

Version 10 Release 0
June, 2016

IBM Opportunity Detect User's Guide

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 127.

This edition applies to version 10, release 0, modification 0 of IBM Opportunity Detect (product number 5725-D16) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 1996, 2016.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. About IBM Opportunity

Detect	1
Integration with IBM Campaign	1
About trigger systems	1
How processing works in a workspace	2
Ancestor and descendent relationships among components.	3
Input data in Opportunity Detect	4
Enable cookies.	5

Chapter 2. Opportunity Detect roles and permissions 7

Permissions for Opportunity Detect.	7
Built-in roles in Opportunity Detect.	8

Chapter 3. Creating workspaces 9

Fields and buttons in the Workspace List	9
--	---

Chapter 4. Adding and deleting components 13

Adding components to workspaces from the Component Palette	13
Adding components within workspaces by saving a copy.	13
About component references.	13
Sharing components across workspaces using component references	16
Deleting components	17
Fields and buttons on the Workspace Component List tab	17
Component list filters	18
Filtering the component list	19
Deleting or modifying component list filters	20
Component type details and examples	20

Chapter 5. Deploying and running workspaces 23

About deployment configurations	23
Creating and deploying a deployment configuration	24
About input mode	25
Fields and buttons on the Deployment tab	26
About data source connectors	29
Data source connector mapping	32
Running workspaces	33
Fields and buttons on the Batch Run tab.	34
The profile data refresh mechanism	36
Fields on the Batch History tab.	36
About batch notifications.	37

Chapter 6. Outcome data in Opportunity Detect 39

Outcome format with the Outcome data source connector	40
---	----

Outcome format with the Expanded Outcome data source connector.	41
Integrating Opportunity Detect with Campaign in batch mode	44
Integrating Opportunity Detect with Campaign in interactive mode.	45
Outcome format with the Web Service data source connector	46

Chapter 7. Component types in Opportunity Detect 49

Component type details and examples	50
---	----

Chapter 8. Common features of components 53

Data Dependencies	53
Dependent Components	55
Effective Window	56
Firing Frequency	56
Properties	56
Incoming Event fields	56
Time Spans	56
Time constants and time units	58
Firing Condition fields.	59
Functions in Opportunity Detect	60
The IsMemberOf function	61

Chapter 9. Building expressions using the Expression Builder. 63

Boolean, comparison, and math operators	63
Value Selector fields	64

Chapter 10. Regular expressions in Opportunity Detect 67

Chapter 11. Date, Math, and Boolean expression components 71

Chapter 12. Simple components 73

Chapter 13. Action components 75

Outcome fields	76
--------------------------	----

Chapter 14. Select components 77

Chapter 15. Container and Container Manipulator components 79

Container components.	79
Container Manipulator components	83
Inserting data into a Container component with a Container Manipulator	85

Deleting data from a Container component with a
Container Manipulator 86

Chapter 16. Pattern components. . . . 87

Pattern component types 87
Negative event modes in Pattern components . . . 88
Calendar time span in Pattern components . . . 88
Rolling time span in Pattern components 91
Pattern Behavior fields 94
Reset Event fields 95

**Chapter 17. Backward Inactivity and
Forward Inactivity components 97**

Backward Inactivity components 97
 Examples of Backward Inactivity Components. . 98
Forward Inactivity components. 98
 Rolling time span in Forward Inactivity
 components 100
 Calendar time span in Forward Inactivity
 components 101
 Examples of Forward Inactivity components . . 102

**Chapter 18. Trend, Spike, and
Exceeded Standard Deviation
components 105**

About time boundaries for trends 105
 Rolling bounded periods 106
 Time boundaries for trend components 106

 End of month boundaries 107
 Beginning points, ending points, and
 end-of-month arithmetic. 108
The Trend component 109
 Example of a Trend component 109
 Specifying a Trend component. 110
 Trend component fields 111
The Spike component. 112
 Example of a Spike component 112
 Specifying a spike component 113
 Spike component fields 114
The Exceeded Standard Deviations component . . 115
 Example of an Exceeded Standard Deviations
 component 116
 Specifying an Exceeded Standard Deviations
 component 117
 ESD component fields 118

**Chapter 19. Artificial transactions in
Opportunity Detect 121**

**Before you contact IBM technical
support 125**

Notices 127

Trademarks 129
Privacy Policy and Terms of Use Considerations 129

Chapter 1. About IBM Opportunity Detect

IBM® Opportunity Detect enables you to look for specified customer behaviors and patterns in your customer data. You define the transactions and patterns that Opportunity Detect looks for, and you specify the data that is written to the database or web servlet when those criteria are met.

You use Opportunity Detect components to build trigger systems in workspaces. When you run a workspace, its trigger systems apply your business logic to streams of data from your transaction and profile data feeds.

Opportunity Detect uses IBM InfoSphere® Streams technology for enhanced performance.

Streams is an advanced analytic platform that allows Opportunity Detect to quickly ingest, analyze and correlate information as it arrives from batch and real time sources. The solution can handle very high data throughput rates.

Integration with IBM Campaign

You can integrate Opportunity Detect with IBM Campaign.

You can use the Expanded Outcome data source connector to store the Outcome data in database tables that Campaign can use. See the section on Expanded Outcome tables in the *IBM Opportunity Detect User's Guide* for details about this integration.

Note: Another product, IBM Interact Advanced Patterns, provides integration with IBM Interact. With this integration, you can apply Pattern component logic to data sent from Interact in real time. See the *IBM Interact and IBM Interact Advanced Patterns Integration Guide* for details.

About trigger systems

To create your business logic, you configure sets of components called trigger systems. Some components send events that activate other components, while others perform operations on data and make the results available to other components.

A trigger system is a time-sensitive detection algorithm that detects relevant patterns over time in a customer's behavior. Each trigger system solves a single detection problem. Multiple trigger systems can share components that hold saved data.

A trigger system produces Outcome data that is written to a database, to a web servlet, or to a queue.

For production, multiple trigger systems are collected into a single workspace, which you run to process your transaction data.

Trigger systems can be complex and sophisticated, but at a minimum every trigger system requires the following.

- A data source
Data sources must include transaction data and can also include static customer profile data.
Data source setup and requirements are described in the *IBM Opportunity Detect Administrator's Guide*.
- A Simple component
Detects discrete events in the stream of input data. A Simple component looks for specified criteria based on a single audience level in a single transaction data source.
- An Action component
Writes Outcome data to a destination specified by the data source connector used for outcomes. This data is available for use by external systems, and can be used in IBM Campaign.

Trigger system development

A workspace can include multiple trigger systems, but it is a good practice during the development and test phase to use a different workspace for each trigger system. This makes it easier to:

- understand each piece of logic as you build it, and
- test and refine the outcomes.

When you have tested all of your trigger systems, you can then use component references to copy them to one workspace for production runs.

Related concepts:

“About component references” on page 13

How processing works in a workspace

As you learn to work with Opportunity Detect, it is helpful to have a basic understanding of the way the application processes transactions.

Trigger systems process one transaction record at a time. Transaction records always include a customer ID, an audience, a timestamp, and data relating to the customer action that created the record.

Trigger systems turn transaction records into events. Event components perform their operations when they receive an incoming event, and then they send an event of their own that activates other event components that are configured to listen for its event.

Event components can also use data components, which are prompted to perform their operations when the event component needs them to complete its processing.

At a high level, Opportunity Detect processes transactions as follows.

1. Transaction and profile data is passed into the engine.
2. State History is loaded. State History contains the following:
 - Incoming event timestamps for components that use incoming events
 - Firing event history for firing frequency evaluation
 - Data saved in Container components

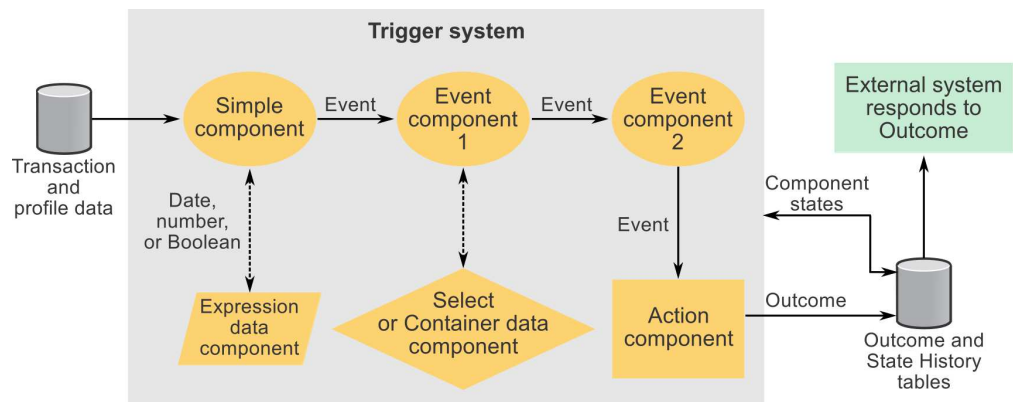
3. Transactions are processed one at a time across all trigger systems in the workspace.
4. Backward Inactivity, Forward Inactivity, and Pattern components configured to use negative mode are processed if they have matured.
5. For each transaction record, an outcome is produced for each trigger system in the workspace, if the criteria in the trigger system are met.
The format of the outcome data is determined by the data source connector used for outcomes.
6. State History is packed and saved.
7. Inactivity events are processed for customers who have had no transactions.
This applies to Backward Inactivity, Forward Inactivity, and Pattern components configured to use negative mode.

Ancestor and descendent relationships among components

Components in a trigger system can be said to have ancestor and descendent relationships with each other. Ancestor and descendent relationships affect many aspects of component behavior.

Data dependencies, deletion of components, and component references are all affected by these relationships.

The following diagram shows an example trigger system.



In the diagram above, Event component 1 requires the Select or Container data, Simple, and Expression data components to perform its operations, which makes them the ancestors of Event component 1. Event component 1 is the descendent of the Select or Container data, Simple, and Expression data components.

Another way to state this is to say that components that are upstream of a component are the ancestors, while downstream components are the descendents.

The following table describes the relationships among the components in the diagram above.

Table 1. Ancestor and descendent component relationships

Component	Ancestors	Descendents
Simple	<ul style="list-style-type: none"> • Expression data 	<ul style="list-style-type: none"> • Event 1 • Select or Container data • Event 2 • Action
Event 1	<ul style="list-style-type: none"> • Select or Container data • Simple • Expression data 	<ul style="list-style-type: none"> • Event 2 • Action
Event 2	<ul style="list-style-type: none"> • Event 1 • Select or Container data • Simple • Expression data 	<ul style="list-style-type: none"> • Action
Expression data	None	<ul style="list-style-type: none"> • Simple • Event 1 • Select or Container data • Event 2 • Action
Select or Container	None	<ul style="list-style-type: none"> • Event 1 • Event 2 • Action
Action	<ul style="list-style-type: none"> • Event 2 • Event 1 • Select or Container data • Simple • Expression data 	None

Related concepts:

“About component references” on page 13

Related tasks:

“Deleting components” on page 17

Related reference:

“Data Dependencies” on page 53

“Time constants and time units” on page 58

Input data in Opportunity Detect

The transaction and profile data you feed into a trigger system is processed one record at a time. For batch file processing using the File type of data source connector, the system requires specific fields, field order, and sort order in your transaction and profile files.

- One field in all data sources must be an ID field used for an audience level, and this must be the first field in each record in the file.
- For batch transaction files only, one field must be a timestamp field, expressed in milliseconds.

- The records in a batch file must be sorted by audience ID first, then by transaction timestamp.

Batch transaction and profile file examples

Note: The following examples are formatted with constant widths for readability. Actual batch transaction and profile files must not use constant widths.

Here is an example of a simple transaction file.

ID	NAME	CALLED_NUMBER	CALL_LENGTH	TRAN_DATE_TIME
001234	David	732-123-4567	15	2012-02-10 09:12:33
001234	David	732-111-5555	48	2012-02-10 10:11:50
002941	Jeremiah	732-777-8888	40	2012-02-10 11:22:44
005555	Anthony	732-333-4444	27	2012-02-10 03:01:02
005555	Anthony	732-32-8945	121	2012-02-10 10:12:30
005555	Anthony	973-597-0022	2	2012-02-10 19:00:21
006789	Tom	732-111-2222	4	2012-02-10 06:54:01

Here is an example of a simple profile file.

ID	AGE	ZIP
001234	25	11111
002941	55	22222
005555	31	33333
006789	60	44444
100382	18	55555

Enable cookies

Enable cookies in your browser to take advantage of all of the features provided in the Opportunity Detect user interface.

Chapter 2. Opportunity Detect roles and permissions

The permissions assigned to users in Opportunity Detect determine what areas of the application they can access and the actions they can perform.

You manage user application access by assigning the desired roles and permissions to individual users, or by assigning users to groups that have the desired roles and permissions. You can use the default roles, or create custom roles with the permissions that you specify. You can not create custom permissions, only custom roles.

You manage roles and permissions from the Users, User Groups, and User Roles & Permissions pages. All of these pages are available under the **Settings** menu.

Tip: For help when you work on these pages, click **Help > Help for this page**, or see the *IBM Marketing Platform Administrator's Guide*.

Permissions for Opportunity Detect

The following table describes permissions that you can assign to roles in Opportunity Detect.

All permissions that have the **Not Granted** status are treated as **Denied**.

Table 2. Permissions in Opportunity Detect

Permission	Description
View only	Can access all of the user interface, in view-only mode.
Design triggers	<ul style="list-style-type: none">• Can create workspaces and design trigger systems.• Can create, modify, and delete all trigger related resources.• Can access Workspace, Component, Audience Level, Data Source, and Named Value List pages.• Can not access the Server Groups page or the Deployment tab of a workspace.• Can not set off a batch run.• Can not administer objects that the web service creates when Opportunity Detect is integrated with Interact.
Run for testing	<ul style="list-style-type: none">• Deploy deployment configurations and run batch deployment configurations on server groups not designated for production.• Can access Server Group page and the Deployment tab of a workspace, but can not designate a server group for production.• Can not deploy deployment configurations or run deployment configurations that use a production server group.
Run for production	<ul style="list-style-type: none">• Deploy deployment configurations and run batch deployment configurations on any server group.• Perform all actions on the Server Group page and the Deployment and Batch Run tabs of a workspace, including designating a server group for production.

Table 2. Permissions in Opportunity Detect (continued)

Permission	Description
Administer real time	<p>Manage objects that the web service creates when Opportunity Detect is integrated with Interact to enable real time mode.</p> <p>Allows the following.</p> <ul style="list-style-type: none"> • Delete workspaces and components created by the web service. • Start and stop real time deployment configurations and update their log level. <p>The user with this permission alone can not start runs for real time deployment configurations.</p> <p>No one, even with this permission, can do any of the following.</p> <ul style="list-style-type: none"> • Delete and update audience levels, data sources, named value lists, server groups, or deployment configurations created by the web service. • Create and deploy deployment configurations created by the web service.

Built-in roles in Opportunity Detect

Four built-in roles are included with Opportunity Detect.

In addition, you can create roles with permissions that you specify. See the *IBM Marketing Platform Administrator's Guide* for details on creating custom roles.

The following table shows the permissions assigned to each built-in role.

Table 3. Built-in roles in Opportunity Detect

Role	Permissions
OpDetectViewer	<ul style="list-style-type: none"> • View only
OpDetectTestDesigner	<ul style="list-style-type: none"> • View only • Design triggers • Run for testing
OpDetectProductionDesigner	<ul style="list-style-type: none"> • View only • Design triggers • Run for production
OpDetectAdmin	<ul style="list-style-type: none"> • View only • Design triggers • Run for testing • Run for production • Administer real time

Chapter 3. Creating workspaces

You build trigger systems in workspaces. This procedure provides the basic steps for setting up workspaces. Details on adding and using components are provided elsewhere in this guide.

Procedure

1. Navigate to **Opportunity Detection > Workspaces** and click **Add**.
The New Workspace window opens.
2. Give the workspace a name and an optional description, and click **OK**.
Four tabs are displayed: Component List, Deployment, Batch Run, and Batch History.
3. Add components to configure the logic you want the workspace to execute.

Tip: See the remainder of this guide for detailed information about adding components.

What to do next

When you have finished creating a workspace, you can add components to the workspace to build your trigger system. To test your trigger system or to run it for production, you first create and deploy a deployment configuration on the Deployment tab. Then you run the workspace on the Batch Run tab.

Related concepts:

Chapter 5, “Deploying and running workspaces,” on page 23

Related tasks:

“Adding components to workspaces from the Component Palette” on page 13

“Sharing components across workspaces using component references” on page 16

“Adding components within workspaces by saving a copy” on page 13

Fields and buttons in the Workspace List

In the Workspace List, you can create, delete, and modify the basic properties of workspaces.

Workspace validation occurs automatically when you select a workspace in the workspace list, when you save, delete, or paste components into a workspace, and when you deploy a workspace. Only valid workspaces can be deployed.

When you navigate to the Workspace Manager, the last workspace you viewed in a previous session is automatically selected.

Table 4. Fields and buttons in the Workspace List














Field or button	Description
 Favorite workspaces folder	Lists workspaces you have added using the Add workspace to favorites button.
 Most recent workspaces folder	When you first open the Workspace Manager, this folder lists the five most recently viewed workspaces. During a session, all of the workspaces you view during that session are listed.
 Deployed workspaces folder	Lists workspaces that have been successfully deployed.
 All workspaces folder	Lists all workspaces in the system.
 Add Workspace	Click to open the New Workspace window, where you can specify the general properties of a new workspace. Clicking OK in this window creates the workspace and adds it to the All Workspaces folder in the Workspace List.
 Edit Workspace	Click to open the Edit Workspace window, where you can modify the general properties of the selected workspace.
 Delete Workspace	Click to delete the selected workspace.
 Add workspace to favorites	Click to add a selected workspace to the Favorites folder.
 Remove workspace from favorites	Click to remove a selected workspace from the Favorites folder.
 Refresh the workspace view	Click to update the list of workspaces, including the validation and deployment status.

Table 4. Fields and buttons in the Workspace List (continued)

Field or button	Description
Workspace status icons	<ul style="list-style-type: none"> •  The workspace is valid. •  The workspace is not valid. •  The workspace was modified after it was deployed. Workspaces that are modified after they are deployed must be deployed again for the changes to go into effect when processing transactions. For example, if a referenced component is changed, all of the instances of that component also change, and this could affect a deployed workspace.
Fields in the New Workspace and Edit Workspace windows	
Name	The name of the workspace
Description	A description of the workspace.
Origin	Indicates whether the workspace was created by a user or by the system. A workspace can be created by the system when Opportunity Detect is integrated with Interact. Read only.
Date Created	Date when the workspace was created. Read only.
Date Modified	Date when the workspace was most recently modified. Read only.
Date Deployed	Date when the workspace was most recently deployed. Read only.
Status	Whether the workspace is valid, not valid, or modified after it was deployed. Read only.

Related tasks:

“Running workspaces” on page 33

Chapter 4. Adding and deleting components

There are three ways to add components to workspaces: by creating a component from the Component Palette, by copying a component within a workspace, and by using a component reference to share a component across workspaces.

Adding components to workspaces from the Component Palette

Use this procedure to add components to workspaces from the Component Palette in Opportunity Detect.

Procedure

1. Navigate to the **Detect > Workspace Manager** page and click a workspace to select it.
2. Click **Component Palette** to expand the component panel.
3. Click a component to open the component editor, where you can configure the logic for the component.
4. Click **Save and Close** to save the component to add it to the selected workspace.

A component must be correctly configured before you can save it.

Related tasks:

Chapter 3, "Creating workspaces," on page 9

Adding components within workspaces by saving a copy

Use this procedure to add a component within a workspace by saving a copy of a component using a new name.

Procedure

1. In a workspace, click a component name on the Component List tab to open the editor for the component.
2. Click **Save As and Close** to open a window where you can enter a name for the copy.

A default name is provided, which you can change. The default name appends "Copy of" to the start of the existing component name.

3. Click **OK** to save the copy.

The copied component is added to the workspace.

Related tasks:

Chapter 3, "Creating workspaces," on page 9

About component references

You can share a component in multiple workspaces by using component references.

When you save a component under a new name or add a component from the Component Palette, you are creating a new object. When you copy and paste a component reference, you are creating another instance of an existing object.

Uses for component references


There are two common situations where you might want to use component references.

- It is a good practice to develop and test your business logic in a modular fashion by building each trigger system in a separate workspace. For production, you often want to include several trigger systems in a single workspace. Component references provide an easy way to copy complete trigger systems from one workspace into another workspace.
- Some components have complex logic that takes some time to configure. When you want to replicate this logic in a different workspace, you can use a component reference.

Workspace origin

When you paste a component reference into a workspace, the name of the workspace from which you copied the component is shown in parentheses after the component name in the component list. This helps you identify the components that are shared across workspaces.

The following rules apply.

- When you change the name of a workspace from which component references are copied, all the component references from that workspace have their workspace origin name changed to the new workspace name.
- If you delete a workspace that is the origin workspace of shared components, the shared components with that workspace origin have their workspace origin name changed to **Deleted_N**, where a number is added to the name to ensure the uniqueness of the Workspace Origin name.
- You can change the workspace origin of a shared component by clicking **Change Workspace Origin**  in the component editor. This is especially useful for components with a workspace origin name of **Deleted_N**.

Component reference rules

The following rules apply to operations you perform on referenced components.

- In general, when you paste a component reference into a workspace, all of the ancestors of that component are also pasted into the workspace.
- There are two exceptions to the rule that ancestors of the referenced component are copied.
 1. When a reference is made to a Container, only the Container and any components it depends upon to calculate its time span are copied and pasted into the new workspace.

This is because Container components are a way to create a sophisticated data structure that you can use within your trigger system. You can configure field names, apply functions to the fields, and set up a time span and aggregation rules. You might want to re-use this structure without including all of the ancestors of the Container.

- If a container depends on a Select or an Expression component to calculate its time span, the Select or Expression is copied, and the ancestors of the Select or Expression are also copied.
 - If a container depends on another Container to calculate its time span, the other Container is also copied, and Container rules regarding the time span are applied.
2. When a reference is made to a Container Manipulator that writes to a Container, the Container is the descendent of the Container Manipulator. However, both the Container Manipulator and the Container it writes to are copied.
- This is because the Container Manipulator is not meaningful without the Container it writes to.

Note: Container Manipulators that read from a Container follow the rule that all of its ancestors are copied, including the Container that it reads from.

- When you make a change to a referenced component or any of its ancestors, your changes are reflected in every workspace in which it is used, including in the workspace in which it was originally created.
- For example, when you add a component that becomes a new ancestor of a referenced component, this component is added in all of the workspaces that use the referenced component.

Component reference examples

The following diagram and accompanying table use an example trigger system to illustrate the rules that apply when you copy and paste a component reference.

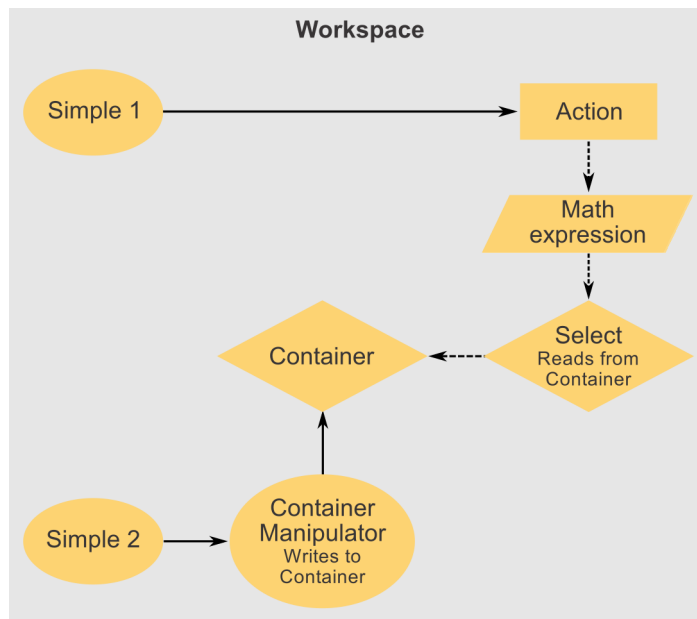


Table 5. Examples of components copied with component references

Copied component	Components that are also copied
Action	All other components in the workspace

Table 5. Examples of components copied with component references (continued)

Copied component	Components that are also copied
Math expression	<ul style="list-style-type: none"> • Select • Container • Container Manipulator • Simple 2
Select	<ul style="list-style-type: none"> • Container • Container Manipulator • Simple 2
Simple 1	No other components
Container Manipulator Writes to Container	<ul style="list-style-type: none"> • Simple 2 • Container <p>This is an exception to the rule that only ancestors are copied. The only ancestor is Simple 2, but the Container is also copied.</p>
Simple 2	No other components
Container	<p>No other components</p> <p>This is an exception to the rule that ancestors are copied. The ancestors are Container Manipulator and Simple 2, but they are not copied.</p> <p>If this Container were to depend on any other components to calculate its time span, those components would also be copied.</p>

Related concepts:

“Ancestor and descendent relationships among components” on page 3

“About trigger systems” on page 1



Related tasks:

“Sharing components across workspaces using component references”

Sharing components across workspaces using component references

Use this procedure to share a component across workspaces.

Procedure

1. Navigate to the **Detect > Workspace Manager** page and select the workspace that contains the component you want to copy.
2. Select the component you want to copy and click  **Copy Component Reference**.
3. In the Workspace List, select the workspace to which you want to add the component reference.
4. Click  **Paste Component Reference**.

A window opens showing the selected component along with all of the other components on which it depends, which will be added to the workspace. You can continue or cancel.

Related concepts:

“About component references” on page 13

Related tasks:

Chapter 3, “Creating workspaces,” on page 9

Deleting components

Use this procedure to delete components from workspaces in Opportunity Detect.


About this task

When you delete a component, all of its descendents are also deleted.

Component deletions affect only the workspace in which the component is deleted. When you delete a component that has been added to a workspace using a component reference, it is deleted within that workspace, but other workspaces that contain the referenced component are not affected.

Procedure

1. Navigate to the **Detect > Workspace Manager** page and click a workspace to select it.

2. Select the component you want to delete and click  **Delete component**, located at the top left of the component list.

A window opens, listing all of the descendent components that will also be deleted. You can cancel the action or complete the deletion.

Related concepts:

“Ancestor and descendent relationships among components” on page 3

Fields and buttons on the Workspace Component List tab

On the workspace Component List tab, you can add component references and manage components in a workspace.

Table 6. Fields and buttons on the workspace Component List tab









Field or button	Description
 Copy Component Reference button	Click this button to copy a selected component and all of the other components on which it depends. You can then paste these components into another workspace.
 Paste Component Reference button	Click this button to paste a previously copied component and all of the other components on which it depends into the workspace.

Table 6. Fields and buttons on the workspace Component List tab (continued)

Field or button	Description
 Delete Component button	When you click this button, the selected component is permanently deleted if it is used in only one workspace. If the component is shared in more than one workspace, it is removed from the current workspace only. When you delete a component, a message is displayed listing all of its child components. These child components are also deleted when you delete the component.
 Show Relations button	Click the arrow next to this icon and select one of the following options. <ul style="list-style-type: none"> • Show Ancestors lists all the components on which this component depends. • Show Descendents lists all the components that depend on this component.
 Create filter button	Click to open a window where you can name a new filter and select the criteria. You can filter based on the component name or the component type, or both. You can select multiple component types in one filter. To make a filter available across workspaces, select the Make this filter available in all workspaces check box.
 Edit filter button	Click to open a window where you can edit the criteria for the selected filter.
 Delete filter button	Click to delete the selected filter. A deleted filter is no longer available.
 Clear all filters button	Click to delete all filters applied to the component list, including custom filters and Show Relations filters. Deleted filters are no longer available.
Name	Name of the component.
Type	Type of the component.
Description	Description of the component, if the creator entered one.
Fields in the Select filter criteria and Edit filter windows	
Filter name	The name you enter appears in the drop-down list of filters.
Name	Enter a string to search for components with names that include this string.
Component types	Expand the panels to show the various component types you can select to include in your filter. You can select multiple types.

Component list filters

You can modify your view of the component list by using filters.

Use the two features described in this section, custom filters and Show Relations, to create filters.

Custom filters that you create are saved in a cookie and are available as long as you do not delete the cookie. Show relations filters are available only while you are viewing a selected workspace. They are discarded when you open a different workspace.

Custom filters

A custom filter that you create can apply to a single workspace, or you can make a filter global so that it is available in all workspaces.

You can create filters based upon the component name and the component type.

Component name filter

This filter searches within the component names for the text you enter. The search is case-insensitive.

Component type filter

This filter searches based on component type. You can select multiple component types.

If you create a filter that uses both the name and type criteria, the component must meet both the name criterion and at least one of the type criteria to be included in the filtered list.


For example, suppose you create the following filter.

- Component name contains abc
- Component type must be Simple or Action

The query finds all components with abc within the name, and then returns only the Simple and Action components from this group.

You can edit or delete any custom filter.

Show relations filter

The **Show relations** filter  allows you to restrict the component list to show only the ancestors or descendents of a selected component.


When you use this filter to show the ancestors or descendents of a component, a filter is automatically added to the **Filter** drop-down list, and it is available for you to use again for the workspace you are viewing.

Filtering the component list


Use this procedure to filter the component list in Opportunity Detect.

Procedure

1. Select a workspace and select the Component List tab.
2. To filter the component list by component name or type, do the following.

- a. Click **Create Filter** . A window opens where you can specify filter criteria.
- b. Enter a name and specify criteria.

The **Make this filter available in all Workspaces** checkbox is selected by default.




- c. Click **OK**.
Your filter is added to the options in the **Filter** drop-down list, and your criteria are applied to the component list.
3. To show only the ancestors or descendents of a component, do the following.
 - a. Select a component.
 - b. Click the arrow next to the **Show relations** icon  and select one of the following options.
 - **Show Ancestors** lists all the components on which this component depends.
 - **Show Descendents** lists all the components that depend on this component.

Deleting or modifying component list filters

Use this procedure to remove filters from the component list in Opportunity Detect.

About this task

You can delete or modify filters as follows.

- A default filter, **All Components**, is always available. Use the **All Components** filter when you want to view the entire list of components after viewing a filtered list. This filter cannot be edited or deleted.
- You can modify a selected custom filter by clicking the **Edit Filter** icon  to open the window where you select filter criteria.
- You can delete a selected custom filter by clicking the **Delete Filter** icon . The deleted filter is no longer available.
- Clicking the **Clear all filters** button  deletes all of the filters you have created, and restores the list to show all components. The deleted filters are no longer available.

Component type details and examples

These are the components that you can use to build trigger systems.

Event components

Table 7. Event components

Component Type	Description
Backward Inactivity	<p>Listens for the occurrence of a specified event and then checks a specified prior time frame to see whether another specified event has occurred. The Backward Inactivity component fires if the specified prior event did not occur.</p> <p>Example: Fire when a customer makes use of the ATM after more than 1 month of not using any teller services.</p>
Container Manipulator	<p>Add and delete data in Container components and perform operations on this data. Can trigger positive events or negative events.</p> <p>Example: When the dollar amount of product a customer purchased in one month exceeds \$200 (value in the container), offer a discount.</p>

Table 7. Event components (continued)

Component Type	Description
Forward Inactivity	<p>Listens for the occurrence of an event and then waits a specified time to see whether or not another event occurs. The Forward Inactivity component fires when the time period expires without the occurrence of the event it for which it was listening.</p> <p>Example: Fire when a web-trade customer, who usually trades once a month, does not trade for two consecutive months.</p>
Pattern (Match All, Counter, Weighted Counter)	<p>Listens for the occurrence of specified events. Fires if the events occur within a specified time frame. In addition, you can configure a Pattern component to fire a negative event if the pattern is no longer matched within the specified time frame.</p> <p>Example: Fire when customer uses his credit card four times a month during a three month period.</p>
Simple	<p>Fires if specified conditions based on transaction attributes are satisfied.</p> <p>A Simple is the only component type that is activated by incoming transactions and not by an event produced by another component. This is why every trigger system requires at least one Simple component as its starting point.</p> <p>Example: Fire when a customer makes a credit card purchase over \$5000 or an international phone call to Italy.</p>

Data components

Table 8. Data components

Component Type	Description
Boolean Expression	Evaluates data and returns True or False.
Container	<p>Holds records from transactions or a profile, as specified by a Container Manipulator. Other components can use this data in calculations and comparisons.</p> <p>Example: Save the total dollar amount of customer purchases in one month.</p>
Date	<p>Returns a date. Specifies date or date range. Often used in Simple components to specify criteria that incoming transactions must meet.</p> <p>Example: A transaction that occurred on June 8, 2013.</p>
Math	<p>Returns a number. Specifies a mathematical formula that uses numeric data from one or more data sources. Common usage includes the following.</p> <ul style="list-style-type: none"> • In Simple components, to specify criteria that incoming transactions must meet. • In Pattern components, to set the aging factor. • In Forward Inactivity and Backward Inactivity components, to set the time span. • In Action components, to include the results of a numerical calculation in the outcome message. <p>Example: In a Math component, calculate 95% of the customer's available account balance, which is available in a Simple component. In a Boolean component, compare this number to credit card transactions, and if a credit card purchase is equal to the number produced by the Math component, fire a positive event that activates a downstream Action component.</p>
Select	A database query that returns a specified set of records drawn from a Container or another Select component. Other components can use this data in calculations and comparisons.

Table 8. Data components (continued)

Component Type	Description
Trend	<p data-bbox="431 258 1412 373">Detects changes in activity measured over defined periods of time. Before you can create a trend component, your trigger system must include a Container or Select component that the trend can use as a data source. There are three trend components: Trend, Spike, Exceeded Standard Deviation (ESD).</p> <p data-bbox="431 394 1412 489">Trend components never fire on their own. They are used internally by Action or Container Manipulator components. You can configure an Action or Container Manipulator component to fire when a Trend Component evaluates to true.</p> <p data-bbox="431 510 1412 562">For example, the average rolling monthly balance has been increasing or decreasing by 10% over the last three months.</p>

The Action component

The Action component is the only one that writes to the outcome destination. When you configure an Action component, you define the data that is written to the outcome destination when all the criteria specified in the components that the Action component depends upon are met. Outcomes can be written to a database table or can be sent to a customized program via a Web Service connector.

Chapter 5. Deploying and running workspaces

You can create and run deployment configurations and view deployment and batch run history on the Workspaces page.

- On the Deployment tab, you can create deployment configurations for workspaces, deploy the configurations, stop and restart deployments, and view deployment history.
- On the Batch Run tab, you can manage run parameters, run deployed workspaces, see the status of the most recent batch runs, and stop and start ongoing runs.
- On the Batch History tab, you can see details about past runs, filtered by deployment and a date range.

Related tasks:

Chapter 3, “Creating workspaces,” on page 9

About deployment configurations

Before you can run a workspace to process data, you create a deployment configuration for it and then deploy it.

When you deploy a configuration, the workspace is compiled into a Streams application and sent with all of its component logic to the Streams server.

When a workspace is successfully deployed on the Streams server, you can select and run a saved deployment configuration to run the workspace.

Each workspace has its own configuration deployment, and multiple configuration deployments can be running simultaneously.

Server group availability in deployment configurations

Opportunity Detect environments typically include at least one test server group and one production server group. For any workspace, you can create a deployment configuration for each server group configured in your system.

The server groups available for selection in a deployment configuration are those that are not yet mapped in any deployment configuration for the workspace you are working with. You can use a server group only once for each workspace, but you can use the same server group in deployment configurations for different workspaces.

For best performance, you would normally run only a single workspace on a production server group.

Option to use different configurations for test and production

Using different deployment configurations, you can test components against new data sources, test new components, and test edits to existing components without affecting the production environment.

For example, you can create a deployment configuration that allows you to deploy a workspace to the test server group to test it, and you can create another configuration that allows you to deploy the workspace to the production server group for production runs. By changing data source mappings, you can separate the test data from the production data in your State History and Outcome tables.

Required permissions

The permissions assigned to a user determine which server groups they can access when creating a deployment configuration, as follows.


- A user with the **Run for testing** permission sees only server groups designated for test usage. The built-in **OpDetectTestDesigner** role has this permission.
- A user with the **Run for production** permission sees only server groups designated for production usage. The built-in **OpDetectProductionDesigner** role has this permission.
- To have access to both test and production server groups, a user must have both of the permissions listed above. The built-in **OpDetectAdmin** role has both of these permissions.

In addition, your system administrator may also create custom roles that include the required permissions.

Creating and deploying a deployment configuration

Use this procedure to create and deploy a deployment configuration.

Procedure

1. Select the valid workspace for which you want to create a deployment configuration and click  **Add a Deployment Configuration** on the Deployment tab.

A panel opens with three tabs: Properties, Data Source Mapping, and History.



2. Complete the fields on the Properties tab and select a server group.
The type of data source connector that you choose for the **Input mode** field determines what connector types are valid for the data sources in your workspace.
3. If you want to override the default data source mapping for the server group, do the following.
 - a. On the Data Source Mapping tab, select the data source you want to change.
 - b. Clear the Server Group Default checkbox.
 - c. Select an alternate mapping for the data source and click **OK**.

Note the following restrictions.

All transaction (input) data sources must be mapped to same connector type.

The system enforces restrictions on what combinations of data source connectors you can use for transaction, profile, Outcome, and State History data sources.

These changes apply only within the deployment configuration; they do not affect the default mappings for the server group.

4. Click  **Save** and then click  **Deploy**.

What to do next

When the deployment configuration is successfully deployed, you can use it to run your workspace.

Related tasks:

“Running workspaces” on page 33

About input mode

The input mode that you select determines what data source connectors are allowed for the data sources used in the workspace. The **Input mode** field is located on the Properties tab of a deployment configuration.

The following table shows the allowed data source connectors for each data source type, by input mode. The system enforces these rules; you cannot select a data source connector that is not allowed.

Table 9. Allowed data source connectors by input mode

Data source type	File connector allowed?	Queue connector allowed?	Web service connector allowed?	Table connector allowed?
Batch file input mode				
Profile	Yes (Batch connector only)	No	No	Yes
Transaction	Yes	No	No	No
Outcome	No	No	No	Yes
State	No	No	No	Yes
Real time file input mode				
Profile	No	No	No	Yes
Transaction	Yes (Real time connector only)	No	No	No
Outcome	No	No	No	Yes
State	No	No	No	Yes
Queue input mode				
Profile	No	No	No	Yes
Transaction	No	Yes	No	No
Outcome	No	Yes	No	Yes
State	No	No	No	Yes
Web service input mode				
Profile	No	No	No	Yes
Transaction	No	No	Yes	No
Outcome	No	No	Yes	Yes
State	No	No	No	Yes

Related reference:

“Fields and buttons on the Deployment tab” on page 26

Fields and buttons on the Deployment tab

On the Deployment tab of a workspace, you can create deployment configurations for workspaces, map data source connectors within the deployment configuration, deploy the configurations, stop and restart deployments, and view deployment history.

Table 10. Fields and buttons on the Deployment tab








Field or button	Description
 Add button	Click to open a panel where you can create a deployment configuration.
 Save button	Click to save a deployment configuration.
 Delete button	Click to delete the selected deployment configuration.
 Deploy button	Click to deploy the selected deployment configuration.
 Start button	Click to re-start a deployment that has been stopped.
 Stop button	Click to stop a running deployment.
 Update logging level	Click to update the logging level during a run. Raising the log level can affect performance.
Status	Indicates whether all data sources are mapped for the deployment configuration. You can not deploy an incomplete deployment configuration.
Deployment Configuration	Names of the available deployment configurations. Select the radio button next to a name to select the deployment configuration, after which you can deploy it, run it, or modify its properties in the panel that opens.
Server Group	Name of the server group the selected deployment uses.
Deployed	Indicates whether the selected configuration has been deployed.
Version	For batch runs, this number shows the version number of the most recent successful deployment. When Opportunity Detect is integrated with Interact, the number shows the version most recently deployed, re-deployed, or undeployed from IBM Interact.
Input mode	The input mode selected on the Properties tab of the deployment configuration.
Input version	The input version entered on the Properties tab of the deployment configuration.

Table 10. Fields and buttons on the Deployment tab (continued)

Field or button	Description
Output version	The output version entered on the Properties tab of the deployment configuration.
Workspace modified on	Timestamp of the most recent workspace modification. By comparing this timestamp to the timestamp in the Deployed On column, you can determine whether the latest version of the workspace is deployed.
Deployed On	Timestamp of the most recent successful deployment of the selected deployment configuration.
Properties tab	
Name	Enter a name for the deployment configuration. Deployment configuration names must be unique across all workspaces.
Server Group	Select from a list of available server groups.
Usage	The usage entered in the definition of the selected server group.
Input mode	Select the mode that matches the type of data source connector you are using for transaction data sources in the deployment. The option selected in this field governs the validation rules applied on Deployment tab.
Input file directory	Applies only when Input mode is Real time file . The directory in which real time files are placed for processing. In a distributed environment, the input and processed directories must be on the Runtime server and Streamsadmin user Read and Write permission.
Processed file directory	Applies only when Input mode is Real time file . The directory in which real time files are placed after they are processed.
Input version	If this is the first time you are deploying, enter 1. Increment this number each time you make any changes in the structure of your transaction data and redeploy. If you are using queue connectors, this number must also be updated in the sending program for transaction data.
Output version	If this is the first time you are deploying, enter 1. Increment this number each time you make any changes in the structure of your Outcome data and redeploy.
Script file name (Optional)	If you use a notification batch file, enter the file name here.
Data Source Mapping tab	
Status	Indicates whether a data source is mapped to a connector.
Data Source	Data sources used in the workspace are listed here. Click a link to view the Data Source Connector Mapping dialog box. You can retain the default mapping used in the server group selected on the Properties tab, or you can deselect the Server Group Default check box to change the mapping.
Type	Lists the configured data source type: Transaction, State, or Outcome.
Connector	Lists the connector used with the data source for this deployment configuration.

Table 10. Fields and buttons on the Deployment tab (continued)

Field or button	Description
Connector Type	<p>Lists the connector type of the data source: Table, Batch file, Real time file, TCP, Web Service Connector, Queue, or Expanded Outcome.</p> <ul style="list-style-type: none"> • The Table connector is used for State History tables, for profile data stored in database tables, and for Outcome data where the XML format used by earlier versions of Opportunity Detect is desired. • The Batch file connector is used for profile and transaction data in flat file format. • The Real time file connector is used for transaction data in fixed width file format. • The TCP connector is used only for Interact Advanced Patterns, which is integrated with Interact. • The Web Service connector can be used for transaction data and Outcomes. Special configuration is needed to use the Web Service connector. For more information, see the <i>IBM Opportunity Detect Administrator's Guide</i>. • The Queue connector can be used for transaction data and Outcomes. Special configuration is needed to use the Queue connector. For more information, see the <i>IBM Opportunity Detect Administrator's Guide</i>. • The Expanded Outcome connector is used for Outcome data sources when the output needs to be stored in database tables structured differently from the format used in the Table connector. Use this connector for integration with IBM Campaign.
Database	Lists the database configured for the data sources that use a Table connector.
Default	Indicates whether the data source uses the default mappings for the server group selected on the Properties tab.
Data Source Connector Mapping window	
Data Source Name	Name of the selected data source.
Connector Type	Available connector types. The connector types that are available are determined by the type of the data source and the input mode.
Table Type	For Outcome data sources, when the connector type is Table , you can select either Outcome or Expanded Outcome for the table type.
Connector	Data source connectors available for mapping.
Database Connection	For Table type connectors, the database connections available for mapping.
Server Group Default	Indicates whether the mapping is the one defined in the server group. Deselect this box if you want to change the data source connector mapping.
History tab	
From Date, To Date, Get History button	Select a date range and click Get History to see details for the successful deployments of the selected deployment configuration.

Table 10. Fields and buttons on the Deployment tab (continued)

Field or button	Description
Status	For batch mode, indicates whether the version of the selected deployment configuration was successfully deployed. When Opportunity Detect is integrated with Interact, Undeploy and Redeploy are additional actions for which success or failure can be indicated.
Version	The number of the deployment for which the row provides details. Includes successful and failed deployments.
Action	For stand-alone Opportunity Detect, the only action is Deploy. When Opportunity Detect is integrated with Interact, Undeploy and Redeploy are additional actions.
Date	Timestamp of completion of the listed action.
Input mode	The input mode selected on the Properties tab of the deployment configuration.
Input version	The input version entered on the Properties tab of the deployment configuration.
Output version	The output version entered on the Properties tab of the deployment configuration.
Message	Additional details for the listed action, including the deployment configuration ID, which is automatically generated when deployment takes place.

Related concepts:

“About data source connectors”

“Data source connector mapping” on page 32

“About input mode” on page 25

Chapter 19, “Artificial transactions in Opportunity Detect,” on page 121

Related reference:

“Fields and buttons on the Batch Run tab” on page 34

“Fields on the Batch History tab” on page 36

About data source connectors

Data source connectors enable the system to connect to the transaction, profile, lookup, State History, and Outcome data sources that you have defined. The types of data source connectors are described in this section.

Note: For Outcomes, the data source connector you use determines the format of the Outcome data.

Batch file connectors

Use Batch file connectors to connect to transaction and profile data that is in file format and that is processed in batches.

Real time file connectors

Use Real time file connectors to connect to transaction data that is in fixed width file format and that is updated frequently. This connector is particularly useful for Call Data Records (CDRs) used by the telco industry. The CDR must first be transformed from binary to ASCII fixed width format.

Opportunity Detect reads these files in real time. You configure your automation system to place the properly formatted file in an input directory that you specify when you create the deployment configuration. After Opportunity Detect consumes the data, the files are automatically moved to another directory that you specify for precessed files.

The input and output directories must be on the runtime server, and must have Streamsadmin permission.

When you create the Real time connector, you enter the start and length of each field. The values must be a positive integer greater than 0.

You can choose to have the connector use a Bloom filter on the data. The Bloom filter eliminates duplicate data based upon user-defined filter fields. Data rejected by Bloom filter is placed under the deployment ID folder: `/home/streamsadmin/OpDetection/deploy/Deployment ID/current/data` You should move this rejected duplicate data on a regular basis, to conserve disk space on the runtime server.

If you have more than one deployment using the Real time file connector, ensure that a different folder is designated for the transaction files for each deployment, to prevent an condition where the file is moved before one of the deployments is finished with it.

Opportunity Detect does not support live updates to files in the input directory used with the real time file connector. Also, you should not use the same input directory for multiple workspaces, as this could lead to undesired behavior.

Table connectors

Use table connectors to connect to the following data sources.

- Profile data that is in database table format
- All State History tables
- Outcome data that you want to be written to database tables

There are four types of table data source connectors.

State

Used to connect to your State History tables.

Profile

Used to connect to your profile data that is in database table format.

Outcome

Used when you want Outcome data to be stored in database tables in the XML format used by previous versions of Opportunity Detect.

Expanded Outcome

Used when you want Outcome data to be stored in database tables in a form that external systems or IBM Campaign can use more easily than the format provided by the Outcome connector.

A table connector can be mapped to only one data source.

Web Service

You can use the Web Service connector to connect to transaction and Outcome data coming from or being sent to the Opportunity Detect web service.

If you use the web service for transaction data and you also use profile data, the profile data must be in database table format.

You do not have to create this type of connector. It exists in the system by default.

For details about the Java classes that must be developed to use the Web Service data source connector, see "Web Service data source connector for input and output in Opportunity Detect" in the *IBM Opportunity Detect Administrator's Guide*.

Queue

You can use this connector for real time operation in conjunction with a supported queue server.

For additional information on setting up Queue data source connectors, see "Real time processing in Opportunity Detect" in the *IBM Opportunity Detect Administrator's Guide*.

Default TCP Connector

Used by the system with IBM Interact Advanced Patterns when integration with IBM Interact is implemented.

You do not have to create this type of connector. It exists in the system by default.

The sharable option

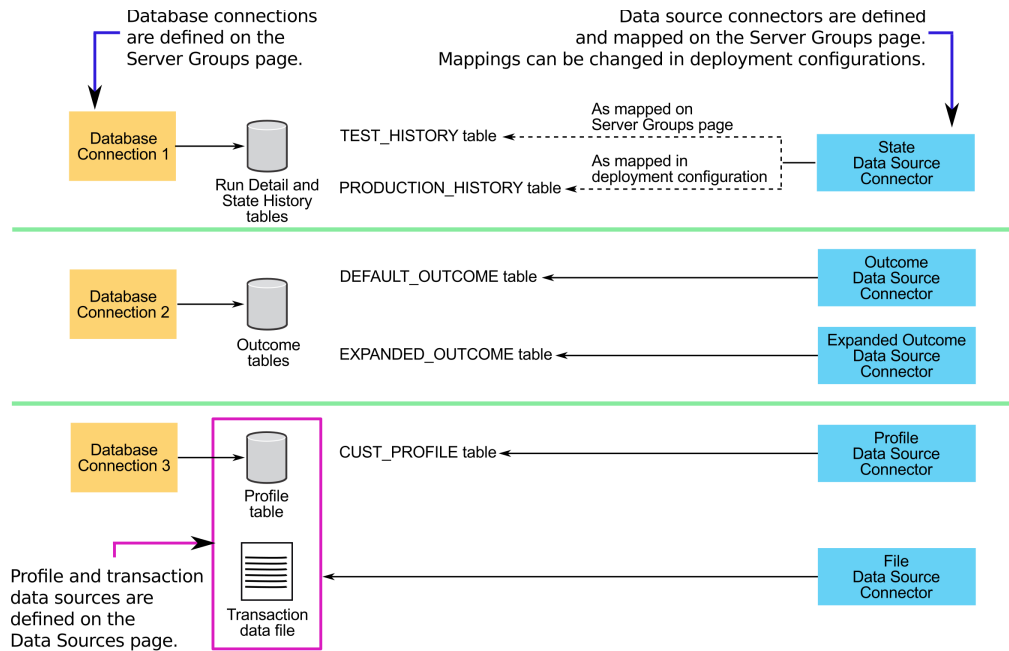
You have the option to make table and queue connectors sharable. The sharable feature works as follows.

- A sharable connector can be used in more than one server group.
- Sharable connectors can be mapped on the Data Source Mapping tab of the Server Groups page and on the Deployment tab of a workspace. Table and queue connectors that are not sharable can be mapped only on the Deployment tab of a workspace.

For example, you might want to have connectors for Outcome and State History tables that are not shared and can be used only by the deployment configuration used for production. You might also want sharable table connectors so you can map them to the same Outcome and State History tables for all test workspaces.

Example of data source connector configuration

The following diagram illustrates a possible data source configuration, showing the relationship between database connections, data source connectors, and the data sources used in Opportunity Detect.



Related concepts:

Chapter 6, “Outcome data in Opportunity Detect,” on page 39

Related reference:

“Fields and buttons on the Deployment tab” on page 26

Data source connector mapping

Data source connectors can be mapped to different data sources in each deployment configuration, which provides flexibility for testing and ramp-up operations.

Default mappings between data source connections and database tables are set when server groups are configured. However, in deployment configurations, you can change the default mappings for a selected server group. Your changed mappings apply only within the deployment configuration and do not affect the global, default mappings for the server group.

Mapping for testing

Suppose you have a server group named Test. You might want to use one set of State History and Outcome tables with the Test server group in one workspace, and another set of State History and Outcome tables with the Test server group in another workspace.

You can do this by changing the mapping of the data source connector on the Data Source Mapping tab for the deployment configuration.

On the other hand, you might want to use the same State History table for several different test workspaces, to reduce the number of State History tables and data source connectors that you need to create.

Mapping for ramp-up

In Opportunity Detect, ramp-up is the process of accumulating State History data for Container and Pattern components. This enables the system to produce meaningful outcomes quickly, rather than waiting for the history to accumulate over time.

For example, trigger system designers often need to introduce a new trigger system into a production workspace that already includes multiple trigger systems. Suppose that the production workspace consists of 10 trigger systems that have been in production for several months. When a new trigger system is introduced, its history can be ramped up by running it in its own workspace against historical transactions, using the same State History table used for the production workspace.

Because the new trigger system is run in isolation, running it against historical transactions does not affect any of the history of the 10 triggers already in production. After ramp-up, the new trigger can then be copied to the production workspace.

To use this technique, you must mark the State History table **Sharable** in the server group.

Having multiple workspaces run against the same State History table has this limitation: the application does nothing to guard against simultaneous access of the same history record. This means that if two workspaces use the same State History table and process the same set of customer IDs, it is possible for each workspace to access the same history record simultaneously. If two workspaces access the same State History record at the same time, the application does not fail, but some data may not be saved.

If the risk of losing data is small, then you could run two sets of triggers simultaneously for the same set of IDs, one with real time feeds and the other with batch feeds.

The application provides a way to guard against inadvertent access of two workspaces against the same State History table. If a table is not marked **Sharable** in the server group, it can be mapped only on the Deployment tab of a workspace, and it can be used by only one deployment configuration at a time.

Related reference:


“Fields and buttons on the Deployment tab” on page 26

Running workspaces

Use this procedure to run a workspace for test or production purposes.

Before you begin

Only valid workspaces can be run. To be valid, a workspace must contain at least one Simple component and one Action component.

If the workspace was modified after it was deployed, indicated by  next to its name, you must deploy it again for the changes to go into effect when processing transactions.

Procedure

1. Select the workspace you want to run.
The Workspace Manager page opens.
2. On the Batch Run panel, select the deployment configuration you want to run.
Only configurations that have been deployed are available for selection.
3. Complete run parameters as desired.
4. Click **Start**.

Related concepts:

Chapter 6, "Outcome data in Opportunity Detect," on page 39

Related tasks:

"Creating and deploying a deployment configuration" on page 24

Related reference:

"Fields and buttons in the Workspace List" on page 9

Fields and buttons on the Batch Run tab

On the Batch Run tab, you can manage run parameters, run deployed workspaces, see the status of the most recent batch runs, and stop and start ongoing runs.

Table 11. Fields and buttons on the Batch Run tab

Field or button	Description
Deployment Configuration	Select a deployment configuration to run and click Start . Only deployment configurations that are successfully deployed and ready to run are listed. If you expect a deployment configuration to be available and it is not listed, check the Streams server logs and correct any problems. Then, start the deployment configuration and try to perform the run again. If any transaction data source in the deployment configuration has been mapped to the Web Service connector, the deployment configuration is not available for batch runs.
Run Parameters	
Feed Path	Location of the feed files.
Inactivity Mode	If your workspace includes any Forward Inactivity components or Pattern components configured with the negative event mode, to enable the component to process data as expected you must select On in this field and also set the date in the Inactivity Date field.
Inactivity Date	Specifies the date the system uses to check for inactivity. Applies only to workspaces that contain Forward Inactivity components or Pattern components configured with the negative event mode, when the Inactivity Mode field is set to On . It is possible that no transactions for a customer whose transaction has activated the component will be processed when the component is due to fire. In those cases, the system uses the date set in the Inactivity Date field to determine whether the component should fire. Typically, you should set this field to the latest transaction timestamp in the batch file being processed.

Table 11. Fields and buttons on the Batch Run tab (continued)




Field or button	Description
Artificial Transaction	<p>Check this box if your workspace includes any components that use an artificial transaction.</p> <p>You can select the following options.</p> <ul style="list-style-type: none"> • Off - No artificial transaction is set. • End of Day - Sends an artificial transaction to indicate the end of day at the end of each unique transaction date in the feed files. • End of Run - Sends an artificial transaction after all transactions are sent for a user, regardless of how many transaction dates are in the feed files.
System Log Level	<p>Set the logging level for a batch run. Optionally, select a system log level for real time runs. Options are as follows, listed in ascending order of detail.</p> <ul style="list-style-type: none"> • Fatal • Error • Warning • Information • Trace • Debug <p>Raising the log level can affect performance.</p> <p>You can also update the logging level during runs by selecting a deployment and clicking  Update Logging level on the Deployment tab.</p>
Run in Recovery	<p>When this box is selected, if a run fails, and you re-start the run after fixing any problems, the system attempts to resume the run where it left off. You must wait three minutes for the automatic save to complete before attempting to re-start the run.</p>
Batch Notification	<p>Select this box if you have a batch notification file and you want to receive notifications.</p>
Most Recent	
 Stop button	<p>Click to stop a batch run.</p>
 Start button	<p>Click to re-start a batch run that has been stopped.</p>
Run id	<p>Unique identifier for the most recent run.</p>
Detail id	<p>A counter that increments each time a recovery run occurs.</p>
Deployment Configuration	<p>The name of the deployment configuration used for the run.</p>
Start Time	<p>Date and time of the beginning of the run.</p>
End Time	<p>Date and time of the end of the run.</p>
Run Time	<p>Duration of the run.</p>

Table 11. Fields and buttons on the Batch Run tab (continued)

Field or button	Description
State	During the run, this column may display a variety of interim states. When a run ends, the state is one of the following. <ul style="list-style-type: none"> • Run Success • Run Failed

Related concepts:

Chapter 19, “Artificial transactions in Opportunity Detect,” on page 121

Related reference:

“Fields and buttons on the Deployment tab” on page 26

“Fields on the Batch History tab”

The profile data refresh mechanism

Opportunity Detect automatically refreshes profile data every six hours. If you require an immediate refresh of profile data, you can stop and restart your deployment.

Fields on the Batch History tab

On the Batch History tab, you can see details about past runs, filtered by deployment and a date range.

Table 12. Fields on the Batch History tab

Field	Description
Deployment Configuration	Select a deployment configuration for which you want to retrieve run history.
From Date, To Date, Get Run History button	Enter or select a date range and then click Get Run History to retrieve information about past runs.
Result	The final status of the run. <ul style="list-style-type: none"> • Run Success • Run Success with Errors • Run Failed
Run Id	Unique identifier for the run.
Detail Id	A counter that increments each time a recovery run occurs.
Run Type	Indicates whether the run is a normal or recovery run.
Start Time	Date and time of the beginning of the run.
End Time	Date and time of the end of the run.
Run Time	The duration of the run.

Related reference:

“Fields and buttons on the Deployment tab” on page 26

“Fields and buttons on the Batch Run tab” on page 34

About batch notifications

The Opportunity Detect engine can run batch files when it completes a run. You can use these files to issue different notifications for each successful and failed engine run.

An example batch file is shipped with the product; you must edit this file as needed to perform the appropriate actions for your system when the Opportunity Detect engine completes a run. If the batch file is empty, no further processing takes place.

The batch file must be a shell script with the `.sh` extension. It must have executable permission (755).

When you run the Opportunity Detect engine from the Batch Run tab in a workspace, check the **Batch Notification** check box and specify the file name to run a batch file when the process completes.

Batch file usage examples

- After an engine process runs successfully, the notification file can call an external process to act on the reported outcome data.
- After an engine process fails to complete, the notification file can send a notification of the failure to the appropriate people.

Batch file required location

You must create a `scripts` directory on the machine where Streams is installed, under the `/home/streamsadmin/OpDetection` directory, if it has not been created previously. Place your batch file in this directory.

Related concepts:

“Outcome format with the Expanded Outcome data source connector” on page 41

Chapter 6. Outcome data in Opportunity Detect

Opportunity Detect generates one outcome record for every set of transactions that meets the criteria specified in the trigger system.

Outcome data can have different formats, depending on the data source connector that you use for your Outcomes when you run your trigger systems.

You assign data source connectors to tables, queues, or a web service when you create server groups. When you deploy a trigger system, you can change this default mapping in the deployment configuration.

How Outcome data is specified in trigger systems

Outcome data is specified in the Action component, in the Outcome panel. This panel has two fields:

- **Message**, which holds a text string that you specify.
- **Additional information**, which holds data captured using an inline expression. The inline expression can choose data from any data source or component that is available to the Action component.

This data can be a single value (scalar), or a row in a data record stored in a Container or Select component (tabular).

Types of data source connectors for Outcomes

For Outcome data, Opportunity Detect provides four types of data source connectors.

Outcome table

Used when you want Outcome data to be stored in database tables in the XML format used by previous versions of Opportunity Detect.

Expanded Outcome tables

Used when you want Outcome data to be stored in database tables in a form that external systems or IBM Campaign can use more easily than the format provided by the Outcome connector.

Web Service

Used when your system is configured to use the Opportunity Detect web servlet for Outcome data, for real time operation. See the *IBM Opportunity Detect Administrator's Guide* for details about the Java classes that must be developed to use the Web Service data source connector.

Queue

Used when your system is configured to use a Queue connector for Outcome data, for real time operation. See the *IBM Opportunity Detect Administrator's Guide* for information on setting up a Queue data source connector.

Related concepts:

“About data source connectors” on page 29

Related tasks:

“Running workspaces” on page 33

Outcome format with the Outcome data source connector

When the Outcome data source connector is used for Outcome data, the system writes the data in the XML format used by previous versions of Opportunity Detect.

Fields in the Outcome table

The following table describes the fields in the Outcome table when you use the Outcome data source connector.

Table 13. Fields in the Outcome table

Field	Description
AUDIENCEID	ID of the audience for which the trigger system fired. Examples of an audience are account, customer, or household.
RUNID	ID of the run. The Run ID helps distinguish between the Outcomes of one run versus the Outcomes of runs before or after it. Because of the Run ID, you do not need to truncate the Outcome table after every run because you can query the table for all of the Outcomes in a specific run.
COMPONENTID	Unique ID of the Action component that fired to generate the Outcome.
AUDIENCETYPE	The single character audience code assigned on the Opportunity Detect Audience Levels page.
OUTCOMEDATE	The timestamp of the final event that caused the Action to fire.
MESSAGE	The data that was specified in the Message and Additional Information fields of the Action component. The data is written in XML format.
ROW_NUMBER	A unique sequence field, generated automatically.

XML format of the message field of the Outcome table

The XML in the message field of the Outcome table contains the data specified in the **Message** and **Additional Information** fields of the Action component.

Here is an example of the XML for a simple Outcome. The Action component in the example includes the following information in the Outcome.

- **Message** field: Send printer paper offer.
- **Additional Information** field:
 - The sum of all values in the field named Amount in a Container component named All Transactions.
 - The field named zipcode in the data source named Customer Profile.

```
<OUTPUT>
<TEXT>
  Send printer paper offer.
</TEXT>
<CONTAINER name="All Transactions" field="Amount" function="SUM">
  123.45
</CONTAINER>
```

```
<DATASOURCE name="Customer Profile" field="zipcode">
  11746
</DATASOURCE>
</OUTPUT>
```

XML format of Outcome values from Container and Select components

Select and Container components can hold values that consist of a single value or a set of fields.

If the value has been aggregated into a single number using a function, it is called a scalar value. Here is an example of the XML for a scalar value where the Outcome includes a field named Amount from a Container component named C1. The Amount field is aggregated using the average function.

```
<CONTAINER name="C1" field="Amount" function="average">
  123.45
</CONTAINER>
```

If the value is a set of fields, it is called a tabular value. Here is an example of the XML for a tabular value where the Outcome includes fields named Field_1, Field_2, and Field_3 from a Select component named S1.

```
<SELECT name="S1">
  <ROW>
    <FIELD name="Field_1">abc</FIELD >
    <FIELD name="Field_2">123.45</FIELD >
    <FIELD name="Field_3">xyz</FIELD >
  </ROW >
</SELECT >
```

Outcome format with the Expanded Outcome data source connector

When the Expanded Outcome data source connector is used for Outcome data, the system writes the data in a form that can be used by IBM Campaign or an external system.

The data produced by Opportunity Detect is called Outcome data.

About the Expanded Outcome tables

The Expanded Outcome connector writes the Outcome data to two database tables, which you must create using scripts provided with Opportunity Detect.

DB2 is the only supported database type for the Expanded Outcome tables.

The tables are:

- A **primary** table that contains the text string specified in the **Message** field in the Action component.
- A **secondary** table that contains the data specified in the **Additional information** field in the Action component.

You provide a base name for the Expanded Outcome tables when you run the ExpandedTable.sql script to create the tables. The script appends the number 1 to the name of the primary table, and appends the number 2 to the name of the secondary table.

For example, if you specify the base name ExpandedOutcome, the script creates two tables: ExpandedOutcome1 and ExpandedOutcome2.

Fields in the Expanded Outcome tables

These descriptions of the fields in the Expanded Outcome tables refer to scalar and tabular values, which are defined as follows:

Scalar

A single unit of data.

Tabular

A data set, as in a database row. In Opportunity Detect Outcomes, tabular data is saved in XML format.

Depending on how you specify the Outcome data, the Outcome can contain either type of value, or both types. If you include tabular data in a Campaign integration, additional processing is required before Campaign can consume it.

Table 14. Fields in the Expanded Outcome primary table

Field	Description	Data type
OUTCOMEID	Unique sequence ID. Used as the primary key to link to the secondary Expanded Outcome table.	Integer
AUDIENCEID	ID of the audience member for which the trigger system fired. Examples of an audience are account, customer, or household. The audience ID is stored as a string. Multi-column audience IDs are not supported.	NVARCHAR(60) If you use Oracle system tables and plan to integrate with Campaign, you must change the data type of this field from NVARCHAR(60) to Varchar2(60) because Campaign does not support the NVARCHAR(60) data type.
AUDIENCELEVEL	The single character audience code assigned on the Opportunity Detect Audience Levels page.	NVARCHAR(60) If you use Oracle system tables and plan to integrate with Campaign, you must change the data type of this field from NVARCHAR(60) to Varchar2(60) because Campaign does not support the NVARCHAR(60) data type.
COMPONENTID	Unique ID of the Action component that fired to generate the Outcome.	Varchar
OUTCOMEDATE	The timestamp of the final event that caused the Action component to fire.	Timestamp
RUNID	ID of the run, for batch mode only. The Run ID helps distinguish between the Outcomes of one run versus the Outcomes of runs before or after it. Because of the Run ID, you do not need to truncate the Outcome table after every run because you can query the table for all of the Outcomes in a specific run.	Integer

Table 14. Fields in the Expanded Outcome primary table (continued)

Field	Description	Data type
MESSAGE	The text string that was specified in the Message field of the Action component.	NVARCHAR(60) If you use Oracle system tables and plan to integrate with Campaign, you must change the data type of this field from NVARCHAR(60) to Varchar2(60) because Campaign does not support the NVARCHAR(60) data type.
PROCESSED	A flag that indicates whether the data has been consumed by Campaign.	Integer

Table 15. Fields in the Expanded Outcome secondary table

Field	Description	Data type
OUTCOMEID	Unique sequence ID. Used as a foreign key to link the record to the primary Expanded Outcome table.	Integer
NAME	The name assigned in the Additional Information field of the Action component.	NVARCHAR(60) If you use Oracle system tables and plan to integrate with Campaign, you must change the data type of this field from NVARCHAR(60) to Varchar2(60) because Campaign does not support the NVARCHAR(60) data type.
VALUE	The scalar and tabular data that was specified in the Additional Information field of the Action component. Tabular values are saved in XML format.	Clob
DATATYPE	For scalar values, the data type can be one of the following. <ul style="list-style-type: none"> • boolean • currency • date • double • integer • string For tabular values, the data type is set to string, because tabular values are stored in XML, and the data type for XML is string.	NVARCHAR(60) If you use Oracle system tables and plan to integrate with Campaign, you must change the data type of this field from NVARCHAR(60) to Varchar2(60) because Campaign does not support the NVARCHAR(60) data type.

XML format of tabular values

Here is an example of the XML for a tabular value, where the record includes these fields:

- Field_1
- Field_2
- Field_3

Example

```

<SELECT name="S1">
  <ROW>
    <FIELD name="Field_1">abc</FIELD >
    <FIELD name="Field_2">123.45</FIELD >
    <FIELD name="Field_3">xyz</FIELD >
  </ROW >
</SELECT >

```

Integrating Opportunity Detect with Campaign in batch mode

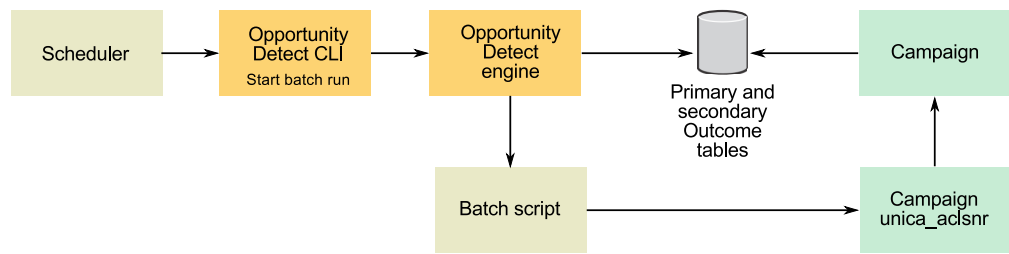
The following example illustrates how you might use the Expanded Outcome data in Campaign, in batch mode.

Before you begin

You must have Campaign and Opportunity Detect installed and running.

About this task

The following diagram illustrates the example described in this procedure.



Procedure

1. Create the Expanded Outcome tables in your database using the script provided with Opportunity Detect.
2. On the Server Groups page in Opportunity Detect, do the following.
 - If a database connection for the database where you created your Expanded Outcome tables does not exist, create one.
 - If an Expanded Outcome data source connector does not exist, create one. If you make the connector sharable, you can map the connector to your primary Expanded Outcome table on the Server Groups page or on the Deployment tab of the workspace. If you do not make the connector sharable, you can map it only on the Deployment tab.
3. Create the Opportunity Detect workspace and configure it to use the Expanded Outcome data source connector for Outcome data, either on the Server Groups page or on the Deployment tab of the workspace.
4. On the Deployment tab of the Opportunity Detect workspace, configure the deployment to call a batch file at the end of a successful run. Create the batch script to call the Campaign listener service, `unica_aclsnr`, to run a Campaign flowchart that you design.
5. Use the Opportunity Detect command line utility, `RemoteControlCLI (CLI)`, to run the workspace. Use your own scheduling utility to run the CLI batch script at the desired interval; for example, daily. When the workspace runs, Opportunity Detect inserts Outcome data into the Expanded Outcome tables.

6. Configure your Campaign flowchart as follows.
 - a. In a Select process, create a new table mapping as follows.
 - Map your main audience in Campaign to the OUTCOMEID field in the primary Expanded Outcome table. This is required so that you can select Outcome records for use in the flowchart. Selection must use the OUTCOMEID field, as the same AUDIENCEID field can be repeated in multiple Outcome records.
 - Map your alternate audience in Campaign to the AUDIENCEID field in the primary Expanded Outcome table. This mapping defines the audience on which rest of the flowchart logic should be performed.

Note: If you plan to use Opportunity Detect Outcome data in multiple flowcharts, save the mapped table information into a table catalog and load this catalog in other flowcharts.
 - b. Select records where the value in the PROCESSED field in the primary Expanded Outcome table is 0.
This value indicates that the record has not been processed yet.
 - c. Set the value in the PROCESSED field in the primary Expanded Outcome table to 1, to indicate that the record has been processed.
You can write SQL in a Select process to set this value.
 - d. In an Audience process, switch the audience from OUTCOMEID to AUDIENCEID.
 - e. Use the Opportunity Detect data as desired in your flowchart.
 - f. Use a MailList process to assign an offer and update contact history.

Integrating Opportunity Detect with Campaign in interactive mode

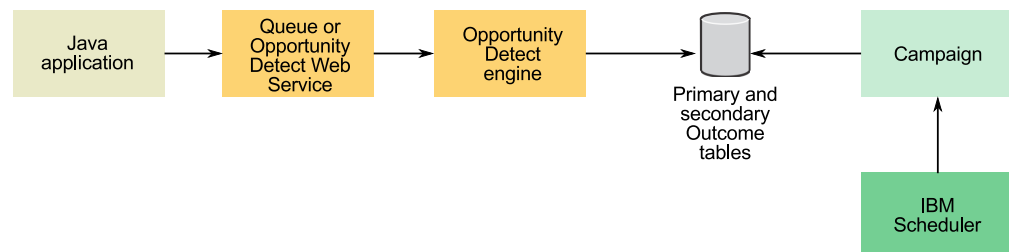
The following example illustrates how you might use the Expanded Outcome data in Campaign, in interactive mode.

Before you begin

You must have Campaign and Opportunity Detect installed and running.

About this task

The following diagram illustrates the example described in this procedure.



Procedure

1. Create the Expanded Outcome tables in your database using the script provided with Opportunity Detect.
2. Do one of the following.

- If you plan to use a queue connector, configure a queue for your transaction data in your queue server.
 - If you plan to use the Web Service, develop the required Java classes.
3. On the Server Groups page in Opportunity Detect, do the following.
 - If a database connection for the database where you created your Expanded Outcome tables does not exist, create one.
 - If an Expanded Outcome data source connector does not exist, create one.

If you make the connector sharable, you can map the connector to your primary Expanded Outcome table on the Server Groups page or on the Deployment tab of the workspace. If you do not make the connector sharable, you can map it only on the Deployment tab.
 4. Configure the Opportunity Detect workspace to use the Web Service or a queue data source connector for transaction data, and the Expanded Outcome data source connector for Outcome data.
 5. Configure your Campaign flowchart as follows.
 - a. In a Select process, create a new table mapping as follows.
 - Map your main audience in Campaign to the OUTCOMEID field in the primary Expanded Outcome table. This is required so that you can select Outcome records for use in the flowchart. Selection must use the OUTCOMEID field, as the same AUDIENCEID field can be repeated in multiple Outcome records.
 - Map your alternate audience in Campaign to the AUDIENCEID field in the primary Expanded Outcome table. This mapping defines the audience on which rest of the flowchart logic should be performed.

Note: If you plan to use Opportunity Detect Outcome data in multiple flowcharts, save the mapped table information into a table catalog and load this catalog in other flowcharts.

- b. Select records where the value in the PROCESSED field in the primary Expanded Outcome table is 0.

This value indicates that the record has not been processed yet.
 - c. Set the value in the PROCESSED field in the primary Expanded Outcome table to 1, to indicate that the record has been processed.

You could write SQL in a Select process to set this value.
 - d. In an Audience process, switch the audience from OUTCOMEID to AUDIENCEID.
 - e. Use the Opportunity Detect data as desired in your flowchart.
 - f. Use a MailList process to assign an offer and update contact history.
6. Use your own scheduling utility or the IBM Marketing Software Scheduler to schedule flowchart runs at the desired interval; for example, every minute.

Outcome format with the Web Service data source connector

You can use a Web Service Connector for incoming transaction data and for outcome data.

When you use the Web Service connector for outcomes, the data is sent to a web servlet. This servlet executes a plugin that can write to a file. You can also do additional programming to enable the plugin to write to a queue or act like a web service.

See the *IBM Opportunity Detect Administrator's Guide* for details about the Web Service connector.

Chapter 7. Component types in Opportunity Detect

Components are the basic building blocks for trigger systems in Opportunity Detect.

Broadly, there are three types of components: those that can send an event to other components, those that return a value that can be used by other components, and the Action component, which writes Outcome data that can be used by external systems.

With these three types of components, you can build logic to suit your business requirements. A set of components configured to carry out your business logic is called a trigger system.

Event components

With the exception of the Simple component, an event component is activated when it receives an event from another component that causes it to evaluate the data against its criteria. After the logic defined within the component is processed, it sends an event to other components.

An event component is said to 'fire' when it sends an event to activate a downstream event component. When you build a trigger system, you specify event components as the incoming events for other event components, creating an unbroken series of events.

An event component fires a positive event when its criteria are met. Some components can also send a negative event when their criteria are not met.

The following event components have additional features.

- The Simple component is the only component that does not require an event to activate it. Instead, it processes all incoming data against its criteria. The Simple component requires at least one transaction data source. When it fires, it sends the first event in the series of events that a trigger system depends upon. This is why all trigger systems require a Simple component.
- The Action component is the only component that writes to a database table or to a customized program via a Web Service connector.

Data components

Data components are invoked by being referenced in another component, which then uses the data returned by the data component in its own evaluation of the customer data. Data components do not fire an event.

The Math, Date, and Boolean components return a single piece of data. The Container and Select components hold one or more rows of data, also called records. Trend components evaluate to true or false.

Action components

When Action components fire, they write the data that you specify to a destination that you specify by choosing the desired data source connector. These outcomes are available for use by external systems.

All trigger systems require an Action component. If a workspace does not contain an Action component, it does not pass validation and you cannot deploy it.

Component type details and examples

These are the components that you can use to build trigger systems.

Event components

Table 16. Event components

Component Type	Description
Backward Inactivity	<p>Listens for the occurrence of a specified event and then checks a specified prior time frame to see whether another specified event has occurred. The Backward Inactivity component fires if the specified prior event did not occur.</p> <p>Example: Fire when a customer makes use of the ATM after more than 1 month of not using any teller services.</p>
Container Manipulator	<p>Add and delete data in Container components and perform operations on this data. Can trigger positive events or negative events.</p> <p>Example: When the dollar amount of product a customer purchased in one month exceeds \$200 (value in the container), offer a discount.</p>
Forward Inactivity	<p>Listens for the occurrence of an event and then waits a specified time to see whether or not another event occurs. The Forward Inactivity component fires when the time period expires without the occurrence of the event it for which it was listening.</p> <p>Example: Fire when a web-trade customer, who usually trades once a month, does not trade for two consecutive months.</p>
Pattern (Match All, Counter, Weighted Counter)	<p>Listens for the occurrence of specified events. Fires if the events occur within a specified time frame. In addition, you can configure a Pattern component to fire a negative event if the pattern is no longer matched within the specified time frame.</p> <p>Example: Fire when customer uses his credit card four times a month during a three month period.</p>
Simple	<p>Fires if specified conditions based on transaction attributes are satisfied.</p> <p>A Simple is the only component type that is activated by incoming transactions and not by an event produced by another component. This is why every trigger system requires at least one Simple component as its starting point.</p> <p>Example: Fire when a customer makes a credit card purchase over \$5000 or an international phone call to Italy.</p>

Data components

Table 17. Data components

Component Type	Description
Boolean Expression	Evaluates data and returns True or False.

Table 17. Data components (continued)

Component Type	Description
Container	<p>Holds records from transactions or a profile, as specified by a Container Manipulator. Other components can use this data in calculations and comparisons.</p> <p>Example: Save the total dollar amount of customer purchases in one month.</p>
Date	<p>Returns a date. Specifies date or date range. Often used in Simple components to specify criteria that incoming transactions must meet.</p> <p>Example: A transaction that occurred on June 8, 2013.</p>
Math	<p>Returns a number. Specifies a mathematical formula that uses numeric data from one or more data sources. Common usage includes the following.</p> <ul style="list-style-type: none"> • In Simple components, to specify criteria that incoming transactions must meet. • In Pattern components, to set the aging factor. • In Forward Inactivity and Backward Inactivity components, to set the time span. • In Action components, to include the results of a numerical calculation in the outcome message. <p>Example: In a Math component, calculate 95% of the customer's available account balance, which is available in a Simple component. In a Boolean component, compare this number to credit card transactions, and if a credit card purchase is equal to the number produced by the Math component, fire a positive event that activates a downstream Action component.</p>
Select	<p>A database query that returns a specified set of records drawn from a Container or another Select component. Other components can use this data in calculations and comparisons.</p>
Trend	<p>Detects changes in activity measured over defined periods of time. Before you can create a trend component, your trigger system must include a Container or Select component that the trend can use as a data source. There are three trend components: Trend, Spike, Exceeded Standard Deviation (ESD).</p> <p>Trend components never fire on their own. They are used internally by Action or Container Manipulator components. You can configure an Action or Container Manipulator component to fire when a Trend Component evaluates to true.</p> <p>For example, the average rolling monthly balance has been increasing or decreasing by 10% over the last three months.</p>

The Action component

The Action component is the only one that writes to the outcome destination. When you configure an Action component, you define the data that is written to the outcome destination when all the criteria specified in the components that the Action component depends upon are met. Outcomes can be written to a database table or can be sent to a customized program via a Web Service connector.

Chapter 8. Common features of components

You build expressions and configure incoming events and time spans when you work with several types of components. In addition, all component editors have a left panel where you configure or view basic attributes. Read this section to understand these commonly used component attributes.

Data Dependencies

The Data Dependencies panel contains a read-only list of transaction data sources that are allowed and used in this component and the audience level associated with these data sources. The fields update as you configure a component.

This panel lists the following information.

Allows Data Source

An attribute of a component that determines which data sources it can use. Values are Any, None, or a single data source.

Allowed data sources are determined as follows.

- In event components, the data sources allowed in the ancestor component that sends the incoming event determine the allowed data sources.
- In Simple components, the data source used in the required Firing Condition determines the allowed data source.
- In data components, all data sources configured in the system are allowed. However, after a data source used in configuring the component, that data source is the only one available in subsequent configuration steps for that component.

Uses Data Source

All direct or indirect data source usage except for Incoming Events. Examples of direct usage occur in Time Spans, Firing Conditions, and values calculated using the Value Selector. An example of indirect usage is a component that refers to a data component such as a Container or Select.

A component can use only one data source, regardless of the number of allowed data sources.

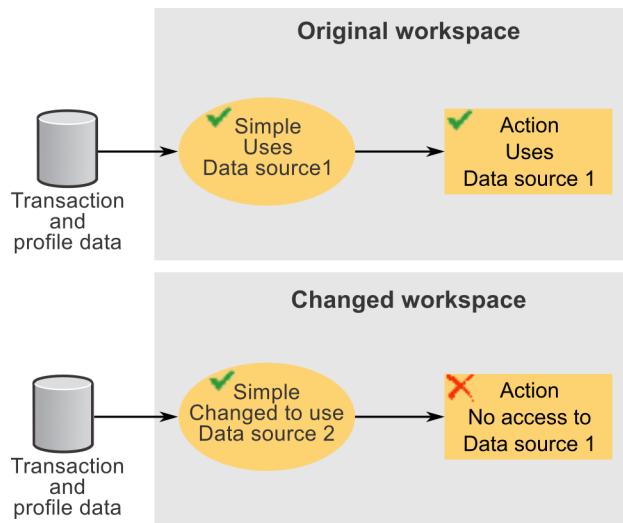
Audience Level

The audience level with which the data source is associated. A trigger system can use data sources from only a single audience.

Opportunity Detect enforces data dependency rules. For example, if you attempt to remove a data source in a component, and a descendent component uses that data source, you can not save the changes. You see an error message that informs you of the reason for the failure and lists the affected descendent components. Before you can make the change, you must edit the descendent components so that they no longer use the data source that you want to remove from the ancestor component.

Example: Changing a data source

In the case illustrated by the diagram below, you would not be able to save the change made in the Simple component until you edited the Action component so that it no longer uses Data source 1.



Special cases in data source availability

Generally, the allowed data sources for a component are the transaction data sources allowed in ancestor components, but there are cases when this rule does not apply.

- Pattern components can be activated by events associated with more than one data source. In such cases, you cannot configure the Pattern component to use transaction data from an ancestor component, because you cannot predict in advance which data source will be involved in activating that component. In the Data Dependencies panel, the **Allows Data Source** field will have a value of None.
- Pattern components that use negative events fire only when the aging out of an event creates a negative condition. Because it is the system's evaluation of this condition (rather than an incoming transaction) that causes the event to fire, no descendent components have access to any transaction data sources.
- Forward Inactivity components fire only when the canceling event does not occur before the time span completes. Because it is the system's evaluation of this condition (rather than an incoming transaction) that causes the event to fire, no descendent components have access to any transaction data sources.

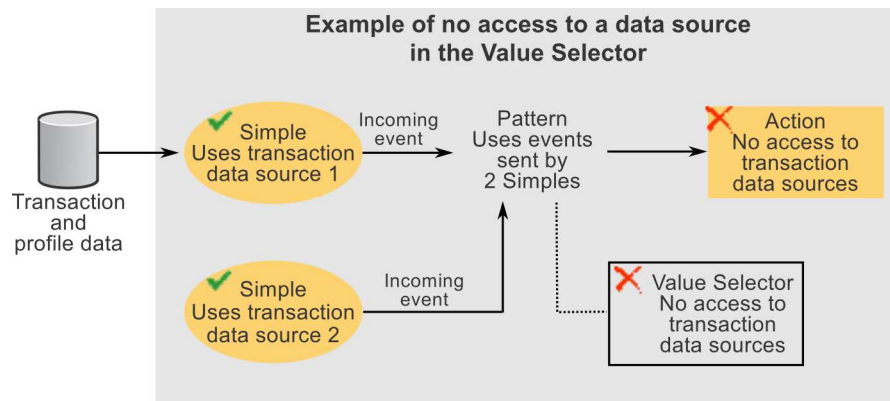
In all of these cases, if profile data exists for the audience level associated with the transaction data source, it is always available for use in configuring a component.

The following examples illustrate data source access in Pattern and Forward Inactivity components.

Example: Pattern components activated by events associated with more than one data source

Pattern components such as Match All and Weighted Count have multiple incoming events. If two of the components sending the incoming events use different data sources, the pattern can not unambiguously determine its data source heritage.

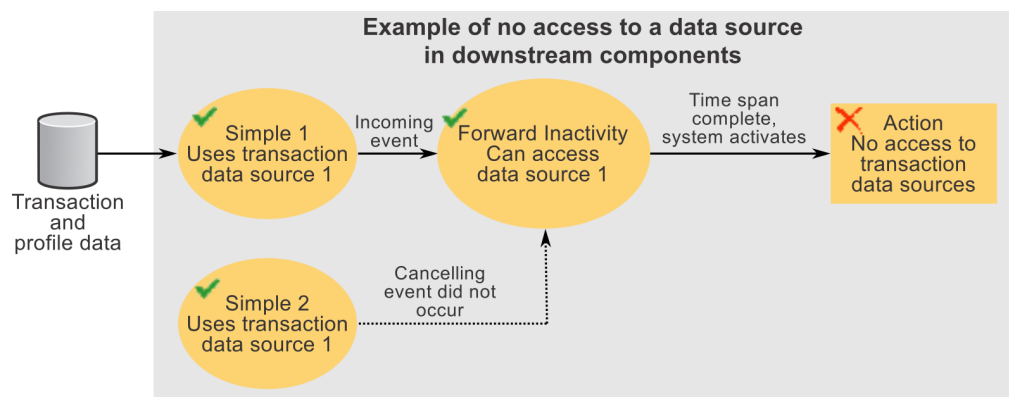
In the diagram below, the Pattern component's allowed data source would be set to None. If you open the Value Selector in the Pattern editor, you would not have access to any transaction data sources.



Example: Forward Inactivity components

A Forward Inactivity component uses incoming and canceling events generated by two Simple components. In the Forward Inactivity's value selector, you can use the transaction data source made available through the component that supplies the incoming event. However, components that are downstream from the Forward Inactivity component have no access to this data source.

This is because it is the system, and not an event generated by a component, that activates the Forward Inactivity for firing when the time span elapses. At the moment when the Forward Inactivity component fires, the system has no access to any data source information.



Related concepts:

“Ancestor and descendent relationships among components” on page 3

Dependent Components

The name and type of any components that use this component. Use this information when you want to make a change to a component, to see what other components might be affected by the change. Read only.

Effective Window

Fields where you can set a start and end date during which an event component is active. Optional.

Related reference:

“Time constants and time units” on page 58

Firing Frequency

Fields where you set a limit on how often an event component produces its event. Optional.

The set of firing frequency fields can be interpreted as follows.

Fire not more than N times in X time units, where a time unit is a combination of type and interval.

For the value of X , you use the Expression Builder. You can select a value based on a constant, a data source field, or a Container, Select, or Math component.

Related reference:

“Time Spans”

Properties

Fields where you can enter a description of the component. Optional.

Incoming Event fields

All event components except the Simple require an incoming event to activate them. The incoming event signals to the component that there is data for it to evaluate against its criteria.

Table 18. Incoming Event fields

Field	Description
Incoming Event	Select a positive or negative event, select the a component type, and select from the available components of that type. Finally, click Add to make your choice one of the incoming events for this component. If you use more than one incoming event, the component is activated when any one of the selected components fires and sends an event.

Time Spans

Time spans are used throughout Opportunity Detect to specify time periods.

Calendar and rolling time spans

There are two types of time span, as follows.

Calendar

Calendar time spans have fixed start and end points, specified using time constants that have defined meanings within the system.

Rolling

Rolling time spans have a defined number of time units, but the start and end points are not fixed.

Time Span fields

Table 19. Time Span fields

Field	Description
Quantity	You can enter a number directly in this field, or use the Value Selector to set the quantity of time units for the time span.
Select Value	You can use Value Selector to set the quantity of time units for the time span.
Time unit fields	<p>You use the two time unit fields to set the type and unit of the time span.</p> <p>The type can be Calendar or Rolling. The behavior of these time span types varies, depending on the component where they are used. Calendar time spans work in conjunction with time constants, which have defined meanings.</p> <p>Tip: See related topics for details.</p> <p>The available time units are:</p> <ul style="list-style-type: none">• Second <p>Applies to calendar time spans only, not to rolling time spans. A time unit of "second" is not valid for a rolling time period. For example, a rolling day has an hour, minute, and second starting point, such as 13:12:44. However, Opportunity Detect does not track units smaller than seconds on the incoming transaction, so there is no way to track units smaller than seconds and therefore no way to track rolling seconds.</p> <ul style="list-style-type: none">• Minute• Hour• Day• Week• Month• Quarter• Year

Related concepts:

"Rolling time span in Forward Inactivity components" on page 100

"Calendar time span in Forward Inactivity components" on page 101

"Calendar time span in Pattern components" on page 88

"Rolling time span in Pattern components" on page 91

Related reference:

"Time constants and time units" on page 58

"Firing Frequency" on page 56

Time constants and time units

When you configure a calendar time span or an effective window, you use time constants and time units. Time constants, the starting points of time units, and the end points of irregular time units have specific definitions within Opportunity Detect.

Time units and time constants are calculated relative to the value of Now within the component.

The value of the Now date constant

The value of the Now date constant is calculated relative to the timestamp of the transaction being processed.

When a trigger system starts processing a transaction, the value of the Now time constant is simply the timestamp of the transaction as received from the transaction data source. However, as a trigger system moves through its logic, the value of Now changes as follows.

- Forward Looking components have a time span that determines how long the component waits after it receives its activating event before it fires its event to activate a child component. The Forward Looking component adds this time span to the transaction timestamp it received, and this becomes the timestamp of the event the Forward Looking component fires.

This event timestamp is passed as the value of the Now time constant for the immediate child component of the Forward Looking component.

- Pattern components can be configured to send a negative event when the aging out of an event creates a negative condition. The timestamp of the negative event becomes the value of the Now time constant for the immediate child component of the Forward Looking component of the Pattern component.
- For an End of Run artificial transaction, the value of Now is the timestamp of the last transaction in the set of data being processed.
- For an End of Day artificial transaction, the value of Now is 12 A.M. of the next day.

Starting points for calendar time units

The starting points for calendar time units are defined as follows.

- A calendar week starts at midnight on Sunday.
- A calendar day starts at midnight.
- A calendar year starts at midnight of January 1.
- Calendar quarters start at midnight on January 1, April 1, July 1, and October 1.
- Beginning of Selected Date is midnight of a day that you select.
- Minimum Date is 1970-01-01 00:00:00
- Maximum Date is 2038-01-19 03:14:07

Date constants that return a date

The following date constants return a date.

- Beginning of current minute
- Beginning of current hour
- Beginning of current day

- Beginning of current week
- Beginning of current month
- Beginning of current quarter
- Beginning of current year
- Beginning of selected date
- Maximum date
- Minimum date

Date constants that return an integer

You might want to use a date constant that returns an integer to determine when a transaction took place. For example, you can create a Boolean expression that returns True when the Current day of the month = 15

The following date constants return an integer.

Table 20. Date constants that return an integer

Date constant	Value
Current Second	1 - 60
Current Minute	1 - 60
Current Hour	1 - 24
Current Day of the Month	1 - 31
Current Day of the Week	1 - 7 Sunday = 1
Current Week	<ul style="list-style-type: none"> • January 1 = day 1 of week 1 • In a leap year, December 30 is in week 53
Current Month	1 - 12
Current Quarter	1 - 4
Current Year	Four digits

Related concepts:

“Ancestor and descendent relationships among components” on page 3

Chapter 19, “Artificial transactions in Opportunity Detect,” on page 121

“Calendar time span in Pattern components” on page 88

“Rolling time span in Forward Inactivity components” on page 100

Related reference:

“Time Spans” on page 56

“Effective Window” on page 56

Firing Condition fields

A firing condition is required for Simple components and is optional for Action components. You can define an expression to serve as a firing condition. The expression can be an existing Boolean expression component, or you can define an inline expression that is available only within the component where it is defined.

Table 21. Firing Condition fields

Field	Description
The firing condition will be evaluated using	If you create more than one expression, the expressions are evaluated using the logical operator you choose here. Options are And and Or . You can also use And and Or logical operators within inline expressions to build nested logic.
Type	Click the Click to add firing condition link and select Add existing Boolean expression or Add new inline expression . A window opens, and you can specify your firing condition.

Functions in Opportunity Detect

Functions perform some additional processing on a record set returned by a Container or Select component. Functions allow you to apply commonly used database functions to the records.

Functions are available in the following component types.

- Container
- Select
- In the Value Selector in a component that allows a Container or Select component as a data source.

Function definitions

The following table describes how functions work.

Function	Description
Average	Use an average of the values in the field.
Count	Counts the number of times this field occurs in the record set. Returns an integer.
Count Distinct	Counts the number of times this field occurs in the record set, excluding fields with duplicate values. Returns an integer.
First	Use the value with the oldest timestamp.
Group By	In Select and Container components, groups together all the records that have identical values in this field. Returns one or more groups of rows. The Group By function can be applied to only a single field in a component.
IsMemberOf	Compares a single value with a single field in each row contained in a Select component. See the expanded explanation elsewhere in this chapter.
Last	Use the value with the most recent timestamp.
Maximum	Use the highest value in the field. Returns an integer.
Minimum	Use the lowest value in the field.
Mode	Use the value of the most frequently occurring value in this field. When a tie situation exists, an age factor is used; the oldest value is returned in this case.
Sum	Use the total of the values in the field.
Standard Deviation	Measures how much variation or dispersion from the average exists in the values of this set of fields, using the STDEVP formula. Returns an integer.

Related concepts:

“The IsMemberOf function”

“Container components” on page 79

Chapter 14, “Select components,” on page 77

The IsMemberOf function

The IsMemberOf (IMO) function provides an efficient method for identifying and acting on large quantities of lookup table data to determine which records contain a specified value.

Any component where you can build a Boolean expression can use the IMO operator. That is, it can be used in Simple, Container Manipulator, Action, and Boolean expression components.

The IMO compares a single value with a single field in each row contained in a Select component. The field used for comparison in the Select component cannot use a function. A function reduces the data returned by a Select component to a scalar value, and the IMO requires a set of values.

Most often the Select is obtains its data from a lookup table, but it can also be obtain its data from a Container component, or another Select component.

The data type of the single value and the data type of the field in the Select must be the same. The value used for comparison is any value that can be created in the Value Selector (for example, integer, double, or string).

Usage

IMO is the most efficient way to check whether a value is equal to any value in a large group of values.

Suppose your organization offers 1000 products for sale. Of the 1000 products, there are 300 for which you send a rebate coupon. On first thought, you might create a Simple component that recognizes each one of the 300 product numbers. However, this method has some drawbacks.

- A Simple Event that identifies 300 product numbers is cumbersome to create.
- A Simple Event that identifies 300 products is difficult to maintain should you need to change the products included in the promotion.
- There is a limit to the number of characters that can be included in every component. The 300 product codes that need to be identified may exceed the length boundary of the Simple component.

Opportunity Detect provides the IMO function to effectively address these issues.

Usage example

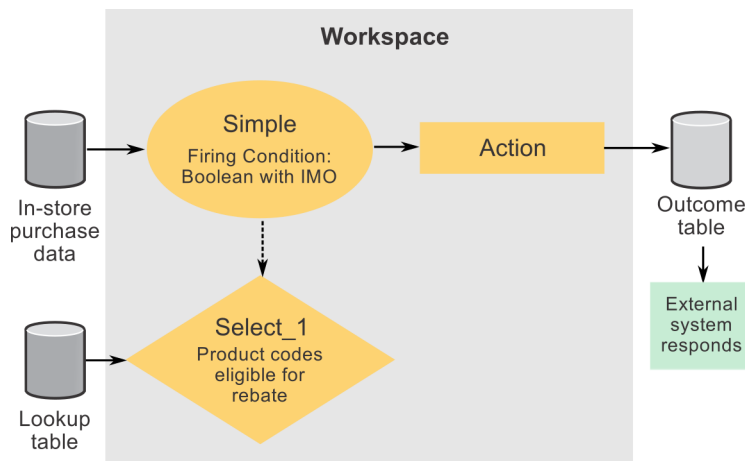
The following steps describe an efficient method for identifying the 300 products in the example above.

1. Create a lookup table listing all 1000 product codes with a flag that indicates whether the product offers a rebate.

Product codes can easily be added to or removed from this table, and the associated flag can be changed as products are included or removed from the promotion.

2. Create a Select component, named Select_1 in this example. This component queries the lookup table for records that have a flag indicating that the product is included in the promotion. Include the product code in the Output panel. This step efficiently identifies only the products that are included in the promotion.
3. Create a Simple component that processes transaction records and uses an inline Boolean expression in its Firing Condition. The Boolean expression uses IMO to compare a product code in a transaction record to the data returned by the Select component. If the Boolean expression is true, the Simple component fires its event. In this example, if the transaction data source is named StorePurchases, and it contains a field named ProductCode, the expression would look like this: `StorePurchases.ProductCode IsMemberOf Select_1`
4. Create an Action component that includes any data from the transaction data source that is needed to process the rebate offer.

The following diagram illustrates this usage example.



Related concepts:

“Functions in Opportunity Detect” on page 60

Chapter 9. Building expressions using the Expression Builder

The Expression Builder is used throughout Opportunity Detect to compare strings and numbers, perform math operations, and apply database functions.

The Expression Builder uses standard mathematical syntax. String constants must be surrounded by double quotes.

For operands, you can enter constant values using the keyboard, or you can use the Value Selector to specify the following.

- Field values from a transaction or profile data source.
If a Named Value list is configured for a field, you must use the Value Selector to add the value.
- Values calculated in expression components.
- Field values from records returned by a Select or Container component.

Your choices differ depending on the component you are working with, and on the data type you select.

CAUTION:

If you enter an operand using the keyboard, and you insert an extra space between operands, the expression does not validate even if you remove the extra space. You must clear the expression and create it again.

Boolean, comparison, and math operators

You can use a variety of Boolean, comparison, and math operators to build an expression.

Boolean operators

Table 22. Boolean operators in the Expression Builder

Operator	Definition	Example
And	Returns TRUE when all conditions are true.	Look for transactions that have a price per share greater than \$99.99 AND were processed on June 15, 2001 AND were for accounts in New York.
Or	Returns TRUE when at least one of the conditions is true.	Look for transactions that have a price per share greater than \$99.99 OR were processed on June 1, 2001 OR were for accounts in New York.

Comparison operators

The following comparison operators are available. They are listed in their order of precedence.

- Equal
- Not Equal
This is available only in the Simple Event component.
- Greater Than
- Greater Than or Equal To

- Less than
- Less Than or Equal To
- Like

Math operators

The following math operators are available. They are listed in their order of precedence.

- Multiply
- Divide
- Add
- Subtract
- Modulo (finds the remainder of division of one number by another)
- Exp (raises a number to the power of another number)

Order of operations

This is the order of precedence for operators.

- multiplication
- division
- addition
- subtraction

If you use a comparison in an expression that uses a Boolean operator, you must enclose the operand that follows the Boolean operator in parentheses whenever the result of the calculation would be ambiguous if you do not use parentheses. For example, you would need parentheses to clarify whether you want `trans.frequency < 3 AND trans.deposit > 2 * (prev.deposit + 50)` or `trans.frequency < 3 AND trans.deposit > (2 * prev.deposit) + 50`.

Value Selector fields

In component editors, you specify values for expressions using the Value Selector.

Table 23. Value Selector fields

Field	Description
Output Data Type	The data type of the value returned by this expression. <ul style="list-style-type: none"> • Integer • Boolean • Currency • Date • Double • String

Table 23. Value Selector fields (continued)

Field	Description
Type	The data type of the field in the source data. Available options vary, depending on the selected Output Data Type. <ul style="list-style-type: none"> • Boolean Expression • Constant • Container • Data Source • Math Expression • Select • Date Expression
Data Source	Available when you select Data Source as the Type. Your configured data sources are available for selection. Availability depends on the component's data source dependency constraints.
Field	Available when you select Data Source, Container, or Select as the Type. The fields in the selected data source that match the selected data type are available.
Value	Available when you select Constant as the Type. You can enter a value here.
Function	Available when you select Container or Select as the Type. You can apply a database function here.
Expression	Available when you select Boolean Expression or Date Expression as the type. The relevant expression components in your workspace are available.

Chapter 10. Regular expressions in Opportunity Detect

In Opportunity Detect, you use regular expressions in two situations.

- When you create a Real time file connector in the Server Groups page, you use a regular expression to match the pattern used in file names.
- When you create a Boolean expression in a component and you select the **Like** operator, you can use a regular expression to set the criteria for comparison.

Opportunity Detect uses the Streams standard toolkit for matching regular expressions. Opportunity Detect supports the POSIX extended regular expressions standard.

The regular expression must conform to the Streams Processing Language requirements, described here: https://www-01.ibm.com/support/knowledgecenter/SSCRJU_3.2.0/com.ibm.swg.im.infosphere.streams.spl-language-specification.doc/doc/primitivetypes.html

Take care that the pattern you specify exactly matches your intent. Some level of testing is always advisable to verify that your patterns are actually matching the required expressions. You can use a trial and error process to design patterns, starting with low complexity and changing them bit by bit to achieve the required result. Pay particular attention to escaping backslashes.

Special characters

Here is a summary of special character usage in POSIX regular expressions.

- Period (.) : Matches any character.
- Anchors (^, \$) : The (^) anchor defines the start of the expression, and the (\$) anchor defines the end of the expression.
- Asterisk (*) : A quantifier that matches a single character or group of characters any number of times.
- Plus (+) : A quantifier that matches a single character or a group, one or more times.
- Question mark (?) : A quantifier that represents optional items.

Bracket expressions

A bracket expression represents a class of characters, any one of which could be a match a single character. For example [a-c] is a bracket expression that will match any of the characters a, b, or c. For example: the regex [a-c]+ will match aaa, abc, ca, etc; or any string containing a sequence of at least one character from the set a, b, or c followed by any number of characters also from that set.

There are other forms of bracket expressions. For example, [a-c] could be also specified as [abc]. Within a bracket expression, there are collating elements. It has the form [.col.]. (There might be other forms.) A collating element is a character or group of characters that act as a single character in a bracket expression. For example, if [.ae.] is a collating element, then it can be used within a bracket expression [[.ae.]bc], which states: match any of the characters "ae", b, or c. In other words, it forces ae to be treated as a single character.

Table 24. Character classes

POSIX	Description	ASCII
<code>[:alnum:]</code>	Alphanumeric characters	<code>[a-zA-Z0-9]</code>
<code>[:alpha:]</code>	Alphabetic characters	<code>[a-zA-Z]</code>
<code>[:blank:]</code>	Space and tab	<code>[\t]</code>
<code>[:cntrl:]</code>	Control characters	<code>[\x00-\x1F\x7F]</code>
<code>[:digit:]</code>	Digits	<code>[0-9]</code>
<code>[:graph:]</code>	Visible characters (that is, anything except spaces, control characters, etc.)	<code>[\x21-\x7E]</code>
<code>[:lower:]</code>	Lowercase characters	<code>[a-z]</code>
<code>[:print:]</code>	Visible characters and spaces (that is, anything except control characters, etc.)	<code>[\x20-\x7E]</code>
<code>[:punct:]</code>	Punctuation and symbols	<code>[!"#\$%&'()*+,-./:;<=>?@[\]^_`{ }~]</code>
<code>[:space:]</code>	All whitespace characters, including line breaks	<code>[\t\r\n\v\f]</code>
<code>[:upper:]</code>	Uppercase letters	<code>[A-Z]</code>
<code>[:xdigit:]</code>	Hexadecimal digits	<code>[A-Fa-f0-9]</code>

Quantification

The question mark makes the preceding token in the regular expression optional. For example, `colou?r` matches both `colour` and `color`.

The star (*) tells the engine to attempt to match the preceding token zero or more times. The plus sign (+) tells the engine to attempt to match the preceding token one or more times.

An additional quantifier allows you to specify how many times a token can be repeated. The syntax is `{min,max}`, where `min` is zero or a positive integer indicating the minimum number of matches, and `max` is an integer equal to or greater than `min` indicating the maximum number of matches. If the comma is present but `max` is omitted, the maximum number of matches is infinite.

For example:

- `{0,1}` is the same as `?`
- `{0,}` is the same as `*`
- `{1,}` is the same as `+`

Omitting both the comma and `max` tells the engine to repeat the token exactly `min` times.

You could use `\b[1-9][0-9]{3}\b` to match a number between 1000 and 9999. `\b[1-9][0-9]{2,4}\b` matches a number between 100 and 99999. Notice the use of the word boundaries.

Grouping

Single characters, or expressions matching single characters, enclosed in parentheses (round brackets), are treated as a regular expression matching a single

character. That is, quantification and other rules apply to the group in the parentheses as a whole.

Alternation

Two regular expressions separated by the special character vertical-line ('|') match a string that is matched by either.

For example, the regular expression "a((bc)|d)" matches the string "abc" and the string "ad".

Single characters, or expressions matching single characters, separated by the vertical bar and enclosed in parentheses, are treated as a regular expression matching a single character.

Example for file name matching

You might create the following regular expression to match timestamp suffixed file names used with the Real time file connector.

```
Detect\.a\.trans\.[0-9]{8,14}
```

This expression matches file names with the common prefix Detect.a.trans and ending with timestamp digits of length greater than 8 and less than 14. This is done because file names can have 8 digits for the basic date (4 for year, 2 for month, 2 for date) and 6 extra digits for more granular timestamps (hh:mm:ss).

```
Detect.a.trans.20100901  
Detect.a.trans.20100908  
Detect.a.trans.20100922  
Detect.a.trans.20101001  
Detect.a.trans.20101008  
Detect.a.trans.20101022  
Detect.a.trans.20101201  
Detect.a.trans.20101208  
Detect.a.trans.20101222  
Detect.a.trans.20101222  
Detect.a.trans.20101223040506  
Detect.a.trans.20101223033240
```

Useful links for POSIX regular expressions

- OpenGroup POSIX regular expression specification:
http://pubs.opengroup.org/onlinepubs/009696899/basedefs/xbd_chap09.html
- <http://www.regular-expressions.info/posix.html>
- Wikipedia regular expressions:
https://en.wikipedia.org/wiki/Regular_expression#POSIX_basic_and_extended
- Wikibooks POSIX regular expressions:
https://en.wikibooks.org/wiki/Regular_Expressions/POSIX_Basic_Regular_Expressions

Chapter 11. Date, Math, and Boolean expression components

Expression components evaluate customer data and return a value that can be used by other components. You can create similar logic by using inline expressions within components, but creating an expression component for frequently used calculations allows you to re-use this logic.

For example, you might want to create a Date expression that returns a date that is three weeks before the current date. You could use this component in multiple workspaces by copying it with a component reference.

When you configure expression components, you use the Expression Builder, which provides options appropriate for the expression type.

There are three expression components.

- Date
- Math
- Boolean

Date components and dates in inline expressions

Date components return a single calendar date.

To create a date in a Date component, you select a date value as the first operand in the Expression Builder, and then add or subtract specified time periods. The date expression always has exactly one date value and one time period.

To create a rolling date in a Date component, you select Now for the first operand. For example, the expression `Now - Period(1, Week)` equals one rolling week prior to Now as defined in the trigger system. The expression `Now - Period(1, Day)` equals one rolling day prior to Now as defined in the trigger system.

To create a calendar date, you can use a time constant in a Date component or an inline expression, depending on the result you want to obtain. For example, Beginning of Current Week is the time constant for the beginning of the current calendar week, relative to Now as defined in the trigger system. You can use Beginning of Current Week in the following ways.

- To obtain the beginning of two calendar weeks ago, use a Date component. In the date expression, select Beginning of Current Week as the first operand and subtract a one week time period.
- To obtain the beginning of the current calendar week, use an inline expression within a component. For example, in a Select component, you use a WHERE clause to compare Beginning of Current Week with a transaction date.

You cannot specify the beginning of the current calendar week with a Date component. The Date component can obtain the transaction date in an expression, but the only thing it can do with it is add or subtract some time span.

Related concepts:

Chapter 9, "Building expressions using the Expression Builder," on page 63

Chapter 12. Simple components

Trigger systems in IBM Opportunity Detect all have Simple components as their basic building block. Simple is the only component type that is triggered by incoming transactions in the transaction data feed, and not by an event produced by another component. This is why every trigger system requires at least one Simple component.

Simple components watch for the occurrence of a single transaction that matches specified criteria. A Simple component fires a positive event when the criteria are met.

A Simple component must contain at least one comparison to a transaction data source field, in the form of a Boolean expression. This Boolean expression can be either an inline expression or a reference to an expression component. Without a transaction, the Simple component would never fire.

Simple component example

In its most basic form, the Simple Event component compares one field from a transaction data source to a specified value.

For example, you could look at the value of the field *Transaction Code* found in the data source *Account Transactions* and test to see if it matches the string constant *ACH Credit*.

Chapter 13. Action components

The purpose of an Action component is to send a notification that a specified behavior has occurred by writing data to a database or sending the data to a web service. Like all event components, an Action component fires and writes its outcome when it receives an incoming event from another event component.

The audience ID is automatically included in each record that the Action component writes. In addition, you can include a static message and values created in the Value Selector.

The data source connector that you choose for the outcome determines the format and location of the data that the Action component writes. When you choose the Table type of data source connector, you can also map the connector to the desired database table.

Action component examples

Examples of the outcome data that an Action can write include the following.

- Customer ID
- The identifier of the Action component that caused the outcome to be written
- The timestamp when the Action component triggered
- A message consisting of a string you specify and optional additional information derived from the data available in the trigger system.

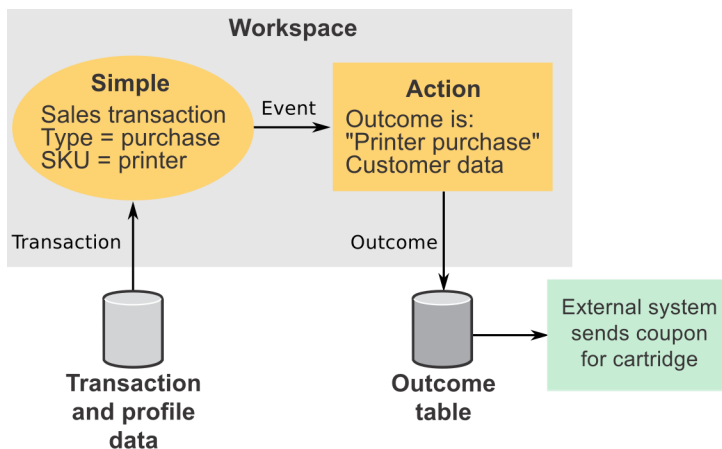
The following diagram illustrates the use of the Action component in a basic trigger system configured as follows.

- A Simple component fires when a transaction data source contains a transaction of type "purchase" with an SKU that indicates that the customer purchased a printer.
These criteria are specified in the Firing Condition panel using inline expressions.
- An Action component has the Simple component as its Incoming Event.

When the Simple component receives a transaction that meets the criteria set in its Firing Condition, the trigger system does the following.

- The Simple component fires and sends an event that activates the Action component.
- The Action component fires and writes an outcome that consists of the string "Printer purchase" and the customer ID.

The outcome is specified in the Outcome panel.



Outcome fields

In Action components, you can define the data that is written to the database or web service.

Table 25. Outcome fields

Field	Description
Message	You can enter any string, which will be written to the database or web service when the Action component fires.
Additional Information	<p>You can use the value selector to choose data from any data source or component that is available to the Action component. Availability depends on the component's data source dependency constraints.</p> <p>You can attach a label to each outcome value that you specify. These labels become the names of fields in the outcome. When you use a queue connector for outcome data, a label is required.</p> <p>When you include data from a Container or Select component, you can select Column or Tabular as the data type. Use Column for fields that are aggregated into a single value, and Tabular for records that contain a set of fields.</p>

Chapter 14. Select components

Select components perform operations that are similar to the SQL SELECT database query with optional WHERE clauses. A Select component holds a set of records that you define, and on which you can perform further operations.

Select components can use data from the following sources.

- Container components
- Other Select components
- Lookup tables

You can give meaningful names to the fields in the records that Select passes to downstream components.

You can also apply functions to the fields, and group and sort the fields to further filter the results.

When you use more than one WHERE clause to refine the record set, you can specify whether to evaluate them using either AND or OR.

Fields in the Output panel of the Select component editor

Table 26. Fields in the Output panel of the Select component editor

Field	Description
Name	Name of the field in the originating data source. Read only.
Data Type	Data type of the originating field. Read only.
Output Name	Name of the field in the record that Select passes on to downstream components. You can supply a different name, or retain the original name. When the Trigger Table data source connector is used to process the trigger system that contains the Select, these labels become the names of fields in the trigger table. Optional.
Output Function	Function to apply to the data in this field. Optional. Tip: For help with functions, see the related topics.
Output Sort	You can apply an ascending or descending sort to the records. Optional. When multiple fields have sort applied, the sort is applied based on the order in which the rows are shown.
Primary Date	If selected, the field is designated as a timestamp field.

Related concepts:

“Functions in Opportunity Detect” on page 60

Chapter 15. Container and Container Manipulator components

Container components store data that you define for a specified length of time. Container components depend on Container Manipulator components to feed them the data that they store.

Container components

Container components provide a way to create a data structure similar to a database table in your trigger systems. You use the Container to define the table that holds the data inserted by a Container Manipulator.

Optionally, you can also define the following.

- The aggregation, or database function, applied to each field in the table
- The number of records that are retained and how long to retain them

The Container never fires an event.

Container usage

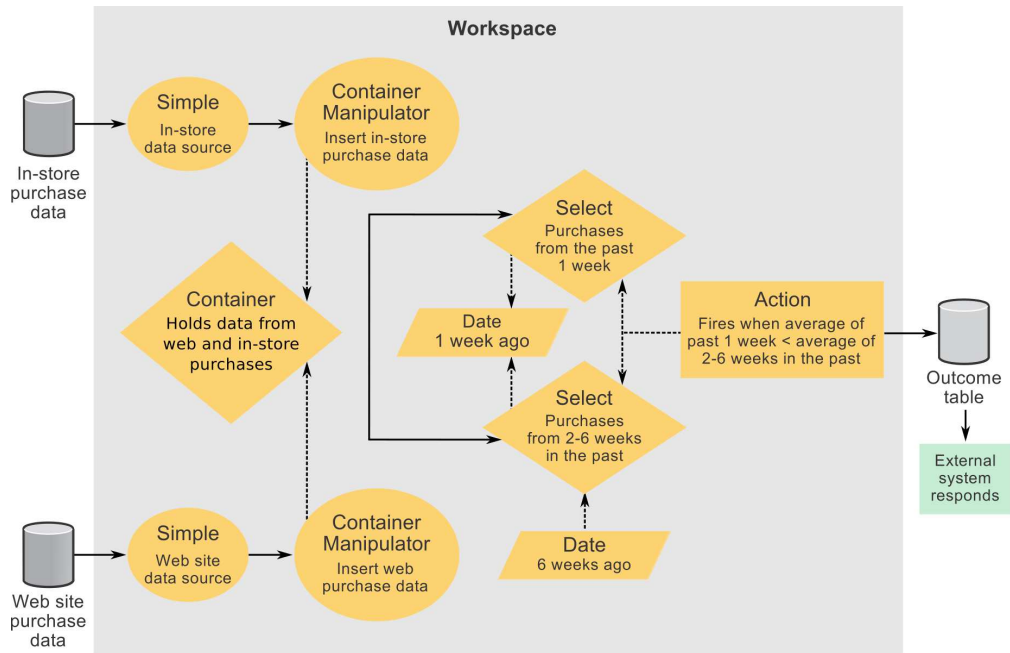
Containers can store data from multiple data sources. A single Container can also be used by multiple trigger systems.

Multiple data sources

Here is an example of a trigger system with a Container that holds data from two data sources. Suppose you have transaction data from two channels: your web site and in-store purchases. Fields in the data from each channel may differ, but both channels include the following fields.

- date of purchase
- SKU number
- quantity
- price

You could use a single Container to hold data from both feeds, and use this Container in a trigger system designed to respond when a customer's activity declines. The trigger system illustrated by the following diagram evaluates the average customer spending from two time periods, and responds when spending decreases.



Multiple trigger systems

Containers can be designed to be useful within a single trigger system, or across multiple trigger systems. For example, a Container could be limited to hold information on only the color and size of garments a customer purchases. You might draw data from this Container for use within a single trigger system.

In contrast, you could define a Container that holds as much data as is required across all the trigger systems that will use it, and then share the Container across trigger systems.

You should consider processing efficiency when deciding whether to create a limited or broad Container. Containers that are expected to hold a very high number of records should generally limit the number of fields per record.

Types of data that a Container can hold

The fields you define in Container components can hold any of the following data types.

- Boolean
- Currency
- Date
- Double
- Integer
- String

Table Definition

For each field you want to define, you select a data type and an aggregation function to apply to the field. You also give the field a name.

In a Container Manipulator, you specify what data is inserted into the fields you define in the Container.

How aggregation works

Use aggregation to apply commonly used database functions to reduce multiple data items to a single data item.

When you enable aggregation in a Container component, rows are grouped according to the time span you specify in the **Aggregate by** field in the Aggregation panel. For example, if the time unit is **Day**, then the timestamps are used to identify which calendar day a row belongs to, and there is one group for each day. If the time unit is **Week**, there is one group for each week, and so on.

Aggregation functions are performed on all the fields of each row within a time span group. You set the function used on each field in the **Aggregation** column in the Table definition panel. The Timestamp field is the exception to this in that it is automatically set to the **Maximum** function and you cannot change it.

For example, suppose you have enabled aggregation and the following functions are applied to the fields.

Table 27. Example container

Field	Aggregation
timestamp	Maximum
integerField	Sum
doubleField	Minimum
stringField	Last Inserted

This aggregation is the equivalent to applying the following select statement to the example container.

```
SELECT MAX(timestamp),  
SUM(integerField),  
MIN(doubleField),  
LAST(stringField)  
WHERE timestamp => [beginning of time unit] AND timestamp < [end of time unit]
```

This select statement is repeated for every group of rows that falls within a time unit. Each statement results in a single row, one per time unit group. These rows are the new contents of the container.

If the example above had used the **Group by** function for stringField instead of the **Last** function, the select statement would look like this.

```
SELECT MAX(timestamp),  
SUM(integerField),  
MIN(doubleField)  
WHERE timestamp => [beginning of time unit] AND timestamp < [end of time unit],  
GROUP BY stringField
```

This results in a single row for every unique stringField value within a time unit group. With **Group by**, a time unit group contains multiple rows if there are multiple unique values in the field where the **Group by** function is applied.

Available aggregation methods

The aggregation methods available for a field depend on the data type. The following table describes all of the aggregation methods.

Table 28. Aggregation methods

Aggregation method	Description
Last Inserted	Use the most recently inserted value in the field.
Minimum	Use the lowest value in the field. In string fields, this returns the string that comes first in an alphabetical sort.
Maximum	Use the highest value in the field. In string fields, this returns the string that comes last in an alphabetical sort.
Sum	Use the total of the values in a numerical field.
And	For Boolean values, use the false value if one is present; if all values are true, use the true value.
Or	For Boolean values, use the true value if one is present; if all values are false, use the false value.

Aging

In the Aging panel of the Container component, you can specify how long data is retained. You select the time unit and the number of time units, such as 2 weeks or 1 day. You also specify whether the time span is calculated on a calendar or rolling basis.

Overflow

In the Overflow panel of the Container component, you can set a limit on the number of records, or rows, that the component stores. You can specify whether to delete rows based on the lowest or highest value of any defined field that you specify.

When a maximum number of rows is reached, the overflow settings determine what happens when a request is made to store another row. The options in the **If the maximum number of rows is exceeded delete** field are as follows.

- Minimum - deletes the item containing the smallest value in the specified field.
- Maximum - deletes the item containing the largest value in the specified field.

Detailed Container example

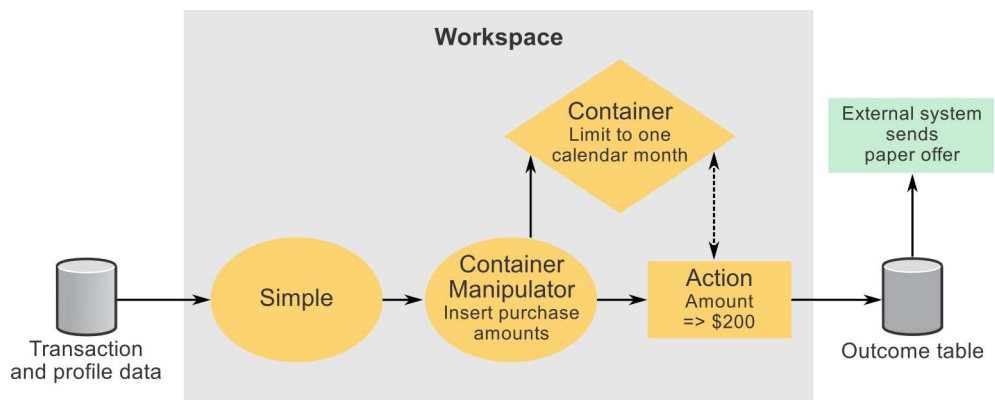
Suppose that your goal is to offer customers a discount for printer paper when they purchase more than \$200 in printer paper in a calendar month. You can set up a trigger system as follows.

- Create a Simple Event component that fires an event when it receives printer paper purchase data.
- Create a Container component to store dollar amounts of printer paper purchases.
 - In the Aging panel, set a calendar month time span to set a limit on the length of time that the data is retained. The Container is cleared at the end of every month.
- Create a Container Manipulator, CM1, that listens for the Simple component event.

- In the Action panel, specify that the purchase amount is inserted into the Container.
- Create an Action component that listens for the CM1 event.
 - In the Firing Condition panel, define a Boolean expression that applies the Sum function to the dollar amount field in the Container and returns True when the amount is equal to or greater than \$200.
 - Optionally, add the same Boolean expression in the Additional Information section of the Outcome panel. This includes the dollar amount in the data that is written to the Outcome table.

Each time a customer purchases printer paper during a calendar month, the dollar amount is saved. If the total amount is equal to or greater than \$200 in any month, the Action component fires for that customer.

Tip: This example uses a limited Container that contains only the data needed for the single trigger system described in this scenario.



Related concepts:

“Functions in Opportunity Detect” on page 60

“Container Manipulator components”

Container Manipulator components

You use Container Manipulator components to add and delete data in Container components.

In the Container Manipulator component, you can map fields from a data source to the fields you defined in the Container.

Firing conditions

Firing conditions are optional. If no firing conditions are set, the Container Manipulator fires a positive event when it completes its processing.

If firing conditions are specified, the Container Manipulator evaluates the data to see whether it meets the firing condition criteria. If the data satisfies the criteria, the Container Manipulator performs its defined operations on a specified Container component and then sends its event. If the data does not satisfy the firing condition criteria, the Container Manipulator does nothing.

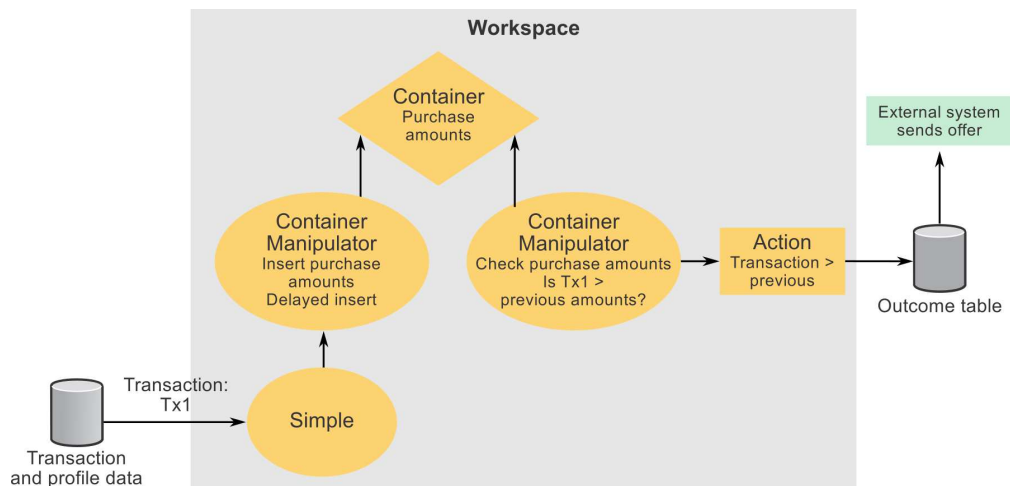
A Container Manipulator component can also be configured to send a negative event when the criteria in its firing condition are not met.

Tip: The only required field in a Container Manipulator is the Incoming Event. Therefore, you can use a Container Manipulator without referring to a Container component. For example, you can create one or more Boolean expressions in the Firing Condition panel that control when the Container Manipulator fires its event.

Delayed insert

Delayed insert provides a special way to insert a value into a container. When you choose the **Delayed insert** option, the system adds the values to the contents of the Container only after all logic in the trigger system has finished processing the transaction.

For example, suppose you have a trigger system like the one shown in the following diagram. It includes a Container that holds purchase amounts. You want to detect any purchase that is larger than the previous purchases made by that customer. If you perform a delayed insert, you can check the incoming purchase against previous purchases before the incoming purchase is inserted into the Container.



Detailed Container Manipulator example

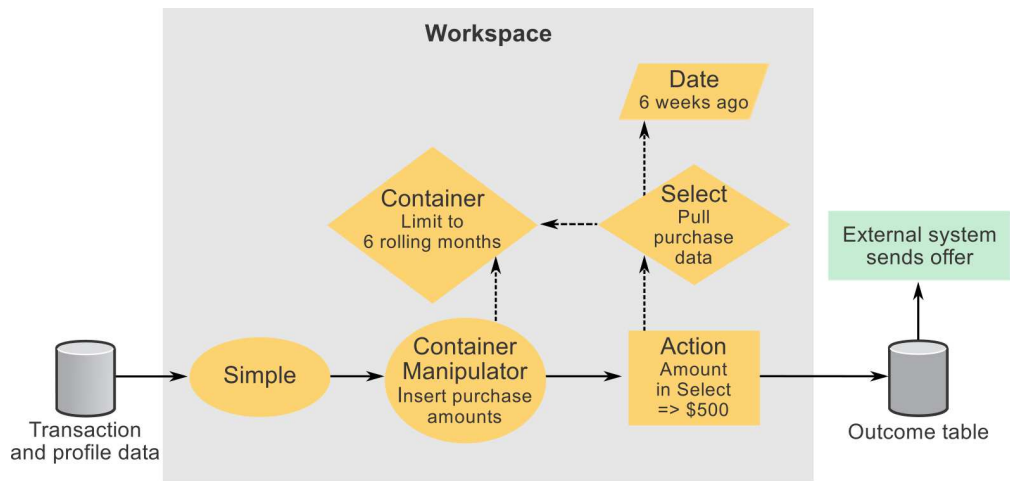
Suppose your goal is to offer a discount to customers who purchase \$500 or more within a 6 week period. You can set up a trigger system as follows.

- Create a Simple Event component that fires an event when a customer makes a purchase.
- Create a Container to store dollar amounts of purchases.
 - In the Aging panel, set a rolling six month limit on the amount of time to retain data, to prevent overload.
- Create a Container Manipulator, CM1, that listens for the Simple component event.
 - In the Action panel, specify that the purchase amount is inserted into the Container.

Tip: Unlike the simplified example shown in the detailed Container example, this Container holds more data than is needed for the current scenario, so additional components are needed to filter the data.

- Create a Date component that returns a date of 6 weeks before the current date.
- Create a Select component.
 - In the Select panel, pull the purchase data from the Container.
 - In the Where panel, use the date returned by the Date component to limit the data to the last six weeks.
- Create an Action component that listens for the CM1 event.
 - In the Firing Condition panel, define a Boolean expression that applies the Sum function to the dollar amount field in the Select and returns True when the amount is equal to or greater than \$500.

Each time a customer makes a purchase, the dollar amount is saved and the total purchases for the last six weeks are evaluated for that customer. When the amount is equal to or greater than \$500, the Action component fires for that customer.



Related concepts:
 “Container components” on page 79

Inserting data into a Container component with a Container Manipulator

Use this procedure to use a Container Manipulator component to insert data into a Container component.

Before you begin

Your workspace should contain the Container component where you want to insert data. Your workspace should also contain any other components whose data you want to use when defining the data to be inserted.

Procedure

1. Open a Container Manipulator editor, give your component a name, and select an incoming event.

2. In the Action panel, select **Insert** as the action, select the desired Container component, and click **Add**.
The Insert Mapping window opens, listing the fields defined in the selected Container component.
All Container components automatically contain a Timestamp field, mapped to the **Now** time constant.
3. Click the link in each **Value** field and use the Value Selector to define the data that the Container stores in each listed field.
You can choose data from any data source or component that is available to the Container Manipulator. Availability depends on the component's data source dependency constraints.
4. Optionally, add one or more firing conditions.
If one or more firing conditions exist, the Container Manipulator does not perform its operations or fire its event unless all the conditions are met.

Deleting data from a Container component with a Container Manipulator

Use this procedure use a Container Manipulator component to delete data from a Container component.

Before you begin

Your workspace should contain the Container component from which you want to delete data.

Procedure

1. Open a Container Manipulator editor, give your component a name, and select an incoming event.
2. In the Action panel, select **Delete** as the action, select the desired Container component, and click **Add**.
A row is automatically added to the action list. By default, this action deletes all data from the Container.
3. Optionally, add a WHERE clause to further specify what data is deleted.
 - a. Click the link in the Value field of the Action list to open the Where window.
 - b. In the Where window, select a field from the drop-down list.
The options include all of the fields in the selected Container.
 - c. Click **Add**.
The field is added to the list in the Where window.
 - d. Click the link next to the field in the list to define the data that the value in the field is compared to, and the operator used in the comparison.
You can choose data from any data source or component that is available to the Container Manipulator. Availability depends on the component's data source dependency constraints.
4. Optionally, add one or more firing conditions.
If one or more firing conditions exist, the Container Manipulator does not perform its operations or fire its event unless all the conditions are met.

Chapter 16. Pattern components

You can use Pattern components to test whether one or more events for a given customer occur over a period of time. A Pattern component collects events that match its criteria and stores these events in the customer's state history. When the specified pattern of events is met, the pattern drops those events from state history and fires a positive event.

Pattern components use a time span to set a boundary around the events it evaluates against its criteria. When you specify the time span, you can choose between Rolling and Calendar spans.

If events held in state history age out of this window, they are dropped and are no longer evaluated against the pattern criteria.

Pattern component types

There are three types of pattern components: Match All, Counter, and Weighted Counter.

The three pattern types work as follows.

Match All

If all of the specified incoming events occur, the pattern fires a positive event.

Counter

If the specified incoming events occur a specified number of times, the pattern fires a positive event.

For example, you might look for cases where a customer makes 3 deposits.

Weighted Counter

You assign a score to each incoming event that you specify, and the pattern fires if a specified total score is reached.

Unlike the Match All pattern, all of the specified incoming events do not have to occur; the total score is what determines whether the pattern criteria are met.

For example, suppose you configure the pattern as follows.

- You select incoming events and assign the following scores.
 - Incoming event 1 has a score of 1.
 - Incoming event 2 has a score of 2.
- You specify a Weighted Counter value of 10.

The event pattern would be true in any of the following cases.

- Incoming event 1 occurs 10 times.
- Incoming event 1 occurs 2 times and incoming event 2 occurs 4 times.
- Incoming event 2 occurs 5 times.

Sequence

You can specify a series of incoming events, and set the number of times the complete sequence of events must occur before the Sequence Pattern component fires.

Negative event modes in Pattern components

By default, a Pattern fires a positive event when its criteria are met, but it does not fire a negative event when the aging out of an event creates a negative condition. However, you can configure the behavior to include negative events.

When you enable negative events, the component alternates between firing positive and negative events, starting with a positive event.

You can select the mode for negative events: Immediate or Delay.

- In Immediate mode, the actions are as follows.
 - When the Pattern component fires a positive event, it retains the events that fulfilled its criteria in the customer's state history. They continue to be used in evaluating the collection of events against the pattern criteria.
 - The component fires a negative event as soon as the pattern is no longer fulfilled by collection of events being held for the customer in state history.
 - The component continues to listen for incoming events and fires a positive event when the pattern is again fulfilled.
- In Delay mode, the actions are as follows.
 - When the Pattern component fires a positive event, it drops the events that fulfilled its criteria from the customer's state history.
 - The delay period, which is specified in the duration field, begins immediately. During the delay period, the component ignores incoming events.
 - At the end of the delay period, the component fires a negative event.

When you run a workspace that includes a Pattern component configured to use the negative event mode, on the Batch Run tab of the workspace, you must set the **Inactivity Mode** and **Inactivity Date** fields.

Inactivity Mode

Set this value to On for workspaces with Forward Inactivity components.

Inactivity Date

Set this to a value greater than the latest transaction date in the last transaction file being processed.

Related reference:

"Pattern Behavior fields" on page 94

Calendar time span in Pattern components

You specify the calendar time span for the pattern using the **Time Span** section of the component editor.

The timestamp on the event, not the time when the event was processed, determines whether the event falls within the time span.

The start and end points of the calendar time span are fixed. When the calendar time span is over, the span resets.

The calendar time span works in conjunction with calendar time units, which have defined start and end points. These start and end points affect the behavior of

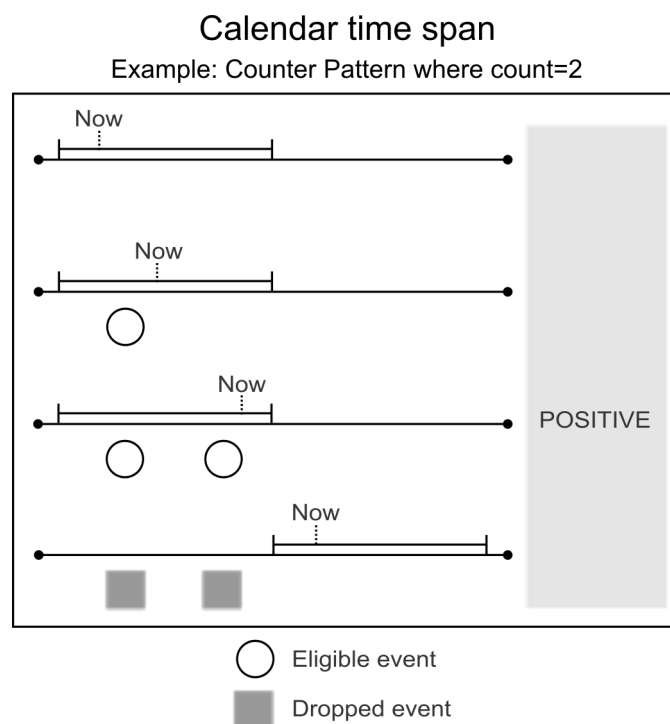
calendar time spans. For example, if the time unit is weeks, the calendar time span ends on Saturday at one second before midnight, and the next time span begins at midnight on Sunday.

The following examples illustrate how the calendar time span works with positive and negative events in Pattern components.

Example: Calendar time span, negative events not enabled

The following diagram illustrates how a calendar time span works when used in a Counter Pattern component where negative events are not enabled.

In the example, the component fires a positive event as soon as its criteria are met within the specified time span. Then it drops the events that caused the component to fire. When the time span expires, it resets.



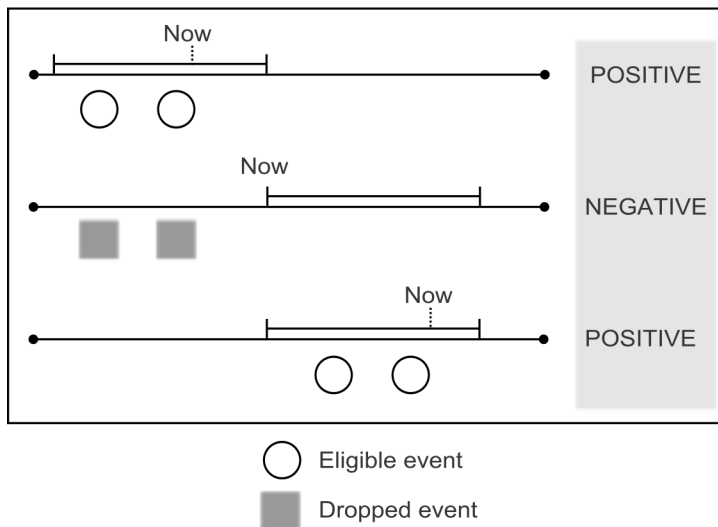
Example: Calendar time span, negative events enabled in Immediate mode

The following diagram illustrates how a calendar time span works when used in a Counter Pattern component where negative events are enabled, and Immediate mode is selected.

In the example, as soon as the component's criteria are met, the component fires a positive event and drops the events that caused the component to fire. When the time span resets to the next calendar period, the component fires a negative event immediately.

Calendar time span with negative events in Immediate mode

Example: Counter Pattern where count=2



Example: Calendar time span, negative events enabled in Delay mode

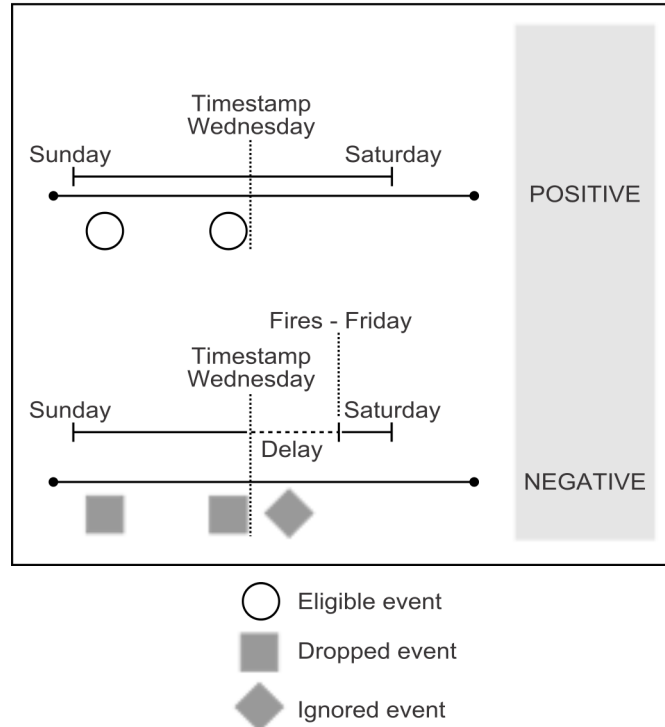
The following diagram illustrates how a calendar time span works when used in a Counter Pattern component where negative events are enabled in Delay mode, and a time span of 1 calendar week and a Delay mode of 2 calendar days are selected.

In the example, the actions are as follows.

- The event that completes the component's pattern has a timestamp of Wednesday. The component fires a positive event immediately and drops all events from the customer's state history.
- The 2 day delay period begins. During the delay period, eligible incoming events are ignored.
- At the end of the delay period, on Friday, the component fires a negative event and begins to collect events again.
- The time span resets at midnight on Sunday (not shown in the diagram).

Calendar time span with negative events in Delay mode

Example: Counter Pattern where count=2
Time span=1 calendar week
Delay=2 calendar days



Related reference:

“Time Spans” on page 56

“Time constants and time units” on page 58

Rolling time span in Pattern components

You specify the rolling time span for the pattern using the **Time Span** section of the component editor.

Events must occur within the specified time span to be eligible for evaluation against the pattern criteria. The start of the rolling time span moves forward in time, but the span you specify always determines the length of the window.

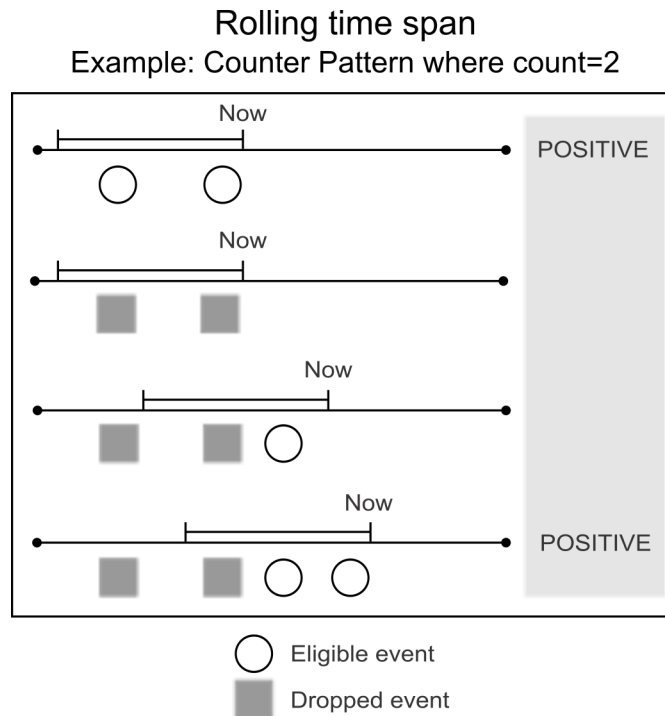
In contrast to the calendar time span, with the rolling time span, events are retained after a positive event is fired. They are dropped only when they age out of the time span.

The timestamp on the event, not the time when the event was processed, determines whether the event falls within the time span.

Example: Rolling time span, negative events not enabled

The following diagram illustrates how a rolling time span works when used in a Counter Pattern component where negative events are not enabled.

In the example, as soon as the component's criteria are met, the component fires a positive event. It drops the events that caused the component to fire. As the time span rolls forward, the event that ages out is dropped, and the component continues to collect eligible events.



Example: Rolling time span, negative events enabled in Immediate mode

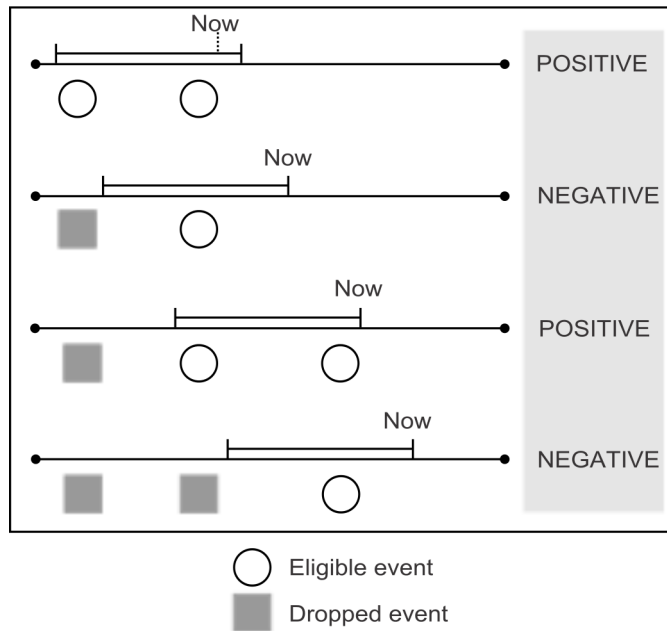
In Immediate mode with the rolling time span, the pattern must fire its first positive event before it ever fires a negative event.

The following diagram illustrates how a rolling time span works when used in a Counter Pattern component where negative events are enabled, and Immediate mode is selected.

In the example, after the first positive event is fired, the component retains the events until they age out. It fires a negative event as soon as an event ages out so that the pattern is no longer fulfilled. The pattern continues to listen for incoming events and fires a positive event when the pattern is again fulfilled.

Rolling time span with negative events in Immediate mode

Example: Counter Pattern where count=2



Example: Rolling time span, negative events enabled in Delay mode

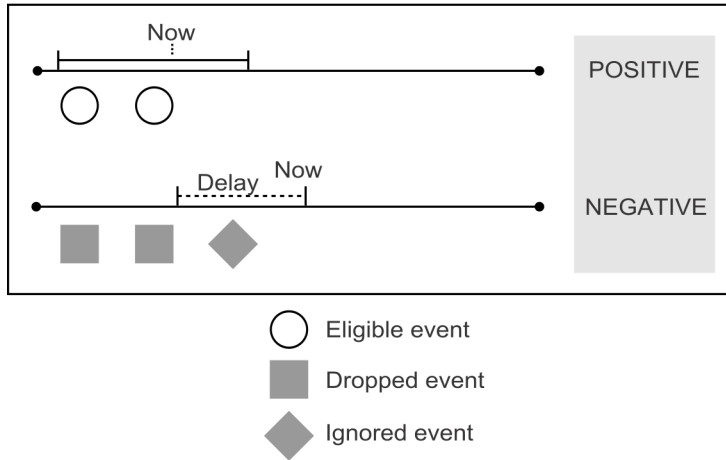
In Delay mode with the rolling time span, the pattern must fire its first positive event before it ever fires a negative event.

The following diagram illustrates how a rolling time span works when used in a Counter Pattern component where negative events are enabled, and Delay mode is selected.

In the example, after the first positive event is fired, the component drops its events, the delay period begins, and the component ignores incoming events. It does not fire the negative event until the delay period you specify is over. At that point the component begins to listen for incoming events again.

Rolling time span with negative events in Delay mode

Example: Counter Pattern where count=2



Related reference:

“Time Spans” on page 56

Pattern Behavior fields

Pattern behavior options allow you to enable the pattern to fire a negative event (by default patterns fire only positive events). They also allow you to delay firing the negative event.

Table 29. Pattern Behavior fields

Field	Description
Incoming events must occur this many times	This field is available only on the Counter Pattern and Sequence Pattern components. Enter an integer to specify how many times the incoming event must occur before the pattern fires a positive event.
Enable negative events	By default, pattern components send only positive events. This checkbox enables the pattern to send negative events.
Delay Mode and Immediate Mode	Delay mode allows you to delay the firing of the negative event for a specified period of time. For details, see “Negative event modes in Pattern components” on page 88.

Related concepts:

“Negative event modes in Pattern components” on page 88

Reset Event fields

You can define reset event when you configure all types of Pattern components. Optional.

Table 30. Reset Event fields

Field	Description
Event	<p>Select a positive or negative event, then select the component type and the component that supplies the event, and click Add.</p> <p>When the component receives a reset event, all incoming events that are being held for evaluation against the pattern criteria are dropped, and the component continues to listen for new incoming events.</p> <p>If negative events are enabled, and the last event that the component fired was positive, when the reset occurs the pattern fires a positive event.</p>

Chapter 17. Backward Inactivity and Forward Inactivity components

You use a Backward Inactivity component when you want to verify that some event did not occur within a specified period of time prior to an incoming event. You use a Forward Inactivity component to set a timer when it receives an incoming event, and optionally to specify another event that turns off the timer.

Backward Inactivity components

When a customer withdraws \$10,000 from her account, you might want to determine whether she has made any other withdrawals larger than \$5,000 within the past two months. For this, you would use a Backward Inactivity component.

The Backward Inactivity component uses the following primary settings.

Triggering event

This is the event that causes the component to execute its logic. The timestamp of this event is compared with the Blocking event's timestamp to determine whether the time span between the two events is greater than the Time span specified in the component.

Time span

The Time span is used in conjunction with the Blocking event. It sets the period of time prior to the Triggering event that is used as a criterion against which the system compares the timestamp of the Blocking event.

Blocking event

A Blocking event is an event prior to the Triggering event that the system evaluates to determine if the timestamp falls within the specified Time span.

The component functions as follows.

- The Backward Inactivity component stores the timestamp of the most recent Blocking event.
- When the Triggering event arrives, the component compares the Blocking event timestamp to the Triggering event timestamp. If the Blocking event timestamp falls before the backward-looking Time span, or if no Blocking event has been received yet, the component fires.
- If the Blocking event timestamp falls within the backward-looking Time span, the component does not fire.
- When no Blocking event is specified in the Blocking Event panel, you must choose the **Use the firing of this component as a blocking event** option described below. The component fires when it receives the Triggering event,

Additional options

In addition, you can use either or both of the following options.

Use the firing of this component as a blocking event

If you select **Use the firing of this component as a blocking event**, then the timestamp of the most recent firing of this component is saved as the most recent Blocking event timestamp.

For example, suppose you have a trigger system where a component named C_1 sends its event as the triggering event to BI_1, a Backward Inactivity component that uses this option.

When C_1 fires for the first time, BI_1 fires, because BI_1 has never fired before, and its blocking event timestamp has not yet been set. After BI_1 fires, the firing event timestamp of BI_1 is then used as the blocking event timestamp. The next time C_1 fires, BI_1 compares the blocking event timestamp to the timestamp of C_1, and BI_1 fires again if the difference is greater than the Time span set in BI_1.

Use creation date if no blocking event

By default, if the Triggering event occurs, and no Blocking event occurred before the triggering event, the component fires. If you check **Use creation date if no blocking event**, the component uses its creation date as the start of the time span if the Blocking event did not occur.

Related reference:

“Time constants and time units” on page 58

Examples of Backward Inactivity Components

This section provides examples of Backward Inactivity components.

Example 1: Store purchase after no online purchases

When a customer makes a store purchase, the Backward Inactivity component evaluates the data for that customer to see if he made an online purchase in the previous 6 months. If an online purchase has not occurred in the past six months, the component fires for that customer.

This component has the following parameters.

- Triggering event-customer makes a store purchase
- Blocking event-online purchase
- Time span-prior 6 months

Example 2: First major deposit after a long interval

When a customer makes a major deposit, the Backward Inactivity component evaluates the data for that customer to see if she made any other deposits or withdrawals in the past month. If these activities have not occurred in the past month, the component fires for that customer.

This component has the following parameters.

- Triggering event-make a major deposit
- Blocking event-major deposit
- Time span-1 month

Forward Inactivity components

In their simplest form, Forward Inactivity components set a timer. They can also look for the absence of a specified event within a specified time span following the event that activates the component. If the specified event does not occur, the component fires.

For example, when a customer makes a purchase, you might want to determine whether he returns within the next month. For this, you would use a Forward Inactivity component.

The Forward Inactivity component requires the following settings.

Incoming Event

The event that activates the component. The timestamp of the incoming event is used as the starting point of the time span.

Time Span

The period of time after the timestamp of the incoming event before the component fires its positive event.

In addition, on the Batch Run tab of the workspace, you must set the **Inactivity Mode** and **Inactivity Date** fields when you run the workspace.

Inactivity Mode

Set this value to **On** for workspaces with Forward Inactivity components.

Inactivity Date

Set this to a value greater than the latest transaction date in the last transaction file being processed.

In the component editor, you can set additional specifications to modify the way the component behaves after it has been activated by the incoming event.

The reset option

You can add an optional reset to the component by checking the **Reset the timer each time the incoming event occurs** box in the Incoming Event panel. It changes the behavior of the Forward Inactivity component as follows.

Reset option is not enabled

After the component is activated by an incoming event, the component ignores any additional instances of the incoming event that have a timestamp that falls within the time span, until the time span is over. When the time span is over, the component fires its event.

Reset option is enabled

After the component is activated by an incoming event, if the component receives an additional instance of the incoming event that has a timestamp that falls within the time span, it resets the time span.

Canceling event

You can set an optional canceling event. If a canceling event occurs after the component is activated, it stops the clock on the time span and the component can again respond to an incoming event. The timestamp of the canceling event is used as the starting point for the reset time span.

Additional time settings

You can use the settings in the Effective Window and Firing Frequency panels to refine the component's behavior.

Rolling time span in Forward Inactivity components

You can set either a calendar or rolling time span in Forward Inactivity components.

For both calendar and rolling time spans, the incoming event timestamp is used as the starting point of the time span.

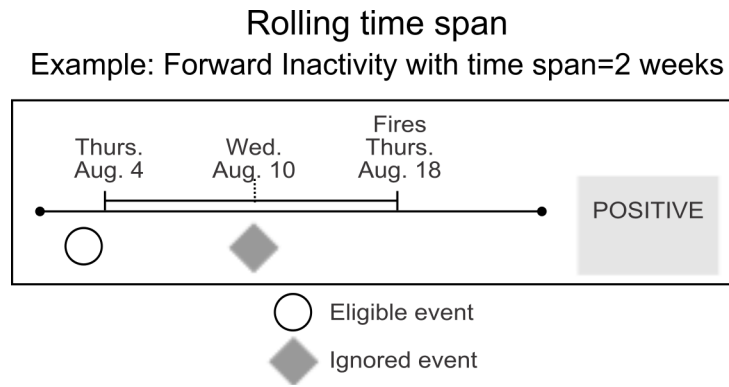
Rolling time span

The rolling time span works differently in Forward Inactivity components from the way it works in Pattern components. The length of the span remains constant, but the start date does not move forward in time as it does with a Pattern.

In the example diagram, a Forward Inactivity has a rolling time span of two weeks, and the component is activated by a transaction with a timestamp of Thursday, August 4. No canceling event is specified.

The time span expires and the component fires 14 days later, on Thursday, August 18.

The component in the following diagram ignores any additional incoming events because reset is not enabled.



Rolling time span with reset

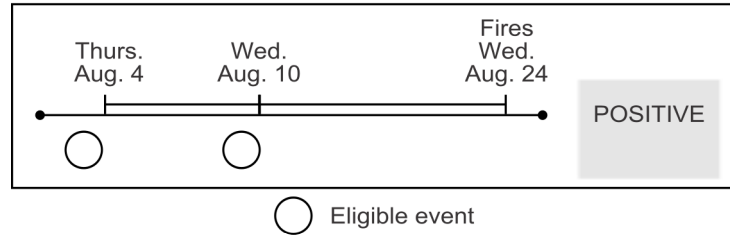
If reset is enabled, the component continues to listen for incoming events even after it is activated.

In the example diagram, a Forward Inactivity has a rolling time span of two weeks, and the component is activated by an event with a timestamp of Thursday, August 4. No canceling event is specified.

With reset enabled, the component would reset its time span if a second incoming event arrives with a timestamp that falls within the time span. Suppose this timestamp is Wednesday, August 10 at 9:47AM. The time span resets, and the first week of this reset time span ends at 9:47AM on Wednesday, August 17. The time span expires and the component fires on at 9:47AM on August 24.

Rolling time span with reset

Example: Forward Inactivity with time span=2 weeks



Related reference:

“Time Spans” on page 56

“Effective Window” on page 56

“Firing Frequency” on page 56

“Time constants and time units” on page 58

Calendar time span in Forward Inactivity components

You can set either a calendar or rolling time span in Forward Inactivity components.

For both calendar and rolling time spans, the incoming event timestamp is used as the starting point of the time span.

Calendar time span

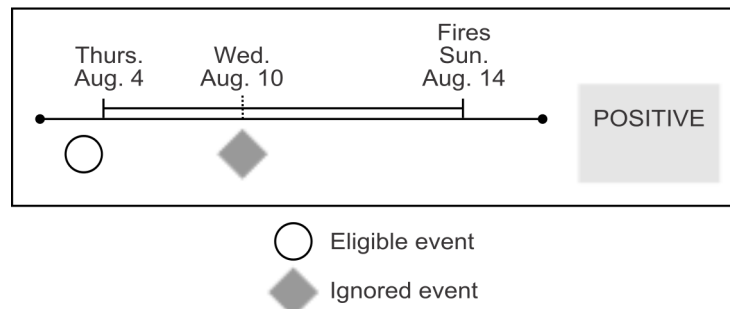
In the example diagram, a Forward Inactivity component has a time span of two calendar weeks, and the component is activated by an event with a timestamp of Thursday, August 4. No canceling event is specified.

The first week of this calendar time span ends at midnight on Sunday, August 7, only three days later. When the second week ends at midnight on Sunday, August 14, the time span expires and the component fires. This is because a week as defined by the time constant always begins at midnight on Sunday.

The component in the example ignores any additional incoming events because reset is not enabled.

Calendar time span

Example: Forward Inactivity with time span=2 weeks

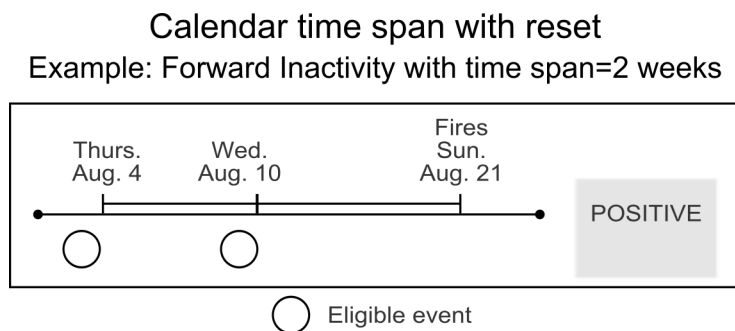


Calendar time span with reset

If reset is enabled, the component continues to listen for incoming events even after it is activated.

In the example diagram, a Forward Inactivity has a calendar time span of two weeks, and the component is activated by an event with a timestamp of Thursday, August 4. No canceling event is specified.

With reset enabled, the component would reset its time span if a second incoming event arrives with a timestamp that falls within the time span. Suppose this timestamp is Wednesday, August 10. The time span resets, and the first week ends at midnight on Sunday, August 14, only four days later. When the second week ends at midnight on Sunday, August 21, the time span expires and the component fires.



Related reference:

“Time Spans” on page 56

“Effective Window” on page 56

“Firing Frequency” on page 56

“Time constants and time units” on page 58

Examples of Forward Inactivity components

This section provides examples of Forward Inactivity components.

Example 1: No additional deposits in one month

On June 10, a customer makes a complaint through a bank's call center. The Forward Inactivity component is activated and starts monitoring to determine whether that customer makes any deposits. If the customer makes a deposit, the Forward Inactivity does not fire. If the customer does not make any deposits, the component fires for that customer so that a follow-up can be initiated.

This component is configured as follows.

- Incoming event-customer makes a complaint
- Canceling Event-customer makes a deposit
- Time Span-one rolling month

Example 2: No additional deposits this month

On June 10, a customer makes a deposit. The Forward Inactivity component starts monitoring to determine whether that customer makes another deposit. If another deposit does not occur by July 1, the component fires for that customer.

This component is configured as follows.

- Incoming Event-customer makes a deposit
- Canceling Event-customer makes a deposit
- Time Span-one calendar month

Chapter 18. Trend, Spike, and Exceeded Standard Deviation components

Trend components detect changes in activity measured over defined periods of time. Before you can create a trend component, your trigger system must include a Container or Select component that the trend can use as a data source.

To be available for use in a trend component, a Select component must have a primary timestamp field in the output.

Three components track trends.

Trend component

Detects simple trends. For example, the average rolling monthly balance has been increasing or decreasing by 10% over the last three months.

Spike component

Detects an unusually large change in activity. For example, a deposit is 50% larger or smaller than any other deposit made in the last 3 months.

Exceeded Standard Deviation (ESD) component

Detects activity outside of the standard variance for a specified time period. For example, a monthly balance that is greater than or less than the average monthly balance for the past 12 months by two standard deviations.

For simplicity, when this document refers to all three types of trend component as a group, it uses the term "trend components."

Trend components never fire on their own. They are used internally by Action or Container Manipulator components. You can configure an Action or Container Manipulator component to fire when a Trend Component evaluates to true.

A trend component can reference a single numeric value from the following components.

- Container
- Select

About time boundaries for trends

Trend components can filter the data in a Container or a Select component based on time periods. Some of these time periods have a natural definition (such as calendar day) and some do not (such as rolling month).

There are two types of time periods: calendar and rolling.

Rolling periods

Start at the time of day noted in the timestamp of the transaction, and go back to that same time of day on an earlier day. The interval (such as day, week, or month) can be specified.

Calendar periods

Calendar days, calendar weeks, and calendar months always begin and end at midnight.

When you define a trend component, you select calendar or rolling for only the last time period, and Opportunity Detect adjusts all previous time periods accordingly. For example, if a spike period is rolling, then the historical period is also rolling. If the spike period is calendar, then the historical period is also calendar.

Rolling bounded periods

Under certain conditions, the system imposes special rules to define boundaries for rolling time periods.

Rolling bounded periods are calendar weekly or monthly periods that do not necessarily start on the first day of the week or the first day of the month.

Rolling bounded periods are required to tight-fit historical periods to a spike period of a calendar day or in some cases a calendar week. The spike period is used in Spike and Exceeded Standard Deviation components.

Time boundaries for trend components

The following table describes the time boundaries that can be used in trends, and explains the rules that define the boundaries of the trend time periods. All examples are shown with granularity of 1 second.

Note: The rolling bounded time boundaries do not appear as settings in the user interface. However, they are included here to explain how the time boundaries behave when the system requires them to tight-fit historical periods to a spike period of a calendar day or in some cases a calendar week.

Time Boundary	Description
Calendar Day	Includes midnight of the transaction day up to but not including midnight of the next day. For example: If the current transaction is dated 2009-03-17 09:22:00, then the boundaries of its calendar day are 2009-03-17 00:00:00 - 2009-03-17 23:59:59 inclusive
Rolling Day	Includes everything greater than this second of the previous day up to and including the second of this current transaction. If the current transaction is dated 2009-03-17 09:22:00, then the boundaries of its rolling day are 2009-03-16 09:22:01 - 2009-03-17 09:22:00 inclusive
Calendar Week	Includes Sunday midnight up to but not including midnight of the following Sunday. For example: If the current transaction is dated (Tuesday) 2009-03-17 09:22:00, then the boundaries of its calendar week are (Sunday) 2009-03-15 00:00:00 - (Saturday) 2009-03-21 23:59:59 inclusive
Rolling Week	Includes everything greater than this second of the previous week up to and including the second of this current transaction. If the current transaction is dated (Tuesday) 2009-03-17 09:22:00, then the boundaries of its rolling week are (Tuesday) 2009-03-10 09:22:01 - (Tuesday) 2009-03-17 09:22:00 inclusive

Time Boundary	Description
Calendar Month	Includes midnight of the first of the month up to but not including midnight of the first of the next month. Note that calendar months vary in length. For example: If the current transaction is dated 2009-03-17 09:22:00, then the boundaries of its calendar month are 2009-03-01 00:00:00 - 2009-03-31 23:59:59 inclusive
Rolling Month	Includes everything greater than this second of this day of the month of the previous month up to and including the exact second of this current transaction. For example: If the current transaction is dated 2009-03-17 09:22:00, then the boundaries of its rolling month are 2009-02-17 09:22:01 - 2009-03-17 09:22:00
Rolling Bounded Week	Used for historical periods when the spike period is of type calendar. For a particular day of the week, includes midnight of that day of the week up to but not including midnight of that same day of the next week. For example: If the current transaction is dated 2009-03-17 09:22:00, then the boundaries of its calendar day are 2009-03-17 00:00:00 - 2009-03-17 23:59:59. Also, the boundaries of the rolling bounded week that precedes the calendar day are: 2009-03-10 00:00:00 - 2009-03-16 23:59:59
Rolling Bounded Month	Used for historical periods when the spike period is of type calendar. For a particular day of the month, includes midnight of that day of the month up to but not including midnight of that same day of the next month For example: If the current transaction is dated 2009-03-17 09:22:00, then the boundaries of its calendar day are 2009-03-17 00:00:00 - 2009-03-17 23:59:59. Also, the boundaries of the rolling bounded month that precedes the calendar day are: 2009-02-17 00:00:00 - 2009-03-16 23:59:59

End of month boundaries

The system calculates end of month boundaries as described in this section.

End date 29 through 31

If the end date is 29 through 31, then the end dates of the each previous month are the same corresponding days of the month unless a month is short and lacks that day of the month. In that case the end date for the short month is the last calendar day of the month in that month.

If a month end date reverts to the last day because the month is short of days, then the month end of the month previous to the shortened month recovers the lost end days.

For example, if the end date is 3/31, then the end dates of previous months are 2/28, 1/31, 12/31, 11/30, 10/31, etc.

Note: These monthly end dates are preserved even if they are embedded in a trend period that ends on 30. For example, if the trend has four three-month time

periods in it, and the end date of the last period is 3/31, then going backward the end dates of the four trend periods are 3/31, 12/31, 9/30, and 6/30.

End date 1 through 28

If the end date is 1 through 28, then the end dates of each previous month are the same corresponding days of the month.

For example if the end date is 3/28, then the end dates of previous months are 2/28, 1/28, 12/28, 11/28, 10/28, etc.

Beginning points, ending points, and end-of-month arithmetic

The way time periods are calculated is defined within the system as described in this section.

Every period has a beginning and end point.

For monthly rolling periods, the beginning and end points are calculated as follows.

- The end point is the date and timestamp of the transaction.
- The beginning point is the previous month with the same day of the month and timestamp as the current transaction.

In contrast, for monthly calendar periods, the beginning and end points are calculated as follows.

- The beginning point is midnight of day x to midnight of day x of the next month.
- To ensure that contiguous time periods are continuous and do not overlap, one of the end points is included in the calculation of the period and the other is not.

With calendar or rolling bounded periods, the midnight of the beginning point is included but the midnight of the ending point is not.

For example: For June, the calendar month includes midnight of June 1 to midnight of July 1. The midnight of June 1 is included in the time period but the midnight of July 1 is not.

With rolling periods, the end point of the transaction date is included, but the beginning point of the same day of the month and the same time of the previous month is not.

For example: For a transaction occurring on June 29 at 23:59:59, the end point of June 29 23:59:59 is included, but the beginning point of May 29 at 23:59:59 is not.

Note: End of month boundaries are calculated moving backward using the end point of the last period as the reference point. So if the end point of a period is 6/30, then the end point of the previous month is 5/30. And if the end point of a period is 7/1, then the end point of the previous month is 6/1.

Around the end of a month, a 1 second difference in two transactions could mean a difference of a day or more in the monthly duration.

For example: A rolling month period that ends on June 29 at 23:59:59 spans from May 29 at 23:59:59 (point not included) to June 29 at 23:59:59 (point included). And a rolling month period that ends on July 1 at 00:00:00 spans from June 1 at 00:00:00 (point not included) to July 1 at 00:00:00 (point included). Here a difference of one second (end date of June 29 at 23:59:59 versus an end date of July 1 at 00:00:00) results in two monthly durations that differ by 1 day.

The Trend component

The Trend component detects either upward or downward changes in data values over a length of time.

The length of time is divided into equal periods. A mathematical function (such as sum or average) is applied to the data collected for each period. The calculated result is evaluated against a specified value to determine whether the values of the period are either consistently increasing or consistently decreasing by a certain percentage. If the percentage change of the values is consistently above the specified value for all of the periods, the component evaluates to true.

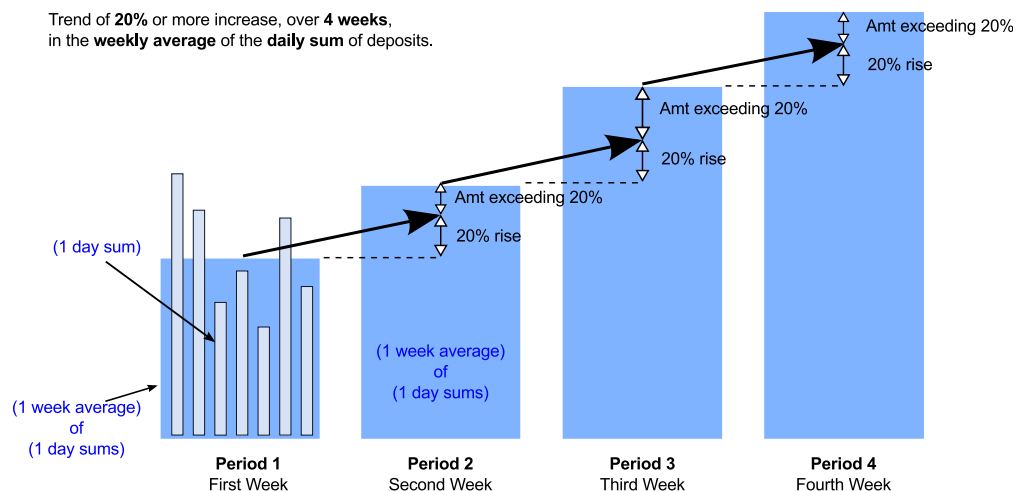
To detect a trend, you specify a field in the data source on which to do the trending. You also specify the time period over which the evaluation is done and the percentage increase or decrease you are looking for. The data values all come from a designated field in a pre-populated Container or Select component.

The trend groups data values by fixed time intervals within the specified period. For example, if a period spans three calendar months beginning January 1 to the end of March, the data values are grouped into intervals for the months of January, February, and March.

Example of a Trend component

In this example, the Trend component is looking for a 20% increase, each week for four weeks, in the weekly average of a daily sum of deposits.

The following figure illustrates data that would cause a Trend component to evaluate to true.



For this trend you would need a Container component with bank transactions (deposits, with date and amount). The Container is configured to sum the transactions for each day. Then to build the Trend component, you would do the following.

- In the **Source** panel, do the following.
 - Use the Value Selector to configure the standard deviation.

You can enter a constant or choose a value that is used to calculate the number of standard deviations. Only numeric data types are allowed. In this example, enter a constant of 2.
 - Select the Container component that contains the data used for comparison.
 - Select the field that sums the transactions for each week.
- In the Trend Period Value panel, set four trend periods of 1 week, and select the **Average** function.
- Optionally, set Advanced Configuration options.

Specifying a Trend component

These steps describe how to build a Trend component.

Procedure

1. Name the component.
2. In the **Source** panel, do the following.
 - a. Define the **Percentage** value that represents the nature of the trend (percentage change) you are looking for.

The value can be any of the following.

 - Positive, to track an increase
 - Negative, to track a decrease
 - A constant value
 - A lookup value from a data source, a Select component, or a Container component
 - If the data dependency allows it, a value from a transaction data source.
 - A calculated math value
 - b. Select the data source by type and name, and identify the field on which to do the trending.
3. In the **Trend Period Value** area, do the following.
 - a. Enter the **Number of trend periods** you want to track. A trend must have at least two periods. (In the next step, you define each time period.)

The trend evaluates to true only if each successive period increases (or decreases) over the previous period by the specified amount. For a trend with four time periods, the percentage of change must hold true for each of the three time period transitions: from period 1 to 2, 2 to 3, and 3 to 4.
 - b. Define the **Period length** for the trend.

For example, you might set the Time Period to be 3 calendar months long, and the Function to be maximum. In that case, the trend period value would be the maximum value in each three calendar month time period within the trend.
4. Optionally, set values in the **Advanced Configuration** panel.

Advanced configuration settings can further refine the criteria used to evaluate the trend, and can help to ensure that data has been tracked long enough, or that there are enough data points, to be statistically significant.
5. Click **Save and Close** to save the component.

Trend component fields

Fields used in defining a Trend component are described in this section.

Table 31. Trend component field

Field	Description
Source	
Percentage	The percent by which the period values must change to cause the component to evaluate as true. To detect an upward trend, set the trend percentage value to a positive value. To detect a downward trend, set it to a negative value.
Type	The component type that supplies the data to be evaluated (Container or Select).
Name	The name of the component that supplies the data to be evaluated.
Field	The field in the selected component that supplies the data to be evaluated.
Trend Period	
Number of trend periods	The number of time periods for which data is evaluated.
Period length	The length of each time period for which data is evaluated.
Advanced Configurations	
Value of last up trend period or first down trend period	Sets a minimum value for the last upward trend period or the first downward trend period.
Number of container data points in each trend period	Sets a minimum number of data points required within each trend period for the trigger to fire. This setting ensures that there are enough tracked activities (such as deposits or withdrawals) to be statistically significant.
Audience ID start date must be before trend period begins	Sets the earliest date on which the component can evaluate to true. For each audience ID, the start date is the date of the first transaction captured by Opportunity Detect for that audience ID. This optional setting allows you to ensure that the trend does not evaluate or fire until the data source that the trend is using (Container or Select component) has existed for a sufficiently long time and has sufficient data. Set this value based upon your data. For example, if you create a new trend that bases a calculation on 3 months of data then: <ul style="list-style-type: none"> • If your Container component contains 3 months of data, you should set this date to the current date. That would allow the trend to start to evaluate to true immediately. • If you have just created the Container today, and have not loaded it with historical data, then set this date to the current date + 3 months. That setting forces the trend to wait for 3 months while the container builds up enough data.

Table 31. Trend component field (continued)

Field	Description
Trend will not fire until this Date	<p>This option sets the earliest date on which the component can evaluate to true.</p> <p>This setting allows you to ensure that the trend does not evaluate or fire until the data source that the trend is using (Container or Select component) has existed for a sufficiently long time and has sufficient data.</p> <p>Set this value based upon your data. For example, if you create a new trend that bases a calculation on 3 months of data then:</p> <ul style="list-style-type: none"> • If your Container component contains 3 months of data, you should set this date to the current date. That would allow the trend to start to evaluate to true immediately. • If you have just created the Container today, and have not loaded it with historical data, then set this date to the current date + 3 months. That setting forces the trend to wait for 3 months while the container builds up enough data.

The Spike component

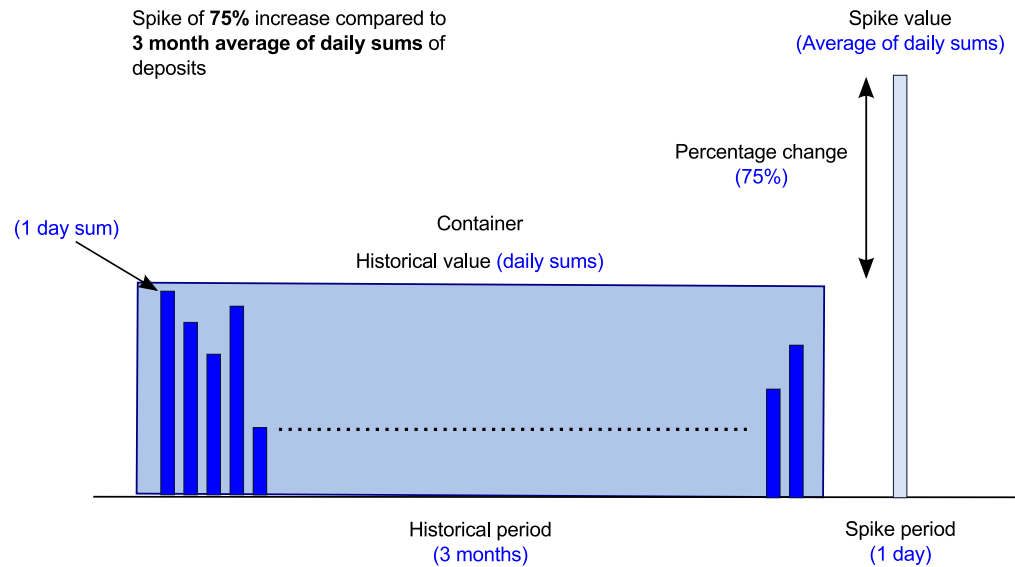
The Spike component looks for a significant jump or a significant drop in comparison to a historical value. Spike components are used as filters for Action components, Container Manipulators, and Boolean expressions.

Based on the parameters you set, the system formulates two values then compares the two values. The Spike component returns true if the **Spike Period Value** is greater than or less than the **Historic Value For Period** by the specified percentage.

Example of a Spike component

In this example, the Spike component is looking for a 75% increase (spike) in the current daily deposit sum above the three month average of daily sums for deposits.

The following figure illustrates data that would cause a Spike component to evaluate to true.



To build a Spike component like this you would need a Container with bank transactions (deposits, with date and amount). The Container is configured to sum the deposits for each day. Then you would do the following.

- In the **Source** panel, do the following.
 - Select the Container component that contains the data used for comparison.
 - Set a percentage value of 75%
 - Select the field that sums the deposits for each day.
-
- In the **Spike Period Value** panel, set the period to 1 day and select the Sum function.
- In the **Historical Value for Period** panel, select a time span of 3 months and use the **Average** function.
- Optionally, set Advanced Configuration options.

Specifying a spike component

These steps describe how to build a Spike component.

Procedure

1. Name the component.
2. In the **Source** panel, do the following.
 - a. Define the **Percentage** value that represents the nature of the trend (percentage change) you are looking for.

The value can be any of the following.

 - Positive, to track an increase
 - Negative, to track a decrease
 - A constant value
 - A lookup value from a profile data source
 - A value from a Container or a Select of a transaction or profile data source
 - If the data dependency allows it, a value from a transaction data source.
 - A calculated math value

- b. Select the data source by type and name, and identify the field on which to do the trending.
3. In the **Spike Period Value** panel, do the following.
 - a. Enter the **Spike period** you want to track and the function to apply to the data within that period.
 For example, you might define the spike value to be the average for two rolling days.
4. In the **Historical Value for Period** panel, define the value to which the spike value is compared.
 For example, you might define it to be the average over three weeks.
5. Optionally, set values in the **Advanced Configuration** panel.
 Advanced configuration settings can further refine the criteria used to evaluate the trend, and can help to ensure that data has been tracked long enough, or that there are enough data points, to be statistically significant.
6. Click **Save and Close** to save the component.

Spike component fields

Fields used in defining a Spike component are described in this section.

Table 32. Spike component fields

Field	Description
Source	
Percentage	The percent by which the period values must change to cause the component to evaluate as true. To detect an upward spike, set the spike percentage value to a positive value. To detect a downward spike, set it to a negative value.
Type	The component type that supplies the data to be evaluated (Container or Select).
Name	The name of the component that supplies the data to be evaluated.
Field	The field in the selected component that supplies the data to be evaluated.
Spike Period Value	
Spike period	The time period that is compared to the historical period.
Function	The mathematical function applied to the data within the period.
Historical Value for Period	
Historical period	The past time period that is used for comparison.
Function	The mathematical function applied to the data within the period.
Advanced Configuration	
Historical value	Sets a minimum for the historical value against which the spike is compared, Enter a value for the minimum historic average. The component does not fire for a customer whose historic base does not meet this value.
Rise or fall from historical value	Sets a fixed rise or fall compared to the historical value.

Table 32. Spike component fields (continued)

Field	Description
Number of container data points in historical period	Sets a minimum number of data points required within each trend period for the trigger to fire. This setting ensures that there are enough tracked activities (such as deposits or withdrawals) to be statistically significant.
Audience ID start date must be before trend period begins	<p>Sets the earliest date on which the component can evaluate to true. For each audience ID, the start date is the date of the first transaction captured by Opportunity Detect for that audience ID.</p> <p>This setting allows you to ensure that the trend does not evaluate or fire until the data source that the trend is using (Container or Select component) has existed for a sufficiently long time and has sufficient data.</p> <p>Set this value based upon your data. For example, if you create a new trend that bases a calculation on 3 months of data then:</p> <ul style="list-style-type: none"> • If your Container component contains 3 months of data, you should set this date to the current date. That would allow the trend to start to evaluate to true immediately. • If you have just created the Container today, and have not loaded it with historical data, then set this date to the current date + 3 months. That setting forces the trend to wait for 3 months while the container builds up enough data.
Spike will not fire until this Date	<p>This option sets the earliest date on which the component can evaluate to true.</p> <p>This setting allows you to ensure that the trend does not evaluate or fire until the data source (Container or Select component) that the trend is using has existed for a sufficiently long time and has sufficient data.</p> <p>Set this value based upon your data. For example, if you create a new trend that bases a calculation on 3 months of data then:</p> <ul style="list-style-type: none"> • If your Container component contains 3 months of data, you should set this date to the current date. That would allow the trend to start to evaluate to true immediately. • If you have just created the Container today, and have not loaded it with historical data, then set this date to the current date + 3 months. That setting forces the trend to wait for 3 months while the container builds up enough data.

The Exceeded Standard Deviations component

The Exceeded Standard Deviations (ESD) component compares the value in a current time period to the average of that value over a historical time period.

The comparison is made in terms of a specified number of standard deviations, with the standard deviation itself being based on the historical assemblage of values. An ESD component detects activity that exceeds the specified number of (positive or negative) standard deviations over specific time period.

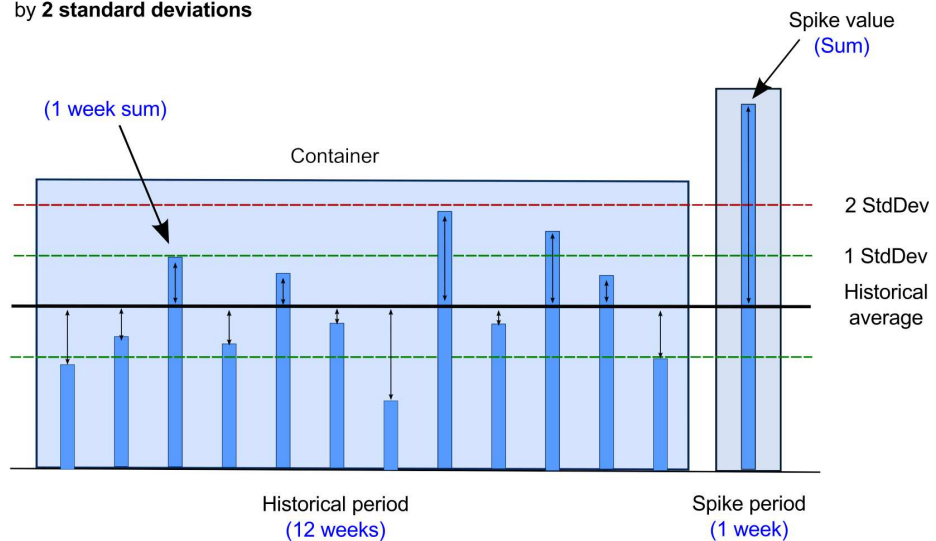
The ESD component allows the detection of a jump or drop that is significant not just in its size but also in comparison to how much the value has typically jumped or dropped for this customer in the past.

Example of an Exceeded Standard Deviations component

In this example, the ESD component is looking for the weekly sum of check card usage to exceed the average of weekly sums over the last twelve weeks by two standard deviations.

The following figure illustrates data that would cause an Exceeded Standard Deviations (ESD) trend to evaluate to true.

The current **weekly sum** of check card usage exceeds the average of **weekly sums** of transactions over the last **12 weeks** by **2 standard deviations**



For this ESD you would need a Container component with check card transactions. The Container is configured to sum the transactions for each week. Then to build the ESD you would do the following.

- In the **Source** panel, do the following.
 - Use the Value Selector to configure the standard deviation.

You can enter a constant or choose a value that is used to calculate the number of standard deviations. Only numeric data types are allowed. In this example, enter a constant of 2.
 - Select the Container component that contains the data used for comparison.
 - Select the field that sums the transactions for each week.
- In the **Spike Period Value** panel, set the period to 1 week and select the **Sum** function.
- In the **Historical Value for Period** panel, select a time span of 12 weeks and use the **Average** function.
- Optionally, set Advanced Configuration options.

Difference between setting a spike period to a rolling week versus a calendar week

There is a difference in behavior when a spike period is set to a rolling week versus when it is set to a calendar week. When using rolling time, the periods are completely full and span the entire week.

When deciding whether to use a calendar week or rolling week for the spike value, keep these guidelines in mind.

- Use a rolling time period when you want to compare a complete period ending at the point of the transaction.
- Use calendar periods when you want to compare a period that is in the process of completion, to a fixed historical period.

A calendar comparison is more appropriate for uncovering abrupt changes; whereas a rolling comparison is more appropriate for changes that are smoothed into even divisions.

In the example above, the spike period was set to one rolling week. In that case, the week of the spike period is a full seven days long, ending at the current time. The spike value returned from that seven day period is compared to the average of the twelve other rolling weeks which begin and end at the same time and day of the current transaction.

If a calendar week were used in the previous example, there would only be data in the current calendar week between the beginning of that week and the current transaction time (which would generally be a fraction of a week). The spike value returned from that calendar week period would be compared to the average of the twelve other calendar weeks which begin and end on strict calendar week boundaries.

Specifying an Exceeded Standard Deviations component

These steps describe how to build an Exceeded Standard Deviation (ESD) component.

Procedure

1. Name the component.
2. In the **Source** panel, do the following.
 - a. Define the **Standard Deviation** value that sets the number of standard deviations above (or below) the historical value for whatever time period you are looking at.

The value can be a constant value or it can be custom to the customer based on a lookup value from a datasource or on a calculated math value.

The value can be any of the following.

 - Positive, to track an increase
 - Negative, to track a decrease
 - A constant value
 - A lookup value from a profile data source
 - If the data dependency allows it, a value from a transaction data source.
 - A calculated math value
 - b. Select the data source by type and name, and identify the field on which to do the trending.
3. In the **Spike Period Value** panel, enter the **ESD period** you want to track and the function to apply to the data within that period.
4. In the **Historical Value for Period** panel, enter the **ESD period** you want to use for comparison.
5. Optionally, set values in the **Advanced Configuration** panel.

Advanced configuration settings can further refine the criteria used to evaluate the trend, and can help to ensure that data has been tracked long enough, or that there are enough data points, to be statistically significant.

- Click **Save and Close** to save the component.

ESD component fields

Fields used in defining an ESD component are described in this section.

Table 33. ESD component fields

Field	Description
Source	
Standard Deviations	The number of standard deviations that cause the component to evaluate as true. To detect an upward trend, set the number of standard deviations value to a positive value. To detect a downward trend, set it to a negative value.
Type	The component type that supplies the data to be evaluated (Container or Select).
Name	The name of the component that supplies the data to be evaluated.
Field	The field in the selected component that supplies the data to be evaluated.
ESD Period Value	
ESD period	The time period that is compared to the historical period.
Function	The mathematical function applied to the data within the period.
Historical Value for Period	
Historical period	The past time period that is used for comparison.
Advanced Configuration	
Historical value	Sets a minimum for the historical value against which the ESD is compared. Enter a value for the minimum historic average. The component does not fire for a customer whose historic base does not meet this value.
Rise or fall from historical value	Sets a fixed rise or fall compared to the historical base.
Percentage rise or fall from historical value	Sets a percentage rise or fall compared to the historical base.
Number of container data points in historical period	Sets a minimum number of data points required within each trend period for the trigger to fire. This setting ensures that there are enough tracked activities (such as deposits or withdrawals) to be statistically significant.

Table 33. ESD component fields (continued)

Field	Description
Audience ID start date must be before trend period begins	<p>Sets the earliest date on which the component can evaluate to true. For each audience ID, the start date is the date of the first transaction captured by Opportunity Detect for that audience ID.</p> <p>This setting allows you to ensure that the trend does not evaluate or fire until the data source that the trend is using (Container or Select component) has existed for a sufficiently long time and has sufficient data.</p> <p>Set this value based upon your data. For example, if you create a new trend that bases a calculation on 3 months of data then:</p> <ul style="list-style-type: none"> • If your Container component contains 3 months of data, you should set this date to the current date. That would allow the trend to start to evaluate to true immediately. • If you have just created the Container today, and have not loaded it with historical data, then set this date to the current date + 3 months. That setting forces the trend to wait for 3 months while the container builds up enough data.
ESD will not fire until this Date	<p>This option sets the earliest date on which the component can evaluate to true.</p> <p>This setting allows you to ensure that the trend does not evaluate or fire until the data source that the trend is using (Container or Select component) has existed for a sufficiently long time and has sufficient data.</p> <p>Set this value based upon your data. For example, if you create a new trend that bases a calculation on 3 months of data then:</p> <ul style="list-style-type: none"> • If your Container component contains 3 months of data, you should set this date to the current date. That would allow the trend to start to evaluate to true immediately. • If you have just created the Container today, and have not loaded it with historical data, then set this date to the current date + 3 months. That setting forces the trend to wait for 3 months while the container builds up enough data.

Chapter 19. Artificial transactions in Opportunity Detect

An artificial transaction (ATX) is a system-generated event that you can use as an incoming event in an event component.

The ATX is available in the Action, Container Manipulator and the Backward Inactivity components. In the Backward Inactivity component, it is available only as a triggering event.

If you use an ATX as an incoming event in a trigger system, then on the Batch Run tab of the workspace you must select an **Artificial Transaction** to have it go into effect.

Trigger systems that use the Web Service data source connector for input should not use an ATX, as it does not function with the Web Service connector.

There are two types of ATX.

- End of Run – An ATX fires at the end of each batch run for each audience ID being processed. If no transactions occur for an audience during a run, the ATX does not fire.
- End of Day – An ATX fires at the end of each day's set of transactions for each audience ID being processed. If no transactions occur for an audience during a day, the ATX does not fire.

When to use an ATX

Both types of ATX are useful when you want to detect a decline in customer activity. The End of Run ATX is most often used for production, and the End of Day ATX is most often used for testing.

Production

In most production use cases, batch files are processed every business day. Batch files for Tuesday, Wednesday, Thursday, and Friday contain transactions for the previous day. The batch file for Monday includes transactions from Friday, Saturday, and Sunday.

For the batch files that contain a transactions from a single day, The End of Run ATX produces the same results as the End of Day ATX. However, on Mondays, when transactions from three days are included, you want to process an account's transactions from all three days before firing a trigger indicating a decline in activity.

The trigger for a decline indicates that a customer did not do something, and it initiates a response urging the customer to resume the activity or asking why the decline occurred. If you respond in this way based on a Saturday transaction, and the customer contradicts that with a transaction on Sunday, you run the risk of sending an inappropriate response, because the customer's activity has not declined. This is when you would use an End of Run ATX to process all available transactions before responding to a decline in activity.

Testing

To simplify testing operations, you often put six months of transactions into a single file. You want to simulate the behavior of a production workspace running every business day with the ATX in End of Run mode. You can not use the End of Run mode, because the trigger indicating a decline would fire only once, after all six months of transactions have been processed.

Therefore, you would use the ATX in End of Day mode to approximate the behavior you would see in production. An End of Day ATX does not account for the Friday-Saturday-Sunday transactions all coming in on Monday, but with transactions from a long time period, it gives test results that are much closer to actual firing behaviors than the End of Run ATX.

Data dependency considerations for artificial transactions

When an event component uses an ATX as its incoming event, the descendent components that depend on it do not have access to any transaction data sources or to data components based on transaction data sources. The reason is that the system initiates the End of Day ATX after all the transactions have been processed.

The descendent components that depend on a component that uses an ATX do have access to a single profile data source. The first descendent component that uses a profile data source determines the profile data source available to all the other descendents of the component that uses the ATX. When you choose this profile data source in a descendent component, it should align with the profile data source used in the logic of the other transaction-based sections of your logic.

ATX in real time deployments

For real time, the ATX is always on and is set to End of Run.

In real time, a transaction set only has one transaction. The result is that the End of Run ATX is sent after every transaction even if there will be more transactions for that audience. When the ATX is sent, the components that are downstream from the component that uses the ATX perform their operations.

You can use an ATX with a real time deployment to activate consistent trigger logic after the transaction has been processed. Most often this is used to perform data calculations after the transaction has finished updating data. This can be useful in a production workspace, which typically contains many triggers. The ATX executes after all the triggers are processed. For example, Container components might be updated by the logic of several triggers before the ATX fires.

Artificial transaction example

The following ATX example illustrates a batch trigger system that a telephone company might use.

Suppose your goal is to take some action when the average of a customer's call minutes used during a week is less than 70 percent of the average minutes per week used in the four weeks prior to that week.

To make the calculation required in this example, you must collect all call minutes for a complete week and compare their average with the average call minutes for the previous four weeks. The End of Day ATX ensures that all transactions for the week are included in the evaluation.

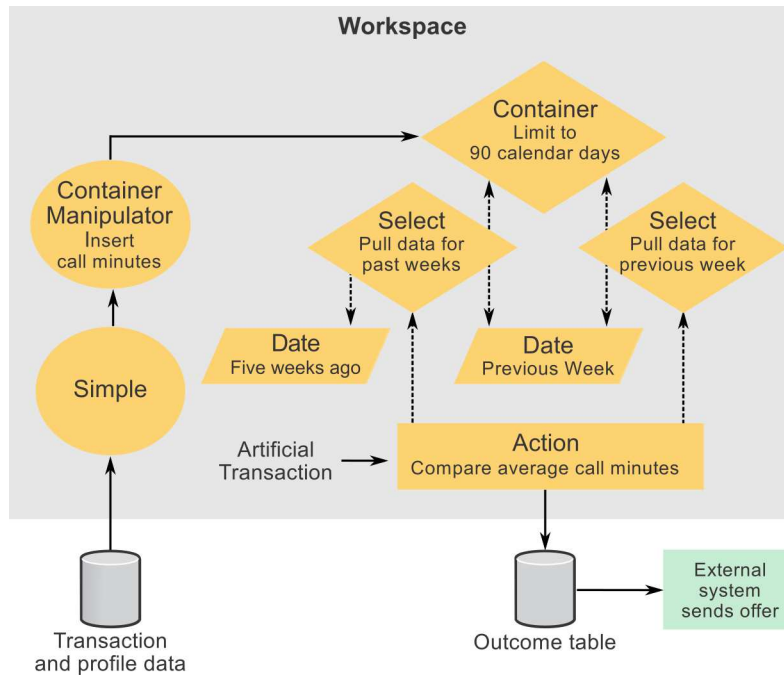
You can set up a trigger system as follows.

- Create a Simple component that fires an event when it receives customer call data.
- Create a Container component holds transactions that occur during a 90-day period.
 - In the Time Span panel, set a limit of 90 calendar days. Transactions with timestamps older than 90 days ago are cleared from the Container.
Typically, you create Containers that can be useful in multiple trigger systems. This is why you specify that this Container stores more transactions than you need for this example.
- Create a Container Manipulator that listens for the Simple component event.
 - In the Action panel, specify that the call data is inserted into the Container.
- Create a Date component named Previous Week.
 - Use the Expression Builder to specify a date seven days in the past. The expression looks like this when complete: `Now - Period(1,week)`.
- Create a Select component named Last Week to gather records from the past week.
 - In the Source panel, select the Container and select the timestamp and the field that holds the call minutes.
 - In the Where panel, create a WHERE clause that limits the data in the Select to records from the previous week.
Specify records with timestamps greater than the date returned by the Previous Week Date component.
- Create a Date component named Five Weeks Ago.
 - Use the Expression Builder to specify a date five weeks in the past. The expression looks like this when complete: `Now - Period(5,week)`.
- Create a Select component named Past Weeks to gather records from the time period between five weeks prior to the current week and the beginning of the current week.
 - In the Source panel, select the Container and select the timestamp and the field that holds the call minutes.
 - In the Where panel, specify the following two WHERE clauses to limit the data in the Select to records from the four weeks prior to the current week.
 - In one clause, specify records with timestamps less or equal to the date returned by the Previous Week Date component.
 - In another clause, specify records with timestamps greater than the date returned by the Five Weeks Ago Date component.
- Create an Action component that listens for an artificial transaction. In the Firing Condition panel, specify the following Boolean expression.
 - Compare the sum of the call minutes in the Current Week Select component to the sum of the call minutes in the Past Weeks Select component. The expression looks like this when complete, where the field that holds the call minutes is named `call_minutes`.
`[Last Week].[call_minutes].Sum() < [Past Weeks].[call_minutes].Sum()/4*0.7`

When you run the example workspace, select an End of Day **Artificial Transaction** on the Batch Run tab of the workspace. The Artificial Transaction fires after the transactions for each day have been processed by the components upstream from the Action component.

When the Action component receives the ATX event, it performs its comparison for each customer, but it fires only when its firing condition criteria are met.

The following diagram illustrates this example.



Related reference:

"Time constants and time units" on page 58

"Fields and buttons on the Batch Run tab" on page 34

"Fields and buttons on the Deployment tab" on page 26

Before you contact IBM technical support

If you encounter a problem that you cannot resolve by consulting the documentation, your company's designated support contact can log a call with IBM technical support. Use these guidelines to ensure that your problem is resolved efficiently and successfully.

If you are not a designated support contact at your company, contact your IBM administrator for information.

Note: Technical Support does not write or create API scripts. For assistance in implementing our API offerings, contact IBM Professional Services.

Information to gather

Before you contact IBM technical support, gather the following information:

- A brief description of the nature of your issue.
- Detailed error messages that you see when the issue occurs.
- Detailed steps to reproduce the issue.
- Related log files, session files, configuration files, and data files.
- Information about your product and system environment, which you can obtain as described in "System information."

System information

When you call IBM technical support, you might be asked to provide information about your environment.

If your problem does not prevent you from logging in, much of this information is available on the About page, which provides information about your installed IBM applications.

You can access the About page by selecting **Help > About**. If the About page is not accessible, check for a `version.txt` file that is located under the installation directory for your application.

Contact information for IBM technical support

For ways to contact IBM technical support, see the IBM Product Technical Support website: (http://www.ibm.com/support/entry/portal/open_service_request).

Note: To enter a support request, you must log in with an IBM account. This account must be linked to your IBM customer number. To learn more about associating your account with your IBM customer number, see **Support Resources > Entitled Software Support** on the Support Portal.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
B1WA LKG1
550 King Street
Littleton, MA 01460-1250
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Privacy Policy and Terms of Use Considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. A cookie is a piece of data that a web site can send to your browser, which may then be stored on your computer as a tag that identifies your computer. In many cases, no personal information is collected by these cookies. If a Software Offering you are using enables you to collect personal information through cookies and similar technologies, we inform you about the specifics below.

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name, and other personal information for purposes of session management, enhanced user usability, or other usage tracking or functional purposes. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

Various jurisdictions regulate the collection of personal information through cookies and similar technologies. If the configurations deployed for this Software Offering provide you as customer the ability to collect personal information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for providing notice and consent where appropriate.

IBM requires that Clients (1) provide a clear and conspicuous link to Customer's website terms of use (e.g. privacy policy) which includes a link to IBM's and Client's data collection and use practices, (2) notify that cookies and clear gifs/web beacons are being placed on the visitor's computer by IBM on the Client's behalf along with an explanation of the purpose of such technology, and (3) to the extent required by law, obtain consent from website visitors prior to the placement of cookies and clear gifs/web beacons placed by Client or IBM on Client's behalf on website visitor's devices

For more information about the use of various technologies, including cookies, for these purposes, See IBM's Online Privacy Statement at: <http://www.ibm.com/privacy/details/us/en> section entitled "Cookies, Web Beacons and Other Technologies."



Printed in USA