Google Analytics in JADE

Version 2018

Contents

Google Analytics in JADE

Introduction

Google Analytics is a useful tool for processing and analyzing usage data obtained from web sites and end-user applications.

JADE allows for the integration of JADE applications with Google Analytics through form event methods to log analytics data, and the **JadeHTTPConnection** class for sending the data the Google Analytics Application Programming Interface (API).

Setting Up Google Analytics

Before using JADE's Google Analytics integration, you first need a Google Analytics account.

Sign up for a free Google Analytics account at https://www.google.com/analytics/.

Note Google Analytics is a third-party service, neither owned nor controlled by JADE.

Once you have set up your Google Analytics account, add the following to your JADE initialization file (**jade.ini**) file.

```
[GoogleAnalytics]
GoogleID=UA-123456789-0
```

Use the Google ID provided to you during account creation instead of the **UA-123456789-0** shown in the above example.

Sending Analytics Data to Google

JADE provides the **JadeHTTPConnection** class for establishing HTTP connections and sending requests. To successfully send a message to Google Analytics, the following steps must be performed.

1. Create a transient **JadeHTTPConnection** object.
2. Set the **url** property of the **JadeHTTPConnection** object to:

   https://www.google-analytics.com/collect.
3. Open a connection using the **JadeHTTPConnection** class **open** method. This method has the **closeFirst** boolean parameter, which determines whether to close the connection if it is already open.
4. Call the **JadeHTTPConnection** class **sendRequest** method, which has the following parameters.
   - The **verb** parameter, of type **String**, is the request to send. For sending data to Google Analytics, this will be **"POST"**.
   - The **additionalHeaders** parameter, of type **String**, is not needed for this request and as such, a null value (**""**) can be passed for this parameter.

- The **optionalPostPutData** parameter, of type **String**, contains the data to send to Google Analytics.

Proper formatting of the data to pass in through the **optionalPostPutData** parameter is the most complex part of the operation. The following must be present in the **String** value.

Name	Parameter	Example	Description
Protocol Version	v	v=1	Should always be set to **1**.
Tracking ID	tid	tid=UA-123456789-0	Should be set to your Google ID.
Client ID	cid	cid=35009a79-1a05-49d7-b876-2b884d0f825b	Unique identifier for the client using the application. The **app.generateUuid** method can generate a value for this.
Hit Type	t	t=screenview	Sort of interaction tracked. Must be one of **pageview**, **screenview**, **event**, **transaction**, **item**, **social**, **exception**, or **timing**.
Application Name	an	an=DemoApp	Name of the application from which the data was collected.

In addition, the following must also be present for **screenview** Hit Types.

Name	Parameter	Example	Description
Screen Name	cd	cd=About Screen	Used to track the name of the screen that the user is on.

Exercise 1 – Setting Up a Google Analytics Account

In this exercise, you will set up a Google Analytics account to use in the rest of this module.

1. In your preferred browser, navigate to https://www.google.com/analytics/.
2. Follow the instructions on the page, to create a new Google Analytics account.
3. Locate your **jade.ini** file, located by default in the **system** directory within your JADE installation directory.
4. Open the **jade.ini** file and add the following section and parameter to the end of the file.

```
[GoogleAnalytics]
GoogleID=UA-123456789-0
```

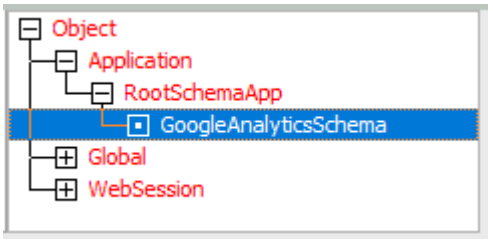
Note Although you can use Google Analytics in JADE without this setting, it is a convenient place to store your Google ID so that it can be accessed by any application within your database.

Exercise 2 – Sending Data to Google Analytics

In this exercise, you will send a hard-coded message to Google Analytics and see it appear in the real-time tracking.

1. Create a new schema called **GoogleAnalyticsSchema**.

2. Navigate to the **GoogleAnalyticsSchema** class, which is a subclass of **RootSchemaApp** (a subclass of **Application**).



3. Add a method called **getGoogleID**, coded as follows, to the **GoogleAnalyticsSchema** class.

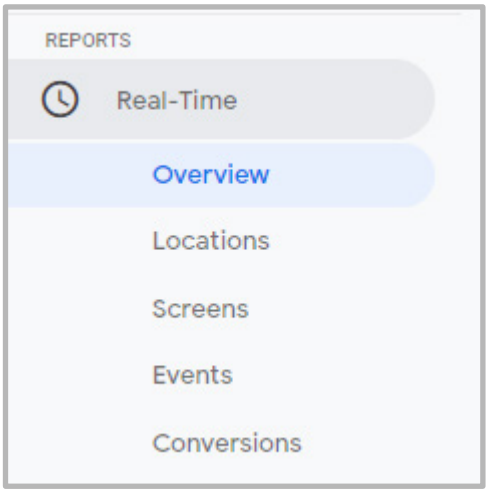
```
getGoogleID() : String;

begin
    return app.getProfileString(
                    app.getIniFileNameAppServer,
                    "GoogleAnalytics",
                    "GoogleID",
                    null);
end;
```

4. Add a JadeScript method called **sendFakeScreenView**, coded as follows.

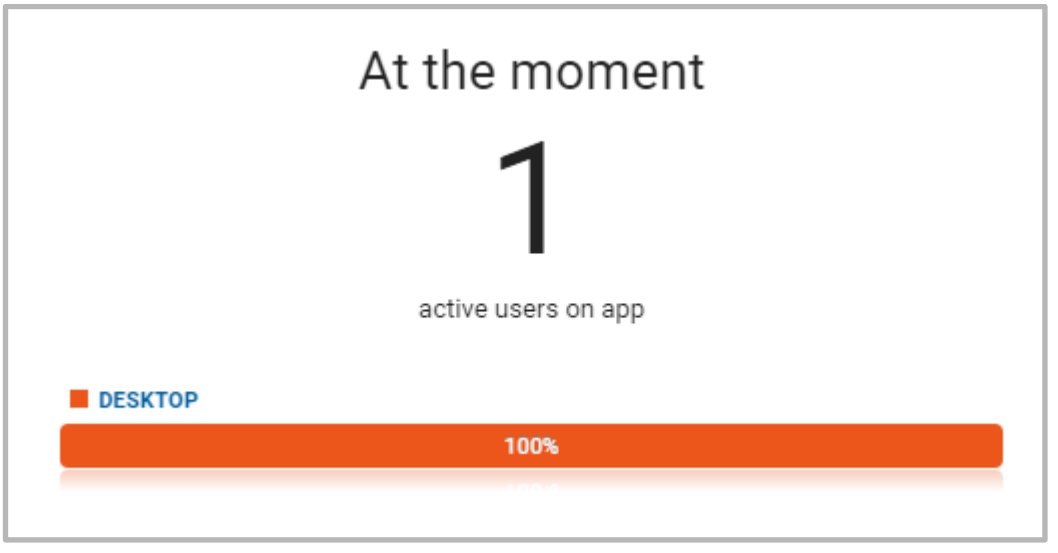
```
sendFakeScreenView();

constants
    MicrosoftUUID = 3;
vars
    msg             : String;
    httpConnection  : JadeHTTPConnection;
begin
    // Create message:
    msg :=    "v=" & "1" & "&"
          & "tid=" & app.getGoogleID & "&"
          & "cid=" & app.generateUuid(MicrosoftUUID).uuidAsString & "&"
          &   "t=" & "screenview" & "&"
          &  "an=" & app.name & "&"
          &  "cd=" & "Not actually a screen!";

    // Set up httpConnection:
    create httpConnection transient;
    httpConnection.url := "https://www.google-analytics.com/collect";
    httpConnection.open(true);

    // Send message
    write "Sending...";
    httpConnection.sendRequest("POST", null, msg);
    write "Sent!";
epilog
    delete httpConnection;
end;
```

5. Using your preferred browser, navigate to https://analytics.google.com.

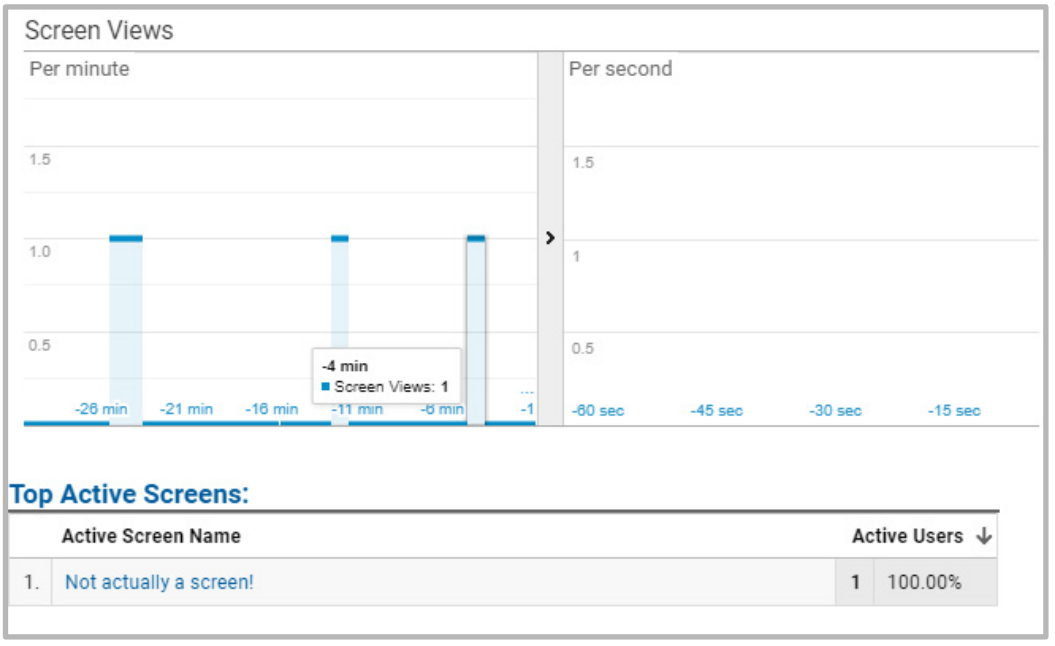
6. Under **REPORTS** in the navigation bar on the left, select **Real-Time** and then **Overview**.



7. Run the **sendFakeScreenView** method. The first thing you see is **Sending…**, possibly a short delay, and then **Sent!** is written to the Jade Interpreter Output Viewer and the dashboard of the Google Analytics real-time view is updated.



The following is an example of the Google Analytics real-time view.



Collecting Analytics Data from Applications

Applications created in the JADE Painter provide for many possible events as users interact with controls.

Each JADE form has a **load** method, which is automatically called when the first form is loaded. This can be useful for collecting screen view data. Much more data can also be collected by using control events; for example, extending the **click** events of buttons or adding data collection to **mouseEnter** and **mouseLeave** events for elements of importance.

When collecting data from user applications, it is useful to create a dedicated collector class for taking the raw user data and processing it into the required form. This decouples the logic required to process data into a form appropriate for Google Analytics away from the form logic (which should be responsible only for the presentation of the GUI).

Once processed, the data is ready to be sent to Google via a POST request over an HTTP connection.

Exercise 3 – Creating a Collector Class

In this exercise, you will create a **Collector** class responsible for consuming application usage data and forming Google Analytics POST messages with them.

1. Add a new class called **Collector**.

2. Add a public method called **create**, coded as follows, to the **Collector** class.

```
create() updating;

constants
    MicrosoftUUID = 3;
begin
    self.googleID := app.getGoogleID;
    self.uuid     := app.generateUuid(MicrosoftUUID).uuidAsString;
end;
```

This method is automatically called every time a **Collector** object is instantiated.

3. Add a protected method called **sendToGoogle**, coded as follows, to the **Collector** class.

```
sendToGoogle(msg : String) protected;

vars
    httpConnection  : JadeHTTPConnection;
begin
    // Set up httpConnection:
    create httpConnection transient;
    httpConnection.url := "https://www.google-analytics.com/collect";
    httpConnection.open(true);

    // Send message
    write "Sending...";
    httpConnection.sendRequest("POST", null, msg);
    write "Sent!";
epilog
    delete httpConnection;
end;
```

Note By taking the responsibility of sending the messages via the HTTP connection as well as generating them based on the provided information, **Collector** is breaking the Single Responsibility Principle of good object-oriented design. This will be resolved in a future exercise.

4. Add a protected method called **generateMessage**, coded as follows, to the **Collector** class.

```
generateMessage(description : String; hitType : String) : String protected;

vars
    msg : String;
begin
    // Common message code
    msg :=    "v=" & "1" & "&"
          & "tid=" & self.googleID & "&"
          & "cid=" & self.uuid & "&"
          &   "t=" & hitType & "&"
          &  "an=" & app.name & "&";

    // Specific message code
    if hitType = "screenview" then
        msg := msg & "cd=" & description;
    endif;

    return msg;
end;
```

5. Add a public method called **postScreenView**, coded as follows, to the **Collector** class.

```
postScreenView(description : String);

vars
    msg : String;
begin
    msg := self.generateMessage(description, "screenview");
    self.sendToGoogle(msg);

end;
```

6. The **Collector** class is used by the forms created in the next exercise, but for now, add a new JadeScript method called **testCollector**, coded as follows.

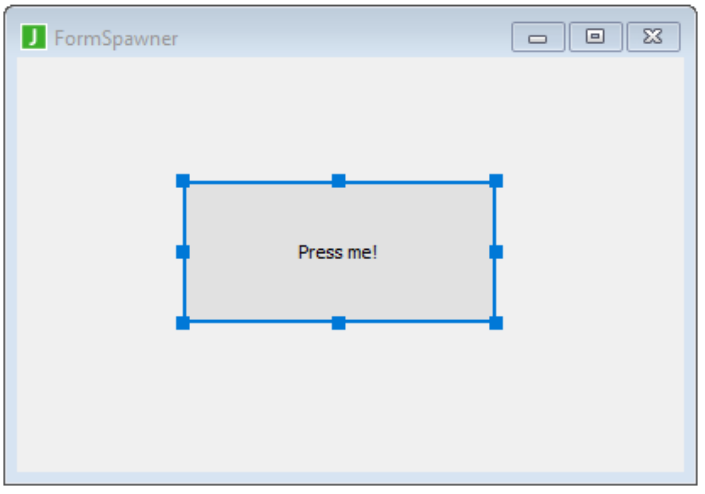
```
testCollector();

vars
    collector : Collector;
begin
    create collector transient;
    collector.postScreenView("I am a test.");
epilog
    delete collector;
end;
```

7. Execute the method and verify that the message is received by Google Analytics.

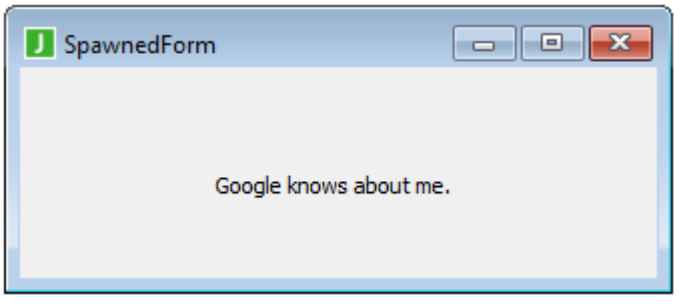
Exercise 4 – Creating a Self-tracking Form

In this exercise, you will create an application that automatically reports to Google Analytics when forms are created.

1. Open the JADE Painter (Ctrl+P).
2. Create a new form called **FormSpawner**, which has a single button called **btnCreateForm** that has the caption **Press me!**



3. Create another new form called **SpawnedForm** with a single label called **lblDescription** that has the caption **Google knows about me.**



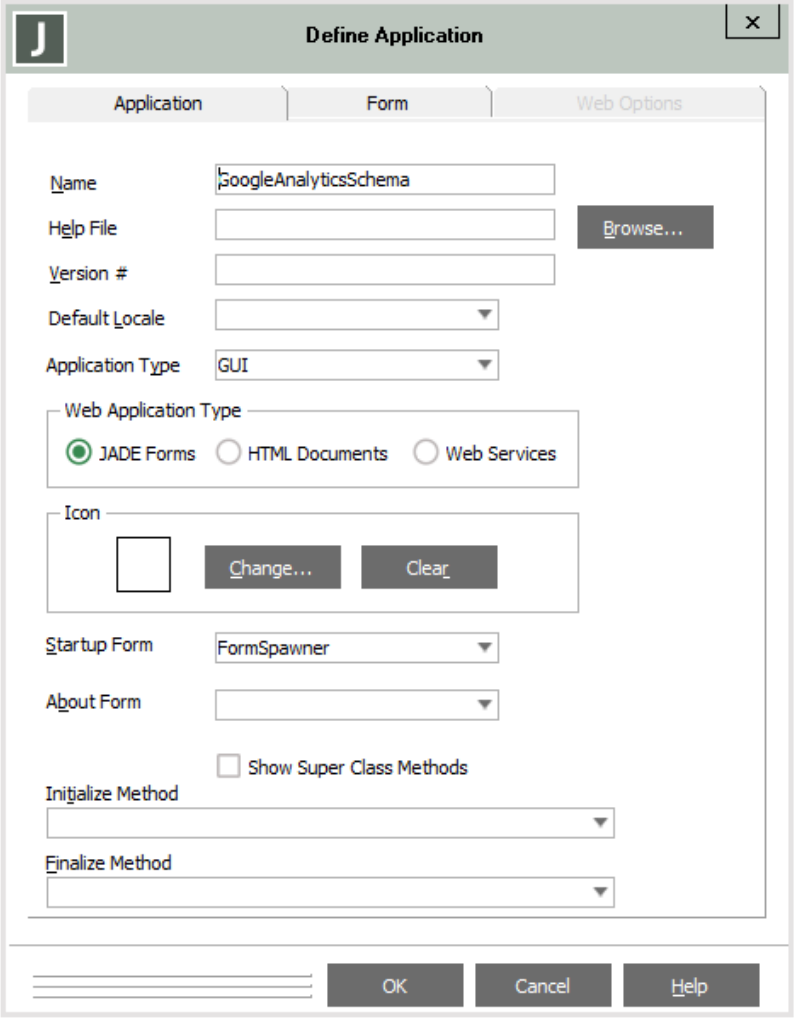
4. Navigate to **FormSpawner** in the Class Browser and code the **click** event method of the **btnCreateForm** control as follows.

```
btnCreateForm_click(btn: Button input) updating;

vars
    form : SpawnedForm;
begin
    create form;
    form.show();
end;
```

5. Navigate to **SpawnedForm** in the Class Hierarchy Browser and code the **load** method of Form Events as follows.

```
load() updating;

vars
    col : Collector;
begin
    create col transient;
    col.postScreenView("Dummy Form");
epilog
    delete col;
end;
```

6. Open the Application Browser (Ctrl+L) and ensure that the **GoogleAnalyticsSchema** application has the Startup Form/Document set to **FormSpawner**. If not, right-click it and specify **FormSpawner** in the Startup Form combo box on the **Application sheet** of the Define Application dialog.



7. Run the **GoogleAnalyticsSchema** application and click **Press me!** button. Notice how there is a slight delay before the **SpawnedForm** is displayed, and if you click repeatedly on the **Press me!** button, you may notice that it becomes unresponsive to some of the clicks.

Exercise 5 – Creating a Sender Class

In this exercise, you will create a **Sender** class that is responsible for sending HTTP requests to Google Analytics. You will also create shared transient **Sender** and **Collector** objects and create references to them on the **Application** class.

1. Create a new class called **Sender**.
2. Create a method called **send**, coded as follows, in the **Sender** class.

```
send(msg : String);

vars
    httpConnection : JadeHTTPConnection;
begin
    // Set up httpConnection:
    create httpConnection transient;
    httpConnection.url := "https://www.google-analytics.com/collect";
    httpConnection.open(true);

    // Send message
    write "Sending...";
    httpConnection.sendRequest("POST", null, msg);
    write "Sent!";
epilog
    delete httpConnection;
end;
```

3. Add a public reference called **collector** of type **Collector** to the **GoogleAnalyticsSchema** class (an **Application** subclass).
4. Add a public reference called **sender** of type **Sender** to the **GoogleAnalyticsSchema** class.
5. Create a method called **initialize**, coded as follows, in the **GoogleAnalyticsSchema** class. (The JADE development environment will warn you that it is reimplementing a superclass method, which is fine.)

```
initialize() updating;

vars
    senderSingleton     : Sender;
begin
    beginTransientTransaction;
    create self.collector sharedTransient;
    commitTransientTransaction;

    senderSingleton     := Sender.firstSharedTransientInstance;
    if senderSingleton = null then
        beginTransientTransaction;
        create self.sender sharedTransient;
        commitTransientTransaction;
    else
        self.sender := senderSingleton;
    endif;
end;
```

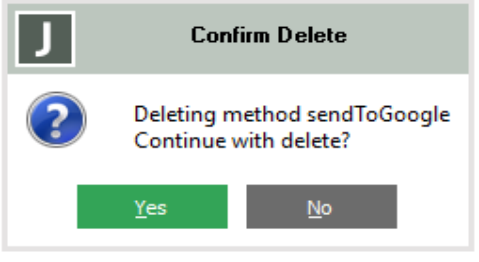
6. Create a new method called **finalize**, coded as follows, in the **GoogleAnalyticsSchema** class. (The JADE development environment will warn you that it is reimplementing a superclass method, which is fine.)

```
finalize() updating;

begin
    beginTransientTransaction;
    delete self.collector;
    commitTransientTransaction;
end;
```

7. Modify the **postScreenView** method in the **Collector** class, as follows.

```
postScreenView(description : String);

vars
    msg : String;
begin
    msg := self.generateMessage(description, "screenview");
//  self.sendToGoogle(msg);
    app.sender.send(msg);

end;
```

8. Delete the **sendToGoogle** method in the **Collector** class (select it in the Class Browser, right-click on it, and then select **Remove**).



Click **Yes** in the Confirm Delete message box.

The **Collector** class is now single responsibility. However, there is still the responsiveness issue in the GUI while Google processes the analytics data.

The Importance of Asynchronous Send Operations

The reason for the unresponsive and delayed forms is that the **load** method of the **SpawnedForm** window does not complete, and therefore the **form.show();** instruction does not execute until Google replies to the **JadeHTTPConnection** class **sendRequest** method with a confirmation that it received the message. This is not instantaneous, and therefore causes slight lags in the user's actions.

The solution is to use asynchronous method calls. An asynchronous method call is where the method runs on a separate application called a *worker*. This allows the caller to continue with the next instruction without waiting for a long-running method call to finish executing.

Note For more information about asynchronous method calls and other multithreading techniques in JADE, see the "Multithreading" module of the JADE Developer's course.

Exercise 6 – Creating a Pool of Workers

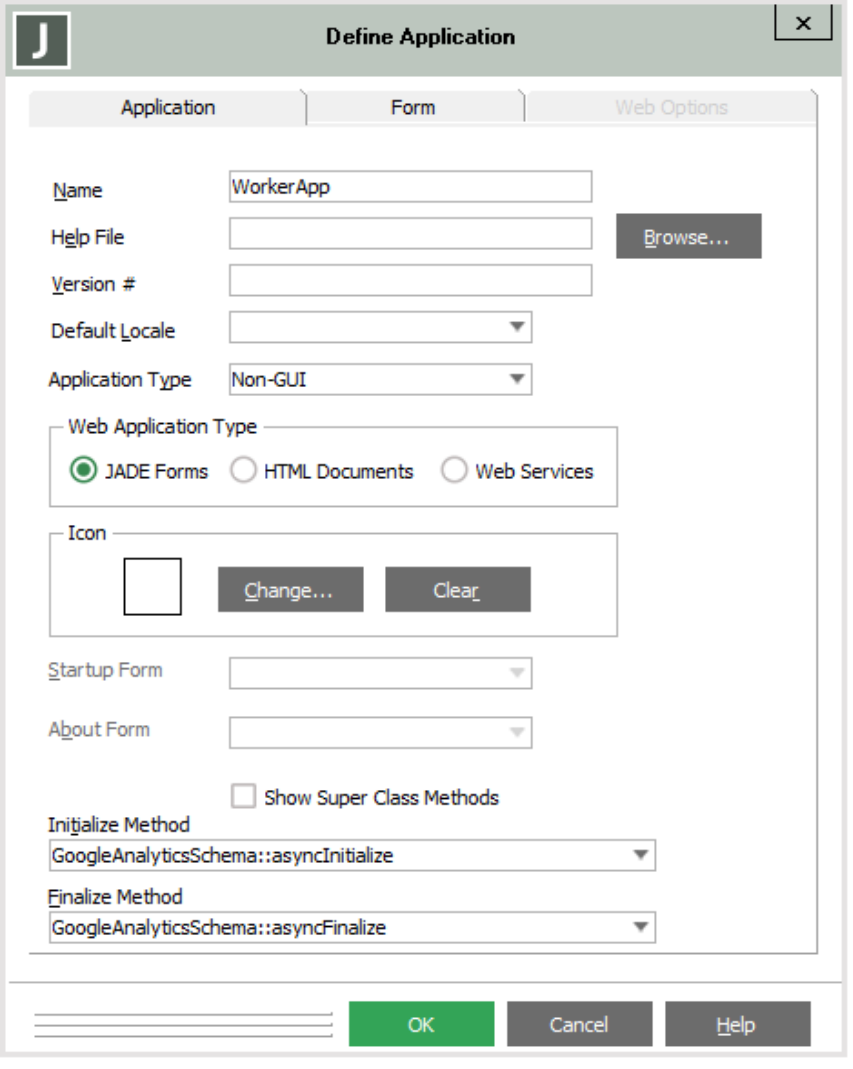
In this exercise, you will create multiple worker applications that will be shared amongst all tracked applications.

1. In the **GoogleAnalyticsSchema** subclass of **RootSchemaApp** (a subclass of **Application**) Application Class, add new methods called **asyncInitialize** and **asyncFinalize**, with only an **inheritMethod** call each. (The JADE development environment warns you that it is reimplementing superclass methods, this is fine.)

```
asyncInitialize() updating;

begin
    inheritMethod();
end;
```

```
asyncFinalize() updating;

begin
    inheritMethod();
end;
```

2. Open the Application Browser (Ctrl+L).
3. Add a new non-GUI application called **WorkerApp**, as follows.



4. Add a new protected reference called **workerPool**, of type **ProcessDict**, to the **Sender** class.

5. Add a **create** method, coded as follows, to the **Sender** class.

```
create() updating;

constants
    WorkerCount = 5;
vars
    i : Integer;
begin
    foreach i in 1 to WorkerCount do
        self.workerPool.add(app.startApplication(currentSchema.name, "WorkerApp"));
    endforeach;
end;
```

This method will create five workers whenever the singleton shared transient is created.

Exercise 7 – Employing the Async Workers

In this exercise, you will have the **Sender** class use the worker applications to perform the HTTP POST requests asynchronously.

1. Add a new method called **sendAsnyc**, coded as follows, to the **Sender** class.

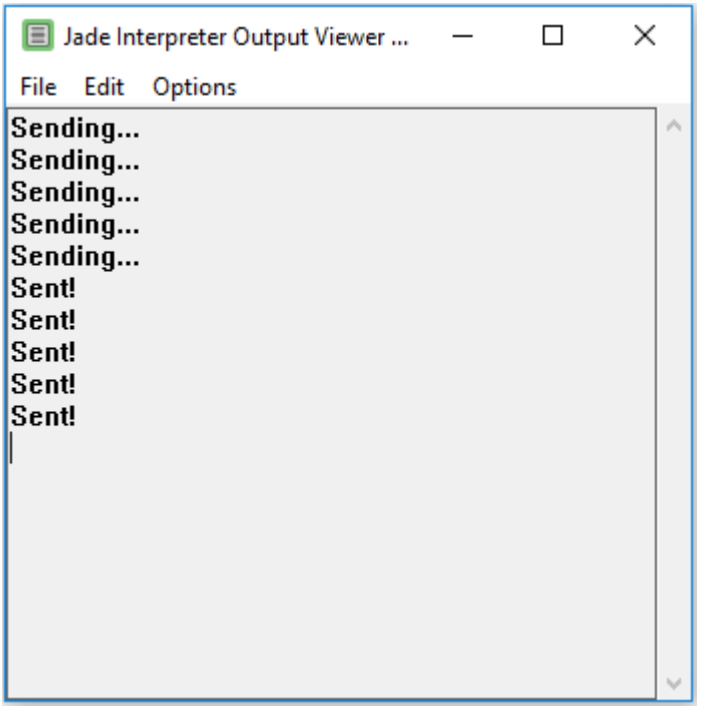
```
sendAsync(msg : String);

vars
    context : JadeMethodContext;
begin
    create context transient;
    context.workerAppName := "WorkerApp";
    context.invoke(self, send, msg);
epilog
    delete context;
end;
```

2. Modify the **postScreenView** method in the **Collector** class as follows.

```
postScreenView(description : String);

vars
    msg : String;
begin
    msg := self.generateMessage(description, "screenview");
//  self.sendToGoogle(msg);
//  app.sender.send(msg);
    app.sender.sendAsync(msg);
end;
```

3. Run the **GoogleAnalyticsSchema** application (closing it first, if it's already open).
4. Click the **Press me!** button multiple times.

You will see that the **SpawnedForm** windows are now displayed instantly, and in the Jade Interpreter Output Window, multiple sends are possible before the first reply is received (if you are fast enough).



Event Tracking

So far, the only data that has been sent to Google Analytics has been screen views, which are useful for tracking how often different screens of an application are viewed by users.

Another useful metric to track is when users trigger specific events. An event could be clicking on a button, entering text into a text box, or checking a check box.

In addition to the common message code, the following must be present for event data messages.

Name	Parameter	Example	Description
Event Category	ec	ec=Delivery Tracking	Specifies the event category. Must not be empty.
Event Action	ea	ea=Checked Delivery	Specifies the event action. Must not be empty.
Event Label	el	el=Delivery Checkbox	Specifies a label for the event. Not strictly necessary, but recommended.

Conversion Tracking

An important use of events is conversions tracking. A conversion is the completion of any activity by a user that is of interest to the application owner. For example, an online shop would be interested in when users choose to purchase an item, and how they navigated through the application before they confirmed a purchase.

Conversions are divided into two main categories: macro conversions and micro conversions. A macro conversion is an end-goal of the business (purchasing an item, signing up to a newsletter, completing a survey) while a micro conversion is a sub-goal of a macro conversion; for example, adding an item to a shopping cart or answering a survey question.

Exercise 8 – Tracking Events

In this exercise, you will add controls to the **SpawnedForm** window and track what percentage of users interact with those controls. You will also modify the **Collector** class to handle event data messages.

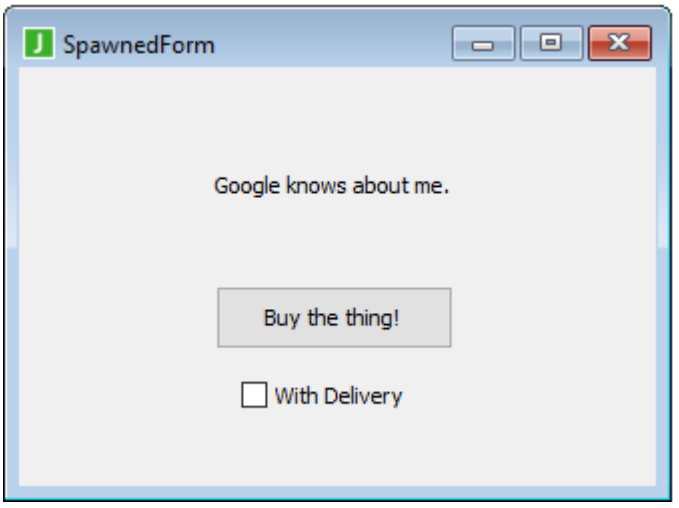
1. In the **Collector** class, modify the **generateMessage** method as follows.

```
generateMessage(description : String; hitType : String) : String protected;

vars
    msg : String;
begin
    // Common message code
    msg :=    "v=" & "1" & "&"
          & "tid=" & self.googleID & "&"
          & "cid=" & self.uuid & "&"
          &   "t=" & hitType & "&"
          &  "an=" & app.name & "&";

    // Specific message code
    if hitType = "screenview" then
        msg := msg & "cd=" & description;
    elseif hitType = "event" then
        msg := msg & "ec=" & description & "&";
        msg := msg & "ea=Clicked" & "&";
        msg := msg & "el=Successful Sale";
    endif;

    return msg;
end;
```

2. Add a method called **postEvent**, coded as follows, to the **Collector** class.

```
postEvent(description : String);

vars
    msg : String;
begin
    msg := generateMessage(description, "event");
    app.sender.sendAsync(msg);
end;
```

3. Open the JADE Painter and modify the **SpawnedForm** window as follows.



   a. Add a button called **btnPurchase** with the caption **Buy the thing!**

   b. Add a check box called **cbDelivery** with the caption **With Delivery**

4. Modify the **change** method of the **cbDelivery** check box as follows.

```
cbDelivery_change(checkbox: CheckBox input) updating;

vars

begin
    if cbDelivery.value then
        app.collector.postEvent("Checked Delivery");
    elseif not cbDelivery.value then
        app.collector.postEvent("Unchecked Delivery");
    endif;
end;
```

5. Modify the **click** method of the **btnPurchase** button as follows.

```
btnPurchase_click(btn: Button input) updating;

vars

begin
    if self.cbDelivery.value then
        app.collector.postEvent("Purchased with delivery");
    elseif not self.cbDelivery.value then
        app.collector.postEvent("Purchased without delivery");
    endif;
    delete self;
end;
```

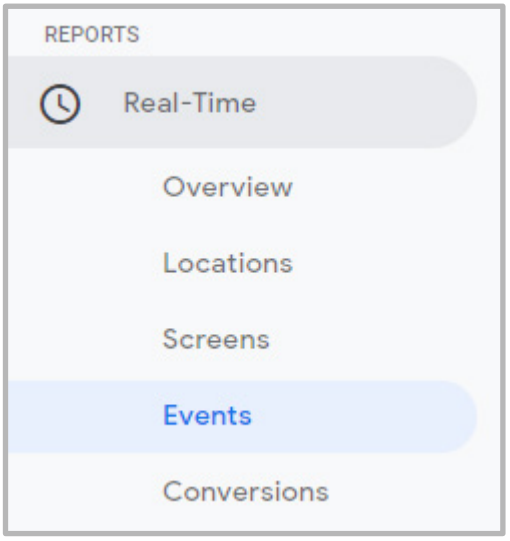
6. Run the **GoogleAnalyticsSchema** application (closing it first, if it's already running).

7. Click the **Press me!** button to display the **SpawnedForm**, then check the **With Delivery** check box.

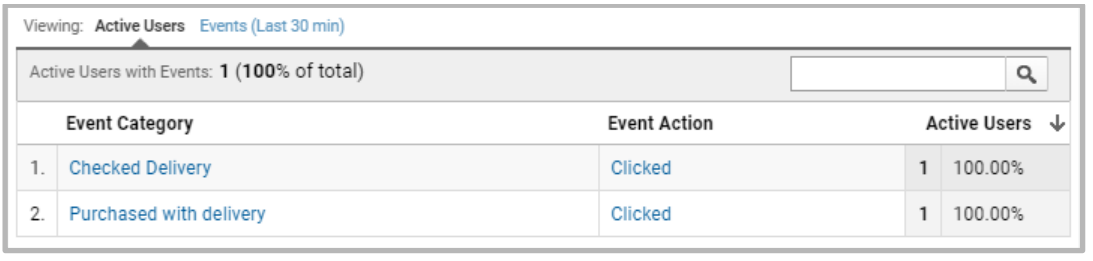
8. Click the **Buy the thing!** button.

9. If the Google Analytics site (https://analytics.google.com) is not already open, open it in your preferred browser.

10. Under **REPORTS** in the left-hand navigation bar, select **Real-Time** and then **Events**.



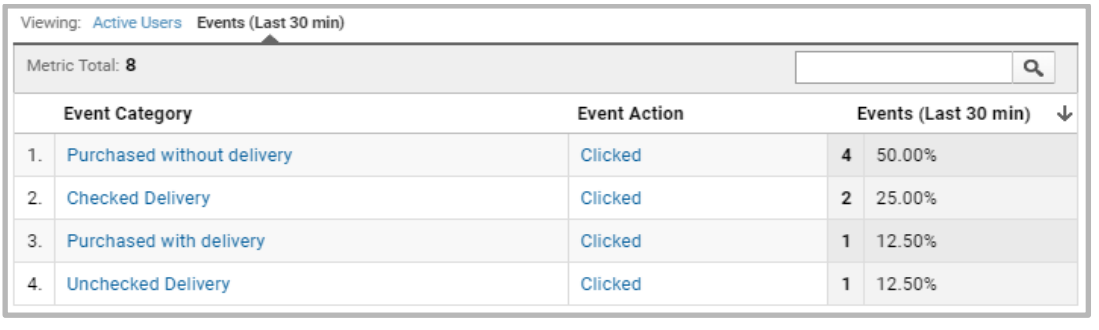
When the HTTP POST operation completes, the following should be displayed.



	Event Category	Event Action	Active Users	
1.	Checked Delivery	Clicked	1	100.00%
2.	Purchased with delivery	Clicked	1	100.00%

11. Reopen the application each time (to appear as a new user) and then try the following actions.
    a. **Buy the thing!** with delivery
    b. **Buy the thing!** without delivery
    c. Open the **SpawnedForm** window but don't **Buy the thing!**
    d. Check and uncheck the **With Delivery** check box

12. On the **Events** page of Google Analytics, click the **Events (Last 30 min)** tab to display the **Events (Last 30 min)** sheet.



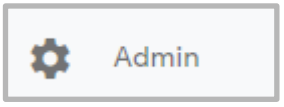
	Event Category	Event Action	Events (Last 30 min)	
1.	Purchased without delivery	Clicked	4	50.00%
2.	Checked Delivery	Clicked	2	25.00%
3.	Purchased with delivery	Clicked	1	12.50%
4.	Unchecked Delivery	Clicked	1	12.50%

When all HTTP POST operations have completed, each type of event recorded is displayed, along with how many times that event occurred and the percentage of users who performed it.

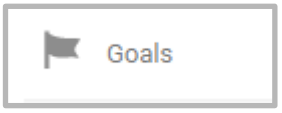
Exercise 9 – Measuring the Conversion Rate

In this exercise, you will add a goal to Google Analytics and measure the conversion rate for that goal. The goal will include the **Purchased without delivery** and **Purchased with delivery** events, and as such, will measure what percentage of users make a purchase.

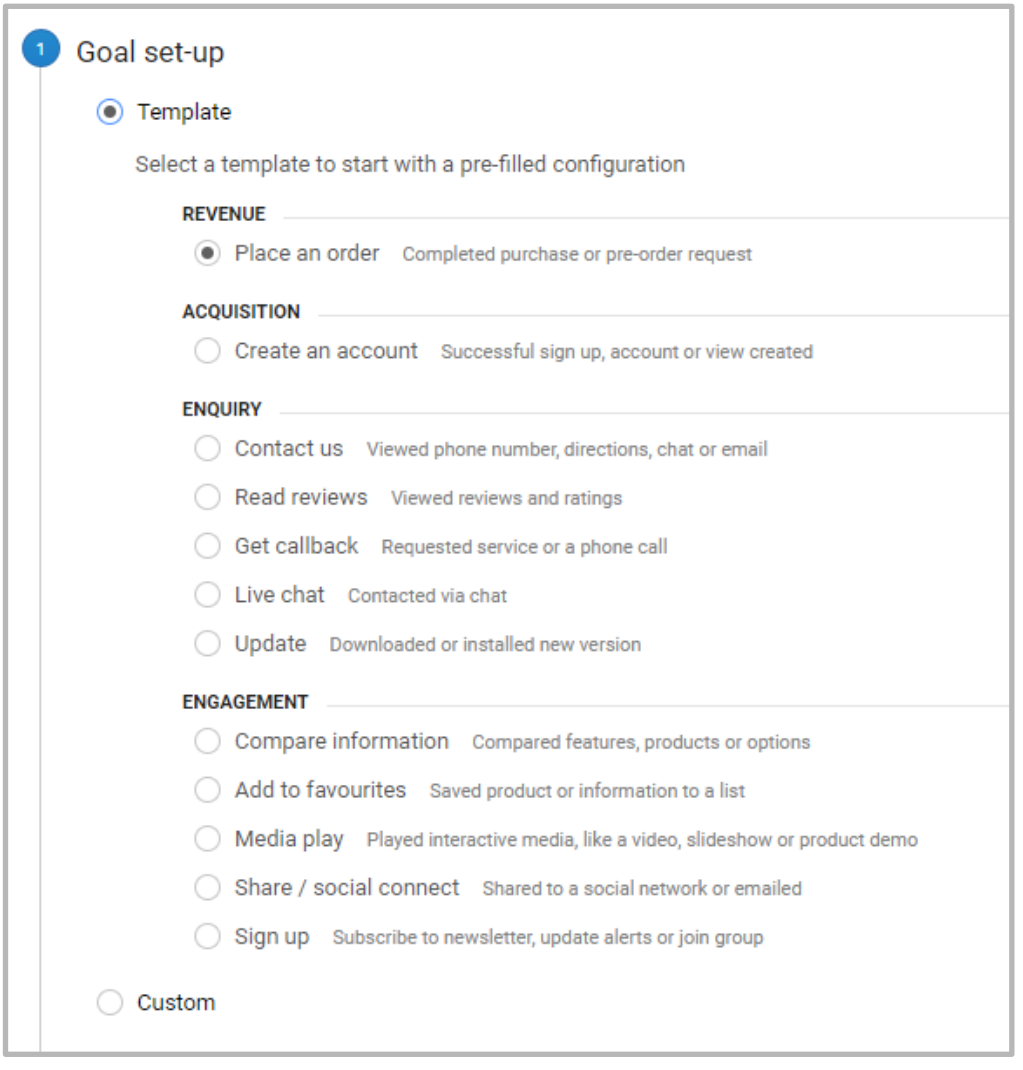
1. From the Google Analytics page, click the **Admin** icon on the left-hand navigation bar.



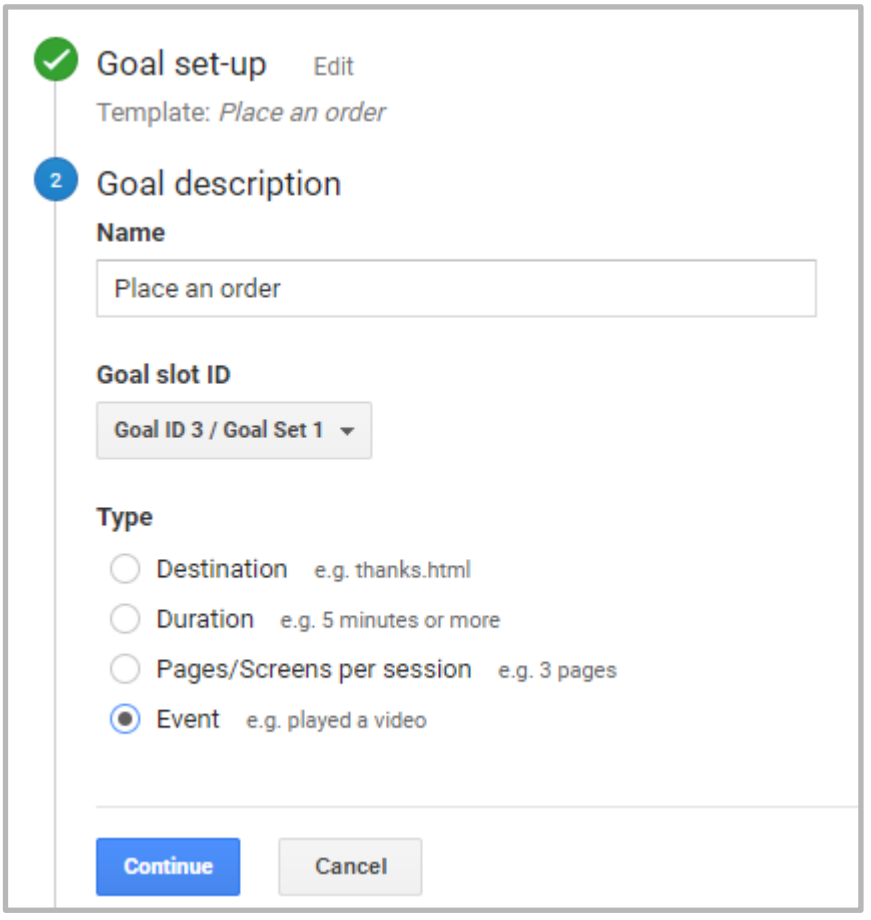
2. Click **Goals**.



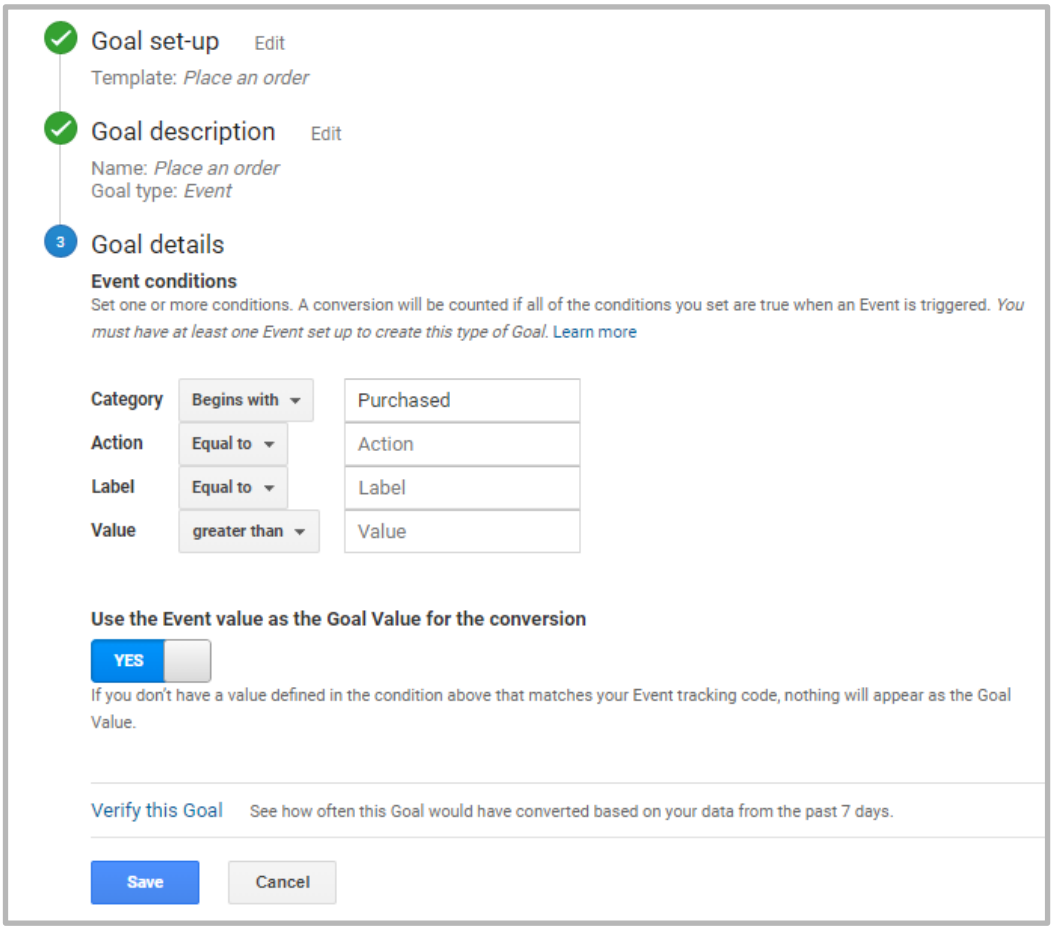
3. Click **+ NEW GOAL**.
4. Select the **Place an order** option button, then **Continue**.



5. Select the **Event** option button, then **Continue**.



6. Select the **Begins With** option for the Category, with **Purchased** as the value.



7. Click **Save**.
8. Reopening the application each time (to appear as a new user), perform the following actions.
   a. **Buy the thing!** with delivery
   b. **Buy the thing!** without delivery
   c. Open the **SpawnedForm** window but don't **Buy the thing!**
9. In the Google Analytics page under **REPORTS** in the left-hand navigation bar, select **Real-Time** and then **Conversions**. The following should then be displayed.