# HP OpenView
# Business Process Insight

## Integration Training Guide
## Defining Business Metrics

**Software Version: 02.00**


**January 2006**

# Legal Notices

## Warranty

*Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.*

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

## Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

## Copyright Notices



## Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc.

Microsoft® is a US registered trademark of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Windows® and MS Windows® are US registered trademarks of Microsoft Corporation.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

## Support

Please visit the HP OpenView web site at:

**http://www.managementsoftware.hp.com/**

This web site provides contact information and details about the products, services, and support that HP OpenView offers.

You can also go directly to the support web site at:

**http://www.hp.com/managementsoftware/support**

HP OpenView online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valuable support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit and track progress on support cases
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and log in. Many also require a support contract.

To find more information about access levels, go to:

**http://www.hp.com/managementsoftware/access_level**

To register for an HP Passport ID, go to:

**http://www.managementsoftware.hp.com/passport-registration.html**

# contents

1

# Business Metrics

This chapter looks at how to create and use business metrics using the OpenView Business Process Insight (OVBPI) Metric Definer.

# Introduction

Once an OVBPI flow is deployed and running, the OVBPI Business Impact Engine maintains basic statistics about the flow, such as:

- The overall state of the flow.
- The number of active flow instances.
- Specific flow instance statistics such as:
  - The start and stop times of each node.
  - Node durations.

These statistics are held in the OVBPI database. The OVBPI database tables are fully described in the *OVBPI System Administration Guide*.

In addition to the standard flow and node statistics that are maintained for all flows, by the Business Impact Engine, you can specify additional flow statistics, known as **business metrics**. For example, you can measure business metrics such as:

- The time taken to process an order.
- The current backlog of flights waiting to get airborne.
- The average value of an order.

Once you have configured one or more business metrics for a flow, you can configure **thresholds**. This allows OVBPI to raise an alert when a business metric exceeds one of your thresholds. For example, you may configure the following thresholds:

- Issue an alert when the time taken to process an order exceeds four hours.
- Issue a warning alert when the current backlog of flights waiting to get airborne is greater than 10. Issue a critical alert when this number exceeds 50.
- Issue an alert when the value of a particular order is more than two standard deviations greater than the normal average order. In other words, issue an alert when a particularly oversized order is in the system.

Once you have defined your metrics and thresholds, the metric values and alerts are reported in the OVBPI Business Process Dashboard.

All metric, threshold and alert data is maintained in the OVBPI database so you can also provide your own reporting directly from these tables, or use a third party reporting tool to produce any custom reports.

# Metric Definer

Once a flow is defined and deployed, you use the OVBPI Metric Definer to define metrics and thresholds for that flow.

## Running the Metric Definer

The Metric Definer runs within a Web browser, thus you need to make sure that you have the OVBPI Servlet Engine running, and that the OVBPI database is up and running.

To run the Metric Definer you can either go to:

```
Start->Programs->HP OpenView->Business Process Insight->Metric Definer
```

or, start a Web browser with the URL:

`http://`*`hostname`*`:`*`44080`*`/ovbpimetricdefiner`

where:

- *`hostname`* is the hostname of your OVBPI installation
- *`44080`* is the default port for the Servlet Engine

For example:

```
http://localhost:44080/ovbpimetricdefiner
```

This works if your Business Impact Engine is installed on the same machine as your Web browser.

## Logon User/Password

When you run the Metric Definer, it asks you to log on as a valid user.

The default user name/password details are:

```
User:     admin
Password: ovbpi
```

This security access is configured in the OVBPI Servlet Engine. Your Web administrator is able to alter this security setting (See the *OVBPI System Administration Guide* for further details).

## Deployed Flows

You can create metrics only for flows that are already deployed.

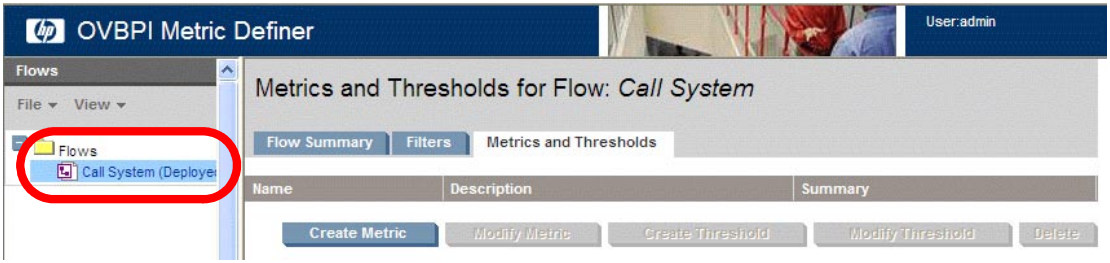
If while you are running the Metric Definer, a new flow is deployed, you can select `View->Refresh` from within the left-hand Navigator frame of the Metric Definer (not the Web browser's toolbar). This refreshes the list of deployed flows available to you within the Metric Definer.

# Creating A Business Metric

To create a business metric you must first select the flow. You simply click on the required flow in the left-hand Navigator frame within the Metric Definer.

For example:

**Figure 1 Metric Definer - Selecting a Flow**




When the flow is selected, the right-hand frame gives you three tab options:

- Flow Summary

  Clicking this tab gives you a picture of the flow definition and some basic details of the flow - such as description and deployment date.

- Filters

  This tab allows you to create and modify filters. This is explained in more detail in Filters on page 24.

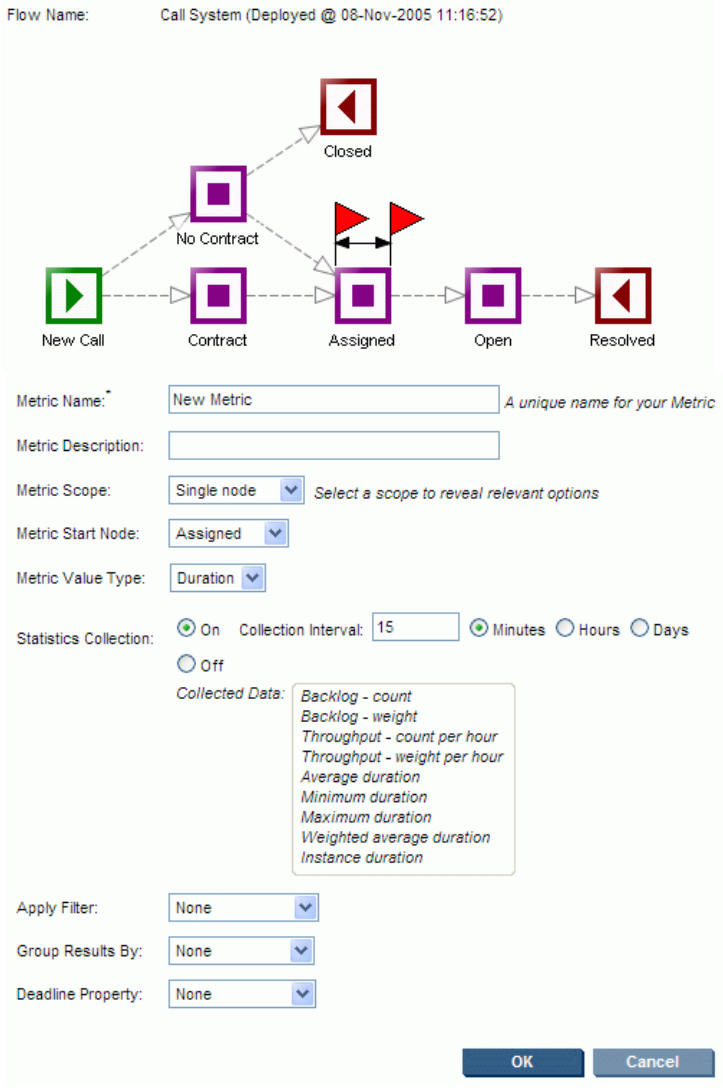
- Metrics and Thresholds

  This tab allows you to create/modify metrics and create/modify thresholds. This is the default tab when you click on a flow in the left-hand frame.

Once a flow is selected, you click on the `Create Metric` button, in the right-hand frame. You are then presented with the details you need to complete to create a new metric.

For example:

**Figure 2 Metric Definer - Create a new Metric**




Let's consider each of the fields in more detail...

## The Flow Diagram

The flow diagram is displayed and two metric flags (red colored flags) are drawn on one of the nodes. These metric flags define the start and end of the metric you are defining. Initially these flags are placed on a single node, and as you define the metric more fully, these flags move to reflect your configuration.

## Metric Name

You need to give your metric a unique name. This name must be unique across all your metrics.

## Metric Description

You can provide your metric with some description text. This is optional.

## Metric Scope

The following options are available to you.

### Whole Flow

Selecting this option causes the metric flags to disappear.

Setting a `Whole Flow` metric means that when a flow instance first starts (at whatever node actually starts the flow instance), a metric instance is started. This metric instance completes when the flow instance completes.

A `Whole Flow` metric is also known as a `Flow Execution Time (FET)` metric. You see this `FET` abbreviation if you access the metric definition directly within the OVBPI database.

### Single Node

Setting a `Single Node` metric means that a metric instance starts when a flow instance enters the specified node. This metric instance completes when the flow instance exits this same node.

A `Single Node` metric is also known as a `Node Execution Time (NET)` metric. You see this `NET` abbreviation if you access the metric definition directly within the OVBPI database.

#### Metric Start Node

When selecting a `Single Node` metric, you are also asked to specify the node itself. As you specify this node, the metric flags within the flow diagram move to highlight this node.

### Multiple Nodes

Setting a `Multiple Nodes` metric means that this metric measures between two nodes within the flow diagram.

A `Multiple Nodes` metric is also known as a `Time Between Nodes (TBN)` metric. You see this `TBN` abbreviation if you access the metric definition directly within the OVBPI database.

#### Metric Start Node

You select the start node for your metric. You specify whether it is the start or completion of this node that starts each metric instance.

#### Metric End Node

You select the end node for your metric. You specify whether it is the start or completion of this node that completes each metric instance.

## Metric Value Type

Here you select the type of metric. There are two metric types available by default:

- Duration

  The metric measures a duration. You measure the time taken from the start of the metric to the end of the metric, as specified in the scope of the metric.

- Weight

  The metric measures the weight. The weight is the value contained in the `weight` attribute as defined within the flow definition.

For example, you may wish to measure the value of each order (assuming that your flow holds the order value within the weight attribute).

You have the capability of adding your own Metric Value Types, known as `Custom` types. (See Chapter 3, Custom Metrics for more details and examples.)

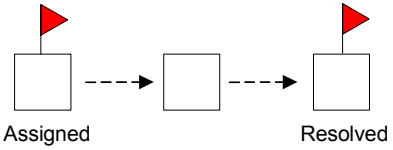
# Statistics Collection

## Recording Metric Values

Selecting the `scope` of the metric determines when the actual metric instance value is recorded.

For example, suppose you configure a metric, and set the metric scope to be from the start of the node `Assigned` to the end of the node `Resolved`.

**Figure 3 Example Metric Definition**




When a flow instance enters the `Assigned` node, a new metric instance is instantiated and OVBPI records the time this metric instance starts. If the metric is defined as a `weight` metric, then OVBPI also records the current weight value.

When this flow instance eventually completes the `Resolved` node, OVBPI completes the associated metric instance and records the time it completes. If the metric is defined as a `duration` metric, then OVBPI calculates the overall duration that this metric instance has taken to complete. If the metric is defined as a `weight` metric, then OVBPI records the final weight value of this metric instance.

As each metric instance starts, OVBPI records the start time for the metric instance. As each metric instance completes, OVBPI determines the duration, or weight value, for the metric. This duration, or weight, represents the “value” for this completed metric instance. So the “value” of the metric instance is recorded at the completion of each metric instance, as defined by the scope of the metric.

## Calculating Statistics

If the “value” of the metric instance is recorded at the completion of each metric instance, as defined by the scope of the metric, then why do you need to enable `Statistics Collection`?

Yes, each metric instance value is recorded as the metric instance completes. But what about being able to calculate the average of all your orders? or whether a particular incoming order rate is faster or slower than normal?

To be able to calculate things such as averages, throughput rates and backlogs, you need to enable Statistics Collection.

## Collection Interval

If you enable Statistics Collection, you need to specify how often you want the statistics to be calculated. This is specified by the `Collection Interval`.

If you were to specify a collection interval of 15 minutes, then every 15 minutes OVBPI would look back through the last 15 minutes of metric data values that had been collected and produce statistics. These statistics would include calculations such as the average of all metric instances completed in the last 15 minutes, the minimum/maximum/standard deviation and throughput of all the metric instances completed within the last 15 minutes, and the count of all metric instances still active at the end of the last 15 minute period (also known as the `backlog`). As you will see in a later section (Creating A Threshold on page 28) you can then set thresholds against these statistics such that (for example) if the average time to complete an order for the last 15 minutes is more than two standard deviations slower than the normal average for completing orders, then raise an alert.

## Collection Interval Timings

Suppose the time is 11:12am and you have just created a new metric and specified a collection interval of 15 minutes. As you create the new metric, within the Metric Definer, it is activated within OVBPI the moment you press the `OK` button. So the question is, does the collection interval run every 15 minutes from 11:12 (the time you created the metric)? Answer.....No!

When you create a metric and specify a collection interval, OVBPI wants to align the collection interval to an “easily understandable” time slot. That is, if you specify a collection period of 15 minutes, it makes sense to run the

collection interval on the hour and at 15, 30 and 45 minutes past the hour. To achieve this, OVBPI typically shortens the first collection interval so that they can be aligned within the hour.

So, in this example, where you defined the metric at 11:12am, the collection intervals runs as follows:

- 11:12 -> 11:15
- 11:15 -> 11:30
- 11:30 -> 11:45
- 11:45 -> 12:00
- etc...

where the first collection interval is less than the specified 15 minutes.

If you specified a collection interval of one day then the first collection period ends that night at midnight. The intervals then run from midnight to midnight.

If you specify a collection interval of one week then the first collection interval ends the following Saturday night at midnight. The week then runs from Saturday midnight to Saturday midnight.

## Choosing a Collection Interval

The time you specify for the collection interval needs some thought.

For example, suppose you wanted to track the average throughput of your orders on a daily basis. That is, you want to calculate your average daily order rate. You could set the collection interval to be one day and this would give you exactly what you want. However, be aware that this order throughput is only calculated each collection interval. That is, the throughput is calculated at the end of each day. Thus you would not see the order throughput rate for today until midnight tonight. And this throughput rate is not changed until the following day (at midnight). That is, each day it shows you yesterday's order throughput rate.

Now, suppose that you want to track the average throughput rate of your orders such that you can set a threshold that raises an alert whenever you start to see orders going through your system too slowly. Setting a collection interval of one day is probably not appropriate. With a collection interval of one day you do not see any statistical data until the next day. Also, if there is a

slow down of orders that lasts for maybe one hour, this may not have any real effect on the overall throughput figures for that day. You probably want to set a smaller collection interval - maybe one hour, maybe less.

So the collection interval determines how frequently OVBPI calculates metric statistics, and these statistics are a view of what has happened over the defined collection interval.

### Do You Need To Collect Statistics?

You may not need a collect any metric statistics.

For example, you may define a metric that simply measures the time taken for each of your orders to be processed. You may then have a threshold that raises an alert whenever any individual order takes more than a certain time to complete. In this situation you do not require any averages, throughputs or standard deviations...you are simply measuring and testing the time taken on a individual order basis.

If your metric is simply measuring the duration (or weight) of flow instances then you do not need to collect statistics. OVBPI is able to monitor thresholds and raise alerts based purely on the active and completed metric instances.

An example might be where you wish to have a metric that measures the time it takes for you to close your priority-1 support calls. If any call is taking longer than four hours then raise an alert. For this sort of metric and alert scenario you do not need to collect statistics. If on the other hand you wanted to monitor whether the average call resolution time for priority-1 call is faster or slower than the normal average resolution time, then you would need to collect statistics.

## Apply Filter

The `Apply Filter` option lets you select a filter to apply to your metric definition. You must have first created a filter before trying to select it when creating (or modifying) a metric.

By selecting a filter, you are able to restrict the number of metric instances recorded. For example, you might apply a filter that filters out only those orders placed by gold customers.

When applying a filter to a metric you must make sure that the attribute(s) tested within your filter actually have values at the time each metric instance is started. Otherwise, your filter always evaluates to false and no metric instances ever start.

See Filters on page 24 for more details about creating metric filters.

## Group Results By

The `Group Results By` field allows you to select one attribute from the flow's related data definition. The metric data and statistics are then collected and grouped by the values within that data attribute.

For example, suppose you are defining the metrics for an airport that has four flight terminals. You might choose to group the results by the airport terminal. This would allow you to see at a glance the performance for each terminal.

The attribute you select for `Group Results By` must be a `String` attribute.

### Group Name

If you select a `Group Results By` attribute, the Metric Definer displays an additional field called `Group Name`. This allows you to specify a display name to be used within the OVBPI Dashboard when displaying the metric data in groups.

Within the OVBPI Dashboard you are able to display the metric data in overall terms, or in groups. When displaying the metric data in groups the OVBPI Dashboard uses your `Group Name` text in the heading of the graph.

## Deadline Property

The `Deadline Property` field allows you to select an attribute of data type `date`, from the flow's related data definition.

By selecting a `Deadline Property` you give the metric the ability to provide a `Deadline` threshold. You are able to specify a threshold that raises an alert if a metric instance is still running relative to the date value specified within your `Deadline Property` attribute.

For example, suppose you have a flow that monitors mortgage applications. You want to set a threshold that raises an alert if a mortgage does not reach a certain step in the flow within 20 days of the mortgage application date. By setting the `Deadline Property` to a data attribute, which contains the mortgage application date within the data definition, you can specify such a threshold.

# Filters

When creating, or modifying, a metric you can apply a filter. But before you can select a filter you must have defined one.

## Creating a Filter

A filter consists of an expression, and the form of the expression is the same as that used when defining event subscription filters within the OVBPI Modeler (see *OVBPI System Administration Guide* for the details of the filter expression syntax).

For example, a filter of:

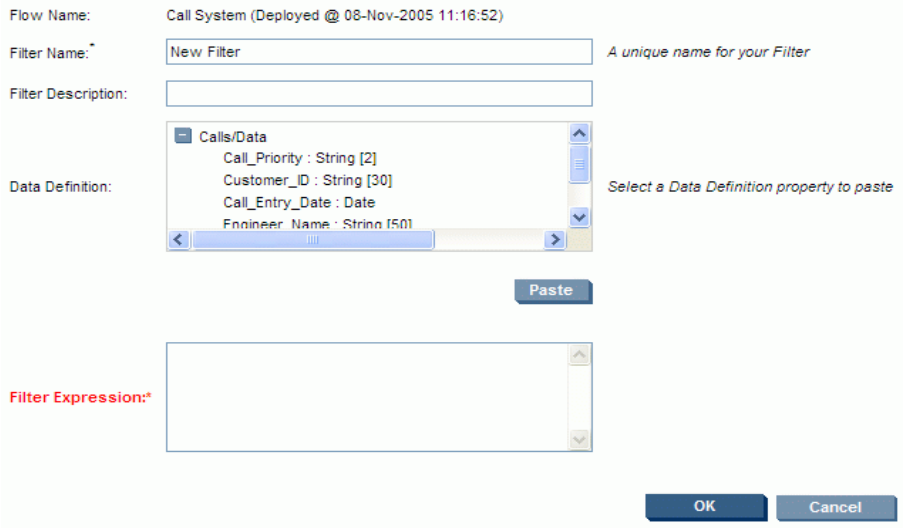
```
data.Customer_Type == "Gold"
```

filters only those instances where the `Customer_Type` data attribute is set to the string value `Gold`.

Within the Metric Definer, once you have selected your flow (in the left-hand navigator frame), you then click on the `Filters` tab in the right-hand frame. You then click the `Create Filter` button and you can create your filter.

An example filter creation screen might look as follows:

**Figure 4 Metric Definer - Filter Creation**



Let's consider each of the fields in more detail...

## Filter Name

When you define a filter you assign it a name. This must be unique across all the filters defined for this flow.

## Filter Description

You can provide description text for this filter.

## Filter Expression

You are required to provide a filter expression. This filter expression involves the testing of data within the flow's related data definition and must produce a true or false result.

To help you formulate your filter expression, the flow's related data definition attributes are displayed for you in the `Data Definition` text box. You can select an attribute within the `Data Definition` text box and click the `Paste` button. Your expression can involve multiple data attributes and you can use AND, OR logic.

All data attribute names are prefixed by the name of the data definition relationship as defined in the flow's definition. That is, if the related data definition is related using the name `data` then the attributes are specified as **`data.`**`attributeName`.

When defining a metric filter, make sure that data attribute(s) you use within your filter expression have values at the start of the metric when each flow instance is running. That is, you cannot filter on (for example) `Customer_Type == "Gold"` if at the start of the metric, the flow instance has not yet set the `Customer_Type` attribute to a value.

## Example Filters

```
data.Priority == 1
```

Filters only those instances where the `Priority` attribute value is set to the number 1 (one).

```
data.Status == "1"
```

Filters only those instances where the `Status` attribute value is set to the string value 1 (one).

```
data.StateValue.in("1", "2", "3")
```

Filters only those instances where the `StateValue` attribute is a string value that is in the set of values 1, 2, or 3.

```
   data.FlightCarrier.starts("QF")
|| data.FlightCarrier.starts("BA")
```

Filters only those instances where the `FlightCarrier` attribute value starts with either the string `QF` or `BA`.

```
   data.FlightCarrier.starts("QF")
&& data.ClientType.contains("Gold")
```

Filters only those instances where the `FlightCarrier` attribute value starts with the string `QF` and the `ClientType` attribute value contains the string `Gold`.

# Modifying A Metric

Once you have defined a metric for a flow, it is listed in the right-hand frame of the Metric Definer, along with any other metrics you have defined.

To **view** the detailed definition of a metric you select the metric (by clicking on the metric definition in the right-hand frame) and then click on the `Modify Metric` button. This shows you the full metric definition and allows you to view, or modify, the metric. Once you have seen the definition of the metric you can click the `Cancel` button and the metric is unchanged. As well as the `Cancel` button there are two other options: `Update` and `Replace`...

Once you have defined a metric within the Metric Definer that metric is active within OVBPI and collecting metric data. If you decide that you wish to make a modification to the metric definition you have two choices:

- **Update** the metric

  You can choose to make the modification to the metric but keep all the previously collected metric data. This is achieved by using the `Update` option.

  By pressing the `Update` button you are telling OVBPI to keep all currently collected metric data, and to start collecting new data according to the newly modified metric definition.

  You obviously need to be careful about using the `Update` button. If you were simply modifying the description of the metric definition then the `Update` button is the button to use. However, you would need to think carefully if you were modifying the metric to add (for example) a filter, as this would mean that any previously collected data and the newly collected data might represent different things and this may lead to confusion when you display all the metric data together.

- **Replace** the metric

  If you are modifying the metric in such a way that it does not make sense to keep the previously collected data, then you should use the `Replace` option. For example, if you are modifying an existing metric definition such that it now has a filter, it may not make sense to keep any previously collected metric data. By modifying the metric definition and clicking `Replace`, you are telling OVBPI to delete all previously collected metric data for this metric definition, and threshold alarms, and start collecting afresh.

# Creating A Threshold

Once you create a metric, the metric data is collected and stored in the OVBPI database. The OVBPI Dashboard allows you to report on this data.

As well as defining metrics, the Metric Definer also lets you define (create) **Thresholds**. For example, suppose you are collecting metrics to measure the time it takes for you to process your orders. You may wish to set up a threshold to alert you when any individual order is taking more than four days to be processed.

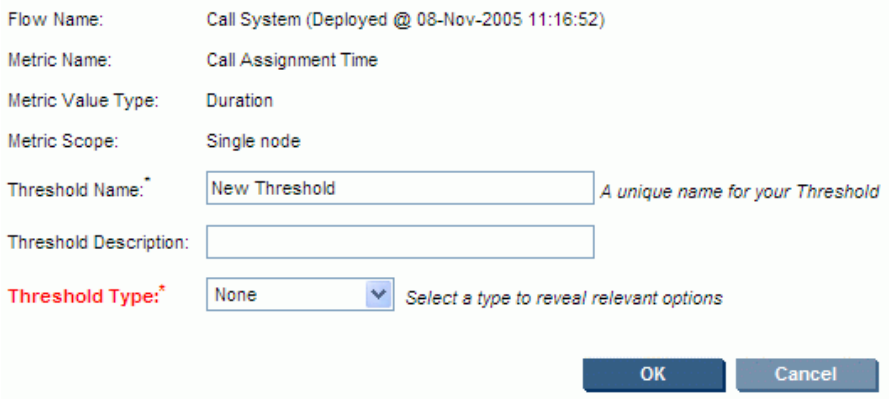
Within the Metric Definer, once you have created a metric, you are then able to create one or more thresholds against this metric.

To create a threshold:

- Select the flow definition - in the left-hand navigator frame
- Select the metric definition - in the right-hand frame
- Click the `Create Threshold` button...

  The screen appears in the right-hand frame, to create a new threshold. An example screen might look as follows:

**Figure 5 Metric Definer - Create Threshold**




The flow and metric details are automatically filled in for you.

You provide a name for this threshold. This name must be unique amongst all the thresholds for this flow.

You can then (optionally) provide a description for this threshold.

You then select the `Threshold Type` from the pull-down list, and this determines the remainder of the screen details. Let's consider the remaining options for this screen...

# Threshold Type

The following threshold types are available:

## Absolute (Duration/Weight/Value)

An absolute threshold allows you to set a threshold against an absolute value.

The text for the threshold type, and the units available on the threshold definition page, are dependent on the `Metric Value Type` as defined for the underlying metric. If the `Metric Value Type` for the metric is `Duration`, then the threshold type is called `Absolute Duration`. If the `Metric Value Type` for the metric is `Weight`, then the threshold type is called `Absolute Weight`. If the `Metric Value Type` for the metric is a custom defined metric (see Chapter 3, Custom Metrics) then the threshold type is called `Absolute Value`.

### Threshold Measure

You can set the threshold measure to be any specific instance, or a recent average/minimum/maximum.

- Any Specific Instance

  This allows you to set a threshold to raise an alert when any metric instance takes longer/shorter than the set time. Or when the weight of any metric instance is greater/smaller than a set value.

  This threshold measure means that the moment a metric instance exceeds the threshold, an alert is triggered. For example, if your threshold is monitoring that orders take less than four hours to complete, the moment an order has been running for four hours the threshold raises the alert.

- Recent

  This allows you to set a threshold to alert when the recent average (or minimum/maximum) is greater/smaller than a set value.

  The term `recent` means the value calculated over the last collection interval of your underlying metric. If the collection interval is 15 minutes then this threshold checks for any alerts every 15 minutes.

## Backlog

The backlog threshold type measures the number of metric instances that are active at the end of each collection interval for the underlying metric. So, setting a backlog threshold type means that the value is tested at the end of each collection interval.

### Threshold Measure

You can test the `Count` of active metric instances, or the `Total weight` of active metric instances.

## Deadline

The `Deadline` threshold type is available only if the underlying metric definition specified a `Deadline Property` attribute.

Setting a deadline threshold type allows you to measure that any metric instance has reached a deadline.

For example, you might say that you wish to raise an alert if a metric instance is still running and the current time is 20 days after the date/time specified in this instance's `Deadline Property`. Or you might like to raise an alert if a metric instance is still running and the current time has reached the date/time specified in this instance's `Deadline Property`.

## Relative (Duration/Weight/Value)

A relative threshold allows you to set a threshold against the standard deviation of your collected metric values.

The text for the threshold type is dependent on the `Metric Value Type` as defined for the underlying metric. If the `Metric Value Type` for the metric is `Duration`, then the threshold type is called `Relative Duration`. If the `Metric Value Type` for the metric is `Weight`, then the threshold type is

called `Relative Weight`. If the `Metric Value Type` for the metric is a custom defined metric (see Chapter 3, Custom Metrics) then the threshold type is called `Relative Value`.

### Threshold Measure

You can set the threshold measure to be any specific instance, or a recent average/minimum/maximum.

- Any Specific Instance

  This allows you to set a threshold to alert when any metric instance takes longer/shorter than a number of standard deviations from your overall average time. Or when the weight of any metric instance is greater/smaller than a number of standard deviations from your overall average weight.

  This threshold measure means that the moment a metric instance hits the threshold, an alert is triggered.

- Recent

  This allows you to set a threshold to alert when the recent average (or minimum/maximum) is greater/smaller than a number of standard deviations from your overall average.

  The term `recent` means the value calculated over the last collection interval of your underlying metric. If the collection interval is 15 minutes then this threshold checks for any alerts every 15 minutes.

## Throughput

The throughput threshold type measures the metric instances that completed in the last collection interval for the underlying metric. So, setting a throughput threshold type means that the value is tested at the end of each collection interval.

### Rate

You can test the `Count` or `Weight` of completed metric instances expressed as a value over a given time period.

## Warning/Minor/Major/Critical Alert

You can specify values for the different alert levels. You do not need to specify values for all alert levels, but you must provide at least one alert level value.

As a threshold crosses any specified alert values, an alert is raised within OVBPI. (See Alerts on page 33 for further details.)

## Alert Message

You can specify a text message to be issued when each alert is raised. This field can contain only simple text.

# Alerts

When thresholds are exceeded, alerts are generated.

The alerts are basically sent to two places:

- The Notification Server

  The alerts are sent to the Notification server and if you have configured any notification subscriptions then these are handled accordingly.

- The OVBPI Database

  The alerts are also written to the OVBPI database. This allows the OVBPI Dashboard to report and show alerts raised. You can also access the alerts data tables directly and produce any customized alerting mechanism you desire.

  See the *OVBPI System Administration Guide* for further details about the OVBPI alert data tables.

## Instance and Statistical Thresholds/Alerts

As discussed in the section Threshold Type on page 29, some thresholds are measured on an “individual instance” basis and some are measured on a “collection interval” basis.

Consider setting an absolute threshold that measures when any specific metric instance duration is greater than four hours. The moment an individual metric instance has been running for more than four hours, an alert is issued. You can think of this threshold as an “instance threshold”, where alerts are issued based on each individual metric instance.

Now consider setting a backlog threshold that raises an alert when the backlog count is greater than 100. A backlog threshold applies to multiple metric instances and thus is calculated at the end of each collection interval (the collection interval of the underlying metric). So this threshold is not calculated for any individual metric instance, instead it is calculated for all metric instances over the last collection interval. You can think of this metric as a “statistical threshold”, where alerts are issued for the whole collection interval.

## Alert Levels

Within the Metric Definer you can choose to raise alerts at the levels Warning, Minor, Major and Critical.

If the alert is based on a statistical threshold, OVBPI additionally raises a Normal alert once the measurement has dropped back below the warning level. A statistical threshold is one that is calculated for each collection interval.

# Options

Within the Metric Definer there are a couple of options in the left-hand navigator frame.

## Refreshing the Metric Definer

If you have just deployed a new flow and you want the Metric Definer to pick up this, and any other, new definition, simply click `View -> Refresh` and the Metric Definer re-reads all deployed flow definitions.

## Exporting Metrics

If you have defined some metrics for a flow you can export them to an external (zip) file, as follows:

1. Select the flow - in the left-hand navigator frame
2. Select `File->Export`

   and save your metric definitions, thresholds and filters to an external file name of your choice.

If you have metrics defined for multiple flows, you can export all metric definitions, thresholds and filters to a single external (zip) file by selecting `File->Export All`

## Importing Metrics

If you have an external (zip) file that contains metric definitions, thresholds and filters as exported from a Metric Definer, you can import them into your Metric Definer, as follows:

1. Select `File->Import`
2. Browse and open the desired input file
3. Click the `Import Definitions` button
4. Select the set of metrics to import
5. Click the `Import Definitions` button
6. Click `OK`

If your import file contains custom metric definitions then you must have previously loaded in the necessary SQL stored procedures used by these custom metrics. If these stored procedures are not present at the time of the metric import, the import fails. (See Chapter 3, Custom Metrics for more details about how to create and work with custom metrics)

# Lab - Defining Metrics

In this lab you define a new flow to monitor the handling of support calls and then define a series of business metrics. You then configure thresholds against these metrics.

## The Scenario

You have been brought in to monitor a call center.

In this center, customers call up with a problem and it gets logged into the system. The customer may or may not have a support contract. If the customer has a support contract, the call is then looked at by a supervisor who assigns a priority and then assigns this call to an engineer within the team. If the customer does not have a contract then the supervisor still looks at the call and decides whether they want to take the time to answer the call. This allows the supervisor to answer calls for “potential” new customers. When a call is assigned to an engineer, they solve the problem, communicate this to the customer, and then mark the call as resolved. If a non-contract call is not assigned to an engineer, the call is simply marked closed and ignored.

This whole call system runs on a database with a front end call management application that allows the call takers, supervisors and engineers to log and update calls.

The call system is very simple. As a call comes in, it is written to the call database and the call state is set to either “Contract” or “NoContract”. As the call goes through the various stages, this state field is updated to say where it is in the process.

The call states are:

- “Contract” - this is a new call from a customer with a support contract
- “NoContract” - this is a new call from a customer with no support contract
- “Assigned” - the call is assigned to an engineer
- “Open” - the call is being worked on by the engineer
- “Resolved” - the call has been resolved
- “Closed” - the call is non-contract and to be ignored

The call center would like to be able to visually see their call system on a dashboard, showing where calls are at any time. They would also like to collect the following metrics:

- The time it takes to assign contract calls

  Raising an alert if the assignment time takes too long, and monitoring the backlog of calls waiting to be assigned over time.

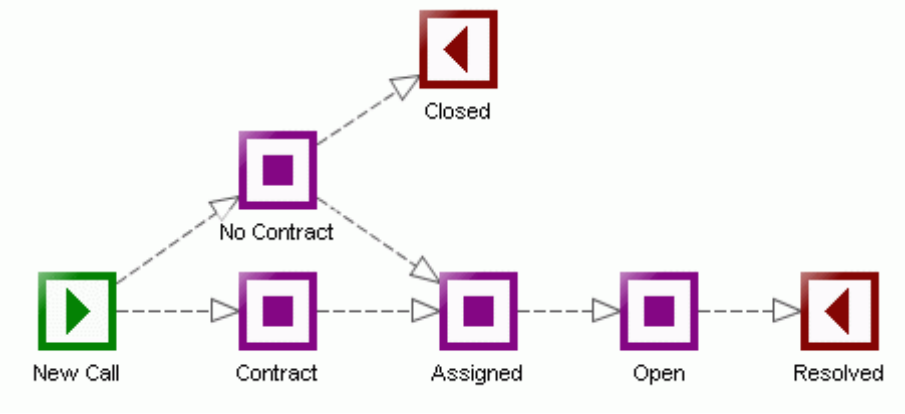
- The time it takes to process a call once it has been assigned

  Raising an alert if a call takes too long to be resolved, and an indication of how fast or slow the engineers are resolving calls compared to their average resolution rate.

## Defining the Flow

The flow definition for this lab is already defined for you. The flow diagram is as follows:

**Figure 6 Flow Diagram**




To load up this flow definition:

- Use the OVBPI Administration Console and start all Components
- Run the OVBPI Modeler
- Locate the file: `labs\CallSystem.zip`

  Import this file into the OVBPI Modeler.

- Deploy the flow: `Call System`

# Understanding the Flow

Now that the flow definition is loaded into the Modeler and deployed to the Business Impact Engine, let's take a few moments to understand how it works.

## Data Definition

The data definition, `Calls/Data`, has the following data attributes:

| Name | Type | Constraint | Unique |
|---|---|---|---|
| Call_ID | String | 30 | Yes |
| Customer_ID | String | 30 | |
| Call_Status | String | 20 | |
| Engineer_Name | String | 50 | |
| Call_Priority | String | 2 | |
| Call_Entry_Date | Date | | |

## Progression Rules

The progression rules are all based around the value of the `Call_Status` attribute.

When a new call comes in, the `Call_Status` attribute is set to either `Contract` or `NoContract`. The valid states then correspond to the names of the nodes.

## Events

There are three event definitions, as follows:

- `Calls/New` with properties:

  ```
  Call_ID
  Customer_ID
  Call_Status
  Call_Entry_Date
  ```

- `Calls/Assigned` with properties:

  ```
  Call_ID
  Engineer_Name
  Call_Priority
  ```

  The `Call_Status` property is set to `Assigned` by event subscription. For contract calls, the supervisor assigns a priority of either `1`, `2` or `3`; `1` being the highest priority. If the supervisor wants to assign and resolve a non-contract call, they assign a priority of `9`.

- `Calls/Update` with properties:

  ```
  Call_ID
  Call_Status
  ```

## Defining the Metrics

The metrics as specified by the customer are as follows:

- The time it takes to assign contract calls

  Raising an alert if the assignment time takes too long, and monitoring the backlog of calls waiting to be assigned over time.

- The time it takes to process a call once it has been assigned

  Raising an alert if a call takes too long to be resolved, and an indication of how fast or slow the engineers are resolving calls compared to their average resolution rate.

So this can be achieved by defining two metrics and then defining a set of thresholds for these metrics.

## Time to Assign Contract Calls

You need to define a metric that measures the time spent in the `Contract` node of the flow. That is, a metric from the start of `Contract` to the end of `Contract`.

To define this...

- Start up the Metric Definer
- Select the `Call System` flow
- Click on `Create Metric`
- Define the metric as follows:
  - Metric Name: `Call Assignment Time`
  - Metric Description: `Measure time to assign a contract call`
  - Metric Scope: `Single Node`
  - Metric Start Node: `Contract`
  - Metric Value Type: `Duration`
  - Statistics Collection: `On`
  - Collection Interval: `5 minutes`
  - Apply Filter: `None`
  - Group Results By: `None`
  - Deadline Property: `None`
- Click `OK`

You have now defined a metric that measures the time that contract calls spend waiting to be assigned.

In the real world you would probably choose the default collection interval or a larger collection interval, however, in this lab situation you are choosing a smaller collection interval just so you have quicker results within the lab time of the class.

### Raise an Alert if Duration Too Long

You now need to define a threshold for this metric to raise an alert if the time it takes to assign any individual call takes too long. In the real world the term "too long" might be four hours, or one day, but for this lab let's make that a lot smaller. Let's make the limit six minutes:

- Within the Metric Definer, select the `Call Assignment Time` metric
- Click `Create Threshold`
- Create a threshold as follows:
  - Threshold Name: `Call Assignment SLA`
  - Threshold Type: `Absolute duration`
  - Threshold Measure: `Any specific instance duration`
  - Duration Units: `Minutes`
  - Test Condition: `Greater than or equal to`
  - Critical Alert: `6`
- Click `OK`

### Call Assignment Backlog

The customer also wants you to monitor the backlog of calls waiting to be assigned:

- Select the `Call Assignment Time` metric
- Click `Create Threshold`
- Create a threshold as follows:
  - Threshold Name: `Call Assignment Backlog`
  - Threshold Type: `Backlog`
  - Threshold Measure: `Count`
  - Test Condition: `Greater than or equal to`
  - Major Alert: `15`
  - Critical Alert: `25`
- Click `OK`

## Time to Process Assigned Calls

To define a metric to measure the time taken to resolve a call once it has been assigned, seems straight forward. You just need to define a metric with a multiple node scope that measures from the start of the `Assigned` node to the end of the `Resolved` node. However, there are some additional considerations:

- The customer only wants to measure this for actual contract calls. So you would want a filter that filtered this metric such that only contract calls (calls with a priority of 1, 2 or 3) were included.
- When a support call is placed, the call comes in with a `Call_Entry_Date`. The SLA that the customer has is that all contract calls are handled within a certain time from the time the call is entered into the system. That is, the call resolution time has a deadline based on the `Call_Entry_Date`.
- You decide that the customer might like to be able to view the processed calls information grouped by call priority. This would enable them to view the day's calls and see not just the volume of calls, but a breakdown of calls by priority.

So, let's define the metric...

But first...you must define the filter:

- Select the `Call System` flow (in the left-hand navigator pane)
- Select the `Filters` tab
- Click `Create Filter`
- Set the filter name to be: `Call On Contract`
- In the `Data Definition` text box click on the `Call_Priority` attribute and then click on the `Paste` button

  This pastes the full name of this data attribute into the `Filter Expression` text box below.
- In the `Filter Expression` text box, complete the expression so that it ends up looking as follows:

  ```
  data.Call_Priority.in("1", "2", "3")
  ```

- Click `OK`

Now to define the metric:

- Select the `Call System` flow
- Click `Create Metric`
- Define the metric as follows:
  - Metric Name: `Call Processing Time`
  - Metric Scope: `Multiple nodes`
  - Metric Start Node: `Assigned (start)`
  - Metric End Node: `Resolved (complete)`
  - Metric Value Type: `Duration`
  - Statistics Collection: `On`
  - Collection Interval: `5 minutes`
  - Apply Filter: `Call On Contract`
  - Group Results By: `Call_Priority`
  - Group Name: `Call Priority`
  - Deadline Property: `Call_Entry_Date`
- Click `OK`

Again, because this is a lab scenario and you want to see results more quickly, you are configuring a small collection interval.

Now to define the thresholds...

### Raise an Alert if Deadline Not Met

The support agreement (SLA) is that contract calls must be resolved within a certain time from when they were entered. Again, being a lab scenario, let's make this time short so you can test it out without having to wait too long.

Let's configure a threshold to issue a critical alert if the time to process a contract call is 35 minutes from the time the call was entered. Let's also issue an additional minor alert if the call reaches 20 minutes from the time the call was entered:

- Select the `Call Processing Time` metric
- Click `Create Threshold`
- Create a threshold as follows:
  - Threshold Name: `Call Time SLA`
  - Threshold Type: `Deadline`
  - Minor Alert: `20 Minutes After`
  - Critical Alert: `35 Minutes After`
- Click `OK`

### Call Processing Speed

The customer also requested to monitor how fast or slow the engineers are resolving calls compared to their average resolution rate:

- Select the `Call Processing Time` metric
- Click `Create Threshold`
- Create a threshold as follows:
  - Threshold Name: `Call Processing Speed`
  - Threshold Type: `Relative duration`
  - Threshold Measure: `Recent Average duration`
  - Test Condition: `Less than or greater than usual`
  - Warning Alert: `1.0`
  - Minor Alert: `1.5`
  - Major Alert: `2.0`
  - Critical Alert: `2.5`
- Click `OK`

Great! You should have the following metrics and thresholds defined for your `Call System` flow:

```
Metric: Call Assignment Time          (Single node, duration)

  Threshold: Call Assignment Backlog  (Backlog)
  Threshold: Call Assignment SLA      (Absolute)

Metric: Call Processing Time          (Multiple node, duration)

  Threshold: Call Processing Speed    (Relative)
  Threshold: Call Time SLA            (Deadline)
```

Now that these metrics are defined within the Metric Definer, they are active. Indeed, your metric collection intervals may have already kicked into action and data may have already started to collect in the OVBPI database. Of course, the statistics at the moment simply record lots of zeros because there are no calls moving through your system. So how are you going to get loads of calls into the system so that you can track and measure their performance? The answer is the **Flow Simulator**...

## The Flow Simulator

The Flow Simulator is a **contributed utility** that allows you to inject events into OVBPI. The big benefit of the Flow Simulator is that you can store these events as `Test Cases`, and then run these test cases as and when you need. The Flow Simulator can drive any flow, and has some pre-defined tokens that allows you to substitute special values such as unique IDs and date/times.

The Flow Simulator is located on the OVBPI product CD, under the `contrib` directory.

To install the Flow Simulator, you just need to copy the entire `contrib` directory (and all its contents) to your OVBPI installation directory. This way you end up with the directory:

*OVBPI-install-dir*\contrib\FlowSimulator

To run the Flow Simulator, you just need to double-click the `FlowSimulator.bat` file in the *OVBPI-install-dir*\contrib\FlowSimulator directory

If you want to learn more about how to use the Flow Simulator, please refer to the documentation provided in the `contrib\FlowSimulator` directory.

## Running the Call Center

For this lab, you are provided with a pre-defined set of test cases to run within the Flow Simulator. These test cases inject contract and non-contract calls through the `Call System` flow, allowing you to run the OVBPI Dashboard and monitor the calls and the metrics.

Let's load up the Flow Simulator:

- Make sure that the Flow Simulator is installed and that you have the directory: *OVBPI-install-dir*\contrib\FlowSimulator
- Run the script: `FlowSimulator.bat`

  This runs the Flow Simulator in a separate GUI

- In the Flow Simulator GUI, select: `File->Open Test Suite`
- Open the file: `labs\Calls.xml`

  The Flow Simulator should load this file and you should now see a number of test cases shown in the top-left-hand corner of the Flow Simulator window.

## Test Cases

The test cases have names that describe what they do. Let's go through them:

- NoContract-Resolved

  This set of events send through a call that is from a non-contract customer. The call is assigned to an engineer and resolved.

- NoContract-Closed

  This set of events send through a call that is from a non-contract customer. The call is not-assigned to an engineer and simply closed.

- Contract-Pri-1

  This set of events send through a call that is from a contract customer. The call is assigned to an engineer with a priority of "1", and resolved.

- Contract-Pri-2

  This set of events send through a call that is from a contract customer. The call is assigned to an engineer with a priority of "2", and resolved.

- Contract-Pri-3

  This set of events send through a call that is from a contract customer. The call is assigned to an engineer with a priority of "3", and resolved.

## Running the Suite of Tests

Now that you have loaded the Flow Simulator with a set of test cases, let's run them:

- Within the Flow Simulator GUI, click on the `Test Suite Runner` tab

  This displays the individual test cases with `Start` and `Stop` buttons for each. It also displays a slider control to adjust how often each test case is run.

  Each test case is made up of a set of events. Below the `Event Injectors` heading you see a start/stop control and a slider control for each event.

  **All the sliders are pre-set** to initial values appropriate for this lab.

- Click the `Start Suite` button at the bottom of the screen and this starts the test suite.

  The sliders are pre-set such that calls take a while to be assigned. You should start to notice a build-up of calls waiting to be assigned.

## Running the OVBPI Dashboard

- With the Flow Simulator still running, start up the OVBPI Dashboard and drill into the `Call System` flow.

  You should start to see a gradual build up of calls in the `Contract` node.

  Notice that as well as seeing the flow diagram for the `Call System` flow you also see dials and tables representing the four thresholds that you defined. You may not see the dials until the first collection intervals have occurred. Remember that statistical thresholds are only updated after each collection interval.

  The OVBPI Dashboard displays the following graphics for defined thresholds:

  — A dial for each statistical thresholds (excluding `Relative` thresholds)

  — An upside-down dial for each `Relative` statistical threshold

  — A table of instance alerts for each instance threshold

  These graphics help you see the overall state of your business at a glance.

- Let the Flow Simulator run for five minute...
- After the first collection interval is complete you should see the dials for your metrics.

## Call Processing Speed Dial

The `Call Processing Speed` dial is shown as an upside-down dial, called a "swing dial", because it is showing the relative movement towards one side or the other. That is, the needle represents the most recent average time taken to process calls. Remember that the most recent average is the average over the most recent collection interval, which is five minutes in this case. The needle points straight down (to the center of the green area) if the most recent average is the same as the overall average (the average since the metric was first defined). If the most recent average is faster than the overall average then the needle swings to the right. If the most recent average is slower than the overall average, then the needle swings back towards the left side.

So the swing dial is used to represent relative thresholds, and the needle position indicates the relative position towards "faster than the average" or "slower than the average".

The value shown under the swing dial is the most recent average value. If this value is greater than the overall average than the needle swings to the right, other wise the needle swings to the left.

The swing dial allows you to visually see whether your call processing time is getting slower (a swing to the right) or getting faster (a swing to the left).

## More Graphs

- Click on the `Call Assignment Backlog` dial and/or the `Call Processing Speed` dial to see historical graphing of the values

  This may not be very exciting until your Call Center events have been running through the system for a few collection intervals.

You can also see a number of different types of graphs for the underlying metrics:

- In the OVBPI Dashboard, on the main `Business Flow & Resource Summary` page - the page that shows the flow diagram and the threshold dials - click on the tab marked: `Metrics`
- Click on the metric called: `Call Processing Time`

  By default this shows you a graph displaying the Average/Minimum/ Maximum values for each collection interval.
- Select the `Data Source` for the graph to be: `Completed - Count`

  This shows you the number of calls that have been resolved within each collection interval.
- Now select `Chart by group` to be: `Yes`

  You now see the number of calls that have been closed within each collection interval, but now showing the numbers for each call priority.

Many of these graphs may not show much data as you have only just started collecting metrics. At the end of this lab you should leave the Flow Simulator running and thus, collect more data that you can drill into during the next lab.

## Adjusting the Flow Simulator

- Once the OVBPI Dashboard is showing alerts for the `Call Assignment SLA` threshold you need to go back to the Flow Simulator and speed up the time it takes for calls to be assigned.
- In the Flow Simulator, move the slider for the event `Calls/Assigned` to the left. You should move the slider until the number to the right of the slider is showing (about) 5000ms.

  This tells the Flow Simulator to handle these events at one event every 5000 milliseconds (five seconds) and thus the calls currently in the `Contract` and `No Contact` nodes are processed more quickly.

  Back in the OVBPI Dashboard, you should notice that calls waiting in the `Contract` node start to reduce.
- After the next collection interval, notice how the dials indicate that there is less of a call assignment backlog and that orders are starting to take a little longer to be processed (the `Call Processing Speed` dial swings to the right).

## End of The Lab

- At this point, you have finished the lab! However, you should leave the Flow Simulator running
- During the rest of the class, make periodic adjustments to the Flow Simulator sliders to simulate changes in the speed at which calls are being handled.

  As you make any changes, leave them in place for about 10 minutes before changing them again.

  For example:

  — Move the `Calls/Update` slider down (left) to a very small time value

  This simulates calls being resolved very quickly.

  — Move the `Calls/Update` slider up (right) to a large value

  This simulates calls being resolved very slowly.

  — Move the `Contract-Pri-1` slider all the way to the right

  This simulates few priority-1 calls coming into the system
- Hopefully when you have breaks during the lectures you can view the metric data in the OVBPI Dashboard and see the affects of your changes.

**Well done! You have reached the end of the lab.**

2

# Metric Engine

The central Business Impact Engine processes all incoming events and progresses all flow instances. Rather than increase the workload of the Business Impact Engine by asking it to also calculate all business metrics, it was decided to create a separate OVBPI component to handle metrics. This component is called the **Metric Engine**.

This chapter looks at the Metric Engine and the SQL data tables that are used to hold metric data.

# How Metrics Work

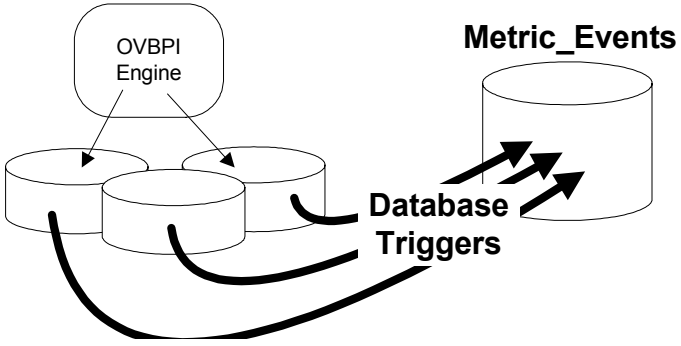
## Metric Events

As flow and node instances are updated, the central Business Impact Engine updates the tables within the OVBPI database. By placing triggers on some of the key data tables, as a flow instance enters a node that starts a metric, a new metric record is generated. The same is true when a flow instance triggers the fact that a metric has completed.

These metric records are written into the data table `Metric_Events`, within the OVBPI database, as and when they occur.

The creation of metric records within the `Metric_Events` table can be shown as follows:

**Figure 7 Creation of Metric Records**




where:

- The Business Impact Engine updates the database tables as normal
- SQL triggers on the various data tables capture the details whenever a metric starts or completes
- The necessary metric data is written to the `Metric_Events` data table

There are four types of metric record written to the `Metric_Events` table:

- `Started`

  Records the starting of a metric instance.

- `Completed`

  Records the completion of a metric instance

- `EndFlow`

  Records the end of a flow instance. If a metric instance is still active, even though its underlying flow instance has now completed, the Metric Engine marks the metric instance as `Aborted`.

- `NewWeight`

  Records the weight value for a metric instance. This record is written when the weight value changes during the life of a metric instance.

Each record written to the `Metric_Events` table contains all the information needed for the recording of each metric.

The `Metric_Events` table is the input for the OVBPI Metric Engine.

## Metric Values

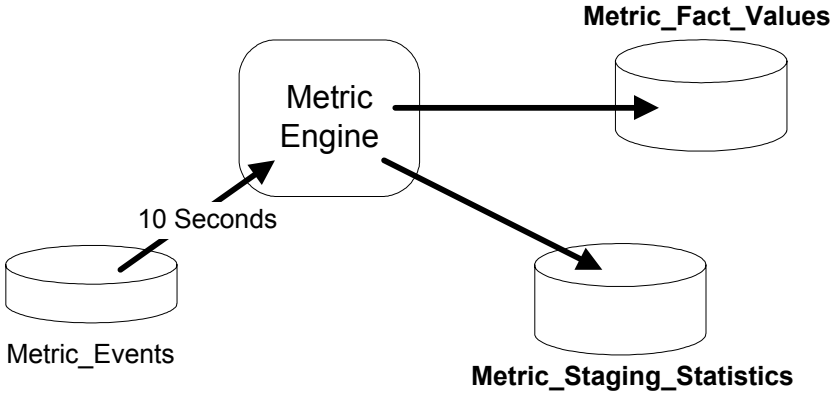
Once the metric record is written into the `Metric_Events` table it is up to the Metric Engine to process this record.

The first stage of the process is to record all the individual metric values. This is where it matches up the start and complete records from the `Metric_Events` table and record the actual duration and/or weight for each metric instance.

As well as producing the values for each metric, the Metric Engine also builds a "work area" known as the `Metric_Staging_Statistics` data table. This table is used as a staging area, or work area, for the future calculation of the metric statistics after each collection interval.

Let's draw the picture and then explain this in more details...

**Figure 8 Metric Values**




where:

- The Metric Engine polls the `Metric_Events` table every 10 seconds.

  This poll period is called the `Metric Event Polling Interval` and is configurable through the OVBPI Administration Console. The default setting is 10 seconds.

- As metric records are read from the `Metric_Events` table, the Metric Engine creates or updates records in the `Metric_Fact_Values` table.

  The `Metric_Fact_Values` table holds one record for each metric instance.

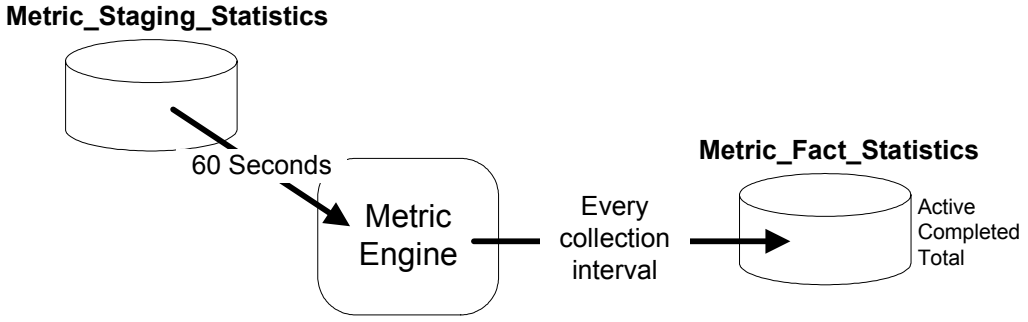
- As well as written records to the `Metric_Fact_Values` table, the Metric Engine needs to keep track of what's been happening over time. So the Metric Engine uses the data table `Metric_Staging_Statistics` to hold intermediate data that enables the calculation of statistics when each metric collection interval occurs.

  You do not really need to know (or worry) about the `Metric_Staging_Statistics` data table as it is just a temporary work area for the Metric Engine. It is just mentioned here because it plays a part in the production of metric statistics.

## Metric Statistics

Metric statistics are produced at the end of each collection interval. These statistics are generated from the data collected within the `Metric_Staging_Statistics` data table. Let's draw the picture and then explain this in more details...

**Figure 9 Metric Statistics**




where:

- The Metric Engine is polling the `Metric_Staging_Statistics` table every 60 seconds.

  This poll period is called the `Statistical Generation Polling Interval` and is configurable through the OVBPI Administration Console. The default setting is 60 seconds.

- If a collection interval has occurred, the Metric Engine writes three summary records into the `Metric_Fact_Statistics` table.

  The Metric Engine produces three records to summarize that collection interval:

  – `Active`

    The records record details for the metric instances that are still active.

  – `Completed`

    The metric instances that have completed within that collection interval

  – `Total`

    The overall total for all metric instances completed over all collection intervals so far.

So the Metric_Fact_Statistics table is probably the table that can grow the fastest. Every collection interval three records are being written to this table for every metric defined on your system. These three records are written every collection interval even if nothing else has happened on your system.

## Group Results By

If you have configured a metric to have a Group Results By property, then the Metric Engine writes more than three summary records at each collection interval.

When a metric is defined to have a Group Results By property, the Metric Engine writes out the following records at the end of each collection interval:

- Three records (Active/Completed/Total) for each current group within the metric.
- Three records (Active/Completed/Total) for the metric overall.

For example, suppose you are collecting metrics for an airport and grouping the metric data by each airport terminal. If there are four airport terminals, then at each collection interval the Metric Engine is writing 15 records to the Metric_Fact_Statistics table. These 15 records consist of three records for each terminal, and three records for the overall statistics.

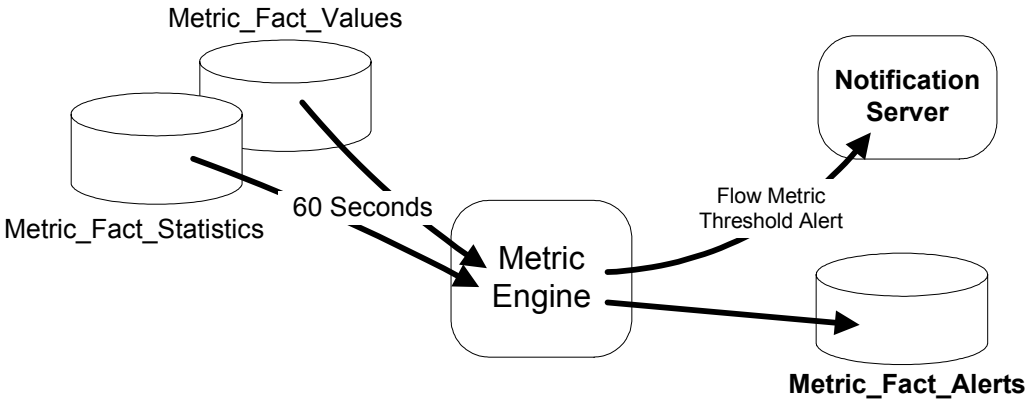
## Metric Alerts

With metric values being recorded in the `Metric_Fact_Values` table as they happen, and metric statistics being calculated and stored in the `Metric_Fact_Statistics` table every collection interval, the Metric Engine can monitor these two tables to see when any thresholds have been exceeded.

As a threshold is exceeded, an alert can be raised.

The handling of alerts looks as follows:

**Figure 10 Metric Alerts**




where:

- The Metric Engine polls the `Metric_Fact_Values` and the `Metric_Fact_Statistics` tables every 60 seconds looking to see if any thresholds have been exceeded.

  This poll period is called the `Threshold Polling Interval` and is configurable through the OVBPI Administration Console. The default setting is 60 seconds.

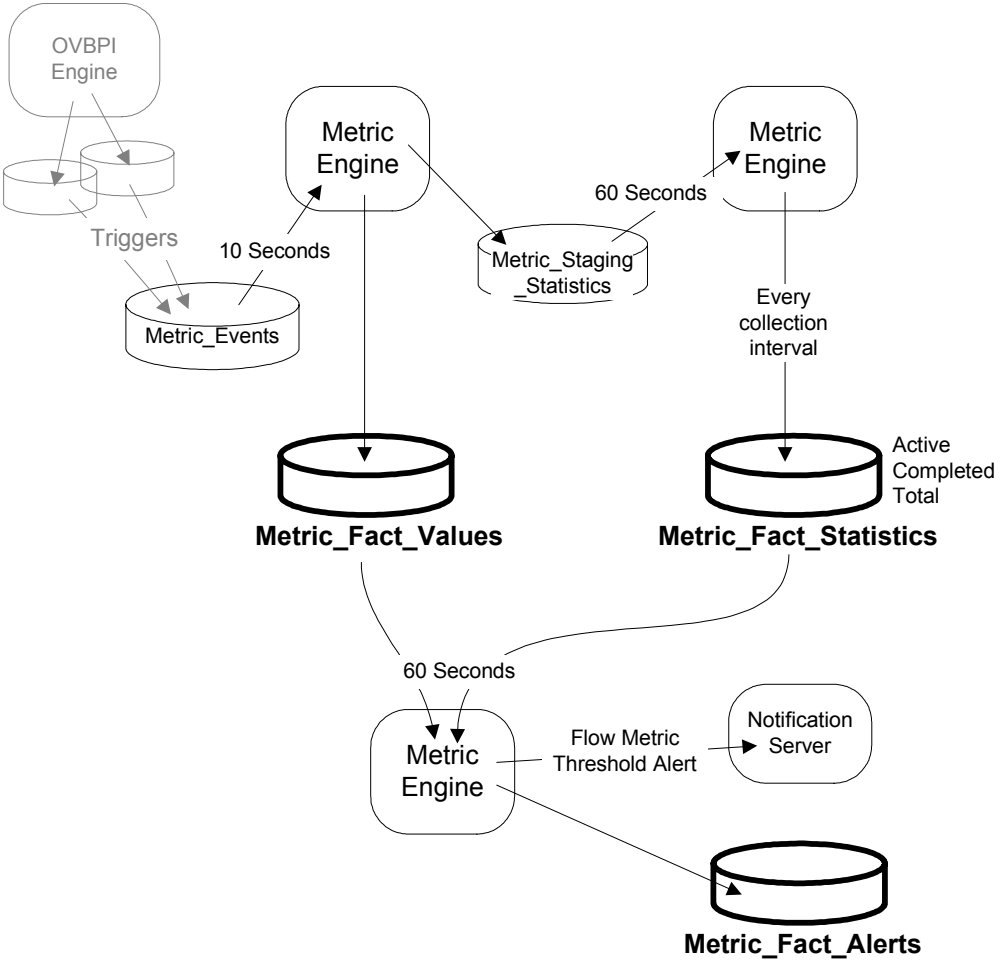
  Instance thresholds are measured against the metric values held in the `Metric_Fact_Values` table. Statistical thresholds are measured against the statistics collected in the `Metric_Fact_Statistics` table.

- If a threshold has been exceeded the Metric Engine first sends a `Flow Metric Threshold Alert` to the OVBPI Notification Server for processing. The Metric Engine then writes the alert into the `Metric_Fact_Alerts` data table.

## The Big Picture

Putting all the diagrams together that describe how metric values, statistics and alerts are processed, gives you the following overview diagram:

**Figure 11 Metrics Overview**



# Alert Timings

When you have configured a threshold you may ask yourself the question: “Exactly when is the alert going to be raised?”

If you look at Figure 11 on page 63 you see that there are some small delays when the Metric Engine is polling the various data tables. This can mean that the alerts may be raised a few seconds/minutes after the threshold has actually been exceeded. And of course, if the threshold is based on a statistical metric then there is also the collection interval to be considered.

Don’t forget that the metric polling periods can all be configured using the OVBPI Administration Console.

Let’s look at the maximum delays that may occur when raising alerts, and using the default metric polling periods.

## Instance Thresholds

Let’s consider thresholds that are set against instances. For example, a threshold that tests an instance taking longer than four hours to get to a certain node within a flow.

There is up to a 10 second delay before metric values are written to the `Metric_Fact_Values` table.

There is up to a 60 second delay before the `Metric_Fact_Values` table is polled to look for instance threshold violations.

So there is up to a 70 second delay before an alert is raised when an instance exceeds a threshold.

The actual delay is less than or equal to 70 seconds because it depends when the violation occurs. For example, the violation might be written to the `Metric_Fact_Values` table one second before the next poll of that table, so the total delay in spotting the violation might be only a few seconds.

When an alert is written to the `Metric_Fact_Alerts` data table, it records both the time the alert occurred and the time the alert was actually spotted by the Metric Engine.

## Statistical Thresholds

When the threshold is set against a statistical value (such as an average value, or a backlog), the potential delays are as follows:

There is up to a 10 second delay before metric values are written to the `Metric_Fact_Values` table.

There is the actual collection interval. If the collection interval is set to 30 minutes then the statistical value is not calculated until that 30 minutes has passed.

There is up to a 60 second delay before the `Metric_Staging_Statistics` table is polled to see if a collection interval has occurred. Then the statistical results are written to the `Metric_Fact_Statistics` table

There is up to a 60 second delay before the `Metric_Fact_Statistics` table is polled to look for instance threshold violations.

So the total, potential delay in raising an alert for a statistical threshold is:

```
10 seconds + 60 seconds + 60 seconds + Collection Interval
```

When an alert is written to the `Metric_Fact_Alerts` data table, it records both the time the alert occurred and the time the alert was actually spotted by the Metric Engine.

# Metric Engine Off/Restart

If you look at Figure 11 on page 63 you realize that you can turn off the Metric Engine and not lose any metric data. With the Metric Engine turned off, the metric events simply accumulate in the `Metric_Events` table.

When the Metric Engine is restarted, after being off for a period of time, the `Metric_Events` table is processed and the metric values are recorded in to the `Metric_Fact_Values` table in the normal way, and the metric statistics recorded in to the `Metric_Fact_Statistics` table in the normal way.

## Alerts

Whilst the Metric Engine is turned off, no alerts are going to be raised. However, when the Metric Engine is restarted the metric data in the `Metric_Events` table is processed and alerts may well be raised. The alerts are raised showing both the time the alert actually occurred on your system and the time the Metric Engine actually spotted the alert.

## Statistical Metrics (“Back Filling”)

When the Metric Engine is restarted, after being off for a period of time, it must calculate the statistics for each metric for each collection interval for all the time the Metric Engine was switched off.

That is, if you have a metric with a collection interval set to (for example) five minutes and you have turned the Metric Engine off for four hours, when you restart the Metric Engine it goes through and calculates statistic records for all the five minute boundaries across the last four hours and writes these into the `Metric_Fact_Statistics` table. In other words, the Metric Engine “back fills” the statistics so that they appear as if the Metric Engine has never been turned off.

This “back filling” of statistical data is fine if the Metric Engine has just been off for a few minutes or hours. In these situations you want the Metric Engine to keep all the metric statistics up to date. But what if you have turned off the Metric Engine for a week? Do you really want the Metric Engine to go through and “back fill” all the statistics for the last week? Possibly not.

The Metric Engine has a user configurable setting that tells the Metric Engine how far back to calculate statistics after a restart. This setting is called:

```
Maximum age of generated statistics on startup
```

and it is measured in `days`.

The default setting is one day.

So, by default, if you turn the Metric Engine off for anything less than a day the Metric Engine calculates all the statistics for the time it was turned off. But if you turn the Metric Engine off for longer than a day, statistics are only calculated back for one day's worth.

# Instance Cleaner Settings

## Metric Instance Cleaner

In the OVBPI Administration Console you can configure the Metric Instance Cleaner to remove metric values (Active and Completed), metric statistics and metric alerts. You can configure to remove some or all of these record types and you can configure the age of the data to be removed.

Metric instances and statistics can grow quite large, so you need to make sure that your OVBPI database, and your server's disc space, is large enough for the amount of metric data you need to have available. By configuring the Metric Instance Cleaner you can help prevent the metric data from growing to large.

## Business Impact Engine Instance Cleaner

The Business Impact Engine has its own Instance Cleaner which can be configured to remove old flow and data instances from the OVBPI database.

If the Business Impact Engine Instance Cleaner removes a flow instance for which there is metric data, that is ok. When the metric record is written to the `Metric_Events` data table, it contains all the data required to record the metric. The Metric Engine does not require any other flow instance or data instance data to calculate the metric values or statistics. All the data required is in the `Metric_Events` record.

Mind you, within the OVBPI Dashboard, when displaying metric details, if a metric links to a flow instance, the Dashboard wants to show this metric data with a live link to the related flow instance data. If at the time of display the related flow instance has been removed by the Business Impact Engine Instance Cleaner, the Dashboard simply shows the metric data without an active link to the flow.

# Metric Database Schema (Star)

As shown in Figure 11 on page 63, the database tables that contains metrics and alerts are:

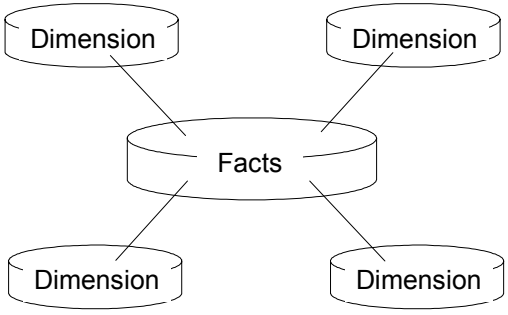
```
Metric_Fact_Values
Metric_Fact_Statistics
Metric_Fact_Alerts
```

but these contain attributes such as `Metric_ID` and `Date_ID`. How do you cross reference these IDs to their actual names?

The Metric Engine maintains its data in a set of “fact” and “dimension” data tables. The idea is that the actual metric data (values, statistics and alerts) is referred to as the “facts” and the indexes into these facts are referred to as “dimensions”. These terms form something known as a “Star Schema”.

A star schema gets its name from the shape it forms when drawn. It is said to “look like the shape of a star”. For example:

**Figure 12 A Star Schema**



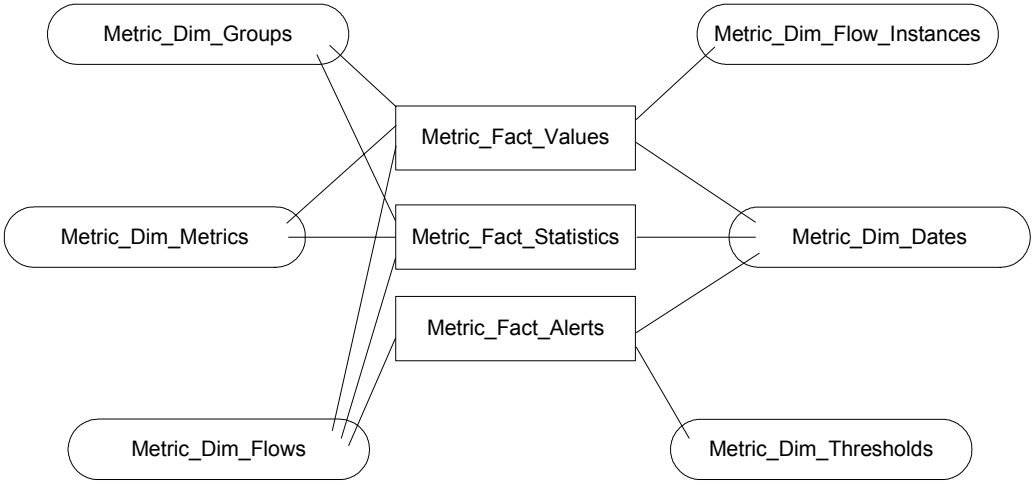

where:

- You have a central facts table
- You have a dimension for each index that you would like to be able to search by.

  If you were to add a new index into your fact table you would be said to have “Added a new dimension to the search ability.” Hence the indexes are called dimensions.
- The dimensions tables can hold additional data for each index.

Some of the main tables in the Metric Engine's star schema are as follows:

**Figure 13 Metric Engine Star Schema**




All these tables are fully described in the *OVBPI System Administration Guide - Appendix A*.

You are encouraged to write custom reports of your own against these metric tables and/or use third party database reporting tools to produce whatever reports you require for your business.

3

# Custom Metrics

What if the OVBPI metrics do not give you the exact metric that you wish to measure? Maybe you need to measure the value of your orders however the value you need is not held in the weight property? Maybe you need to measure the percentage of orders that have gone through a particular node?

If the metric you need to record is not available using the standard out-of-the-box metrics, you can add your own.

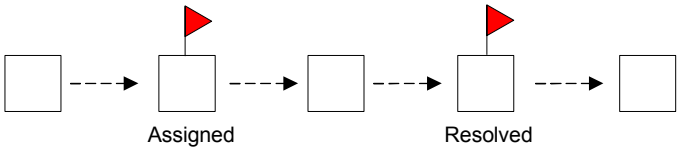
This chapter looks at how to add and use your own custom metrics.

# Metric Scope

The scope of the metric determines when the start of each metric instance is recorded and when the end of the metric instance is recorded.

For example, suppose you configure a metric, and set the metric scope to be from the start of the node `Assigned` to the end of the node `Resolved`:

**Figure 14 Metric Scope**




When a flow instance enters the `Assigned` node, a new metric instance is started. When this flow instance leaves the `Resolved` node, the metric instance is completed.

In How Metrics Work on page 56 you learned that the start and end of the metric instance is captured by triggers on the Business Impact Engine tables. These triggers capture the starting metric value and write this to the `Metric_Events` table. The triggers also capture the final metric value at completion and write this to the `Metric_Events` table.

The triggers that capture the start and end of a metric instance, call predefined SQL stored procedures to capture the metric value. You are able to provide your own custom stored procedures and have these called when a metric starts and completes. Thus you can write a custom stored procedure to capture whatever value you are required to measure.

# Defining A Custom Metric

There are two main parts to defining a custom metric:

1. The stored procedure

   Writing the stored procedure to determine the actual metric value.

2. The Metric Definer

   Making this stored procedure available within the Metric Definer such that users can define metrics that use this stored procedure.

## The Stored Procedure

For your stored procedure to be called correctly it must accept the following parameters:

```
Metric Id
Flow Id
Flow Instance Id
Flow Instance Identifier
Flow Instance start time
Weight
Data Definition ID
Data Instance ID
Event type
Event time
Index
```

### Parameter Values

Let's consider the values to be contained in these input parameters:

- `Metric Id`

  This is the database unique ID for the metric.

- `Flow Id`
  `Flow Instance Id`
  `Flow Instance Identifier`
  `Flow Instance start time`
  `Weight`

  These are the IDs and values for the flow instance for which this metric value is to be determined.

- `Data Definition ID`

  This is the unique ID for the flow's related data table entry in the `Data_Objects` table.

- `Data Instance ID`

  This is the unique ID of this flow's related data instance entry within the related data table.

- `Event type`

  This is a string attribute that contains one of two values:

  — `Started`

  This is the start of a new metric instance.

  — `Completed`

  This is the completion of the metric instance.

- `Event time`

  The time that this metric is being recorded.

- `Index`

  It may be that a flow instance passes through the node that starts the metric, more than once. You can use the index value to code for this.

  Zero (`0`) means that it is the first time through this node. One (`1`) means this is the second time through this node, etc.

## SQL Parameter Syntax

As the SQL syntax for MSSQL and Oracle is different when defining stored procedures, here are two examples for how to define the parameters:

For MSSQL:

```
@Metric_ID           NVARCHAR(36),
@Flow_ID             NVARCHAR(36),
@FlowInstance_ID     NVARCHAR(36),
@FlowInstIdentifier  NVARCHAR(40),
@FlowInstStartTime   DATETIME,
@Weight              FLOAT,
@DataDefinition_ID   NVARCHAR(36),
@DataInstance_ID     NVARCHAR(36),
@EventType           NVARCHAR(12),
@EventTime           DATETIME,
@Idx                 INTEGER
```

For Oracle:

```
Metric_ID           VARCHAR2,
Flow_ID             VARCHAR2,
FlowInstance_ID     VARCHAR2,
FlowInstIdentifier  VARCHAR2,
FlowInstStartTime   DATE,
Weight              FLOAT,
DataDefinition_ID   VARCHAR2,
DataInstance_ID     VARCHAR2,
EventType           VARCHAR2,
EventTime           DATE,
Idx                 NUMBER
```

## The Metric Value

Once your stored procedure has determined the metric value, it needs to write this value into the `Metric_Events` data table. To do this, there is a pre-defined stored procedure called `METRIC_SEND_EVENT`.

The `METRIC_SEND_EVENT` stored procedure accepts the following parameters:

```
Metric Id
Flow Id
Flow Instance Id
Flow Instance Identifier
Flow Instance start time
Weight
Data Definition ID
Data Instance ID
Event type
Event time
Metric Value
Index
```

These are the same parameters as were passed into your stored procedure, with the addition of the `Metric Value` parameter.

As the SQL syntax for MSSQL and Oracle is different when calling stored procedures, here are two examples for how to define the call to the `METRIC_SEND_EVENT` stored procedure:

`For MSSQL:`

```
EXECUTE METRIC_SEND_EVENT @Metric_ID,
                          @Flow_ID,
                          @FlowInstance_ID,
                          @FlowInstIdentifier,
                          @FlowInstStartTime,
                          @Weight,
                          @DataDefinition_ID,
                          @DataInstance_ID,
                          @EventType,
                          @EventTime,
                          @Value,
                          @Idx
```

```
For Oracle:

    METRIC_SEND_EVENT (Metric_ID,
                       Flow_ID,
                       FlowInstance_ID,
                       FlowInstIdentifier,
                       FlowInstStartTime,
                       Weight,
                       DataDefinition_ID,
                       DataInstance_ID,
                       EventType,
                       EventTime,
                       Value,
                       Idx);
```

## Metric Backlog

Your stored procedure is called both at the start of the metric and the completion of the metric. Indeed, your stored procedure is able to test whether it is being called for the start or completion of a metric by testing the `EventType` parameter.

Suppose the custom metric that you are measuring always has a zero start value. In this case you might decide that you do not need to write any metric record at the start of the metric, and only write a metric record at the completion of the metric. That is, only calculate the metric value and call `METRIC_SEND_EVENT` if the `EventType` is `Completed`.

Only writing completing metric records cuts down on some metric processing time and works just fine. However, if you choose to not write any metric record at the start of the metric, you are not able to observe and graph the backlog for your metric. Without any start-of-metric records there is no way to show how many metric instances are currently active and hence no way to represent the metric backlog. So if you want to be able to monitor your custom metrics, including the backlog, then call `METRIC_SEND_EVENT` for all invocations of your stored procedure.

# The Metric Definer

Once you have defined and installed your stored procedure to capture and record your metric values, you need to make this available to the Metric Definer so that users can associate this stored procedure with a metric definition.

The Metric Definer looks in the table `METRIC_CustomTypes` to locate any custom metric types that are defined.

## Defining a Custom Type

You need to add a record to the `METRIC_CustomTypes` table to describe your new custom metric and connect it to your stored procedure.

The `METRIC_CustomTypes` table has the following columns:

- `CustomMetricName`

  You specify the name for your custom metric. This name appears in the `Metric Value Type` attribute in the Metric Definer when defining a metric. The user are able to select this metric from the pull-down list.

- `CustomMetricDescription`

  You provide a description for your custom metric. This is purely for your own documentation purposes.

- `CustomSPName`

  This is where you specify the name of your custom stored procedure.

- `ValueUnits`

  You can specify the display name to be used within the OVBPI Dashboard when displaying the values of this custom metric.

Here is an example SQL command to create a new metric custom type definition:

```
INSERT INTO METRIC_CustomTypes
     (CustomMetricName,
      CustomMetricDescription,
      CustomSPName,
      ValueUnits)
   VALUES
     ('Calls Resolved ON Contract',
      'Calculates percentage calls resolved ON contract.',
      'BPI_Contract_Resolved_Calls',
      'Percent');
```

where:

- The new metric type is called `Calls Resolved ON Contract`
- The associated stored procedure that writes out the metric values is called `BPI_Contract_Resolved_Calls`
- The display text for the measurement units for this metric type is `Percent`

# Example - A Data Property

Suppose the metric you wish to record is contained in a data property but this data property is not the weight of the flow. You can set up a custom metric that simply pulls out the data property and writes that as the metric value instead of the weight.

Suppose your data definition contains the property `Discount`, and that is the property you wish to measure as your metric.

Your stored procedure needs to locate the value for this property and return its value. The stored procedure is passed a number of parameters, and these include:

- `Data Definition ID`

  This is the unique ID for the flow's related data table entry in the `Data_Objects` table.

- `Data Instance ID`

  This is the unique ID of this flow's related data instance entry within the related data table.

You can use the `Data Definition ID` to get the name of the related data table, and then use the `Data Instance Id` to look up the actual data row within this related data table.

## The Stored Procedure

Let's look at an example stored procedure (for MSSQL):

```
CREATE PROCEDURE BPI_Discount_Value
    @Metric_ID NVARCHAR(36), @Flow_ID NVARCHAR(36),
    @FlowInstance_ID NVARCHAR(36), @FlowInstIdentifier NVARCHAR(40),
    @FlowInstStartTime DATETIME, @Weight FLOAT,
    @DataDefinition_ID NVARCHAR(36), @DataInstance_ID NVARCHAR(36),
    @EventType NVARCHAR(12), @EventTime DATETIME, @Idx INTEGER
AS
  DECLARE @DiscountValue FLOAT
  DECLARE @DataTableName NVARCHAR(128)
  DECLARE @DataSqlStmt   NVARCHAR(256)
BEGIN

  -- (1) Use the DataDefinition_ID to get the name of the data table.

  select @DataTableName = InstanceTable
     from Data_Objects do
     where do.model_id = @DataDefinition_ID;

  -- (2) Now use DataInstance_ID to look up the actual data record
  --     and pull out the discount column value.

  CREATE TABLE Temp_BPI_CustomTable (tValue FLOAT)
  SET @DataSqlStmt = 'insert into Temp_BPI_CustomTable(tValue) ' +
                     ' select Discount from ' + @DataTableName +
                     ' where id = ''' + @DataInstance_ID + ''''
  EXECUTE (@DataSqlStmt)
  SELECT @DiscountValue = tValue from Temp_BPI_CustomTable
  DROP TABLE Temp_BPI_CustomTable

  -- (3) Now write the metric value

  EXECUTE METRIC_SEND_EVENT @Metric_ID, @Flow_ID, @FlowInstance_ID,
      @FlowInstIdentifier, @FlowInstStartTime, @Weight, @DataDefinition_ID,
      @DataInstance_ID, @EventType, @EventTime, @DiscountValue, @Idx

END;
```

where:

- This defines a stored procedure called `BPI_Discount_Value`
- Part `(1)` shows the SQL to determine the name of the related data table. This is the data table that holds the flow instance's related data record.

- Part **(2)** shows how to use this related data table name to locate the data record and pull out the discount attribute value.

  The SQL you logically want to use at this point is:

  ```
  select @DiscountValue = Discount
      from @DataTableName dot
      where dot.id = @DataInstance_ID;
  ```

  However, MSSQL does not let you issue a select statement in this form where the table name is a variable. Hence you need to create a temporary table and use the `Execute()` command to execute the select statement.

- Part **(3)** shows the writing of the metric value to the `Metric_Events` table.

## The Custom Metric Type Definition

Once the stored procedure has been installed into your database you need to tell the Metric Definer about it.

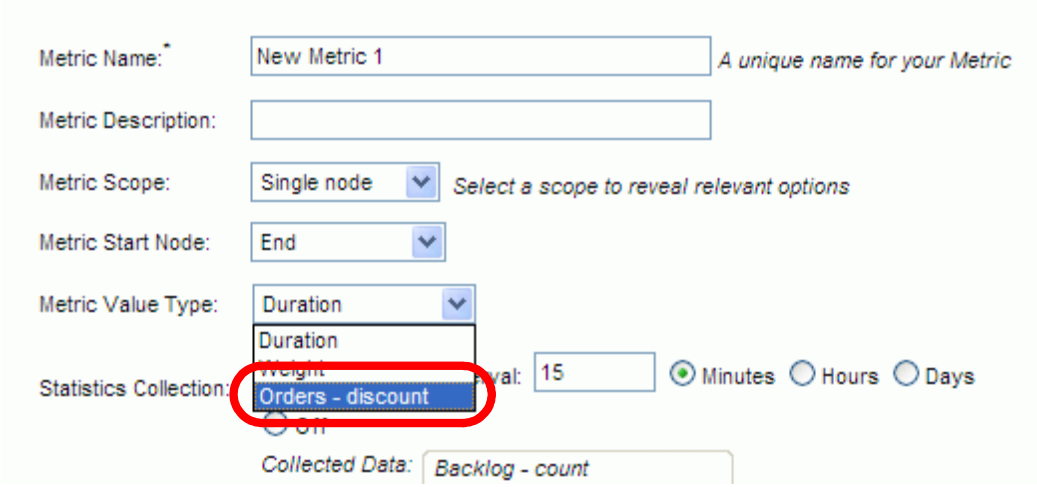
You create an entry in the `METRIC_CustomTypes` table defining a name for your custom metric and linking this to your stored procedure.

For example:

```
INSERT INTO METRIC_CustomTypes (CustomMetricName,
                                CustomMetricDescription,
                                CustomSPName,
                                ValueUnits)
      VALUES('Orders - discount',
             'Calculates the percentage discount given on this order.',
             'BPI_Discount_Value',
             'Percent');
```

When you defines a metric in the Metric Definer you are now able to select your stored procedure by selecting the `Metric Value Type` of `Orders - discount`. This is shown in Figure 15 on page 83.

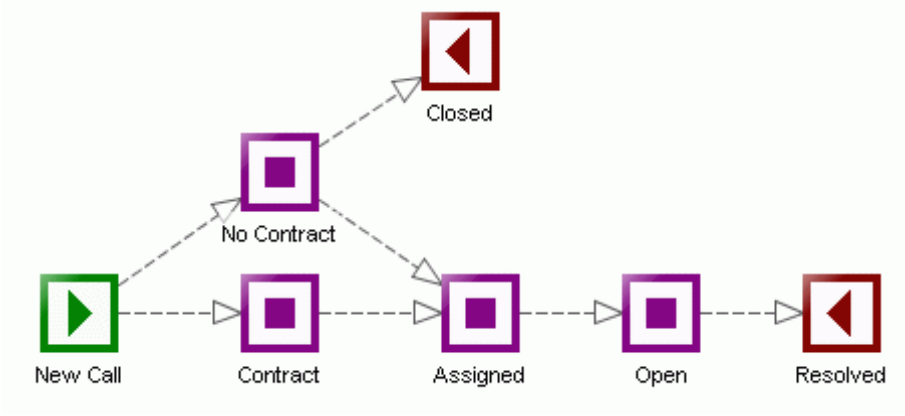
**Figure 15 Custom Metric Type**



# Example - Percentage Path Flow

Suppose you have a flow that looks as follows:

**Figure 16 Flow with Multiple Paths**




This flow is the `Call System` flow from an earlier lab (Lab - Defining Metrics on page 37). It monitors support calls as they come into the call center. The calls include some that are under a support agreement and some that are not. What's more, the supervisor may choose to handle a non-contract call because he/she feels the caller is a potential customer and therefore worth looking after.

Your job is to measure the calls that are being processed and report the following percentages:

- The percentage of contract calls being resolved
- The percentage of non-contract calls being resolved
- The percentage of non-contract calls being closed

In other words, you need to measure the different outcomes of the flow and show these outcomes as percentages.

This requires defining some custom metrics.

## Metric Scope

You want to be able to measure the state of each flow instance when it has completed. So you want to define some stored procedures that are run at the completion of each flow instance. This means that when you come to configure these metrics within the Metric Definer (once all the stored procedures are defined and in place) you set the metric scope to be `Whole Flow`. This ensures that the metric is collected at the start and the end of each flow instance.

## The Stored Procedure(s)

You need to write a stored procedure for each outcome of the flow. That is, you write a stored procedure that measures the flow instances that end at the `Closed` node. You write another stored procedure that measures the flow instances that terminate at the `Resolved` node having gone through the `Contract` node, and another procedure that measures the flow instances that terminate at `Resolved` node having gone through the `NoContract` node.

But what is it that you measure? What value do you return as the metric value? And how do you calculate this value?

You write the stored procedures to return the value of either 0 or 100.

Let's consider the stored procedure that is going to measure the flow instances that have terminated at the `Resolved` node having come through the `Contract` node. The stored procedure uses the flow instance Id passed in, to find out whether this flow instance actually terminated at the `Resolved` node. It does this by accessing the `Node_Instance` table and looks to see if there is a completed node instance for the `Resolved` node for this flow instance. If the flow instance did terminate at the `Resolved` node, then the stored procedure looks to see if the `Contract` node is also completed for this flow instance. If both of these tests are true, then this flow instance terminated at the `Resolved` node having also completed the `Contract` node, and therefore the stored procedure returns a value of 100. If the flow instance did not terminate at the `Resolved` node or did not complete the `Contract` node, the stored procedure returns a value of 0.

By returning a metric value of either 0 or 100, the Metric Engine can collect all the metric values. Every collection interval the Metric Engine calculates, amongst other things, the average of these values. Thus, the Metric Engine produces the average of your metric values, which is indeed the percentage.

## Contract Resolved Calls

The stored procedure for counting the instances of contract calls that are resolved looks as follows (for MSSQL):

```
CREATE PROCEDURE BPI_Contract_Resolved_Calls
      @Metric_ID NVARCHAR(36), @Flow_ID NVARCHAR(36),
      @FlowInstance_ID NVARCHAR(36), @FlowInstIdentifier NVARCHAR(40),
      @FlowInstStartTime DATETIME, @Weight FLOAT,
      @DataDefinition_ID NVARCHAR(36), @DataInstance_ID NVARCHAR(36),
      @EventType NVARCHAR(12), @EventTime DATETIME, @Idx INTEGER
AS
  DECLARE @RetVal FLOAT
  DECLARE @Count1 FLOAT
  DECLARE @Count2 FLOAT
BEGIN

  -- For this flowinstanceID, is the 'Resolved' node completed
  --                          and the 'Contract' node completed
  -- If so  = set value to 100
  -- If not = set value to 0

  -- (1) If this is the start of the metric, then simply return
  IF (@EventType = 'Started')
  BEGIN
      RETURN;
  END

  -- Assume a false outcome
  SET @RetVal = 0

  -- (2) Did the flow instance terminate at the 'Resolved' node
  --     (For this flowInstanceID, is the 'Resolved' node 'Completed')
  SET @Count1 = 0
  SELECT @count1 = count(ni.flowinstance_id)
     from node_instance ni, nodes
      where ni.node_id = nodes.node_id
      and ni.flowinstance_id = @FlowInstance_ID
      and (nodes.nodename = 'Resolved' and ni.status = 'Completed')
```

```
  -- Only do this check if this instance terminated at the 'Resolved' node.
  IF (@Count1 != 0)
  BEGIN
     -- (3) Check whether the 'Contract' node is also 'Completed'
     SET @Count2 = 0
     SELECT @Count2 = count(ni.flowinstance_id)
        from node_instance ni, nodes
         where ni.node_id = nodes.node_id
         and ni.flowinstance_id = @FlowInstance_ID
         and (nodes.nodename = 'Contract' and ni.status = 'Completed')
  END

  -- (4) If it did go through the nodes, set the return value to 100
  IF (@Count1 != 0 AND @Count2 != 0)
  BEGIN
      SET @RetVal = 100
  END

  -- (5) Now write the result to the metric engine

  EXECUTE METRIC_SEND_EVENT @Metric_ID, @Flow_ID, @FlowInstance_ID,
      @FlowInstIdentifier, @FlowInstStartTime, @Weight, @DataDefinition_ID,
      @DataInstance_ID, @EventType, @EventTime, @RetVal, @Idx

END;
```

where:

- Step **(1)** tests if this is the start of the metric, simply return. You only wish to measure completed instances
- Step **(2)** tests that this flow instance has actually completed at the `Resolved` node
- Step **(3)** tests that, if the flow instance has completed at the `Resolved` node, did it also complete the `Contract` node
- Step **(4)** tests whether all conditions have been met. If so, it sets the return value to be `100`.
- Step **(5)** sends the metric result to the Metric Engine

## Non-Contract Resolved Calls

The stored procedure for counting the instances of non-contract calls that are resolved looks as follows (for MSSQL):

```
CREATE PROCEDURE BPI_Off_Contract_Resolved_Calls
       @Metric_ID NVARCHAR(36), @Flow_ID NVARCHAR(36),
       @FlowInstance_ID NVARCHAR(36),
       @FlowInstIdentifier NVARCHAR(40), @FlowInstStartTime DATETIME,
       @Weight FLOAT, @DataDefinition_ID NVARCHAR(36),
       @DataInstance_ID NVARCHAR(36), @EventType NVARCHAR(12),
       @EventTime DATETIME, @Idx INTEGER
AS
  DECLARE @RetVal FLOAT
  DECLARE @Count1 FLOAT
  DECLARE @Count2 FLOAT
BEGIN

  -- (1) If this is the start of the metric, then simply return
  IF (@EventType = 'Started')
  BEGIN
      RETURN;
  END

  -- Assume a false outcome
  SET @RetVal = 0

  -- (2) Did the flow instance terminate at the 'Resolved' node
  --      (For this flowInstanceID, is the end node 'Completed')
  SET @Count1 = 0
  SELECT @count1 = count(ni.flowinstance_id)
     from node_instance ni, nodes
      where ni.node_id = nodes.node_id
      and ni.flowinstance_id = @FlowInstance_ID
      and (nodes.nodename = 'Resolved' and ni.status = 'Completed')

  -- Only do this check if this instance terminated at the 'Resolved' node.
  IF (@Count1 != 0)
  BEGIN
     -- (3) Check whether the 'No Contract' node is also 'Completed'
     SET @Count2 = 0
     SELECT @Count2 = count(ni.flowinstance_id)
        from node_instance ni, nodes
         where ni.node_id = nodes.node_id
         and ni.flowinstance_id = @FlowInstance_ID
         and (nodes.nodename = 'No Contract' and ni.status = 'Completed')
  END
```

```
  -- (4) If it did go through the nodes, set the return value to 100
  IF (@Count1 != 0 AND @Count2 != 0)
  BEGIN
      SET @RetVal = 100
  END

  -- (5)

  EXECUTE METRIC_SEND_EVENT @Metric_ID, @Flow_ID, @FlowInstance_ID,
      @FlowInstIdentifier, @FlowInstStartTime, @Weight, @DataDefinition_ID,
      @DataInstance_ID, @EventType, @EventTime, @RetVal, @Idx

END;
```

where:

- Step **(1)** tests if this is the start of the metric, simply return. You only wish to measure completed instances
- Step **(2)** tests that this flow instance has actually completed at the `Resolved` node
- Step **(3)** tests that, if the flow instance has completed at the `Resolved` node, did it also complete the `No Contract` node
- Step **(4)** tests whether all conditions have been met. If so, it sets the return value to be `100`.
- Step **(5)** sends the metric result to the Metric Engine

## Non-Contract Closed Calls

The stored procedure for counting the instances of non-contract calls that are simply closed (ignored), looks as follows (for MSSQL):

```
CREATE PROCEDURE BPI_Off_Contract_Closed_Calls
     @Metric_ID NVARCHAR(36), @Flow_ID NVARCHAR(36),
     @FlowInstance_ID NVARCHAR(36), @FlowInstIdentifier NVARCHAR(40),
     @FlowInstStartTime DATETIME, @Weight FLOAT,
     @DataDefinition_ID NVARCHAR(36), @DataInstance_ID NVARCHAR(36),
     @EventType NVARCHAR(12), @EventTime DATETIME, @Idx INTEGER
AS
  DECLARE @RetVal FLOAT
  DECLARE @Count1 FLOAT
BEGIN

  -- (1) If this is the start of the metric, then simply return
  IF (@EventType = 'Started')
  BEGIN
      RETURN;
  END

  -- Assume a false outcome
  SET @RetVal = 0

  -- (2) Did the flow instance terminate at the 'Closed' node
  -- (For this flowInstanceID, is the end node 'Completed')
  SET @Count1 = 0
  SELECT @count1 = count(ni.flowinstance_id)
     from node_instance ni, nodes
      where ni.node_id = nodes.node_id
      and ni.flowinstance_id = @FlowInstance_ID
      and (nodes.nodename = 'Closed' and ni.status = 'Completed')

  -- (3) If it did go through the 'Closed' node, set the return value to 100
  IF (@Count1 != 0)
  BEGIN
      SET @RetVal = 100
  END

  -- (4)

  EXECUTE METRIC_SEND_EVENT @Metric_ID, @Flow_ID, @FlowInstance_ID,
      @FlowInstIdentifier, @FlowInstStartTime, @Weight, @DataDefinition_ID,
      @DataInstance_ID, @EventType, @EventTime, @RetVal, @Idx
END;
```

where:

- Step **(1)** tests if this is the start of the metric, simply return. You only wish to measure completed instances
- Step **(2)** tests that this flow instance has actually completed at the `Closed` node
- Step **(3)** tests whether the condition has been met. If so, it sets the return value to be `100`.
- Step **(4)** sends the metric result to the Metric Engine

## The Custom Metric Type Definitions

Once the stored procedures are installed into your database you need to tell the Metric Definer about them and give them names.

You create an entry in the `METRIC_CustomTypes` table defining a name for your custom metric and linking this to your stored procedure.

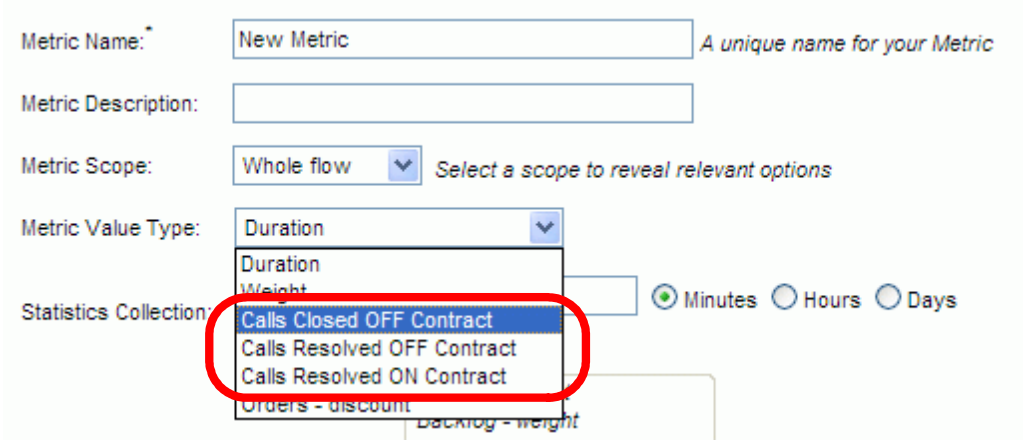
For example, to load in the three stored procedures, you could issue the following commands:

```
INSERT INTO METRIC_CustomTypes (CustomMetricName,
                                CustomMetricDescription,
                                CustomSPName,
                                ValueUnits)
      VALUES('Calls Resolved ON Contract',
             'Calculates the percentage calls resolved ON contract.',
             'BPI_Contract_Resolved_Calls',
             'Percent');

INSERT INTO METRIC_CustomTypes (CustomMetricName,
                                CustomMetricDescription,
                                CustomSPName,
                                ValueUnits)
      VALUES('Calls Resolved OFF Contract',
             'Calculates the percentage calls resolved OFF contract.',
             'BPI_Off_Contract_Resolved_Calls',
             'Percent');

INSERT INTO METRIC_CustomTypes (CustomMetricName,
                                CustomMetricDescription,
                                CustomSPName,
                                ValueUnits)
```

```
VALUES('Calls Closed OFF Contract',
       'Calculates the percentage OFF contract calls closed.',
       'BPI_Off_Contract_Closed_Calls',
       'Percent');
```

When you define a metric in the Metric Definer, you are now able to select your stored procedures by selecting one of the defined `Metric Value Types` as shown in Figure 17 on page 92

**Figure 17 Custom Percentage Metric Types**




## Defining The Metrics

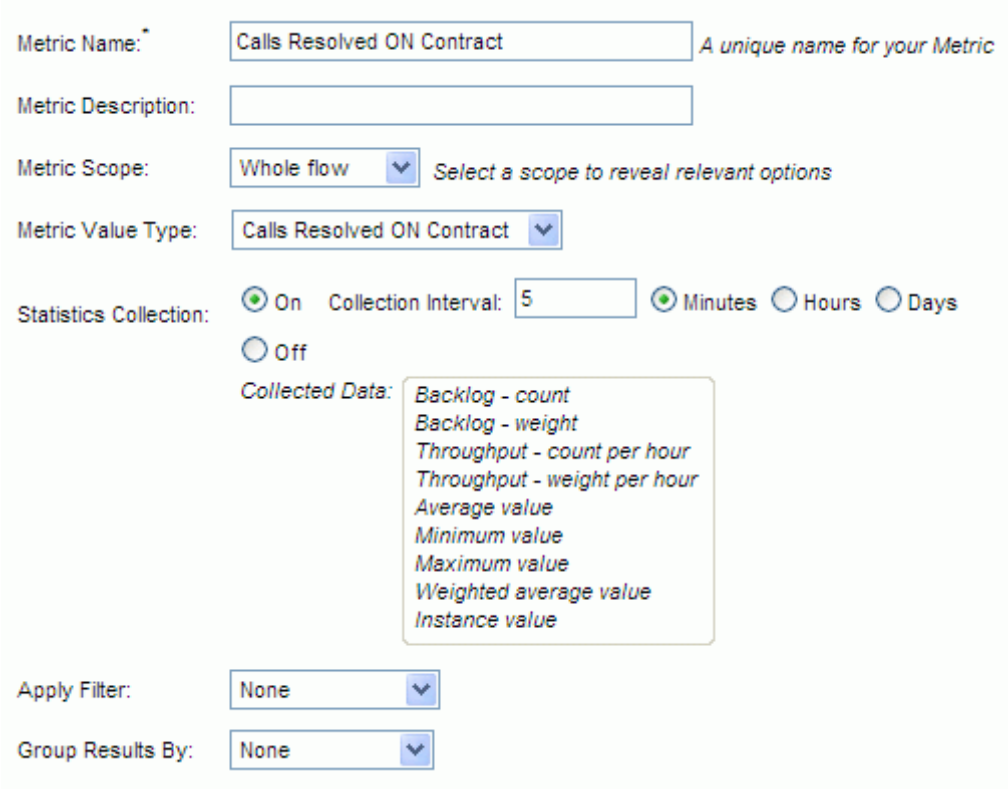
As there are three possible outcomes for the `Call System` flow, you define three metrics. Each metric must produce results for every flow instance going through the `Call System` flow, as this enables the Metric Engine to produce an average. So you define three metrics, each with a scope of `Whole Flow`.

You can set the collection interval to be whatever you require. Remember that the collection interval determines the frequency with which the percentage is calculated and updated.

With each metric definition, you select the `Metric Value Type` from the three custom metric types you defined.

For example, to configure the metric that measures the percentage of contract calls that are resolved, you define the metric as follows:

**Figure 18 Metric Definition - Resolved Contract Calls**




You would define two more metrics, to measure the non-Contract Resolved calls and the non-Contract Closed calls.

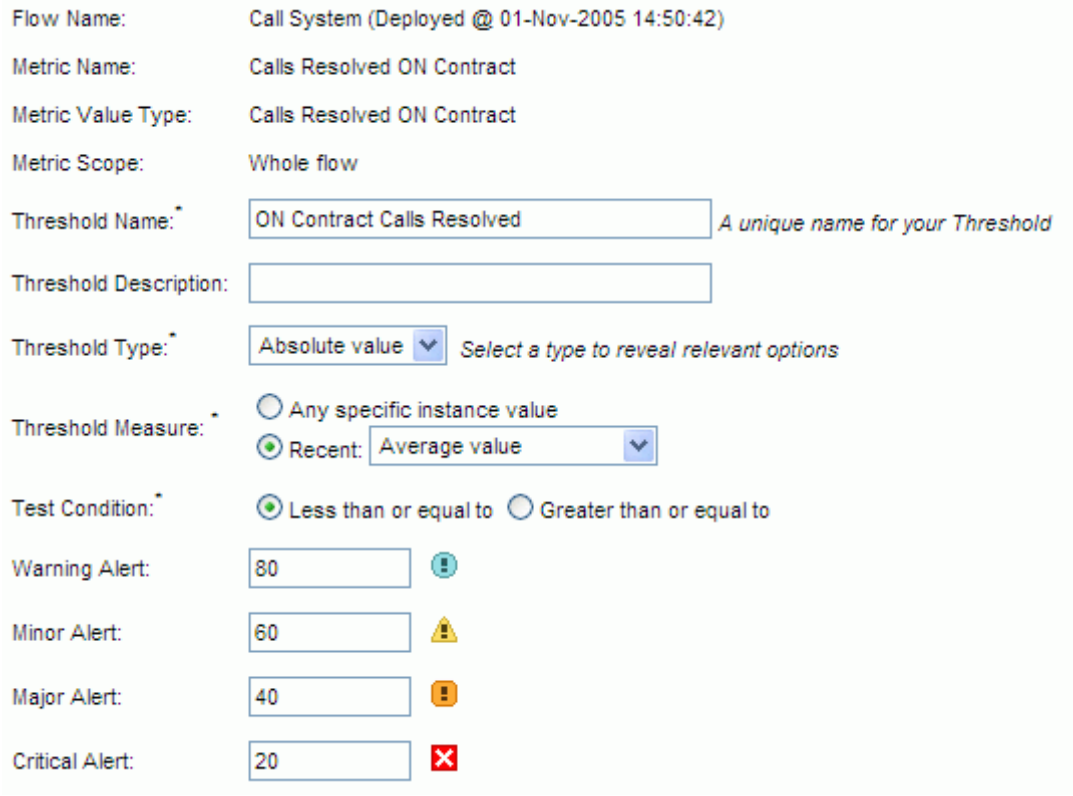
## Defining Thresholds

To see your percentages within the OVBPI Dashboard as dials, you can define thresholds for each metric.

You can define thresholds that measure the recent average as this is the percentage. This allows you to set percentage values as the ranges for warning, minor, major and critical.

For example:

**Figure 19 Threshold Definition - Resolved Contract Calls**



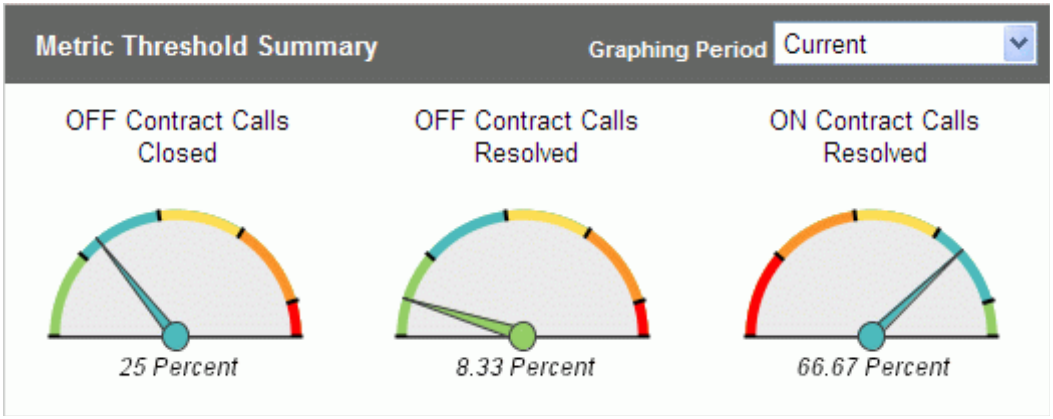

This threshold alerts you whenever the percentage of resolved contract calls drops below 80 percent.

## The OVBPI Dashboard

Now that the metrics and thresholds are defined, you can view the percentages within the OVBPI Dashboard. When you view the `Call System` flow, the three threshold dials show the percentages of your calls.

For example:

**Figure 20 Dashboard - Percentages**




These dials give you the current averages (or percentages in this case). You can use the `Graphing Period` pull-down to show the averages (percentages) over a longer period.

You can also click on any of these dials to see the average (percent) since the metric was defined. This allows you to see the percentages over time and spot trends.

# SQL Errors

When you write a stored procedure to produce a custom metric value, your stored procedure is invoked when the Business Impact Engine updates the OVBPI database tables. The updating of the OVBPI tables and the running of your stored procedure all occur within a single database transaction. This means that if your stored procedure encounters a problem it may affect the Business Impact Engine transaction and thus the flow instance. It all depends on the underlying database being used by OVBPI.

## OVBPI on Oracle

If you are running OVBPI against an Oracle database then any problems within your metric stored procedure are logged in the Business Impact Engine's log file, and this does **not** affect the Business Impact Engine or the processing of the flow instance.

The Oracle database system allows the stored procedure to throw an error, and have this treated separately within the overall Business Impact Engine's transaction. The Business Impact Engine is able to trap, and log, any errors and keep processing.

Obviously the metrics are not recorded correctly if the stored procedure is in error, but the flow keeps running.

## OVBPI on MSSQL

If you are running OVBPI against an MSSQL database then any problems within your metric stored procedure cause the Business Impact Engine to abort the entire transaction. The error is logged to the Business Impact Engine's log file and it causes both the metric and the current flow instance to not be updated correctly.

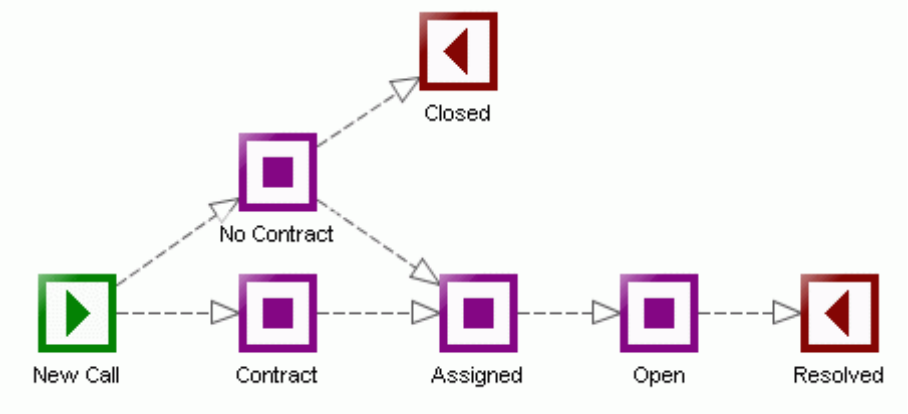
# Lab - Custom Metrics

This lab gives you experience setting up custom metrics.

## The Call System Flow

For this lab you are going to continue working with the `Call System` flow you set up during the previous lab (Lab - Defining Metrics on page 37). The flow diagram is as follows:

**Figure 21 Flow Diagram**




This flow monitors support calls as they come into the call center. The calls include some that are under a support agreement and some that are not. The supervisor may choose to handle a non-contract call because he/she feels the caller is a potential customer and therefore worth looking after.

## The Required Metrics

Your job is to measure the calls that are being processed, and report the following percentages:

- The percentage of contract calls being resolved
- The percentage of non-contract calls being resolved
- The percentage of non-contract calls being closed

In other words, you need to measure the different outcomes of the flow and show these outcomes as percentages.

The way to set up these metrics is fully explained earlier in this chapter in the section: Example - Percentage Path Flow on page 84. You are to refer to that section for guidance as needed.

Your job in this lab is to do the following:

- Define the three required stored procedures, and define metric type entries such that the Metric Definer knows about your stored procedures.

  To help you get started, the SQL for defining the stored procedure `BPI_Contract_Resolved_Calls`, and for defining the metric value type `Calls Resolved ON Contract`, is located in the file:

  ```
  labs\custom_ResolvedContractCalls.sql
  ```

  This file contains the SQL for working with a MSSQL database.

- Define metrics to use these metric types.

  Define three metrics. All of these metric are to be set to a scope of `Whole Flow`. These metrics should invoke your stored procedures. Set the collection interval to be five minutes.

- Define a threshold for each of your metrics, to measure the recent average and set alerts between the values of 0 and 100.

- Use the Flow Simulator to send through more calls and monitor the percentages in the OVBPI Dashboard.

  Make sure when you use the Flow Simulator you ensure the `Instance ID` is greater than any previous run of the Flow Simulator. That is, make sure that the Flow Simulator injects new instances with new unique IDs.

When you have the OVBPI Dashboard showing the percentages, take some time to look at the other metrics that you defined in the first lab.

If you kept the Flow Simulator running since the first lab, you should be able to use the OVBPI Dashboard to view the details of your metrics over time.

For example:

- In the OVBPI Dashboard, on the main `Business Flow & Resource Summary` page - the page that shows the flow diagram and the threshold dials - click on the tab marked: `Metrics`
- Click on the metric called: `Call Processing Time`

  By default this shows you a graph displaying the Average/Minimum/Maximum values for each collection interval.
- Select the `Data Source` for the graph to be: `Completed - Count`

  This shows you the number of calls that have been resolved within each collection interval.
- Now select `Chart by group` to be: `Yes`

  You now see the number of calls that have been closed within each collection interval, but now showing the numbers for each call priority.

  This shows you the completion statistics since the first lab when you set up the `Call Processing Time` metric.

Feel free to explore all your metrics using the OVBPI Dashboard and the many options available to you.

**Well done! You have reached the end of the lab.**

4

# Further Topics

This chapter looks at additional topics to do with OVBPI metrics.

# OVBPI 1.1 Metric Tables

The OVBPI release 1.1 had the ability to define metrics. These metrics were stored in the two tables, `Metrics` and `Metric_Values`. The OVBPI 2.0 release has completely replaced the old metric system, and as such, uses a new set of data tables.

For compatibility reasons OVBPI 2.0 maintains two database views called `Metrics` and `Metric_Values`. These views provide OVBPI 1.1 views onto the new OVBPI 2.0 metric data.

If you are running existing 1.1 database reports against these new views be aware of a subtle change. There are more than just `TBN` metric types.

The metric type for OVBPI 1.1 used to always be `TBN`. The metric type can now have four possible values:

- `TBN` - Time between nodes
- `NET` - Node execution time
- `FET` - Flow execution time
- `CUSTOM` - A custom metric

In OVBPI 1.1 terms, the three types `TBN`, `NET` and `FET` are all just `TBN` metrics.

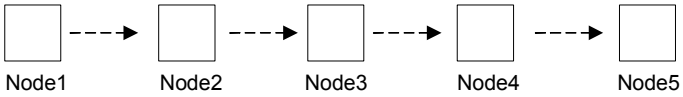
These views, and all the metric tables, are fully described in the *OVBPI System Administration Guide*.

# Metric/Threshold Activation

When does a metric definition become active? The answer is, the moment you click the `OK` button within the Metric Definer. This means that metrics only begin to be recorded from the time you click the `OK` button when defining a new metric.

Suppose you have a flow as follows:

**Figure 22 Simple Flow**




and you already have some flow instances that are sitting in `Node3`.

You then define a metric to measure from the start of `Node2` to the end of `Node4`.

The flow instances that are currently in `Node3` are past the start of the metric so you do not see any start metrics for these flow instances. When these flow instances move out of `Node4`, end metric records are generated.

When you modify a metric definition and click the `Replace` button, all previous metric history and alerts are removed, and the metric starts recording data afresh as if it were a brand new metric definition.

## Detecting Thresholds

When you modify a time-based threshold be careful if you update the alert time.

Suppose you define a metric for the flow shown in Figure 22 on page 103, and this metric measures the time it takes for instances to move from the start of `Node2` to the end of `Node4`. You then define a threshold to raise an alert if any specific instance takes longer than 10 minutes.

To test this out, you then send in a flow instance and progress it to `Node3`. You leave the flow instance in `Node3` for 10 minutes and wait for the alert to be issued.

After waiting for four minutes you think “Why am I waiting all this time?” so you decide to modify the threshold definition and update it to issue an alert if any specific instance takes longer than just one minute. You assume that you now only have to wait one more minute to see the alert appear.

You wait...and wait...and after 15 minutes of waiting, and seeing no alert, you think something has gone wrong.

You then assume that the alert is not going to happen, so you decide to progress the flow instance and see what happens when it reaches the end of `Node4`. Sure enough, when the flow instance exits `Node4`, you suddenly see an alert raised within the OVBPI Dashboard. This alert shows that the metric instance did indeed take far longer than one minute to be completed.

So what happened?

It all has to do with the way the Metric Engine looks for threshold violations while metrics are still active.

The Metric Engine constantly monitors active metric instances in case they exceed a threshold while still active. That is, if you want an alert raised when an instance takes longer than (for example) one hour to be completed, you don’t want to be told this once the instance has completed. You want the alert raised the moment the instance has been running for more than your specified time period.

To identify active metric instances that have been running for longer than a specified time period, the Metric Engine monitors active metric instances. But the Metric Engine only needs to monitor the metric instances that have started within the specified time period. In this example, the original threshold time period was set to 10 minutes. So every poll period the Metric Engine checks all metric instances that have started within the last 10 minutes and looks to see if they are still running. If they are, then they have exceeded the threshold and an alert is raised.

The problem came about when you updated the threshold definition and changed the alert time period from 10 minutes to one minute. This meant that the Metric Engine started monitoring active instances, but only within the last one minute period. Thus the original instance that you had started was out of the time range for active monitoring. This is why no alert was raised the moment the metric instance took longer than one minute.

Of course, when the metric instance finally does complete, the Metric Engine detects this and raises an alert to say that the metric instance took too long.

So, be careful about updating threshold times and making them shorter. This may cause alerts to not be raised until the completion of some of the metric instances.

# Instance Alerts

If you create a threshold and define threshold alerts for Warning, Minor, Major and Critical, do not be surprised if you see a metric instance raise (for example) a Warning alert followed by a Critical alert. That is, just because you have specified alerts at all levels, an individual metric instance may not raise them all. Let's explain...

Thresholds are checked every 60 seconds. (This time period is configurable within the OVBPI Administration Console.) When the threshold values are checked for a metric instance, an alert is raised only for the highest (most severe) alert that has occurred within that 60 seconds. So, for example, if a metric instance has raised a Minor alert, a Major alert and a Critical alert all within the last 60 seconds, the Metric Engine only raises a single alert showing that the metric instance has now gone critical.

# Deadline Metric Value is Fixed

With a deadline metric, the value of the deadline property is read at the start of the metric instance.

Suppose you have created a metric and specified a `Deadline Property`. As each instance of this metric is instantiated, the value within the deadline property is read from the flow's related data definition, and stored with the metric instance. Thresholds are then measured against this deadline attribute value held within the metric instance. If the value within the deadline attribute (within the data definition) changes during the life of the metric instance, the metric instance does not see this. The metric instance uses the attribute value it stored when it started.

# No Collection Interval Defined

It is possible to define a metric to have no collection interval. For example, if you are measuring the time an instance takes to get to a particular node, and raising an alert if this time is greater than four hours, then you do not need to set any collection interval.

The OVBPI Dashboard uses the metric's collection interval to make certain calculations. For example, the `Business Health Scorecard` page displays the most severe alert for a flow over the last collection interval for each defined metric. But what period does it use for metrics that have no collection interval defined?

For metrics that have no collection interval specified, the OVBPI Dashboard uses a period of:

```
2 * Threshold Polling Interval
```

The `Threshold Polling Interval` is configurable within the OVBPI Administration Console, and defaults to 60 seconds.

# Dashboard and Alerts

## Business Health Scorecard Page

When you run the OVBPI Dashboard, the `Business Health Scorecard` page displays each deployed flow name and shows a `Business Flow Metric Status` column.

The idea is that as business metric alerts occur, the `Business Health Scorecard` page reflects these in the `Business Flow Metric Status` column. Because instance alerts (see Instance and Statistical Thresholds/ Alerts on page 33) have no way of being reset, the home page of the OVBPI Dashboard has been written to only include alerts that have occurred within the last collection interval across all the thresholds defined for this flow.

For thresholds where there is no collection interval configured see the discussion in No Collection Interval Defined on page 107.

So, the `Business Flow Metric Status` column represents the highest (most severe) alert that has occurred within the last collection interval across all the thresholds defined for this flow.