

# HP Business Availability Center

for the Windows and Solaris operating systems

Software Version: 8.00

---

## Business Process Insight Integration Training Guide

### Defining Business Process Monitors

Document Release Date: January 2009

Software Release Date: January 2009



## Legal Notices

### Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

### Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

### Third-Party Web Sites

HP provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. HP makes no representations or warranties whatsoever as to site content or availability.

### Copyright Notices

© Copyright 2006 - 2009 Hewlett-Packard Development Company, L.P.

### Trademark Notices

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

Intel®, Pentium®, and Intel® Xeon™ are trademarks of Intel Corporation in the U.S. and other countries.

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

Microsoft®, Windows®, Windows NT®, and Windows® XP are U.S registered trademarks of Microsoft Corporation.

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

Unix® is a registered trademark of The Open Group.

## Documentation Updates

This guide's title page contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

**<http://h20230.www2.hp.com/selfsolve/manuals>**

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

**<http://h20229.www2.hp.com/passport-registration.html>**

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

## Support

You can visit the HP Software Support web site at:

**<http://www.hp.com/go/hpsoftwaresupport>**

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

HP Software Support Online 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 valued support customer, you can benefit by using the HP Software Support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into 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 sign in. Many also require a support contract.

To find more information about access levels, go to:

**[http://h20230.www2.hp.com/new\\_access\\_levels.jsp](http://h20230.www2.hp.com/new_access_levels.jsp)**

To register for an HP Passport ID, go to:

**<http://h20229.www2.hp.com/passport-registration.html>**

# Contents

<b>1 Business Monitors</b> .....	<b>9</b>
Introduction .....	10
Monitor Definer .....	12
Running the Monitor Definer .....	12
Logon User/Password .....	13
Deployed Processes .....	13
Creating a Business Monitor .....	14
The Process Diagram .....	15
Monitor Name .....	16
Monitor Description .....	16
Monitor Scope .....	16
Monitor Type .....	17
Statistics Collection .....	18
Filters, Groups and Deadlines .....	21
Apply Filter .....	21
Group Results By .....	22
Deadline Property .....	22
Filters .....	23
Creating a Filter .....	23
Example Filters .....	25
Modifying a Monitor .....	26
Creating a Threshold .....	27
Creating a Threshold Using the Monitor Definer .....	27
Threshold Type .....	28
Warning/Minor/Major/Critical Violation .....	31
Violation Message .....	31
Violations .....	31
Instance and Statistical Thresholds/Violations .....	32

Violation Levels . . . . .	32
Options . . . . .	33
Refreshing the Monitor Definer . . . . .	33
Exporting Monitors . . . . .	33
Importing Monitors . . . . .	34
Monitor Definer Help . . . . .	34
Lab - Defining Monitors . . . . .	35
The Scenario . . . . .	35
Defining the Process . . . . .	36
Understanding the Process . . . . .	37
Data collection Intervals . . . . .	38
Defining the Monitors and KPIs . . . . .	39
The Process Simulator . . . . .	47
Running the Call Center . . . . .	47
End of The Lab . . . . .	52
<b>2 Custom Monitors . . . . .</b>	<b>53</b>
Monitor Scope . . . . .	54
Defining a Custom Monitor . . . . .	55
The Stored Procedure . . . . .	55
The Monitor Definer . . . . .	60
Example - A Data Property . . . . .	62
The Stored Procedure . . . . .	63
The Custom Monitor Type Definition . . . . .	68
Example - Percentage Path Process . . . . .	69
Monitor Scope . . . . .	70
The Stored Procedure(s) . . . . .	70
The Custom Monitor Type Definitions . . . . .	79
Defining The Monitors . . . . .	80
Defining KPIs and objectives . . . . .	82
The BPI Business Heath screen . . . . .	83
SQL Errors . . . . .	84
BPI on Oracle . . . . .	84
BPI on MSSQL . . . . .	84
Lab - Custom Monitors . . . . .	85
The Call System Process . . . . .	85

The Required Monitors.....	85
<b>3 Further Topics.....</b>	<b>89</b>
Monitor/Threshold Activation.....	90
Pre-Dated Events (BPI_GeneratedDate).....	90
Detecting Thresholds.....	91
Instance Violations.....	93
Deadline Monitor Value is Fixed.....	93
Redeploying Processes/Monitors.....	94
Deleting Process Monitors/Thresholds.....	95
Avoiding Notification Storms.....	96





---

# 1 Business Monitors

This chapter looks at how to create and use business monitors using the Business Process Insight (BPI) Monitor Definer.

# Introduction

Once a BPI process is deployed and running, the BPI Business Impact Engine maintains basic statistics about the process, such as:

- The overall state of the process.
- The number of active process instances.
- Specific process instance statistics such as:
  - The start and stop times of each step.
  - Step durations.

These statistics are held in the BPI Engine database.

The data collected is then forwarded on to HP Business Availability Center (BAC) where it can be used to generate Key Performance Indicators (KPI). By default BAC maintains volume and backlog KPIs for all nodes within the flow based on the process value property, but these can be modified to track other parameters such as the instance counts.

In addition to the standard process and step statistics that are maintained for all processes, by the Business Impact Engine, you can specify additional process statistics, known as **business process monitors**. For example, you can create business process monitors such as:

- The time taken to process an order.
- The orders processed by the warehouse that have not been credit checked.
- The calls resolved in a support center even though the customer has no support contract.
- Any delays to re-scheduled flights beyond their agreed new take off times.

These additional monitors are also made available to BAC where they can be statistically processed allowing measurements such as:

- The average backlog of flights waiting to get airborne
- The average delay to orders over \$100,000
- The percentage of phone registration requests requiring manual intervention

When you have defined a business process monitor, you can:

- create a business process threshold using the Monitor Definer.
- create a KPI objective using Dashboard administration.

Business process thresholds enable BPI to raise a violation when an instance of a business process monitor exceeds one of the defined thresholds; for example, you can configure a threshold to issue a violation when:

- the time taken to process an order exceeds four hours.
- a mortgage application is not handled by an agreed date.

Using a KPI you can configure **objectives** to:

- Issue a warning violation when the current backlog of flights waiting to get airborne is greater than 10. Issue a critical violation when this number exceeds 50.
- Issue a violation when the average for orders is below a specified level.

Once defined, your monitors, thresholds and violations are reported in the BPI Business Process Insight screens within the Business Availability Center.

# Monitor Definer

Once a process is defined and deployed, you use the BPI Monitor Definer to define monitors and thresholds for that process.

## Running the Monitor Definer

The Monitor Definer runs within a Web browser inside BAC, thus you need to make sure that you have the BPI Servlet Engine running, that the BPI Engine database is up and running, and that BAC is also available.

To run the Monitor Definer you need to login to BAC as an administrator and get to Business Process Insight under the Admin menu. You initially start BAC using:

```
Start->Programs->HP Business Availability Center ->  
Open Business Availability Center
```

or, start a Web browser pointing at the BAC server with the URL:

```
http://fully_qualified_hostname/topaz/
```

where:

- *fully\_qualified\_hostname* is the hostname complete with DNS domain extension of your BAC Server installation

For example:

```
http://bac.acme.com:/topaz/
```

From within BAC use the Admin menu to select Business Process Insight. From the Business Process Insight screen select the Monitor Definer tab.

## Logon User/Password

When running BAC, it asks you to log on as a valid user.

The default user name/password details are:

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

To change the admin user password refer to the *BAC documentation*.

## Deployed Processes

You can create monitors only for processes that have been defined to track individual process instances are already deployed.

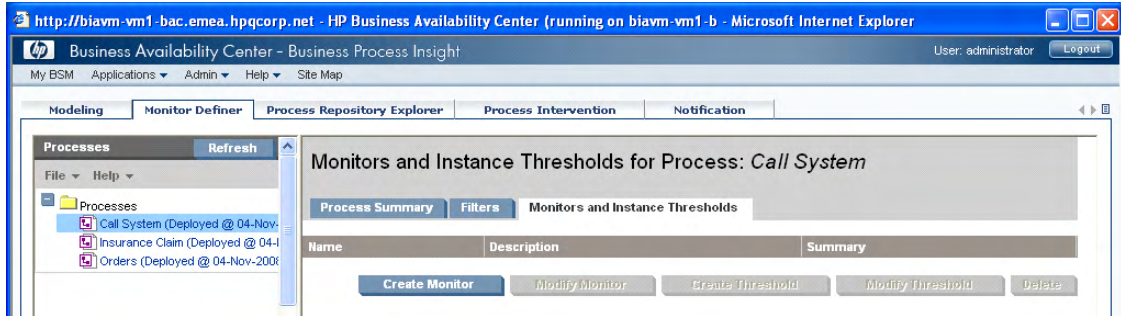
If a new process is deployed when you are running the Monitor Definer, you can select the `Refresh` button from within the left-hand Navigator frame of the Monitor Definer (not the Web browser's toolbar). This refreshes the list of deployed processes available to you within the Monitor Definer.

# Creating a Business Monitor

To create a business monitor you must first select the process. You simply click on the required process in the left-hand Navigator frame within the Monitor Definer.

For example, consider [Figure 1](#):

**Figure 1 Monitor Definer - Selecting a Process**



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

- Process Summary

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

- Filters

This tab allows you to create and modify filters. Filters are explained in more detail in the section [Filters](#) on page 23.

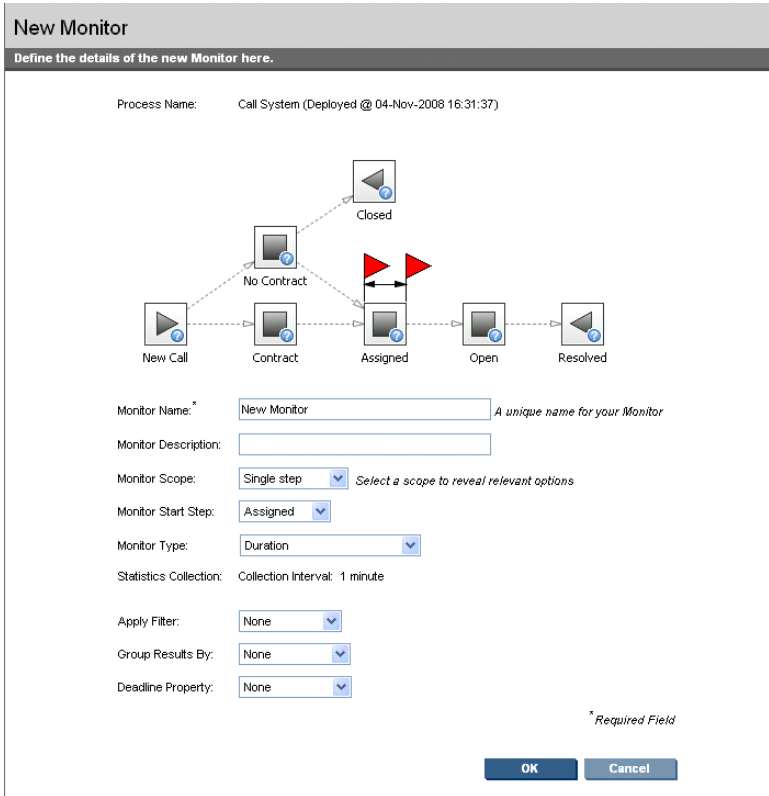
- Monitors and Instance Thresholds

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

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

For example, consider [Figure 2](#) on page 15.

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



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

## The Process Diagram

The process diagram is displayed and two monitor flags (red colored flags) are drawn on one of the steps. These monitor flags define the start and end of the monitor you are defining. Initially these flags are placed on a single step<sup>1</sup>, and as you define the monitor more fully, these flags move to reflect your configuration.

1. The node is selected from the Monitor Start Step field which has the nodes listed in alphabetical order.

## Monitor Name

You need to give your monitor a unique name. This name must be unique across all your monitors.

## Monitor Description

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

## Monitor Scope

The following options are available to you.

### Whole Process

Selecting this option causes the monitor flags to disappear.

Setting a `Whole Process` monitor means that when a process instance first starts (at whatever step actually starts the process instance), a monitor instance is started. This monitor instance completes when the process instance completes.

### Single Step

Setting a `Single Step` monitor means that a monitor instance starts when a process instance enters the specified step. This monitor instance completes when the process instance exits this same step.

#### Monitor Start Step

When selecting a `Single Step` monitor, you are also asked to specify the step itself. As you specify this step, the monitor flags within the process diagram move to highlight this step.



## Multiple Steps

Setting a `Multiple Steps` monitor means that this monitor measures between two steps within the process diagram.

### Monitor Start Step

You select the start step for your monitor. You also specify whether it is the start or completion of this step that starts each monitor instance. Note that the monitor flags within the process diagram do not show whether the monitor is to be started at the start or completion of the step.

### Monitor End Step

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

## Monitor Type

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

- **Duration**

The monitor measures a duration; this Monitor measures the time taken from the start of the monitor to the end of the monitor, as specified in the scope of the monitor.

- **Process Value**

The monitor measures the process value. The process value is the value contained in the `process value` attribute as defined within the process definition. For example, you may wish to measure the value of each order (assuming that your process definition holds the order value within the `process value` attribute).

You have the capability of adding your own Monitor Types, known as `Custom` types. (See [Chapter 2, Custom Monitors](#) for more details and examples.)

## Statistics Collection

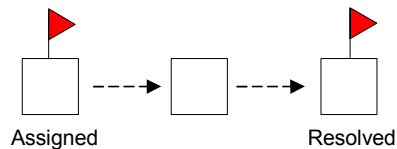
From BPI version 8 onwards, statistical processing of monitors is performed within BAC. The `Statistics Collection` settings are hardcoded. The `Collection Interval` is common to all monitors and processes and is defined within the BPI Admin Console as the collection interval.

### Recording Monitor Values

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

For example, suppose you configure a monitor, and set the monitor scope to be from the start of the step `Assigned` to the end of the step `Resolved`, as shown in [Figure 3](#).

**Figure 3 Example Monitor Definition**



When a process instance enters the `Assigned` step, a new monitor instance is instantiated and BPI records the time this monitor instance starts. If the monitor is defined as a `process value` monitor, then BPI also records the current process value.

When this process instance eventually completes the `Resolved` step, BPI completes the associated monitor instance and records the time it completes. If the monitor is defined as a `duration` monitor, then BPI calculates the overall duration that this monitor instance has taken to complete. If the monitor is defined as a `process value` monitor, then BPI records the final process value of this monitor instance.

Where the monitor is of type `process value`, BPI also records any changes that occur to the process value property of the process instance. This allows up to date information to be forwarded to BAC to facilitate backlog value calculations to be performed.

As each monitor instance starts, BPI records the start time for the monitor instance. As each monitor instance completes, BPI determines the duration, or process value, for the monitor. This duration, or value, represents the

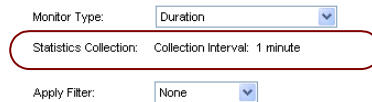
“value” for this completed monitor instance. So the “value” of the monitor instance is recorded at the completion of each monitor instance, as defined by the scope of the monitor.

## Business Availability Center and Collection Intervals

From BPI 8.00, BPI is configured to send data samples to Business Availability Center and so the statistics collection setting within the Monitor Definer is already configured for you.

For example, when you are creating a monitor, the statistics collection part of the dialog might look as shown in [Figure 4](#) on page 19.

**Figure 4 Statistics Collection with BAC**



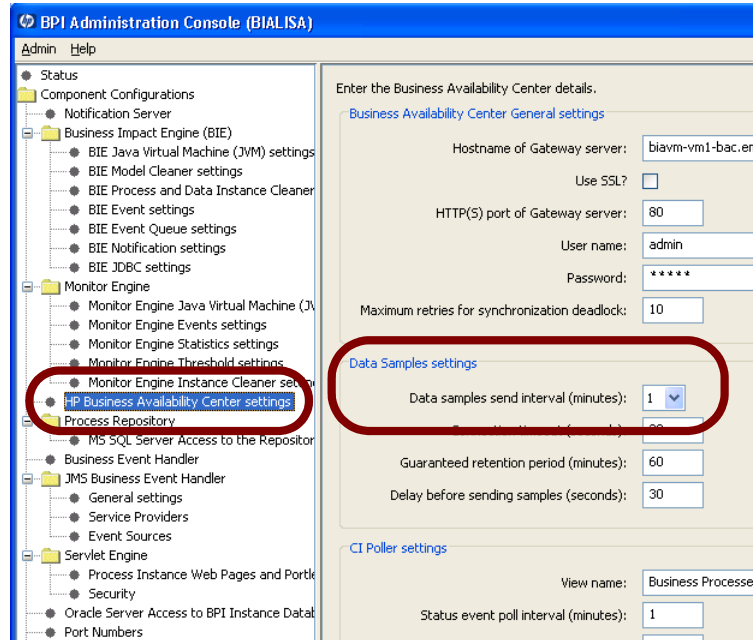
The screenshot shows a dialog box with three rows of settings. The first row is 'Monitor Type:' with a dropdown menu set to 'Duration'. The second row is 'Statistics Collection:' with a text field containing 'Collection Interval: 1 minute'. The third row is 'Apply Filter:' with a dropdown menu set to 'None'. A red rounded rectangle highlights the 'Statistics Collection:' row.

Notice that the collection interval is fixed for all monitors.

The collection interval shown for the monitor corresponds to the `Data sample send interval` that is configured for the Business Availability Center settings within the BPI Administrator Console - refer to the *BPI System Administration Guide* for details of how to configure the Business Availability Center settings.

For example, the collection interval shown in [Figure 4](#) is set to 1 minutes. This is because the `Data samples send interval` that is configured within the BPI Administration Console for the Business Availability Center data samples settings, has been set to 1 minute, as shown in [Figure 5](#).

**Figure 5 BPI Admin Console - BAC Data Samples Send Interval**



The integration to Business Availability Center is configured within the BPI Administration Console. The collection interval for all your monitors is set to the time period specified for the Data samples send interval.

If you change the data samples send interval within the BPI Administration Console, then the collection interval of all of your monitors is automatically updated to this newly configured send interval time period. This update of the monitor collection interval occurs the moment you apply the changes within the BPI Administration Console - there is no need to restart the Monitor Definer or any BPI services.

## Filters, Groups and Deadlines

Let us consider the remaining options you can specify when creating a monitor as shown in [Figure 3](#) on page 18. The remaining options are shown again here in [Figure 6](#).

**Figure 6 Creating a new Monitor - continued**



Apply Filter:

Group Results By:

Deadline Property:

### Apply Filter

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

By selecting a filter, you are able to restrict the number of monitor 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 monitor you must make sure that the attribute(s) tested within your filter actually have values at the time each monitor instance is started. Otherwise, your filter always evaluates to false and no monitor instances ever start.

See [Filters](#) on page 23 for more details about creating monitor filters.

## Group Results By

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

For example, suppose you are defining the monitors for an airport that has four flight terminals. You might choose to group the results by the airport terminal. This allows 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 Monitor Definer displays an additional field called `Group Name`. This allows you to specify a display name to be used within the BPI Business Process Monitor Graph screen when displaying the monitor data in groups.

Within BAC you are able to display the monitor data in overall terms, or in groups. When displaying the monitor data in groups BAC uses your `Group Name` text in the heading of the graphs.

## Deadline Property

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

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

For example, suppose you have a process that measures mortgage applications. You want to set a threshold that raises a violation if a mortgage does not reach a certain step in the process within a number of days agreed with the customer. By storing the agreed date as a data attribute and then setting the `Deadline Property` to that data attribute within the data definition, you can specify such a threshold.

# Filters

When creating, or modifying, a monitor 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 BPI Modeler (see *BPI Reference 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 Monitor Definer, once you have selected your process (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 shown in [Figure 7](#).

**Figure 7 Monitor Definer - Filter Creation**

Process Name: Call System (Deployed @ 14-Mar-2008 15:18:38)

Filter Name:  A unique name for your Filter

Filter Description:

Data Definition: 

- Calls/Data
  - Call\_Priority : String [2]
  - Customer\_ID : String [30]
  - Call\_Entry\_Date : Date
  - Engineer\_Name : String [50]

Select a Data Definition property to paste

Filter Expression:

Let us 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 process.

## 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 process's related data definition and must produce a true or false result.

To help you formulate your filter expression, the process'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 process'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 monitor filter, make sure that data attribute(s) you use within your filter expression have values at the start of the monitor when each process instance is running. That is, you cannot filter on (for example) `Customer_Type == "Gold"` if at the start of the monitor, the process 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 Monitor

Once you have defined a monitor for a process, it is listed in the right-hand frame of the Monitor Definer, along with any other monitors you have defined.

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

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

- **Update** the monitor

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

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

You obviously need to be careful about using the `Update` button. If you are simply modifying the description of the monitor definition then the `Update` button is the button to use. However, you need to think carefully if you are modifying the monitor to add (for example) a filter, as this means 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 monitor data together.

- **Replace** the monitor

If you are modifying the monitor 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 monitor definition such that it now has a filter, it may not make sense to keep any previously collected monitor data. By modifying the monitor definition and clicking `Replace`, you are telling BPI to delete all previously collected monitor data for this monitor definition, and threshold violations, and start collecting afresh.

## Creating a Threshold

Once you create a monitor, the monitor data is collected and stored in the Engine database. You can use the BPI Application Health and Report pages to report on this data.

As well as defining monitors, the Monitor Definer also lets you define (create) **Thresholds**. For example, suppose you have a monitor that is measuring the time it takes for you to process your orders. You may wish to set up a threshold to alarm when any individual order is taking more than four days to be processed.

The Monitor Definer is only able to define “instance” based thresholds. That is, thresholds where you wish to trigger a violation if any particular monitor instance breaches a user-specified value. For example, trigger a violation for a monitor instance if it takes longer than two days to reach a particular step in the process. To set “statistical” thresholds, such as a threshold that measures whether a monitor instance has taken longer than the most recent average, you need to configure KPI objectives within Business Availability Center.

The data samples are sent from BPI to Business Availability Center. The BPI Monitor Engine calculates the monitor values and statistics for the most recent collection interval (which is the same as the data samples send interval) and sends this set of data to Business Availability Center to be stored against the corresponding configuration item (CI) within the uCMDB. You then use Dashboard administration to configure you KPI objectives for this data. For more details refer to the Dashboard Help.

## Creating a Threshold Using the Monitor Definer

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

To create a threshold:

- Select the process definition - in the left-hand navigator frame
- Select the monitor 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 appear as shown in [Figure 8](#).

**Figure 8 Monitor Definer - Create Threshold**

Process Name: Call System (Deployed @ 14-Mar-2008 15:18:38)  
Monitor Name: Call Assignment Time  
Monitor Type: Duration  
Monitor Scope: Single step  
Threshold Name:\*  *A unique name for your Threshold*  
Threshold Description:   
Threshold Type:\*  *Select a type to reveal relevant options*

The process and monitor details are automatically filled in for you.

You provide a name for this threshold. This name must be unique across all the thresholds for this process.

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 us consider the remaining options for this screen...

## Threshold Type

Let us first consider the complete list of threshold types available within the Monitor Definer, and then discuss in more detail the differences when the Business Availability Center data sample integration with BPI is configured.

The complete list of threshold types available within the Monitor Definer are as follows:

- Absolute Duration/Value
- Deadline
- Relative Duration/Value

## Absolute (Duration/Process Value/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 `Monitor Type` as defined for the underlying monitor. If the `Monitor Type` for the monitor is `Duration`, then the threshold type is called `Absolute duration`. If the `Monitor Type` for the monitor is `Process Value`, then the threshold type is called `Absolute process value`. If the `Monitor Type` for the monitor is a custom defined monitor (see [Chapter 2, Custom Monitors](#)) then the threshold type is called `Absolute value`.

### Threshold Measure

From BPI version 8.00 the threshold measure is can be for any specific instance. BAC KPI objectives can be used to threshold recent average values.

- Any Specific Instance

This allows you to set a threshold to raise a violation when any monitor instance takes longer/shorter than the set time. Or when the process value of any monitor instance is greater/smaller than a set value.

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

## Deadline

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

Setting a deadline threshold type allows you to measure whether any monitor instance has reached a deadline.

For example, you might say that you wish to raise a violation if a monitor 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 a violation if a monitor instance is still running and the current time has reached the date/time specified in this instance's `Deadline Property`.

## Relative (Duration/Process Value/Value)

A relative threshold allows you to set a threshold against the standard deviation of your monitor values for all sample data collected for the Monitor.

The text for the threshold type is dependent on the Monitor Type as defined for the underlying monitor. If the Monitor Type for the monitor is Duration, then the threshold type is called Relative duration. If the Monitor Type for the monitor is Process Value, then the threshold type is called Relative process value. If the Monitor Type for the monitor is a custom defined monitor (see [Chapter 2, Custom Monitors](#)) then the threshold type is called Relative value.

### Threshold Measure

The Threshold Measure is for Any Specific Instance.

- Any Specific Instance

This allows you to set a threshold to alarm when any monitor instance duration takes longer/shorter than a number of standard deviations. Or when the process value of any monitor instance is greater/smaller than a number of standard deviations.

The standard deviation is calculated using all previous samples for the Monitor.

This threshold measure means that the moment a monitor instance reaches the threshold, a violation is triggered, it does not need to wait until the monitor instance completes.

## Warning/Minor/Major/Critical Violation

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

As a threshold crosses any specified violation value, a violation is raised within BPI. (See [Violations](#) on page 31 for further details.)

## Violation Message

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

## Violations

When thresholds are exceeded, violations are generated.

The violations are sent to two places:

- The Notification Server

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

- The BPI Engine Database

The violations are also written to the BPI Engine database. This allows the BPI Business Process Insight screens within BAC to report and show violations raised.

## Instance and Statistical Thresholds/Violations

As discussed above, BPI instance monitor thresholds are measured on an “individual instance” basis and BAC KPI objectives are measured on a “collection interval” basis.

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

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

## Violation Levels

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

If the violation is based on a statistical threshold, BPI additionally raises a Normal violation 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 Monitor Definer there are some options in the left-hand navigator frame.

## Refreshing the Monitor Definer

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

## Exporting Monitors

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

1. Select the process - in the left-hand navigator frame
2. Select `File->Export->Monitors` for selected process  
and save your monitor definitions, thresholds and filters to an external file name of your choice.

## Exporting Monitors for All Processes

If you have monitors defined for multiple processes, you can export all monitor definitions, thresholds and filters to a single external (zip) file by selecting `File->Export->Monitors` for all processes

## Importing Monitors

If you have an external (zip) file that contains monitor definitions, thresholds and filters as exported from a Monitor Definer, you can import them into your Monitor 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 monitors to import
5. Click the `Import Definitions` button
6. Click `OK`

If your import file contains *custom* monitor definitions then you must have previously installed the necessary SQL stored procedures used by these custom monitors. If these stored procedures are not present within the BPI Engine database at the time of the monitor import, the import fails. (See [Chapter 2, Custom Monitors](#) for more details about how to create and work with custom monitors.)

## Collection Intervals

When you import a monitor from an earlier version of BPI where the statistic collection interval could be specified on the individual monitors then this collection interval continues to be stored by BPI. However, the collection interval that appears within the Monitor Definer, and therefore the collection interval used for this monitor, is set to the value of the data sample send interval.

In other words, when a monitor is imported, the imported collection interval is stored with that monitor but it will not be used.

## Monitor Definer Help

If you select `Help->Help Topics` the Monitor Definer help system is displayed in a separate window.

# Lab - Defining Monitors

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

## 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 support 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

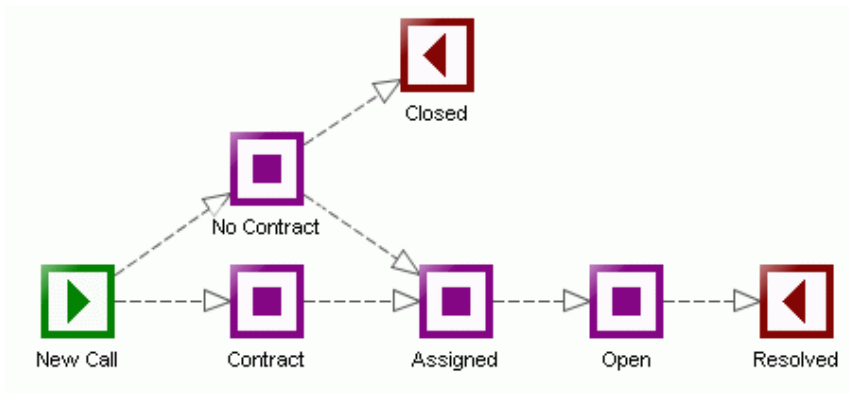
The call center would like to be able to visually see their call system on a dashboard, showing where calls are at any time. They would also like to collect the following measurements:

- The time it takes to assign contract calls  
Raising a violation if the assignment time takes too long, and measuring the backlog of calls waiting to be assigned over time.
- The time it takes to process a call once it has been assigned  
Raising a violation 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 recent average resolution rate.

## Defining the Process

The process definition for this lab is already defined for you. The process diagram is as shown in [Figure 9](#) on page 36.

**Figure 9 Call System Process Diagram**



To load up this process definition:

- Use the BPI Administration Console and start all Components.
- Run the BPI Modeler.
- Locate the file: labs\CallSystem.zip  
Import this file into the BPI Modeler.
- Deploy the process: Call System

## Understanding the Process

Now that the process definition is loaded into the BPI 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:

<b>Name</b>	<b>Type</b>	<b>Constraint</b>	<b>Unique</b>
<code>Call_ID</code>	String	30	Yes
<code>Customer_ID</code>	String	30	
<code>Call_Status</code>	String	20	
<code>Engineer_Name</code>	String	50	
<code>Call_Priority</code>	String	2	
<code>Call_Entry_Date</code>	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 steps.

### 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 the 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

## Data collection Intervals

The data collection interval used must be carefully considered. If the chosen period is too long then data may not be available with sufficient immediacy. For example a 1 day collection interval would mean that data from today would not be forwarded to BAC till midnight and so any KPIs and objectives would not indicate what is happening within the process today.

On the other hand, if the period is set too short then plots of KPIs will be excessively noisy. If the period is left at 1 minute then there are likely to be many samples where there were no data points. For this lab there may not be any Priority 3 calls that complete in any given minute.

For this lab set the collection interval to 5 minutes.

- On the BPI server run the BPI Admin Console
- Select the HP Business Availability Center settings screen from the navigation pane
- Change the Data sample send interval to 5 minutes

## Defining the Monitors and KPIs

The monitors as specified by the customer are as follows:

- The time it takes to assign contract calls

Raising a violation if the assignment time takes too long, and measuring the backlog of calls waiting to be assigned over time. Remember that backlogs are measured using BAC KPIs and not directly by BPI monitors and thresholds.

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

Raising a violation 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 recent average resolution rate.

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

## Time to Assign Contract Calls

You need to define a monitor that measures the time spent in the Contract step of the process. That is, a monitor from the start of Contract to the end of Contract.

To define this...

- Start up BAC and login as an administrator
- From the Admin menu select Business Process Insight
- Select BPI Monitor Definer tab
- Select the Call System process
- Click on Create Monitor
- Define the monitor as follows:
  - Monitor Name: Call Assignment Time
  - Monitor Description: Measure time to assign a contract call
  - Monitor Scope: Single Step
  - Monitor Start Step: Contract
  - Monitor Type: Duration
  - Apply Filter: None
  - Group Results By: None
  - Deadline Property: None
- Click OK

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

In the real world you would probably want to choose a longer 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 a Violation if Duration Too Long

You now need to define a threshold for this monitor to raise a violation 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 Monitor Definer, select the `Call Assignment Time` monitor
- Click `Create Threshold`
- Create a threshold as follows:
  - `Threshold Name`: `Call Assignment SLA`
  - `Threshold Type`: `Absolute duration`
  - `Duration Units`: `Minutes`
  - `Test Condition`: `Greater than or equal to`
  - `Critical Violation`: `6`
  - `Violation message`: `Call assignment time has exceed the SLA`
- Click `OK`

## Call Assignment Backlog

The customer also wants you to monitor the backlog of calls waiting to be assigned. If more than 15 calls are awaiting assignment then it should be considered to be a major violation. Should the backlog exceed 25 it would be considered critical:

- From the Admin menu select `Dashboard`
- Select the `Call System` process from the left hand navigation pane and expand its contents
- Select the `Call Assignment Time` monitor

The monitor will have three automatically defined KPIs, for `Backlog`, `Volume` and because this is an absolute duration monitor, `Duration`. BAC also automatically defines `Backlog` and `Volume` KPIs to all nodes within the flow as node monitors. These `Backlog` and `Volume` KPIs default to measuring the process value attribute. In the case of this `Call Center` process there is no value attribute so these KPIs would always have a value of zero. However the `Business rule` used to calculate the KPI can be changed.

- Modify the Backlog KPI and change the Business rule and specify the customer's backlog monitoring objectives.
  - Edit the Backlog KPI by clicking on its edit button
  - Change the Business rule to BPI Monitor Backlog Count Rule
  - Change the Operator to < since any backlog less than 15 is considered OK by the customer
  - Set the OK < value to 15
  - Set the Major < value to 25
  - Click on the Save button

The BPI monitor instance thresholds are displayed in a table on the Business Health screen. The BAC objectives are displayed graphically using a dial. It would be useful to graphically display a summary of the Call Assignment Times too. By setting a Objectives against the Duration KPI a dial will be displayed.

- Modify the Duration KPI and specify the customer's assignment time monitoring objectives.
  - Edit the Duration KPI by clicking on its edit button
  - Change the Operator to < r
  - Set the OK < value to 330 seconds
  - Set the Major < value to 360 seconds
  - Click on the Save button

Since there is no process value data attribute in this process change the Business rule for the monitors Volume KPI.

- Modify the Volume KPI and change the Business rule.
  - Edit the Volume KPI by clicking on its edit button
  - Change the Business rule to BPI Monitor Volume Count Rule
  - Click on the Save button

For the same reason it would be good to change the Business rules for all the automatically created Volume and Backlog KPIs for all the nodes in the process.

For each node listed under the Call System process in the left hand navigation pane:

- Open up the underlying hierarchy to expose the node's monitor
  - Edit the Backlog KPI by clicking on its edit button
  - Change the Business rule to BPI Monitor Backlog Count Rule
  - Click on the Save button
  - Edit the Volume KPI by clicking on its edit button
  - Change the Business rule to BPI Monitor Volume Count Rule
  - Click on the Save button

## Time to Process Assigned Calls

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

- The customer only wants to measure this for actual contract calls. So you want a filter that filters this monitor such that only contract calls (calls with a priority of 1, 2 or 3) are included.
- 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 return to the Monitor Definer and define the monitor...

But first...you must define the filter:

- Select the `Call System` process (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 monitor:

- Select the Call System process
- Click Create Monitor
- Define the monitor as follows:
  - Monitor Name: Call Processing Time
  - Monitor Scope: Multiple steps
  - Monitor Start Step: Assigned (start)
  - Monitor End Step: Resolved (complete)
  - Monitor Type: Duration
  - Apply Filter: Call On Contract
  - Group Results By: Call\_Priority
  - Group Name: Call Priority
  - Deadline Property: None
- Click OK

Now to define the thresholds...

### Raise a Violation if Call Processing Time target is not met

The support agreement is that contract calls must be resolved within a certain time from when they were assigned. 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 an instance threshold to issue a critical violation if the time to process a contract call is 35 minutes from the time the call was assigned. Let's also issue an additional minor violation if the call reaches 20 minutes from the time the call was assigned:

- Select the **Call Processing Time monitor**
- Click **Create Threshold**
- Create a threshold as follows:
  - **Threshold Name:** Call Time SLA
  - **Threshold Type:** Absolute Duration
  - **Minor Violation:** 20 Minutes After
  - **Critical Violation:** 35 Minutes After
- Click **OK**

In addition to defining an instance threshold which allows the customer to track the individual calls that are exceeding the set thresholds, it would be useful to specify BAC KPI objective to get a graphical feedback about which calls are exceeding the SLA.

- Return to the **Dashboard Administration** screen
- Find the KPIs for the newly defined **Call Processing Time monitor**
- Switch the **Backlog** and **Volume KPIs'** business rules to being the **Count** variant rather than the default value rule
- Edit the **Duration KPI** to define the objectives
  - **Operator:** <
  - **OK:** 1200 seconds
  - **Minor:** 1800 seconds
  - **Major:** 2100 seconds
  - Click **Save**

While in the Dashboard Administration screens, another implicit set of KPIs are created by BAC for the process as a whole, these are the Call System\_Monitor KPIs. As with the other automatically created ones, these KPIs default to using the process value. These all need to be changed to using the count version of the business rules.

### Call Processing Speed

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

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

Great! You should have the following monitors and thresholds defined for your Call System process:

Monitor: Call Assignment Time (Single step, duration)

Threshold: Call Assignment SLA (Absolute)

Monitor: Call Processing Time (Multiple step, duration)

Threshold: Call Processing Speed (Relative)

Threshold: Call Time SLA (Absolute)

Now that these monitors are defined within the Monitor Definer, they are active. Indeed, your monitor collection intervals may have already kicked into action and data may have already started to collect in the BPI 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 **Process Simulator**...

## The Process Simulator

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

The Process Simulator is located under the `BPI-server-install-dir\contrib\ProcessSimulator` directory.

To run the Process Simulator, you just need to double-click the `ProcessSimulator.bat` file.

If you want to learn more about how to use the Process Simulator, please refer to the PDF documentation that is provided in the `contrib\ProcessSimulator` directory.

## Running the Call Center

For this lab, you are provided with a pre-defined set of test cases to run within the Process Simulator. These test cases inject contract and non-contract calls through the `Call System` process, allowing you to run the BPI Application and see the calls and the monitors.

Let's load up the Process Simulator:

- cd to the directory: `BPI-install-dir\contrib\ProcessSimulator`
- Run (Double-Click) the script: `ProcessSimulator.bat`  
This runs the Process Simulator in a separate GUI
- In the Process Simulator GUI, select: File->Open Test Suite
- Open the file: `labs\Calls.xml`

The Process Simulator should load this file and you should now see a number of test cases shown in the top-left-hand corner of the Process 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 Process Simulator with a set of test cases, let's run them:

- Within the Process 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 BPI Application

- With the Process Simulator still running, start up the BAC and from the `Applications` menu select `Business Process Insight`. Go to the `Health` tab and choose the `Call System` process from the drop down menu.

In the lower half of the screen select the `Business Health` tab.

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

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

The BPI Business Health displays the following graphics for defined thresholds:

- A dial for each KPI with defined objectives
- A table of instance violations for each instance threshold

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

- Let the Process Simulator run for five minute...
- After the first collection interval is complete you should see values displayed in the dials for your monitors.

For example, after a while the threshold dials and violation tables might look as shown in [Figure 10](#).

### **Figure 10 Monitor Threshold Summary Table**

## More Graphs

- Click on the Duration Call Processing Time dial and/or the Backlog Call Assignment Time 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 monitors:

- In the BPI Business Health screen, clicking on the Duration Call Processing Time dial pops up a page with historical data for the minimum, average and maximum durations. Yours will not be very exciting at this stage. At the end of the graph there will be 3 small rings representing the day's data points.
- Click on the ring for the day's average duration.

By default this shows you a graph displaying the Average/Minimum/Maximum values for each collection interval.

- Now select `Graph by group`

This will popup a new graph window showing you the average call processing time for calls in each of the priority groups. Note that these graphs are plotting the raw data points, whereas the graph on the previous screen defaulted to showing the data aggregated over 1 hour intervals. If you did not change the data collection interval to 5 minutes these graphs will be very noisy with many points at zero.

- Change the `Data Aggregation` method to `Completed - Count`

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 measurements. At the end of this lab you should leave the Process Simulator running and thus, collect more data that you can drill into during the next lab.

## Adjusting the Process Simulator

- Once the Business Health screen is showing violations for the `Call Assignment SLA` threshold you need to go back to the Process Simulator and speed up the time it takes for calls to be assigned.
- In the Process 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 Process Simulator to handle these events at one event every 5000 milliseconds (five seconds) and thus the calls currently in the `Contract` and `No Contact` steps are processed more quickly.

Back in BAC, you should notice that calls waiting in the `Contract` step 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 Time` dial swings up).

## End of The Lab

- At this point, you have finished the lab! However, you should **leave the Process Simulator running**.
- Over time, make periodic adjustments to the Process 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
- You can view the monitor data in BAC and see the affects of your changes.

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

---

## 2 Custom Monitors

What if the BPI monitors do not give you the exact monitor 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 process value property? Maybe you need to measure the percentage of orders that have gone through a particular step?

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

Custom monitors are written as stored procedures within the BPI Engine database. Any data item that can be returned by a stored procedure can be used as a customer monitor. Given a full understanding of the schema of this database it would be possible to return any desired data as a custom monitor. However, since the schema is not published this chapter will cover the two most likely situations:

- Returning a data property other than the process value property.
- Returning the path taken through the process flow. BPI monitors are only concerned with individual instances, but BAC can use these individual instance paths to calculate the percentages of instances using a particular path through the process.

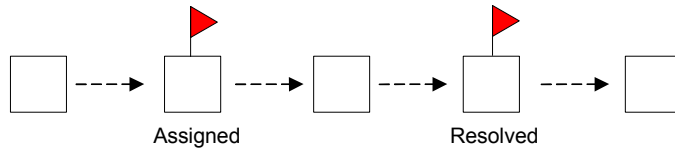
If you need to define other custom monitors please contact your support representative for further assistance on database structure.

# Monitor Scope

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

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

**Figure 11 Monitor Scope**



When a process instance enters the `Assigned` step, a new monitor instance is started. When this process instance leaves the `Resolved` step, the monitor instance is completed.

The start and end of the monitor instance is captured by triggers on the Business Impact Engine tables. These triggers capture the starting monitor value and write this to the BPI Engine database. The triggers also capture the final monitor value at completion and write this to the BPI Engine database.

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

# Defining a Custom Monitor

There are two main parts to defining a custom monitor:

1. The stored procedure

Writing the stored procedure to determine the actual monitor value.

2. The Monitor Definer

Making this stored procedure available within the Monitor Definer such that users can define monitors that use this stored procedure.

## The Stored Procedure

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

```
Monitor Id
Process Id
Process Instance Id
Process Instance Identifier
Process Instance start time
Process Instance start time in milliseconds
Process Value
Data Definition ID
Data Instance ID
Event type
Event time
Event time in milliseconds
Index
```

Many of these parameters refer to identifiers that make it possible to extract data associated with the process instance from the BPI Impact Engine database.

Your stored procedure is passed these parameters and then must pass them on to another supplied stored procedure, `METRIC_SEND_EVENT_V2`, along with the desired monitor value. This procedure sends the monitor data to the BPI Monitor Engine.

## Parameter Values

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

- Monitor Id

This is the database unique ID for the monitor.

- Process Id  
Process Instance Id  
Process Instance Identifier  
Process Instance start time  
Process Instance start time in milliseconds  
Process Value

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

- Data Definition ID

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

- Data Instance ID

This is the unique ID of this process'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 monitor instance.

- Completed

This is the completion of the monitor instance.



- Event time  
Event time in milliseconds  
The time that this monitor is being recorded.
- Index  
It may be that a process instance passes through the step that starts the monitor, more than once. You can use the index value to code for this.  
Zero (0) means that it is the first time through this step. One (1) means this is the second time through this step, 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:

```
@Monitor_ID          NVARCHAR (36) ,
@Process_ID          NVARCHAR (36) ,
@ProcessInstance_ID  NVARCHAR (36) ,
@ProcessInstIdentifier NVARCHAR (40) ,
@ProcessInstStartTime DATETIME,
@ProcessInstStartTimeLongMillis BIGINT,
@ProcessValue        FLOAT,
@DataDefinition_ID   NVARCHAR (36) ,
@DataInstance_ID     NVARCHAR (36) ,
@EventType           NVARCHAR (12) ,
@EventTime           DATETIME,
@EventTimeLongMillis BIGINT,
@Idx                 INTEGER
```

### For Oracle:

```
Monitor_ID          VARCHAR2,
Process_ID          VARCHAR2,
ProcessInstance_ID  VARCHAR2,
ProcessInstIdentifier VARCHAR2,
ProcessInstStartTime DATE,
ProcessInstStartTimeLongMillis NUMBER,
ProcessValue        FLOAT,
DataDefinition_ID   VARCHAR2,
DataInstance_ID     VARCHAR2,
EventType           VARCHAR2,
EventTime           DATE,
EventTimeLongMillis NUMBER,
Idx                 NUMBER
```

## The Monitor Value

Once your stored procedure has determined the monitor 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_V2`. (The `_V2` at the end of the procedure name stands for “version 2”.)

The `METRIC_SEND_EVENT_V2` stored procedure accepts the following parameters:

```
Monitor Id
Process Id
Process Instance Id
Process Instance Identifier
Process Instance start time
Process Instance start time in milliseconds
Process Value
Data Definition ID
Data Instance ID
Event type
Event time
Event time in milliseconds
Monitor Value
Index
```

These are the same parameters as were passed into your stored procedure, with the addition of the `Monitor 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_V2` stored procedure:

**For MSSQL:**

```
EXECUTE METRIC_SEND_EVENT_V2 @Monitor_ID,
                              @Process_ID,
                              @ProcessInstance_ID,
                              @ProcessInstIdentifier,
                              @ProcessInstStartTime,
                              @ProcessInstStartTimeLongMillis,
                              @ProcessValue,
                              @DataDefinition_ID,
                              @DataInstance_ID,
                              @EventType,
                              @EventTime,
                              @EventTimeLongMillis,
                              @Value,
                              @Idx
```

### For Oracle:

```
METRIC_SEND_EVENT_V2 (Monitor_ID,  
    Process_ID,  
    ProcessInstance_ID,  
    ProcessInstIdentifier,  
    ProcessInstStartTime,  
    ProcessInstStartTimeLongMillis,  
    ProcessValue,  
    DataDefinition_ID,  
    DataInstance_ID,  
    EventType,  
    EventTime,  
    EventTimeLongMillis,  
    Value,  
    Idx);
```

## Monitor Backlog

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

Suppose the custom monitor that you are measuring always has a zero start value. In this case you might decide that you do not need to write any monitor record at the start of the monitor, and only write a monitor record at the completion of the monitor. That is, only calculate the monitor value and call `METRIC_SEND_EVENT_V2` if the `EventType` is `Completed`.



Only writing completion monitor records cuts down on some monitor processing time and works just fine. However, if you choose to not write any monitor record at the start of the monitor, you are not able to observe and graph the backlog for your monitor. Without any start-of-monitor records there is no way to show how many monitor instances are currently active and hence no way to represent the monitor backlog. So if you want to be able to monitor your custom monitors, including the backlog, then call `METRIC_SEND_EVENT_V2` for **all** invocations of your stored procedure.

Remember that by default BAC creates an implicit KPI for the monitor's backlog.

## Stored Procedure Ownership

When you create your custom monitor stored procedure, you need to ensure that your stored procedure is owned by the same user that owns the BPI data tables. The default user name for this is `hpbpiuser`, however this name is configurable at BPI installation time.

If your custom monitor stored procedure is not owned by the same user as the BPI data tables, the BPI Business Impact Engine may not be able to invoke it, and this may cause the BPI Business Impact Engine to log errors and be unable to execute.

## The Monitor Definer

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

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

## Defining a Custom Type

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

The `METRIC_CustomTypes` table has the following columns:

- `CustomMetricName`

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

- `CustomMetricDescription`

You provide a description for your custom monitor. 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 BPI Business Health screens when displaying the values of this custom monitor.
- ParameterVersion  
This needs to be set to the number 2.

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

```
INSERT INTO METRIC_CustomTypes
    (CustomMetricName,
     CustomMetricDescription,
     CustomSPName,
     ValueUnits,
     ParameterVersion)
VALUES
    ('Calls Resolved ON Contract',
     'Calculates percentage calls resolved ON contract.',
     'BPI_Contract_Resolved_Calls',
     'Percent',
     2);
```

where:

- The new monitor type is called Calls Resolved ON Contract.
- The associated stored procedure that writes out the monitor values is called BPI\_Contract\_Resolved\_Calls.
- The display text for the measurement units for this monitor type is Percent.

## Example - A Data Property

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

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

When a data definition is deployed in the Modeler an associated data table is created in the BPI Impact Engine's database. The columns in the database use the data property names. So your stored procedure is going to need to find the name of the associated data table. Each entry in the table is indexed by a unique `Data Instance ID`.

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:

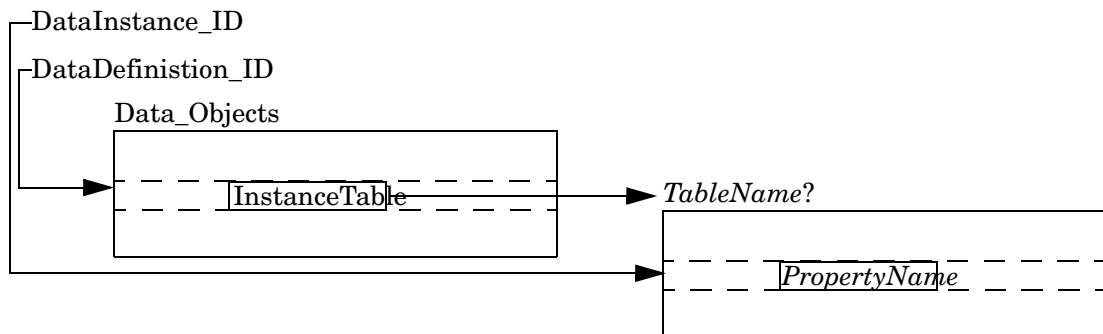
- `Data Definition ID`

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

- `Data Instance ID`

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

You can use the `Data Definition ID` to get the name of the associated 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 both an MSSQL example, and an Oracle example, for how to write the stored procedure.

### MSSQL Example

Here is an example stored procedure for MSSQL:

```
CREATE PROCEDURE BPI_Discount_Value
    @Monitor_ID NVARCHAR(36), @Process_ID NVARCHAR(36),
    @ProcessInstance_ID NVARCHAR(36), @ProcessInstIdentifier NVARCHAR(40),
    @ProcessInstStartTime DATETIME, @ProcessInstStartTimeLongMillis BIGINT,
    @ProcessValue FLOAT, @DataDefinition_ID NVARCHAR(36),
    @DataInstance_ID NVARCHAR(36), @EventType NVARCHAR(12),
    @EventTime DATETIME, @EventTimeLongMillis BIGINT, @Idx INTEGER
AS
    DECLARE @DiscountValue FLOAT
    DECLARE @DataTableName NVARCHAR(128)
    DECLARE @DataSqlStmnt 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 @DataSqlStmnt = 'insert into Temp_BPI_CustomTable(tValue) ' +
        ' select Discount from ' + @DataTableName +
        ' where id = '' + @DataInstance_ID + ''''
    EXECUTE (@DataSqlStmnt)
    SELECT @DiscountValue = tValue from Temp_BPI_CustomTable
    DROP TABLE Temp_BPI_CustomTable

    -- (3) Now write the monitor value.

    EXECUTE METRIC_SEND_EVENT_V2 @Monitor_ID, @Process_ID, @ProcessInstance_ID,
        @ProcessInstIdentifier, @ProcessInstStartTime,
        @ProcessInstStartTimeLongMillis, @ProcessValue,
        @DataDefinition_ID, @DataInstance_ID, @EventType,
        @EventTime, @EventTimeLongMillis, @DiscountValue, @Idx

END;
```

where:

- This defines a stored procedure called `BPI_Discount_Value`.
- Step (1) shows the SQL to determine the name of the related data table. This is the data table that holds the process instance's related data record.
- Step (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 can create a temporary table and use the `Execute()` command to execute the select statement.

- Step (3) sends the monitor result to the Monitor Engine.



The above MSSQL example may have performance issues. The creating and deleting of temporary tables within MSSQL can have a significant impact on your system's performance!



## Oracle Example

Here is an example stored procedure for Oracle:

```
CREATE or REPLACE PROCEDURE BPI_Discount_Value(
  Monitor_ID VARCHAR2, Process_ID VARCHAR2, ProcessInstance_ID VARCHAR2,
  ProcessInstIdentifier VARCHAR2, ProcessInstStartTime DATE,
  ProcessInstStartTimeLongMillis NUMBER, ProcessValue FLOAT,
  DataDefinition_ID VARCHAR2, DataInstance_ID VARCHAR2,
  EventType VARCHAR2, EventTime DATE, EventTimeLongMillis NUMBER,
  Idx NUMBER)
IS
  DiscountValue FLOAT;
  DataTableName VARCHAR2(128);
  DataSqlStmnt VARCHAR2(256);
BEGIN
  -- (1) Use the DataDefinition_ID to get the name of the data table
  select InstanceTable into DataTableName
     from Data_Objects do
     where do.model_id = BPI_Discount_Value.DataDefinition_ID;

  -- (2) Now use DataInstance_ID to look up the actual data record
  --     and pull out the discount column value
  DataSqlStmnt := 'select Discount from ' || DataTableName || ' where id = :1';
  EXECUTE IMMEDIATE DataSqlStmnt INTO DiscountValue using DataInstance_ID;

  -- (3) Now write the monitor value.
  METRIC_SEND_EVENT_V2(Monitor_ID, Process_ID, ProcessInstance_ID,
    ProcessInstIdentifier, ProcessInstStartTime,
    ProcessInstStartTimeLongMillis, ProcessValue,
    DataDefinition_ID, DataInstance_ID, EventType,
    EventTime, EventTimeLongMillis, DiscountValue, Idx);
END;
```

where:

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

- Step (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 Discount into DiscountValue
   from :DataTableName dot
   where dot.id = :DataInstance_ID;
```

However, Oracle does not let you issue a select statement in this form where the table name is a variable. Hence you need to build the SQL statement in a variable, and then execute this variable passing in the DataInstance\_ID.

- Step (3) sends the monitor result to the Monitor Engine.

### Modifying the example to recover your own data attributes

The supplied example will retrieve a particular example data attribute. This example can be used as a template to retrieve any data attribute from any data definition with only minor modifications.

```

CREATE or REPLACE PROCEDURE BPI_Discount_Value (1
  Monitor_ID VARCHAR2, Process_ID VARCHAR2, ProcessInstance_ID VARCHAR2,
  ProcessInstIdentifier VARCHAR2, ProcessInstStartTime DATE,
  ProcessInstStartTimeLongMillis NUMBER, ProcessValue FLOAT,
  DataDefinition_ID VARCHAR2, DataInstance_ID VARCHAR2,
  EventType VARCHAR2, EventTime DATE, EventTimeLongMillis NUMBER,
  Idx NUMBER) (2
IS
  DiscountValue FLOAT;
  DataTableName VARCHAR2(128);
  DataSqlStmt VARCHAR2(256);
BEGIN

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

  select InstanceTable into DataTableName
     from Data_Objects do (1
        where do.model_id = BPI_Discount_Value.DataDefinition_ID;

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

  DataSqlStmt := 'select Discount from ' || DataTableName || ' where id = :1';
  EXECUTE IMMEDIATE DataSqlStmt INTO DiscountValue using DataInstance_ID;

  -- (3) Now write the monitor value.

  METRIC_SEND_EVENT_V2(Monitor_ID, Process_ID, ProcessInstance_ID,
    ProcessInstIdentifier, ProcessInstStartTime,
    ProcessInstStartTimeLongMillis, ProcessValue,
    DataDefinition_ID, DataInstance_ID, EventType,
    EventTime, EventTimeLongMillis, DiscountValue, Idx); (2
END;

```

The values to be modified to suit your own example would be:

- ① The name of the stored procedure. Each custom monitor needs its own uniquely named procedure.
- ② The variable name used within the procedure, the name of DiscountValue is appropriate in this example but might not be in yours.
- ③ The name of the data property from the data definition defined within the modeler.

## The Custom Monitor Type Definition

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

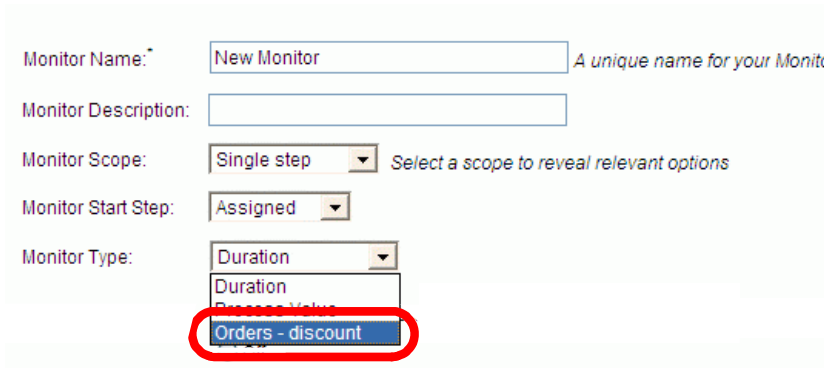
You create an entry in the `METRIC_CustomTypes` table defining a name for your custom monitor and linking this to your stored procedure.

For example:

```
INSERT INTO METRIC_CustomTypes (CustomMetricName,  
                                CustomMetricDescription,  
                                CustomSPName,  
                                ValueUnits,  
                                ParameterVersion)  
VALUES('Orders - discount',  
      'Calculates the percentage discount given on this order.',  
      'BPI_Discount_Value',  
      'Percent',  
      2);
```

When you define a monitor in the Monitor Definer you are now able to select your stored procedure by selecting the Monitor Type of `Orders - discount`, as shown in [Figure 12](#).

**Figure 12 Custom Monitor Type**



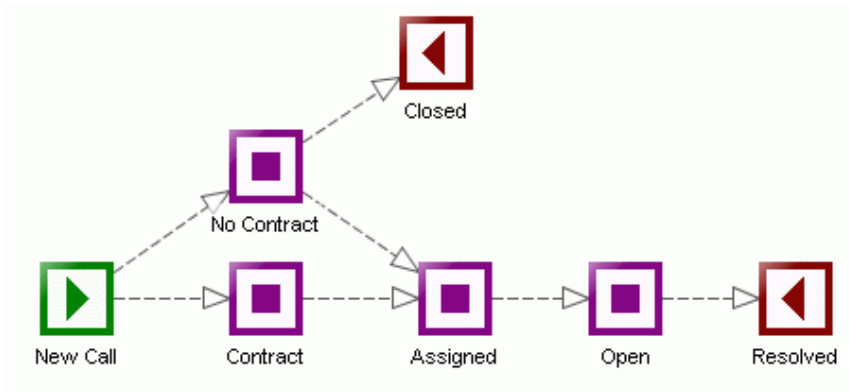
The screenshot shows the Monitor Definer interface with the following fields:

- Monitor Name:  A unique name for your Monitor
- Monitor Description:
- Monitor Scope:  Select a scope to reveal relevant options
- Monitor Start Step:
- Monitor Type: 
  - Duration
  - Process Value
  - Orders - discount** (highlighted with a red circle)

## Example - Percentage Path Process

Suppose you have a process that looks as shown in [Figure 13](#).

**Figure 13 Process with Multiple Paths**



This process is the Call System process from an earlier lab ([Lab - Defining Monitors](#) on page 35). 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 process and show these outcomes as percentages.

This requires defining some custom monitors.

## Monitor Scope

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

## The Stored Procedure(s)

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

The `nodes` and `node_instance` tables describe the nodes within your process flows. These two tables can be joined using their respective `node_id` columns. The `nodes.nodename` column holds the names of the nodes as defined in the modeler and `node_instance.flowinstance_id` can be matched to the `ProcessInstance_ID` parameter.

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

You write the stored procedures to return the value of either `0` or `100`. Remember that BPI monitors examine only single instances. When looking at a single instance of a process it either completely follows a particular path through the process, and scores `100%` or it completely didn't follow the path and scores `0%`. The BPI Monitor Engine can collect all the monitor values for the individual instances and forward them to BAC. BAC's statistics functions then average these values to generate the percentage of flow instances following the different paths.

Let's consider the stored procedure that is going to measure the process instances that have terminated at the `Resolved` step having come through the `Contract` step. The stored procedure uses the process instance Id passed in to find out whether this process instance actually terminated at the

Resolved step. It does this by accessing the `Node_Instance` table and looks to see if there is a completed step instance for the `Resolved` step for this process instance. If the process instance did terminate at the `Resolved` step, then the stored procedure looks to see if the `Contract` step is also completed for this process instance. If both of these tests are true, then this process instance terminated at the `Resolved` step having also completed the `Contract` step, and therefore the stored procedure returns a value of 100. If the process instance did not terminate at the `Resolved` step or did not complete the `Contract` step, the stored procedure returns a value of 0.

## Backlogs

In most situations it is useful to be able to measure the backlog for a monitor, i.e. to see how many, and possibly the value, of all uncompleted instances of a monitor. In the case of these process path monitors however this does not make sense. The monitors are all Whole Flow monitors, they all start when the process instance starts. But at this stage it is impossible to know where the instance will complete and so it does not make sense to view an active instance as a backlogged `CallResolvedOnContract` or `NonContractClosedCall` until it is known where the instance will complete. So for these examples monitor records will only be written on completion so that no backlog data is generated.

## Contract Resolved Calls (MSSQL example)

The stored procedure for counting the instances of contract calls that are resolved looks as follows<sup>1</sup> (for MSSQL):

1. We will present an Oracle example for this stored procedure afterwards. It will also be necessary to create procedures to test the other paths, to avoid repetition these will be shown in MSSQL only.

```

CREATE PROCEDURE BPI_Contract_Resolved_Calls
    @Monitor_ID NVARCHAR(36), @Process_ID NVARCHAR(36),
    @ProcessInstance_ID NVARCHAR(36), @ProcessInstIdentifier NVARCHAR(40),
    @ProcessInstStartTime DATETIME, @ProcessInstStartTimeLongMillis BIGINT,
    @ProcessValue FLOAT, @DataDefinition_ID NVARCHAR(36),
    @DataInstance_ID NVARCHAR(36), @EventType NVARCHAR(12),
    @EventTime DATETIME, @EventTimeLongMillis BIGINT, @Idx INTEGER
AS
    DECLARE @RetVal FLOAT
    DECLARE @Count1 FLOAT
    DECLARE @Count2 FLOAT
BEGIN
    -- For this flowinstanceID, is the 'Resolved' step completed
    --                               and the 'Contract' step completed
    -- If so = set value to 100
    -- If not = set value to 0

    -- (1) If this is the start of the monitor, then simply return.
    --     NOTE: This means that no backlog information is available.
    IF (@EventType = 'Started')
    BEGIN
        RETURN;
    END

    -- Assume a false outcome
    SET @RetVal = 0

    -- (2) Did the process instance terminate at the 'Resolved' step
    --     (For this flowInstanceID, is the 'Resolved' step '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 = @ProcessInstance_ID
        and (nodes.nodename = 'Resolved' and ni.status = 'Completed')

    -- Only do this check if this instance terminated at the 'Resolved' step.
    IF (@Count1 != 0)
    BEGIN
        -- (3) Check whether the 'Contract' step 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 = @ProcessInstance_ID
            and (nodes.nodename = 'Contract' and ni.status = 'Completed')
    END
END

```



```

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

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

EXECUTE METRIC_SEND_EVENT_V2 @Monitor_ID, @Process_ID, @ProcessInstance_ID,
    @ProcessInstIdentifier, @ProcessInstStartTime,
    @ProcessInstStartTimeLongMillis, @ProcessValue,
    @DataDefinition_ID, @DataInstance_ID, @EventType,
    @EventTime, @EventTimeLongMillis, @RetVal, @Idx
END;

```

where:

- Step (1) tests if this is the start of the monitor, simply return. You only wish to measure completed instances. This means that BAC is unable to show any backlog numbers for this monitor.
- Step (2) tests that this process instance has actually completed at the Resolved step.
- Step (3) tests that, if the process instance has completed at the Resolved step, did it also complete the Contract step.
- Step (4) tests whether all conditions have been met. If so, it sets the return value to be 100.
- Step (5) sends the monitor result to the Monitor Engine.

## Contract Resolved Calls (Oracle example)

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

```
CREATE or REPLACE PROCEDURE BPI_Contract_Resolved_Calls (
  Monitor_ID VARCHAR2, Process_ID VARCHAR2,
  ProcessInstance_ID VARCHAR2, ProcessInstIdentifier VARCHAR2,
  ProcessInstStartTime DATE, ProcessInstStartTimeLongMillis NUMBER,
  ProcessValue FLOAT, DataDefinition_ID VARCHAR2, DataInstance_ID VARCHAR2,
  EventType VARCHAR2, EventTime DATE, EventTimeLongMillis NUMBER,
  Idx NUMBER)
IS
  RetVal FLOAT;
  Count1 FLOAT;
  Count2 FLOAT;
BEGIN

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

  -- (1) If this is the start of the monitor, then simply return.
  -- NOTE: This means that no backlog information is available.
  IF (EventType = 'Started')
  THEN
    RETURN;
  END IF;

  -- Assume a false outcome
  RetVal := 0;

  -- (2) Did the process instance terminate at the 'Resolved' step
  -- (For this flowInstanceID, is the end step 'Completed')
  Count1 := 0;
  SELECT count(ni.flowinstance_id) INTO count1
    from node_instance ni, nodes
    where ni.node_id = nodes.node_id
    and ni.flowinstance_id = BPI_Contract_Resolved_Calls.ProcessInstance_ID
    and (nodes.nodename = 'Resolved' and ni.status = 'Completed');

  -- Only do this check if this instance terminated at the 'Resolved' step.
  IF (Count1 != 0)
  THEN
    -- (3) Check whether the 'Contract' step is also 'Completed'
    Count2 := 0;
    SELECT count(ni.flowinstance_id) INTO Count2
      from node_instance ni, nodes
```

```

        where ni.node_id = nodes.node_id
        and ni.flowinstance_id =
BPI_Contract_Resolved_Calls.ProcessInstance_ID
        and (nodes.nodename = 'Contract' and ni.status = 'Completed');
    END IF;

    -- (4) If it did go through the step(s) we asked for,
    --     set the return value to 100
    IF (Count1 != 0 AND Count2 != 0)
    THEN
        RetVal := 100;
    END IF;

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

    METRIC_SEND_EVENT_V2 (Monitor_ID, Process_ID, ProcessInstance_ID,
        ProcessInstIdentifier, ProcessInstStartTime,
        ProcessInstStartTimeLongMillis, ProcessValue, DataDefinition_ID,
        DataInstance_ID, EventType, EventTime, EventTimeLongMillis, RetVal, Idx);

END;
```

where:

- Step (1) tests if this is the start of the monitor, simply return. You only wish to measure completed instances. This means that BAC is unable to show any backlog numbers for this monitor.
- Step (2) tests that this process instance has actually completed at the Resolved step.
- Step (3) tests that, if the process instance has completed at the Resolved step, did it also complete the Contract step.
- Step (4) tests whether all conditions have been met. If so, it sets the return value to be 100.
- Step (5) sends the monitor result to the Monitor 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
    @Monitor_ID NVARCHAR(36), @Process_ID NVARCHAR(36),
    @ProcessInstance_ID NVARCHAR(36), @ProcessInstIdentifier NVARCHAR(40),
    @ProcessInstStartTime DATETIME, @ProcessInstStartTimeLongMillis BIGINT,
    @ProcessValue FLOAT, @DataDefinition_ID NVARCHAR(36),
    @DataInstance_ID NVARCHAR(36), @EventType NVARCHAR(12),
    @EventTime DATETIME, @EventTimeLongMillis BIGINT, @Idx INTEGER
AS
    DECLARE @RetVal FLOAT
    DECLARE @Count1 FLOAT
    DECLARE @Count2 FLOAT
BEGIN

    -- (1) If this is the start of the monitor, then simply return.
    -- NOTE: This means that no backlog information is available.
    IF (@EventType = 'Started')
    BEGIN
        RETURN;
    END

    -- Assume a false outcome
    SET @RetVal = 0

    -- (2) Did the process instance terminate at the 'Resolved' step
    -- (For this flowInstanceID, is the end step '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 = @ProcessInstance_ID
        and (nodes.nodename = 'Resolved' and ni.status = 'Completed')

    -- Only do this check if this instance terminated at the 'Resolved' step.
    IF (@Count1 != 0)
    BEGIN
        -- (3) Check whether the 'No Contract' step 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 = @ProcessInstance_ID
            and (nodes.nodename = 'No Contract' and ni.status = 'Completed')
    END

END
```

```

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

-- (5)

EXECUTE METRIC_SEND_EVENT_V2 @Monitor_ID, @Process_ID, @ProcessInstance_ID,
    @ProcessInstIdentifier, @ProcessInstStartTime,
    @ProcessInstStartTimeLongMillis, @ProcessValue,
    @DataDefinition_ID, @DataInstance_ID, @EventType,
    @EventTime, @EventTimeLongMillis, @RetVal, @Idx

END;

```

where:

- Step (1) tests if this is the start of the monitor, simply return. You only wish to measure completed instances. This means that BAC is unable to show any backlog numbers for this monitor.
- Step (2) tests that this process instance has actually completed at the Resolved step.
- Step (3) tests that, if the process instance has completed at the Resolved step, did it also complete the No Contract step.
- Step (4) tests whether all conditions have been met. If so, it sets the return value to be 100.
- Step (5) sends the monitor result to the Monitor 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
    @Monitor_ID NVARCHAR(36), @Process_ID NVARCHAR(36),
    @ProcessInstance_ID NVARCHAR(36), @ProcessInstIdentifier NVARCHAR(40),
    @ProcessInstStartTime DATETIME, @ProcessInstStartTimeLongMillis BIGINT,
    @ProcessValue FLOAT, @DataDefinition_ID NVARCHAR(36),
    @DataInstance_ID NVARCHAR(36), @EventType NVARCHAR(12),
    @EventTime DATETIME, @EventTimeLongMillis BIGINT, @Idx INTEGER
AS
    DECLARE @RetVal FLOAT
    DECLARE @Count1 FLOAT
BEGIN

    -- (1) If this is the start of the monitor, then simply return.
    --     NOTE: This means that no backlog information is available.
    IF (@EventType = 'Started')
    BEGIN
        RETURN;
    END

    -- Assume a false outcome
    SET @RetVal = 0

    -- (2) Did the process instance terminate at the 'Closed' step
    --     (For this flowInstanceID, is the end step '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 = @ProcessInstance_ID
        and (nodes.nodename = 'Closed' and ni.status = 'Completed')

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

    -- (4)

    EXECUTE METRIC_SEND_EVENT_V2 @Monitor_ID, @Process_ID, @ProcessInstance_ID,
        @ProcessInstIdentifier, @ProcessInstStartTime,
        @ProcessInstStartTimeLongMillis, @ProcessValue,
        @DataDefinition_ID, @DataInstance_ID, @EventType,
        @EventTime, @EventTimeLongMillis, @RetVal, @Idx
END;
```

where:

- Step (1) tests if this is the start of the monitor, simply return. You only wish to measure completed instances. This means that BAC is unable to show any backlog numbers for this monitor.
- Step (2) tests that this process instance has actually completed at the Closed step.
- Step (3) tests whether the condition has been met. If so, it sets the return value to be 100.
- Step (4) sends the monitor result to the Monitor Engine.

## The Custom Monitor Type Definitions

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

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

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

```
INSERT INTO METRIC_CustomTypes (CustomMetricName,
                                CustomMetricDescription,
                                CustomSPName,
                                ValueUnits,
                                ParameterVersion)
VALUES('Calls Resolved ON Contract',
       'Calculates the percentage calls resolved ON contract.',
       'BPI_Contract_Resolved_Calls',
       'Percent',
       2);

INSERT INTO METRIC_CustomTypes (CustomMetricName,
                                CustomMetricDescription,
                                CustomSPName,
                                ValueUnits,
                                ParameterVersion)
VALUES('Calls Resolved OFF Contract',
       'Calculates the percentage calls resolved OFF contract.',
       'BPI_Off_Contract_Resolved_Calls',
       'Percent',
       2);
```

```

INSERT INTO METRIC_CustomTypes (CustomMetricName,
                                CustomMetricDescription,
                                CustomSPName,
                                ValueUnits,
                                ParameterVersion)
VALUES('Calls Closed OFF Contract',
      'Calculates the percentage OFF contract calls closed.',
      'BPI_Off_Contract_Closed_Calls',
      'Percent',
      2);

```

When you define a monitor in the Monitor Definer, you are now able to select your stored procedures by selecting one of the defined Monitor Types as shown in [Figure 14](#)

**Figure 14 Custom Percentage Monitor Types**

The screenshot shows the 'Monitor Definer' interface with the following fields:

- Monitor Name:**  *A unique name for your Monitor*
- Monitor Description:**
- Monitor Scope:**  *Select a scope to reveal relevant options*
- Monitor Start Step:**
- Monitor Type:** 
  - Duration
  - Process Value
  - Calls Closed OFF Contract** (highlighted with a red circle)
  - Calls Resolved OFF Contract
  - Calls Resolved ON Contract

## Defining The Monitors

As there are three possible outcomes for the Call System process, you define three monitors. Each monitor must produce results for every process instance going through the Call System process, as this enables BAC to produce an average. So you define three monitors, each with a scope of Whole Process.

With each monitor definition, you select the Monitor Type from the three custom monitor types you defined.



For example, to configure the monitor that measures the percentage of contract calls that are resolved, you define the monitor as shown in [Figure 15](#).

**Figure 15 Monitor Definition - Resolved Contract Calls**

Please insert the file  
CallsResolvedOnContract\_Monitor.vsd  
here every time I try I'm getting errors  
from frame maker

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

## Defining KPIs and objectives

The BPI monitors report on the individual instances. To see percentages the data is forwarded to BAC and there averaged. To display this within the Business Health screens within BAC as dials, you must define objectives for the automatically created KPIs for each monitor.

### **Figure 16 Threshold Definition - Resolved Contract Calls**

Please insert the picture from

`CallsResolvedOnContract_EditKPI.vsd`

here, I keep getting errors from Frame when I try

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

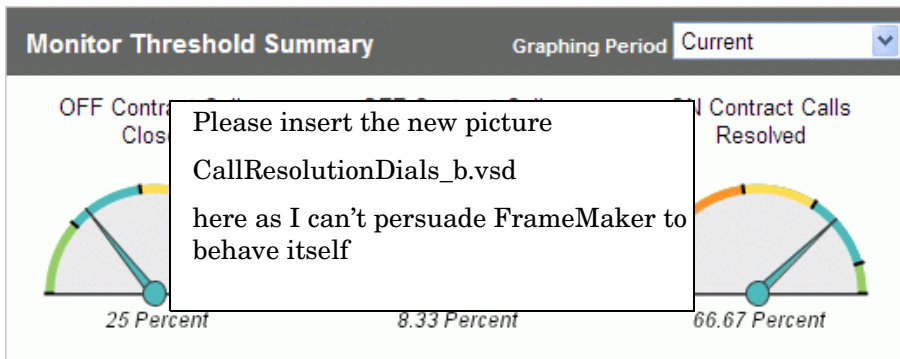
For each monitor defined in BPI, BAC automatically creates three KPIs, Backlog, Volume and the one for the monitor's value. In the case of the monitors for process paths there is no point in returning a backlog, it does not make sense. The volume will be the same for each path, as each path's monitor completes for all process instances. Therefore neither the Backlog or Volume KPIs need to be kept for these monitors, they should be deleted.

## The BPI Business Health screen

Now that the monitors and KPI objectives are defined, you can view the percentages within the BPI Business Health screen. When you view the Call System process, the three threshold dials show the percentages of your calls.

For example, you might see the dials as shown in [Figure 17](#).

**Figure 17 Business Process Health screen - Percentages**



These dials give you the current averages (or percentages in this case). By clicking on the dials you can obtain graphs of the percentages over time.

## SQL Errors

When you write a stored procedure to produce a custom monitor value, your stored procedure is invoked when the Business Impact Engine updates the BPI database tables. The updating of the BPI 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 process instance. It all depends on the underlying database being used by BPI.

### BPI on Oracle

If you are running BPI against an Oracle database then any problems within your monitor stored procedure are logged in the Business Impact Engine's log file, and this does **not** cause any problems to the Business Impact Engine or the processing of the process 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 values are not recorded correctly if the stored procedure is in error, but the process keeps running.

### BPI on MSSQL

If you are running BPI against an MSSQL database then any problems within your monitor 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 monitor and the current process instance to not be updated correctly.

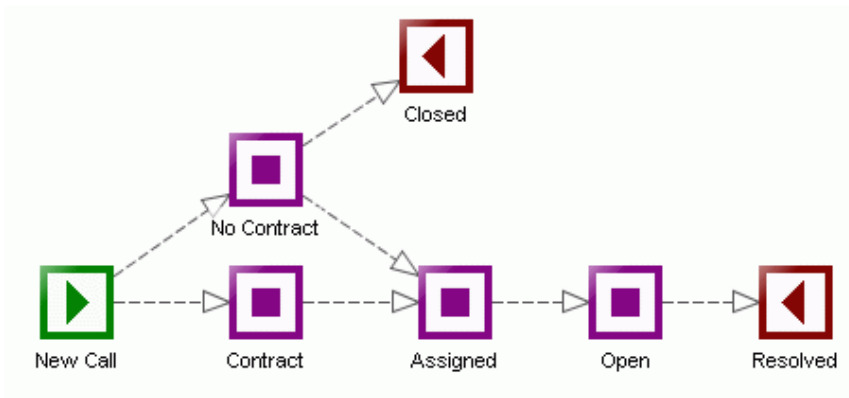
# Lab - Custom Monitors

This lab gives you experience of setting up custom monitors.

## The Call System Process

For this lab you are going to continue working with the Call System process you set up during the previous lab ([Lab - Defining Monitors](#) on page 35). The process diagram is as follows:

**Figure 18 Process Diagram**



This process 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 Monitors

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 process and show these outcomes as percentages.

The way to set up these monitors is fully explained earlier in this chapter in the section: [Example - Percentage Path Process](#) on page 69. 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 monitor type entries such that the Monitor 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 monitor 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 monitors to use these monitor types.

Define three monitors. All of these monitors are to be set to a scope of `Whole Process`. These monitors should invoke your stored procedures. Ensure that the collection interval is set to be five minutes, if it is not fix this in the BPI Admin Console.

- Go to the Admin -> Dashboard screen and remove the Backlog KPI for the three new monitors.
- Edit the Value KPIs and set objectives of between 0 and 100 for the three monitors.

— For `Calls Resolved On Contract` you could use:

- Operator: `>=`
- OK: `>= 80`
- Warning: `>= 70`
- Minor: `>= 60`
- Major: `>= 50`

- For Calls Closed Off Contract you could use:
  - Operator: <=
  - OK: <= 20
  - Warning: <= 30
  - Minor: <= 40
  - Major: <= 45
- For Calls Resolved Off Contract you could use:
  - Operator: <=
  - OK: <= 10
  - Warning: <= 15
  - Minor: <= 20
  - Major: <= 25

- Use the Process Simulator to send through more calls and view the percentages in the BAC.

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

When you have the BPI Business Health screen showing the percentages, take some time to look at the other monitors that you defined in the first lab.

If you kept the Process Simulator running since the first lab, you should be able to use the BPI Business Health screen to launch views of your monitors over time.

For example:

- Follow the Monitor Status Details link above the dials
- Choose the Calls Resolved on Contract monitor, this will popup a new window, explore the different tabs available here.
- Go to the Monitor Graph tab
- For these process path monitors the minimum and maximum percentages are obviously going to be 0 and 100%, so there is no point in displaying them. You can remove them from the graph by changing the Data Aggregation Method to just Average.

- Change the Graph Type to Line.
- Change the monitor to Call Processing Time
- Increase the Graphing Period
- Now select Chart by group to be: Yes

Feel free to explore all your monitors using the BPI Business Health screens of BAC and the many options available to you.

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



---

## 3 Further Topics

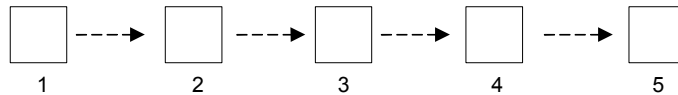
This chapter looks at additional topics to do with BPI monitors.

# Monitor/Threshold Activation

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

Suppose you have a process as shown in [Figure 19](#).

**Figure 19 Simple Process**



and you already have some process instances that are sitting in step 3.

You then define a monitor to measure from the start of step 2 to the end of step 4.

The process instances that are currently in step 3 are past the start of the monitor so you do not see any start monitors for these process instances. When these process instances move out of step 4, end monitor records are generated.

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

## Pre-Dated Events (BPI\_GeneratedDate)

As just explained, a monitor definition becomes active from the moment you click the `OK` button within the Monitor Definer. If you then sent in business events containing `BPI_GeneratedDate` values that are earlier than when you defined the monitor, any process activity resulting from these events would **not** create monitor instances. This is because these events, and therefore the resultant process/step activity, would be seen as having occurred **before** the monitor had been defined.

## Detecting Thresholds

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

Suppose you define a monitor for the process shown in [Figure 19](#) on page 90, and this monitor measures the time it takes for instances to move from the start of step 2 to the end of step 4. You then define a threshold to raise a violation if any specific instance takes longer than 10 minutes.

To test this out, you then send in a process instance and progress it to step 3. You leave the process instance in step 3 for 10 minutes and wait for the violation 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 a violation 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 violation appear.

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

You then assume that the violation is not going to happen, so you decide to progress the process instance and see what happens when it reaches the end of step 4. Sure enough, when the process instance exits step 4, you suddenly see a violation raised within the BPI Application Health page. This violation shows that the monitor instance did indeed take far longer than one minute to be completed.

So what happened?

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

The Monitor Engine constantly monitors active monitor instances in case they exceed a threshold while still active. That is, if you want a violation 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 violation raised the moment the instance has been running for more than your specified time period.

To identify active monitor instances that have been running for longer than a specified time period, the Monitor Engine monitors active monitor instances. But the Monitor Engine only needs to monitor the monitor 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 Monitor

Engine checks all monitor 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 a violation is raised.

The problem came about when you updated the threshold definition and changed the violation time period from 10 minutes to one minute. This meant that the Monitor 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 violation was raised the moment the monitor instance took longer than one minute.

Of course, when the monitor instance finally does complete, the Monitor Engine detects this and raises a violation to say that the monitor instance took too long.

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

## Instance Violations

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

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

## Deadline Monitor Value is Fixed

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

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

## Redeploying Processes/Monitors

Suppose you have defined monitors and thresholds on a deployed process. What happens if you need to redeploy the process?

If you redeploy a process that has monitors and thresholds defined, the new process is deployed and the deployer defines all the monitors and thresholds for this new process. In other words, the newly deployed process ends up having the same set of monitors and thresholds defined for it. However, the newly defined monitors and thresholds are new, and therefore the new monitors and thresholds do not inherit any history from the previous version of the monitors or thresholds. This is however not true for BAC statistics data. When a process is redeployed, the previous BAC data is kept and it is still possible, for instance, to display the previous statuses of the processes KPIs.

When redeploying a process that currently has monitors and thresholds defined for it, the deployer may not be able to set up all the monitors and thresholds. For example, if you are deploying a new version of the process where you have removed a step, then the deployer is not able to redefine any monitors that were based on that step. So when you redeploy a process, the deployer tries to set up all the monitors and thresholds as defined on the superseded process definition. However, depending on what changes are within the newly deployed process, you may not end up with all the monitors in place. You need to check your monitor and threshold definitions within the Monitor Definer to ensure that they are set up as you require on the newly redeployed process.

If you undeploy a process definition then any subsequent redeployment results in the process being deployed but with no monitor definitions. So if you are planning to redeploy the process again later it is important to save the monitor definition from the monitor definer. BAC KPIs are similarly removed when the process is undeployed and will not be available after the process is subsequently redeployed.



Once you have defined a set of monitors and thresholds for a deployed process, it is a good idea to export these monitor and threshold definitions from within the Monitor Definer (see [Exporting Monitors](#) on page 33) and save them to a file. This allows you to reinstate these monitor and threshold definitions should you ever have the scenario where you redeploy a process and find that the monitor and threshold definitions have not been applied to the newly deployed process.

## Deleting Process Monitors/Thresholds

If you undeploy a process from the BPI Business Impact Engine, the process is undeployed and any monitor, threshold, KPI and objective definitions for this process are deleted.

## Avoiding Notification Storms

When you are configuring your thresholds, and any notification subscriptions, you need to give some thought to how many violations this might end up sending. That is, configuring your thresholds to send out potentially hundreds of email notifications may not be helpful. If notifications are configured, remember that they are potentially sent out for each process instance that causes a violation. This could result in very large numbers of emails being sent. To limit this, BPI imposes a maximum number of notifications that will be sent out per threshold per violation polling interval.

The way that the notifications are sent out is that part of the BPI Monitor Engine polls a table of alerts in its database. By default the polling interval is 5 minutes but this can be configured using the Monitor Engine Threshold Setting option within the BPI Admin console. Also on this screen, in the BPI Admin console, is the option Maximum number of individual notifications per threshold per polling interval. This limit defaults to 10.

It can be better to work on the principal of sending out fewer notification emails but have the recipient then log into BAC and check the notification in there, that way they are able to see the details of violating instances and also see the effect on the business process.





