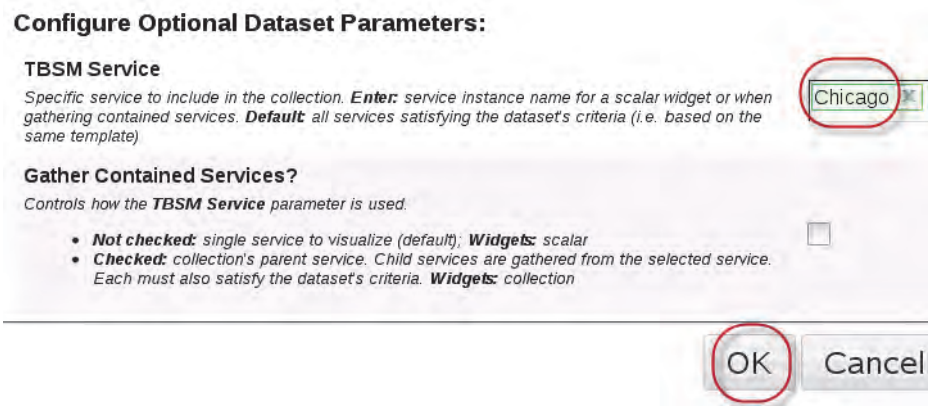


8. Use the previous process to create a third status image clone and change the **Title** field and **TBSM Service** field values to **Dallas**.

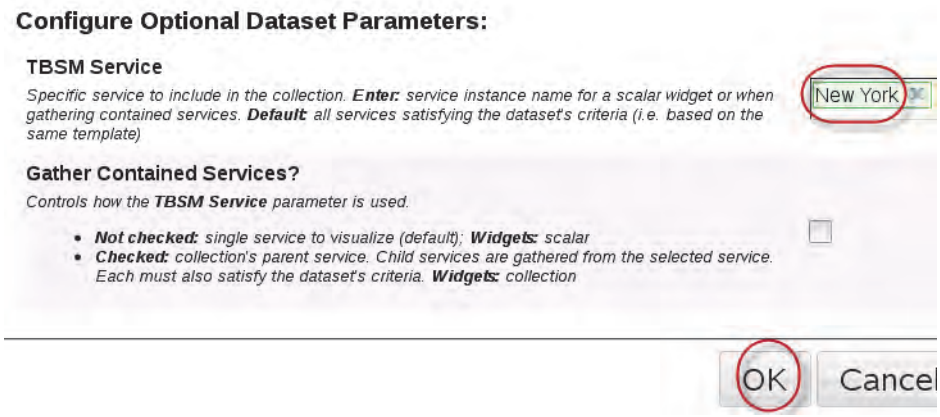


9. Move the **Dallas** status gauge widget to the north central area of Texas on the map image.
10. Click the widget, edit the widget properties, and select the **Widget > Transparent no title** menu.

11. Use the previous process to create another status image clone and change the **Title** field and **TBSM Service** field values to **Chicago**.

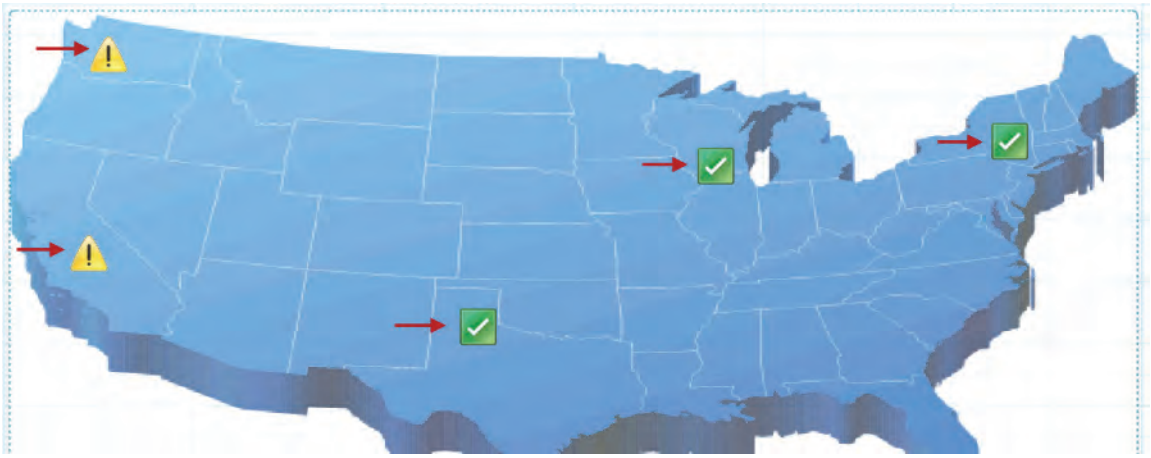


12. Move the **Chicago** status gauge widget to the northern tip of Illinois on the map image.
13. Click the widget, edit the widget properties, and select the **Widget > Transparent no title** menu.
14. Use the previous process to create another status image clone and change the **Title** field and **TBSM Service** field values to **New York**.



15. Move the **New York** status gauge widget to New York on the map image.
16. Click the widget, edit the widget properties, and select the **Widget > Transparent no title** menu.

The final status gauge arrangement on the map image is shown in the following screen image.



17. Save your work but do not close the page editor. Before you complete the page configuration, you create event wires in Unit 8, Exercise 4 on page 8-38. Click **Save** above the widget palette section.

Exercise 4 Linking pages with wires

In this exercise, you create *event wires* that send NodeClickedOn events from each status gauge in the top-level page (dayTrader Revenue Status) to the second-level page (dayTrader Revenue Detail). Each event wire is configured to deliver context information to all subscribing widgets in the second-level page. Events that are delivered at the page level are automatically sent to all subscribing widgets on the target page. You create and configure only one wire for each status gauge with this method. Figure 6, “dayTrader event wire logical layout diagram,” on page 8-38 shows a logical view of the event wires and the relationship of each top-level widget and the second-level target page.

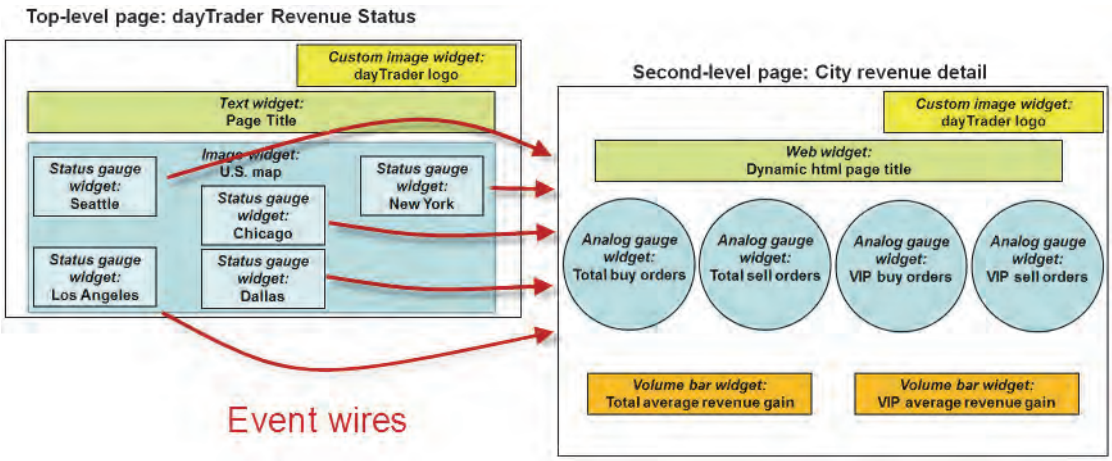


Figure 6 dayTrader event wire logical layout diagram

Complete the following tasks to create the event wires and complete the page configuration.

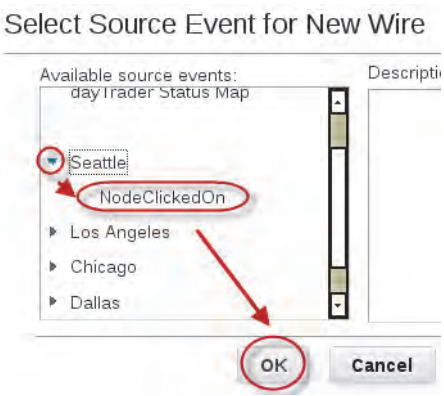
1. Click the **Show Wires** icon in the page and widget control menu bar in the upper right of the page.

The **Summary of wires** table is shown at the top of the page. By default, no wires are defined.

2. Create the first event wire.
   a. Click **New Wire** in the **Summary of wires** table.

   b. Select the source event for the wire. Click **Seattle** in the **Available source events** column and select the associated **NodeClickedOn** entry.

   c. Click **OK**.



   d. Configure the event wire to send the NodeClickedOn event to all widgets in the second-level page. Click the **Default** folder and select **dayTrader Revenue Detail**.

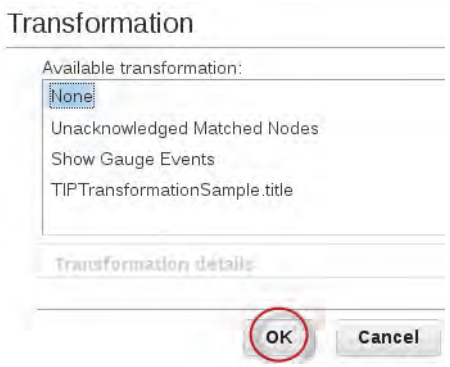
Important: Do not select any of the individual widgets that are listed in the dayTrader Revenue Detail page as the wire target. The page is selected as the target so that the source event is sent to all available widgets in the page. Select a specific widget as a target only if you require the source event to apply to only that target widget.

e. Configure the wire to automatically open and show the target page. Select **Load the selected target page** and **Switch to the selected target page**.

f. Click **OK**.

g. Do not select an event transformation option. Click **None** in the **Available transformation** column.

h. Complete the event wire configuration. Click **OK**.

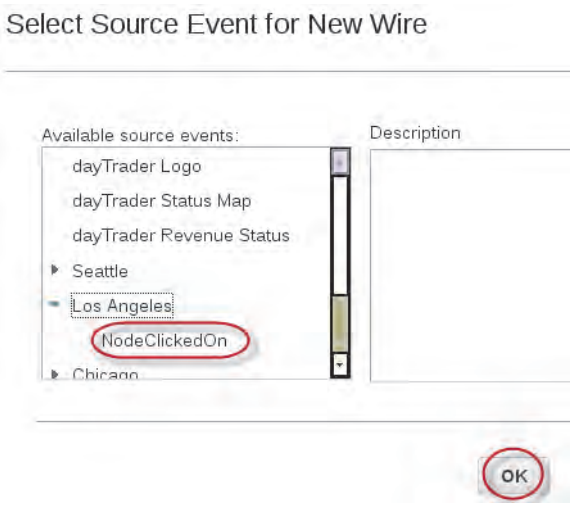


The event wire configuration is shown in the **Summary of wires** table.

3. Create the second event wire for the **Los Angeles** status gauge with the same process that you used to create the first event wire.
   a. Click **New Wire** in the **Summary of wires** table.

   b. Select the source event for the wire. Click **Los Angeles** in the **Available source events** column and select **NodeClickedOn**.

   c. Click **OK**.

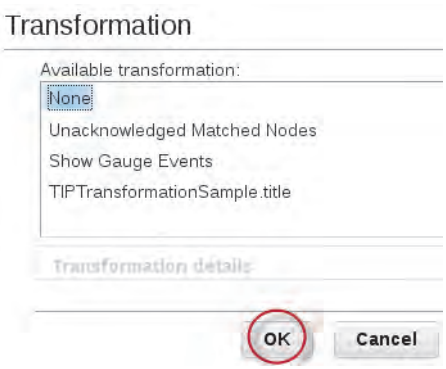


   d. Configure the event wire to send the NodeClickedOn event to all widgets in the second-level page. Click the **Default** folder and select **dayTrader Revenue Detail**.

e. Configure the wire to automatically open and show the target page. Select **Load the selected target page**, **Switch to the selected target page**, and click **OK**.

f. Do not select an event transformation option. Click **None** in the **Available transformation** column.

g. Complete the event wire configuration. Click **OK**.

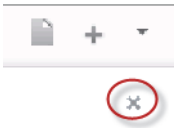


4. Use the previous wire creation process to create the third event wire to connect the **Dallas** status gauge to the **dayTrader Detail Page**.

5. Use the previous wire creation process to create the third event wire to connect the **Chicago** status gauge to the **dayTrader Detail Page**.

6. Use the previous wire creation process to create the third event wire to connect the **New York** status gauge to the **dayTrader Detail Page**. New York status gauge.

7. Close the **Summary of wires** table. Click the **Close Table** (**X**) icon in the upper right of the table. Do not click the X icon that closes the browser.



8. Save the **dayTrader Revenue Status** page. Click **Save and Exit** above the page palette section.

The completed page looks like the following image.

Exercise 5 Testing the dashboard design

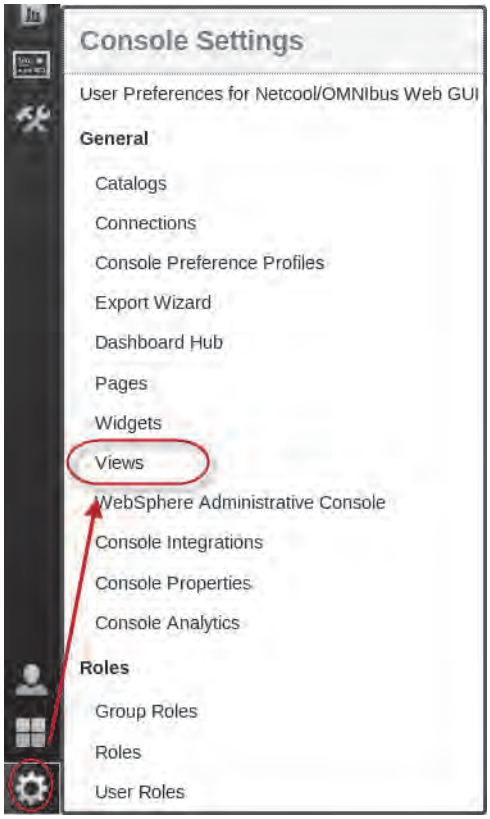
In this exercise, you do the following tasks to test the dashboard design:

- Add the dashboard pages to the view that you created in Unit 1, Exercise 3 on page 1-10.
- Log on to the Jazz for Service Management console as a designated mobile user and verify that the dashboard pages that you created function properly.

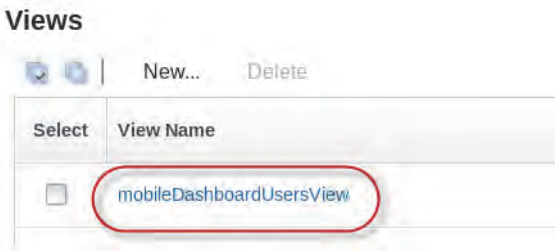
You log on to the DASH console as the user ID **whill**, which is assigned the **mobileDashboardUsersRole** authorization role during the laboratory setup.

Adding the dashboard pages to an existing view

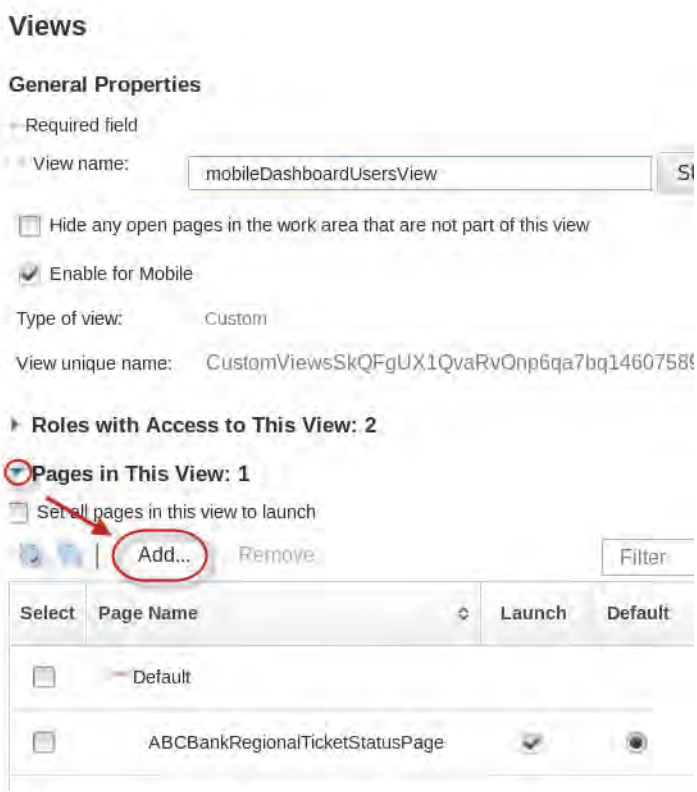
1. Add the two dashboard pages that you created in this exercise to the **mobileDashboardUsersView** view.
   a. Select the **Console Settings > Views** task.



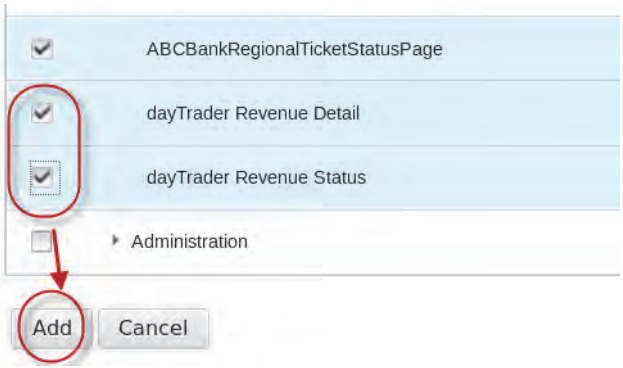
   b. Click **mobileDashboardUsersView** to open the view configuration form.



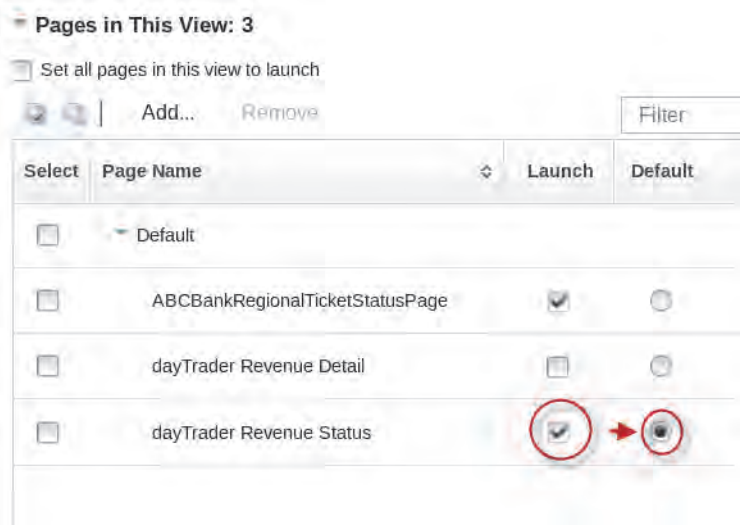
c. Click to expand the **Pages in This View** section and click **Add**.



d. Expand the **Default** folder, select **dayTrader Revenue Detail**, **dayTrader Revenue Status**, and click **Add**.



e. Update the configuration in the **Pages in This View** section. Select **dayTrader Revenue Status** in the **Launch** column and the **Default** column.



f. Scroll to the bottom of the page and click **Save**.

Logging in to the DASH console

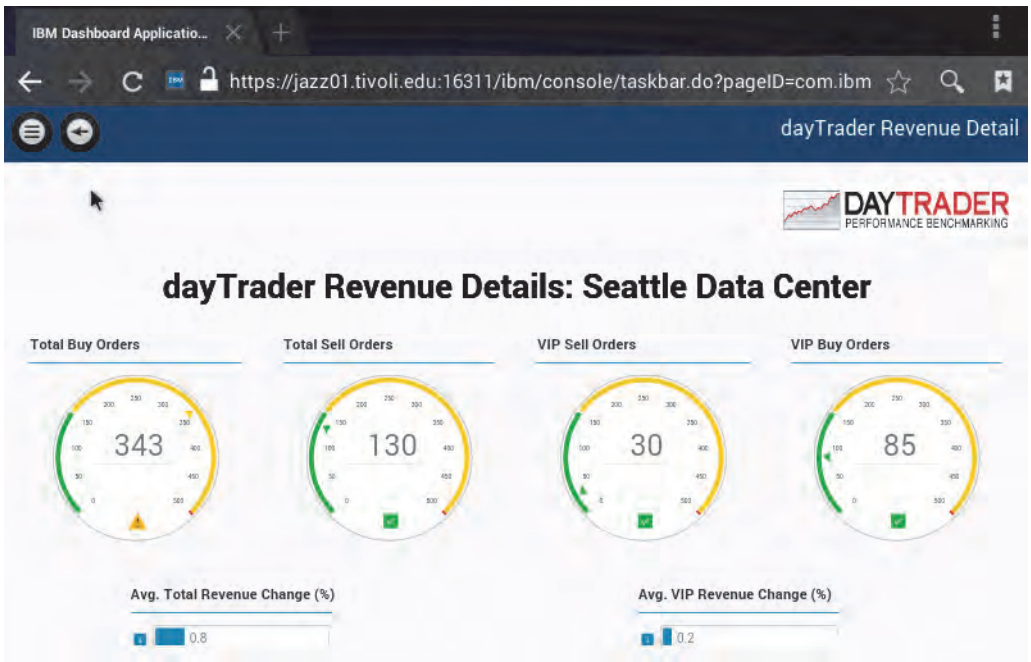
1. Log out of the Jazz for Service Management console. Click the person icon in the console taskbar and select the **Log out** menu.

2. Log on to the console with the user ID **whill** and the password **object00**.

   The **mobileDashboardUsersView** view that you created in Unit 1, Exercise 3 on page 1-10 and modified in this exercise is opened and the **dayTrader Revenue Status** page is shown.

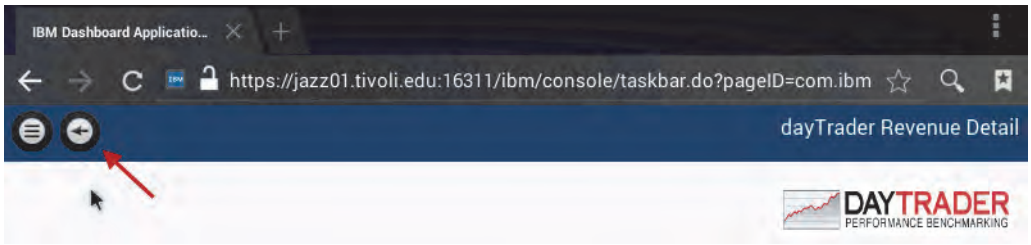
Note: The dashboard page images were taken from a mobile device. The images in your laboratory environment looks slightly different, but are functionally equivalent.

3. Test the event wire configuration between the dashboard pages.
   a. Click the **Seattle** status gauge in the upper left of the map widget.



The dayTrader Revenue Details page is shown with the Seattle data center contextual data.

b. Switch to the dayTrader Revenue Status page. Click the arrow icon on the left of the page title bar.



c. Click the status gauge for the other data centers and verify that the dayTrader Revenue Detail page shows the correct contextual data.

Unit 9 Networks for Operations Insight dashboards exercises

This unit has no student exercises.

Unit 10 Exporting and importing customizations exercises

In these exercises, workshop students export their DASH customizations and finalize their dashboard design and development and present their dashboards to the workshop.

Exercise 1 Exporting DASH customizations

In this exercise, you export all of the DASH customizations that you created during the workshop exercises. You then delete some customizations, import your customization archive, and examine the effects of the import process.

There are two ways to export DASH customizations:

- The Console Settings > Export Wizard task
- The consolecli.sh command line interface

In this exercise, you will use the Export Wizard to export your customizations and use the consolecli.sh command to import your customizations.

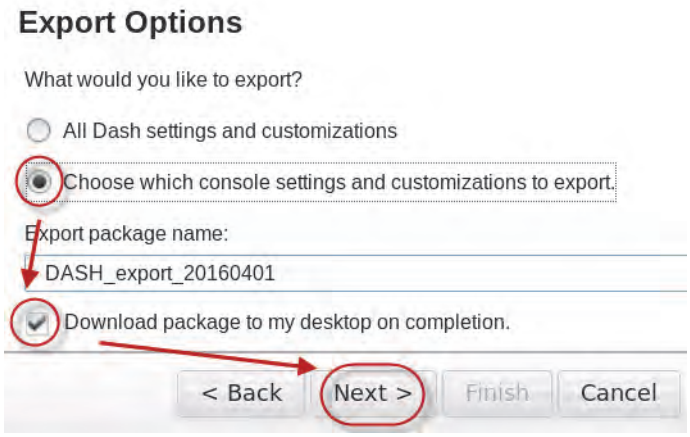
To complete this exercise, you must complete the following high-level tasks:

- Export dashboard customizations
- Delete selected dashboard pages
- Import dashboard customizations

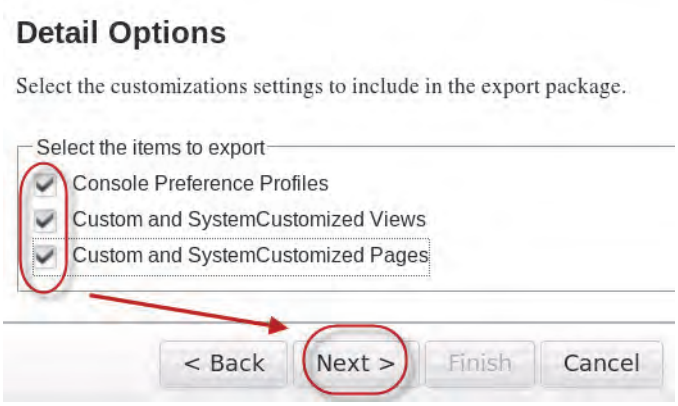
Exporting dashboard customizations

1. Select the **Console Settings > General > Export Wizard** task in the DASH console.
2. Click **Next**.

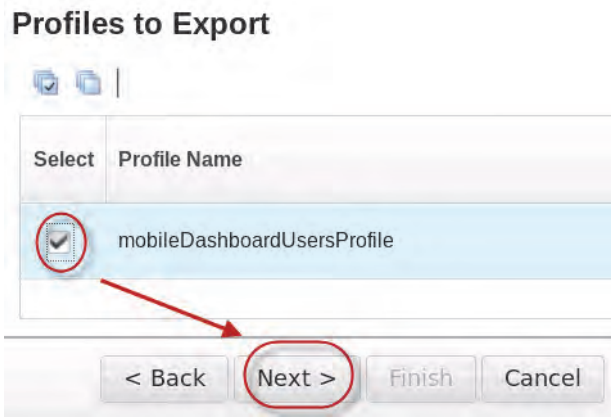
3. Select **Choose which console settings and customizations to export**, select **Download package to my desktop on completion**, and click **Next**.



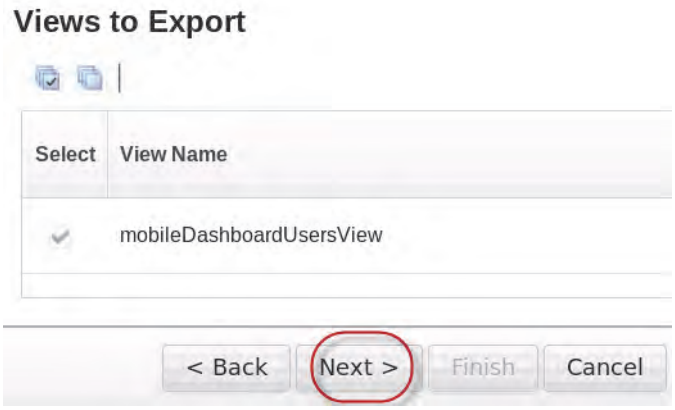
4. In the **Detail Options** page, select **Console Preference Profiles**, **Custom and SystemCustomized Views**, **Custom and SystemCustomized Pages**, and click **Next**.



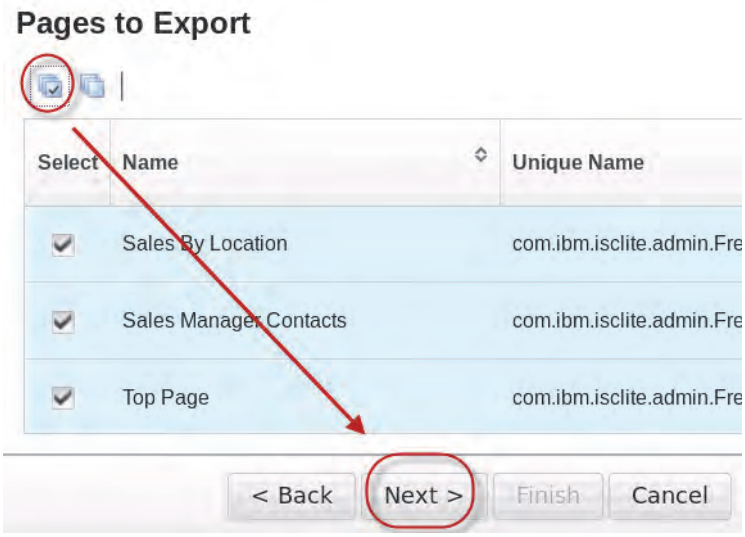
5. Select all profiles to export and click **Next**.



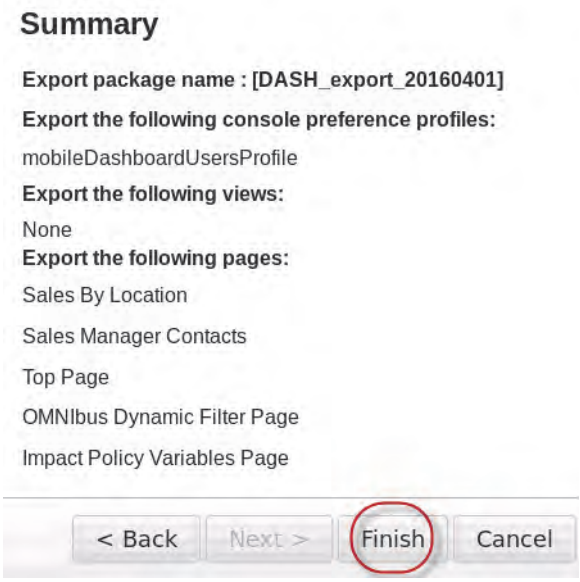
6. Because you selected all of the custom profiles, the configured views are automatically selected. Click **Next**.



7. Export all custom pages. Click the **Select all items** icon and click **Next**.

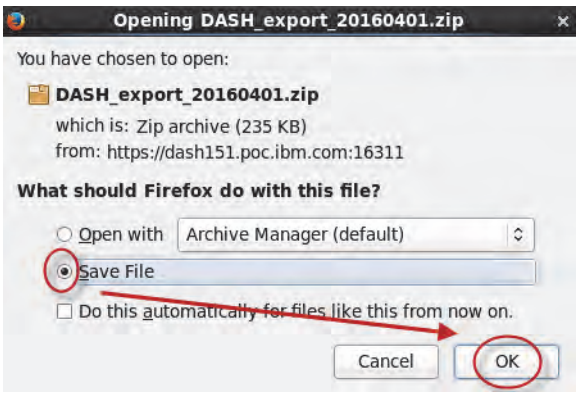


8. Click **Finish** in the **Summary** page.

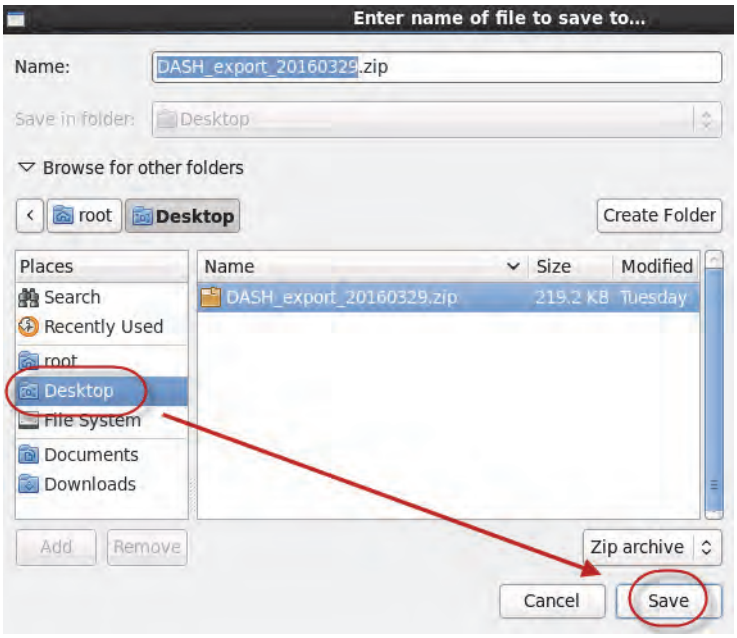


A save file window opens.

9. Select **Save File** and click **OK**.



10. In the file browser window, select the **Desktop** folder, use the **default file name**, and click **Save**.

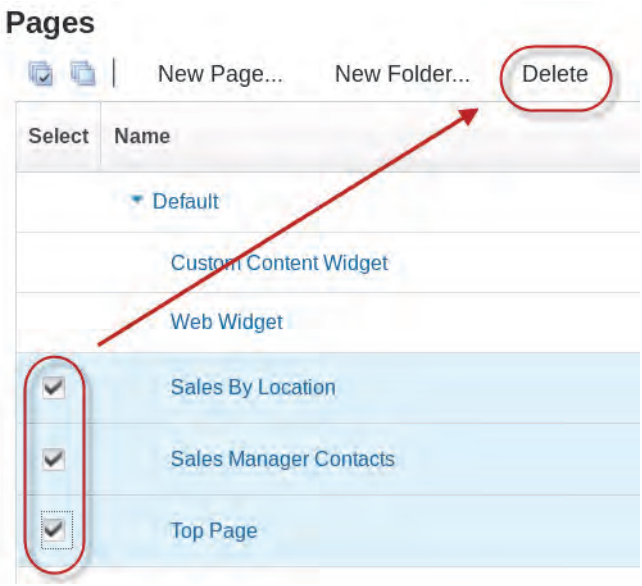


Note: Your default file name will be different from the one that is shown in the screen image.

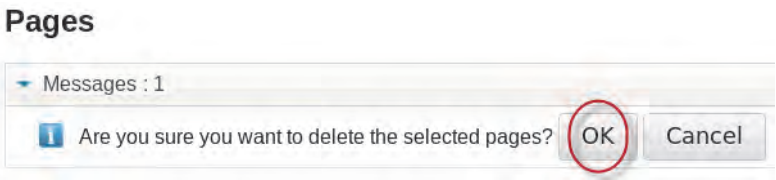
11. Close the Export Wizard page. Click the **X** icon in the **Export Wizard** tab at the top of the console.

Deleting dashboard pages

1. Select the **Console Settings > General > Pages** task.
2. Select the **Sales By Location**, **Sales Manager Contacts**, and **Top Page** pages and click **Delete**.



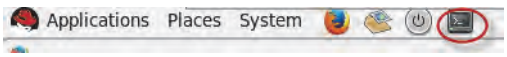
3. Click **OK** to confirm the file deletions.



4. Close the Pages page. Click the **X** icon in the **Pages** tab at the top of the console.
5. Verify that the pages are deleted. Move the mouse over the **Default** folder in the console taskbar.

Importing dashboard customizations

1. Click the **Terminal** icon at the top of the dash151 image to open a command window.



2. Copy the export archive file to the /opt/IBM/JazzSM/ui/input directory. Enter the following commands:

   ```
   cd /opt/IBM/JazzSM/ui/bin
   ls /root/Desktop/*.zip
   cp /root/Desktop/<archive_file_name>.zip /opt/IBM/JazzSM/ui/input/data.zip
   ```

   where **<archive_file_name>** is the name of the file on your desktop. If prompted to overwrite an existing data.zip file, enter **y**.

3. Import the data.zip archive into the DASH server. Enter the following command to import all customizations. Enter the following command:

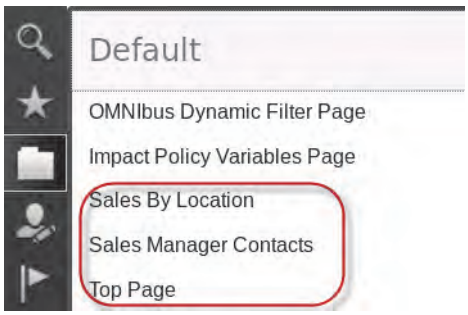
   ```
   ./consolecli.sh Import --username smadmin --password object00
   ```

   The import takes 5-10 seconds.

   ```
   CTGWA4240I CMS data source is used to work with remote launch entries and is an
   optional setting. It will not be created since required parameters are missing
   CTGWA4017I The command completed successfully.
   [root@dash151 bin]#
   ```

4. To see the updated page list, log off and log on to the DASH console with the user ID **smadmin**, password **object00**.

5. Verify that the previously deleted pages are restored.



Appendix A Dashboard Java Server Pages

This appendix shows the Java Server Page (JSP) file and cascading style sheet (CSS) files that are used to create dynamic dashboard page banners in these exercises.

dayTradeHeader3.jsp

```
<%-- Load java util libraries --%>
<%@ page import="java.util.*" %>
<html>
<head>
<title>dayTrader revenue detail</title>
<link href="dynamicTextExample.css" rel="stylesheet" type="text/css">
</head>
<body>
<h1> dayTrader Revenue Details:
<%=request.getParameter("TbsmServiceInstanceName")%> Data Center</h1>
</body>
</html>
```

dynamicTextExample.css

```
h1 {
color:black;
text-align: center;
font-family: Arial, Verdana;
}
```

IBM Training