

Introduction to the Oracle Stream Explorer

Fast Data and Event Processing without Software Coding

ORACLE WHITE PAPER | MARCH 2015





Table of Contents

Introduction	2
Understanding Shapes, Streams, References and Explorations	3
Shapes	3
Streams and References	3
Explorations and Patterns	4
Case Study: Implementing the Minority Report Mall Scene	6
Part One: Creating the Solution Design for the Scenario	7
Part Two: Implementing the Artifacts in Oracle Stream Explorer	9
Conclusion	23
Appendix A: Scripts and Samples used in the Case Study	23
Appendix B: Creating a Message-Driven Bean that Greets	29



Introduction

Events are everywhere. From the moment we wake up until the next day, millions of events happen without our knowledge. According to any popular dictionary, an event is something that just happens. It can be a thermostat being adjusted by a homeowner, a credit card being processed at the grocery store, or a car passing through an automated toll station on a highway. Virtually everything in the final analysis can be viewed as an event, and many types of events may be related. Paying attention to events is important because they tell us what is going on and provide us with awareness of the current situation around us and in the wider world.

The technique of analyzing event relationships and their consequences is called event processing. Event processing handles events while they are still in motion because that is the perfect moment to take action, like paying attention to a low fuel warning light on your car to avoid running out of gas on a highway. Once events happen, they become past tense and therefore become nothing more than a record of a fact. Analyzing historical facts still has its value, but depending of the events that occurred and the situation those events represented, it may signify a missed opportunity or a threat that went unnoticed. For this reason, event processing has never been as important as it is now, especially for twenty-first-century enterprises.

Most people think that many, if not all, industries already use event processing somehow, but the reality is that few of them are actually using it. Most enterprises use EDA (Event-Driven Architecture) to allow the exchange of events between different systems. But despite the usage of inherent EDA characteristics such as loose coupling and message routing; there is no actual event processing in place, only event delivery. Just to name a few, Automated Trading and Online Gaming are examples of industries using event processing. But those are event-driven industries by nature; event processing is part of their core business. Why then, if event processing is that important, are other industries still failing to leverage it?

The main reason is likely the level of abstraction of current technologies. When compared to business intelligence, event processing technologies offers a low level of abstraction. Even the most powerful event processing technologies like OEP (Oracle Event Processing) are focused on the developer community. In this context, OEP offers a complete set of tools to create, test, debug and deploy event processing applications; that include extreme low latency and high throughput, just like fault-tolerant capabilities. But this comes with a high price, which is exposing the technical details of the technology to the person interested in doing the analysis.



There is a clear need in the industry for a product that provides event processing power but without the need to understand all the technical details that can confuse business users. This was the reason Oracle launched the new Oracle Stream Explorer product; to provide business users with a platform that allows the creation of event processing applications through an intuitive and simple user interface. Oracle Stream Explorer is a web-enabled application available to be used both on-premise and via SaaS in the Oracle Cloud, capable of providing a zero-coding environment for people interested in analyzing streams of events in real-time.

The goal of this paper is to provide the basic information necessary to start building applications using Oracle Stream Explorer. It will provide a solid introduction to the main product features and will show step-by-step how to develop a sample application based on an interesting case study.

Understanding Shapes, Streams, References and Explorations

This section will be focused in providing a solid introduction to the most important artifacts created during the development of event processing applications. It is highly recommended that first time users working with Oracle Stream Explorer spend some time reading this section before venturing into the product.

Shapes

According to many psychologists, when human beings start learning something new, they mentally break down objects into simple geometric forms called geons. Geons are simple 2D or 3D forms such as cylinders, bricks, wedges, circles or rectangles. When observed together, all geons can describe a higher geon such as a car, but even when they are observed individually, geons also mean something because in the human brain, every geon is matched against a structural representation of these objects. In the real world, these so called geons are known as shapes.

Shapes are the fundamental building blocks in which all the objects are classified. Every object is a shape therefore it has a structural representation. Building event processing applications is no different. Every piece of data being analyzed or helping in the analysis is a shape. You need to create a shape to provide a working representation of a data set. Without creating a shape, there is no way in Oracle Stream Explorer to process any data.

From the technical point of view, a shape in Oracle Stream Explorer is a type of artifact that provides metadata about a data set. It has a name, a list of attributes and their respective data types, and it is commonly used when associated with a source type. A source type determines from where the data will come from, and behind the scenes each source type is implemented through an OEP adapter. Another important aspect about shapes is that they are never persisted. During the analysis of stream of events, data is brought into memory but as soon as the analysis finishes all the data is discarded.

Streams and References

Events in general can be classified according to their temporal state. There is previously processed data that represent facts from the past and there are the ongoing events that represent what is happening right now. It is undeniable that event processing is all about discovering what is happening right now but, in most cases, with only the events happening right now, there is not enough data to understand the entire context. As mentioned before, the



analysis of historical facts still has its value, and in the world of event processing, this value is based on the idea of past events being used to provide more context to the current stream of events.

For instance, imagine that many ships carrying commercial products continuously send events about their locations and from those events you need to detect and alert when a particular ship is delayed which means product delivery may be delayed. Assuming that the event holds at least the identifier of the ship, you would still need an external data source to find out where the ship is coming from, registry, etc. In this context, the events coming from the ships are the ongoing events and they represent what is happening right now, and the details about each ship represents previously processed data, therefore they are facts from the past.

In Oracle Stream Explorer, any unbounded sequence of data used to represent an ongoing event is called stream. A stream is continuous; it never stops and provides the raw material for the event processing analysis. But most analysis requires some of this contextual or historical data to be useful. Any static data that represents data used to add more context to a stream it is called reference. References are static and the name came from its usage: they are used to reference already processed data. From the temporal state perspective, the streams represent what is happening right now and the references represents facts from the past.

Explorations and Patterns

During the development of event processing applications, there are a set of common activities that must be performed to achieve some desired business goal, such as creating junctions, temporal constraints, performing aggregation and filtering and customizing the output result. In most cases, these activities must be repeatedly performed until the desired business goal is reached. In Oracle Stream Explorer, this is accomplished through an artifact called an exploration.

Every exploration needs to be associated with a list of sources, which can be a stream or a reference. But, at least one stream must exist in the list of sources, and it must be the first source on the list. The other sources can be other streams or references, and there is no limit to the number of sources used in the list. Due to the relationship between sources and explorations, all sources intended to be used in explorations need to be created in advance, prior to the exploration.

The output result of an exploration is invariably a new stream. For this reason, once the exploration is published, it can be used in the list of sources of other explorations, just like any other stream. Figure 1 shows the relationship between sources and explorations.

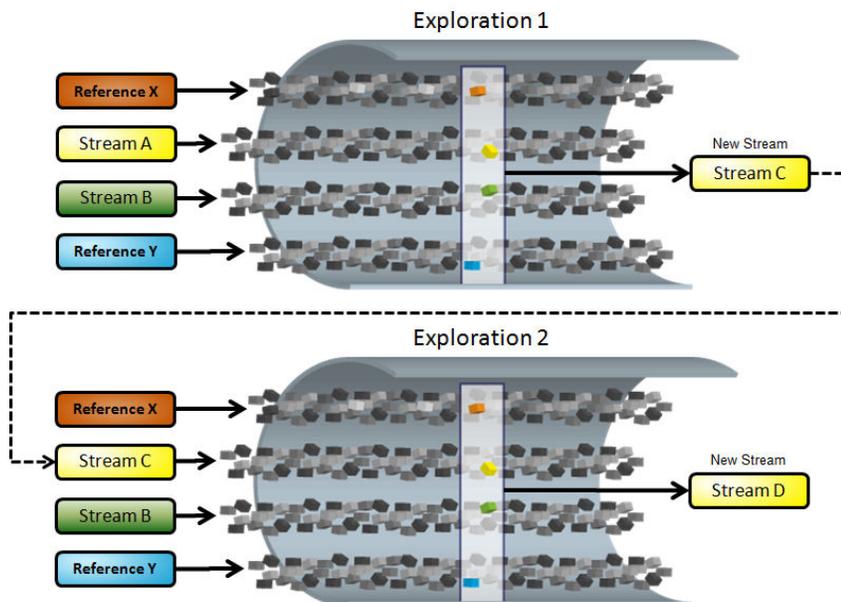


Figure 1. The relationship between sources and explorations.

Working with explorations can be very time consuming, especially if the scenario being explored requires complex logic. For this reason, Oracle Stream Explorer makes available a collection of pre-built explorations called patterns. Patterns are generic solutions to recurring common problems, and they help to keep the focus on the business problem instead of on the implementation details.

The usage of patterns is fairly simple: After choosing which pattern needs to be implemented, specific data or key fields will be required, and after providing all of them the entire exploration will be automatically generated. The following patterns are currently available in Oracle Stream Explorer:

- » **Top N:** Used to obtain the first "N" events from the event stream.
- » **Bottom N:** Used to obtain the last "N" events from the event stream.
- » **Up Trend:** Detects when a numeric field shows a specified trend change higher in value.
- » **Down Trend:** Detects when a numeric field shows a specified trend change lower in value.
- » **Fluctuation:** Detects when a given field value changes in a specific upward or downward fashion within a specific time window.
- » **Eliminate Duplicates:** Eliminates duplicate events in the event stream.
- » **Detect Duplicates:** Detects when a given data field has duplicate values within a specified period of time.
- » **Detect Missing Event:** Detects when an expected event does not occur within a specific time window.
- » **W:** Detects when a given field value rises and falls in "W" fashion over a specified time window.
- » **Inverse W:** Use this pattern to detect inverse W.

Once an exploration is built it can be used as a source for another exploration or it can be exported. The decision to export the exploration will be normally determined when the exploration is meant to become an OEP application; originally created in Oracle Stream Explorer to reach the business goal but to be later enhanced in OEP to handle technical details such as integration, security, performance, scalability and fault-tolerance.

As previously mentioned, Oracle Stream Explorer has OEP as its foundation, and the generated explorations are nothing more than EPNs (Event Processing Networks) deployed as OEP applications. Figure 2 shows an example of an EPN. Because explorations are EPNs, they can be exported from Oracle Stream Explorer and imported again

into a development environment such as Oracle Fusion Middleware JDeveloper. When the exploration is exported, a JAR file is generated with common EPN artifacts such as event types, adapters, caches, channels, processors and CQL (Continuous Query Language) statements.

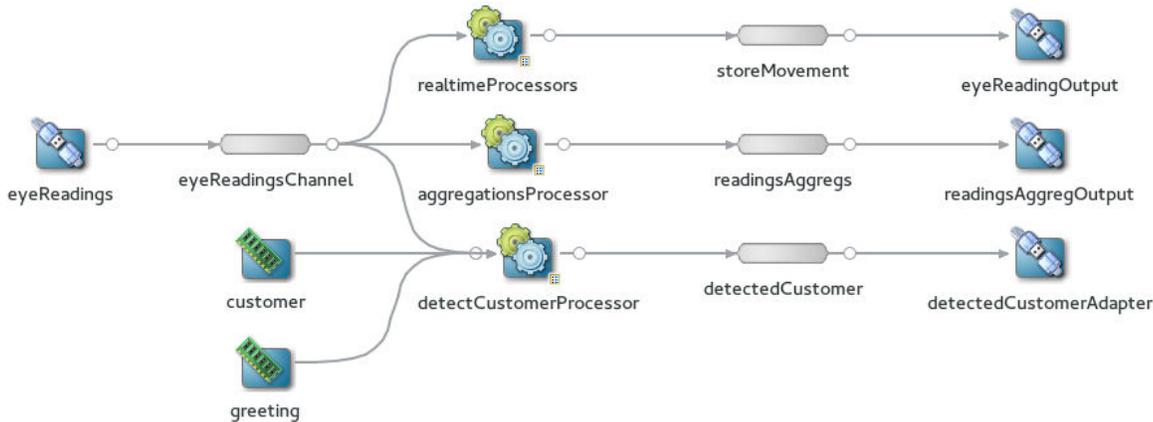


Figure 2. Example of an EPN generated from the implementation of an exploration.

Case Study: Implementing the Minority Report Mall Scene

In July of 2002, 20th Century Fox's studios released *Minority Report*. Directed by Steven Spielberg and starring Tom Cruise as the main character, the film rapidly became one of the most successful science fiction movies ever made. The film revolves around the story of a special police program called PreCrime. Using three mutated humans stored in a special chamber where they experience precognitive visions, the PreCrime program is designed to stop murders before they happen, deploying the police force in preemptive fashion to arrest the future murderers. In the film, the PreCrime program is established in Washington D.C in the year of 2050, and Tom Cruise plays the role of John Anderton; a police captain responsible for the PreCrime program. Besides presenting a very interesting story, the film also has lots of scenes in which high technology is used, revealing how the near future could be.

In one of those scenes, John Anderton needs to get in into the chamber where the mutated humans are, but he faces a challenge since he is now the main suspect of a future crime. In the year of 2050, nearly all public places have eye scanners that can identify every citizen. So in order to not be apprehended he has no choice but to get rid of its own eyes and have then surgically exchanged with a pair of eyes that belonged to a deceased person known as Yakamoto. After the surgery and already in possession of Mr. Yakamoto eyes, he finds himself into an odd situation when, looking for some new clothes, he enters in a GAP store and it is surprised with a greeting message saying "Hello Mr. Yakamoto, welcome back to the GAP".

What this scene taught us is that the technology of sensors combined with event processing can provide an individualized experience for customers, reacting appropriately when needed and with information about the customer's personal preferences. The sensor in this context are the eye scanners that continuously read people's retinas and send the results to a capable event processing technology for processing. The event processing part comes into play when it has to, from a stream of multiple events; detect which one of them is near the store and, after recognizing who the person is, greeting them with a custom message.

This scenario would not be so challenging if it was not for the fact that the event processing system must trigger the custom message while the person is near the store. If the current business intelligence techniques were used in this scenario, the events from the scanned people would be stored in a staging database where a scheduled job would load the last "N" scans that were near the store, possibly an hour, day or even a week later. To make sense, the

custom message must be delivered while the person is in close proximity the store, a situation that may only last for a few seconds.

This is when the technology of event processing comes into play making it feasible to actually build a system that could make this type of science fiction scene possible. Event processing enables the analysis of what is happening right now, handling events while they are still ongoing and detecting complex relationships between them. The mall scene from the Minority Report film will be the case study that is going to be developed in this section. Using Oracle Stream Explorer, this case study is going to be built through explorations, covering the detection of when a person is near the store to the moment in which he or she is recognized to finally, receive a custom greeting message. In order to make the implementation feasible, the following assumptions will be assumed:

- » Each eye scan event holds the person retina and his or her location.
- » All customer information is available through a database table.
- » The custom greetings are available through a database table.

The development of this case study will be divided in two parts. The first part will cover all the details related to the solution design, where all the necessary artifacts will be identified. The second part will cover the development of those artifacts in Oracle Stream Explorer, showing how to create them and how to handle the configuration details.

Part One: Creating the Solution Design for the Scenario

When designing a solution that is going to be implemented in Oracle Stream Explorer, you need to detail all the artifacts that will be required, starting with the necessary shapes. Considering that shapes are the representation of all data that is going to flow through the event processing application, the first step of the solution design is to build of a conceptual model, separating the shapes into two categories: the shapes already available and the shapes that are going to be needed. Figure 3 shows the conceptual model of this case study.

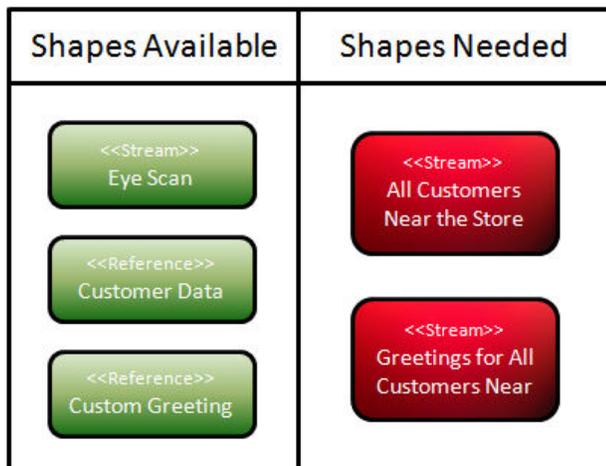


Figure 3. Shapes in the conceptual model of the case study.

In order to classify each shape according to its usage, the conceptual model may include stereotypes. Shapes associated with ongoing events are stereotyped as streams and shapes associated with already processed data are stereotyped as references. All the shapes found in the needed category may be stereotyped as streams because, from the Oracle Stream Explorer perspective, they derive from explorations and the output result of an exploration is invariably a new stream. For this reason, it would not be considered a mistake using the stereotype exploration instead of a stream, because in the final analysis every exploration is a stream.

The next step is the inclusion of the causality relationships between all the shapes, emphasizing how the processing of one or more shapes will result in the creation of a new shape. In this context, the term processing refers to the usage of any activity of junction, temporal constraint, aggregation or filtering, performed during the execution of an exploration. It is also important that during this step the attributes of each shape be included, just like the mapping between the attributes of the shapes available and the shapes needed. Figure 4 shows the conceptual model with the causality relationships.

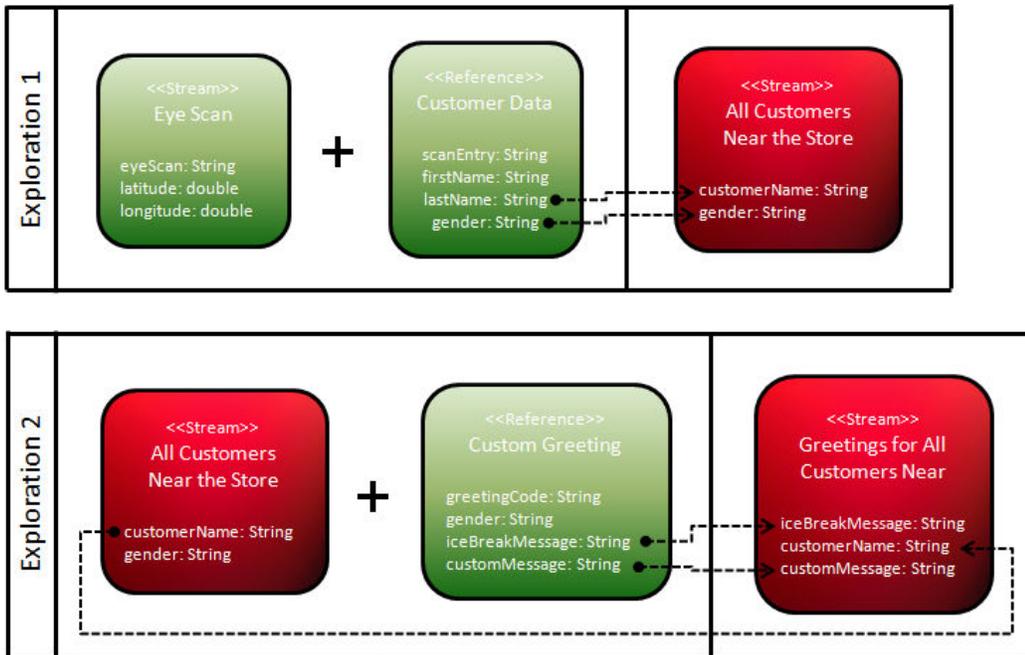


Figure 4. Conceptual model refined to include the causality relationships.

Each lane shown on figure 4 represents an exploration that needs to be built in Oracle Stream Explorer. Finding out in advance how many explorations will need to be built is important because it can reveal how much development effort the scenario will demand from the implementation perspective. While there is no rule that dictates that causalities need to be implemented in different explorations, it is considered a best practice to break down the business problem into smaller explorations, promoting better reuse of the artifacts and moving the complexity toward a reasonable level.

Each exploration must have a meaningful name. As a rule of thumb, consider that each exploration must be named with the shape name that it intends to create. If business rules need to be applied during the execution of the exploration, it is considered a best practice to provide a description of these business rules. For instance, the first exploration has as business rule the utilization of the latitude and longitude attributes present in the eye scan stream to detect if the customer is near the store or not. Also considered as a business rule, the eyeScan attribute from the eye scan stream needs to be correlated with the scanEntry attribute present in the customer reference, in order to find out who the person near the store is.

The target audience of this description will be the person responsible for the implementation of the artifacts in Oracle Stream Explorer, which can be the same person that builds the conceptual model or it can be someone different. In most cases, the same person that builds the conceptual model also implements the artifacts since Oracle Stream Explorer provides a high level of abstraction in regards to implementation details. For this reason, people with few, if any, software development skills can use the product directly, being responsible for the end-to-end implementation.

If different people need to cooperate to implement the scenario, providing a description of the business rules can help in the communication throughout the entire project implementation. This is particularly important when the solution designer and the implementer belong to different companies or, when they are geographically separated. Figure 5 shows the final version of the conceptual model.

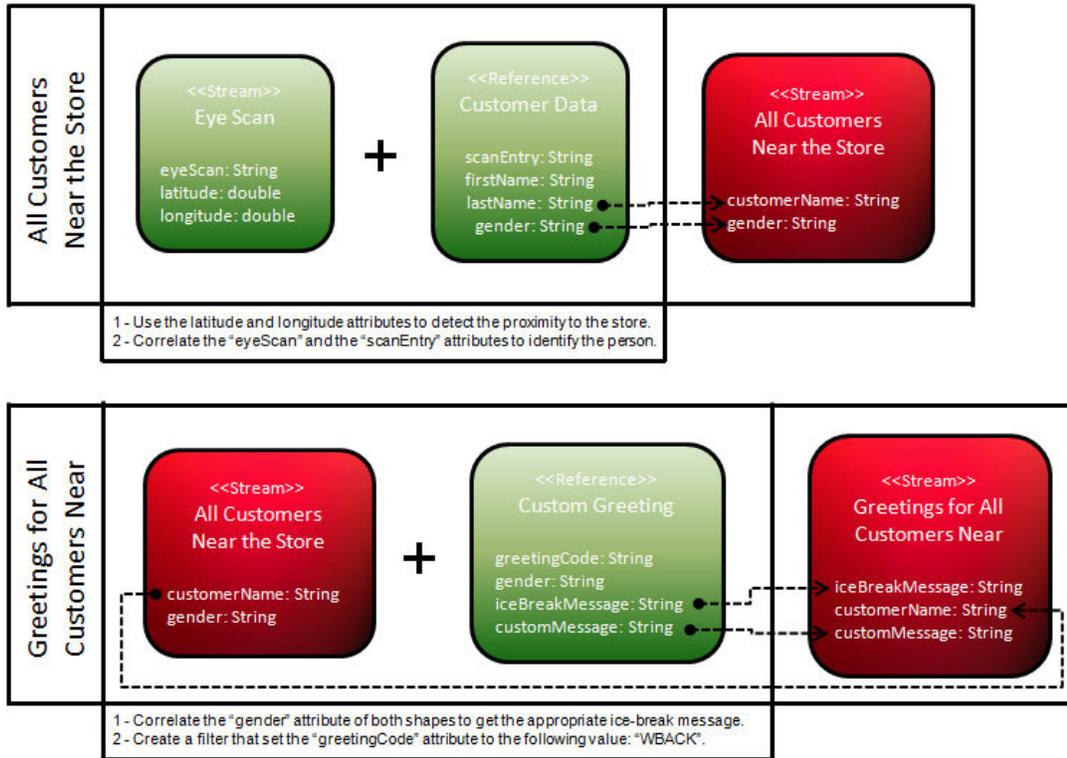


Figure 5. Final version of the conceptual model with the description of the business rules.

The conceptual model detailed in figure 5 provides a solid foundation for the implementation of the case study. In most cases, the creation of the conceptual model will be elaborated mentally during the implementation of the business scenarios. The steps shown in this section are not intended to be used as a solution design methodology. Instead, they show how a typical event processing application should be designed and what kinds of concerns are commonly found in this type of solution.

Part Two: Implementing the Artifacts in Oracle Stream Explorer

Before starting the implementation of the artifacts in Oracle Stream Explorer, it is necessary to make sure that some prerequisites are satisfied. First, a running database with DDL and DML permissions needs to be available. The out-of-the-box supported databases are Oracle, SQL Server 2005 and Derby. Other JDBC compliant databases can be used, but deployment of their JDBC drivers may be necessary. Secondly, an SQL script needs to be executed to create and populate the two database tables that will be used during the implementation. This SQL script can be found in the Appendix A of this paper. Finally, a CSV file will be used during the implementation of the eye scan stream artifact. Sample data from the content of this file, just like the URL to download its complete version can also be found in the Appendix A.

Considering that two sources that act as references to a database table will need to be created in Oracle Stream Explorer, a database connection pool needs to be set up. In order to be able to set up a database connection pool,

administrative access to the Oracle Event Processing Visualizer will be necessary. With the Oracle Stream Explorer up and running, open a web browser and access the following URL: <http://host:port/wlevs>. In this URL, host is the IP address or hostname where the server is running and port is the HTTP-enabled port configured for external access. Figure 6 shows the login screen of the Oracle Event Processing Visualizer.

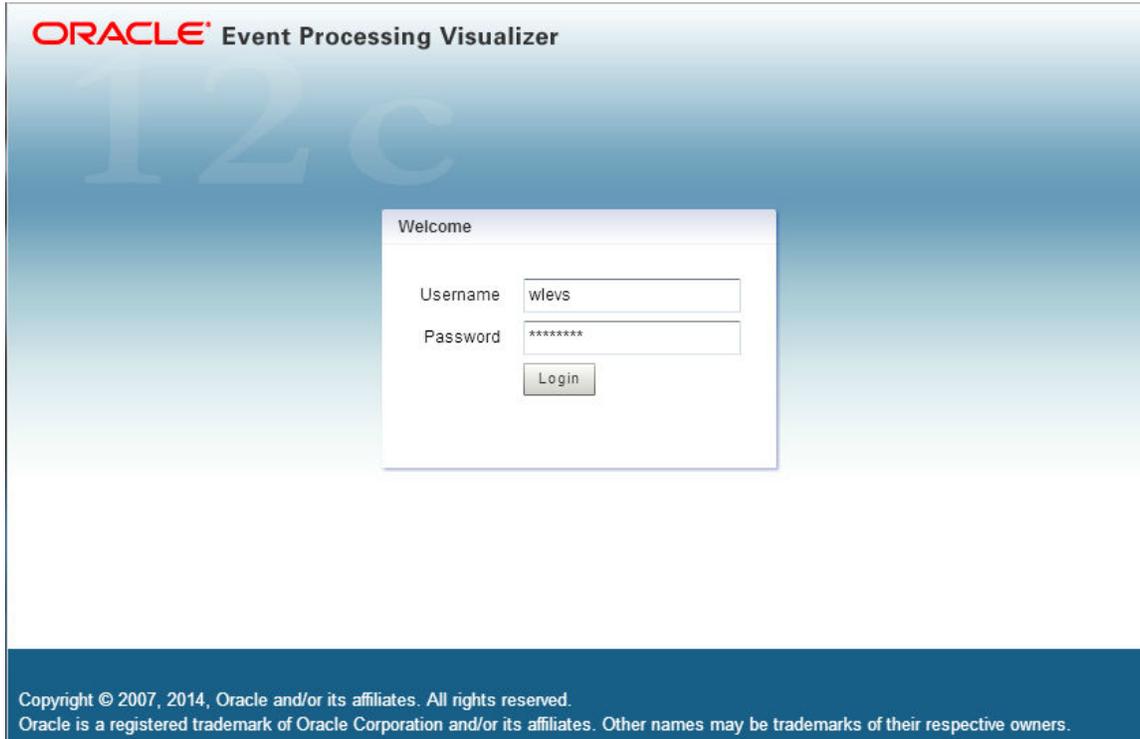


Figure 6. Oracle Event Processing Visualizer login screen.

Appropriate credentials will be needed to access this console. If you do not have these credentials, ask your Oracle Stream Explorer administrator to create them for you. Once logged in to the console, access the server instance where Oracle Stream Explorer is running. This will open the window where all the server details will be available via tabs. To start the creation of the database connection pool, click in the [DataSource](#) tab as shown in figure 7.

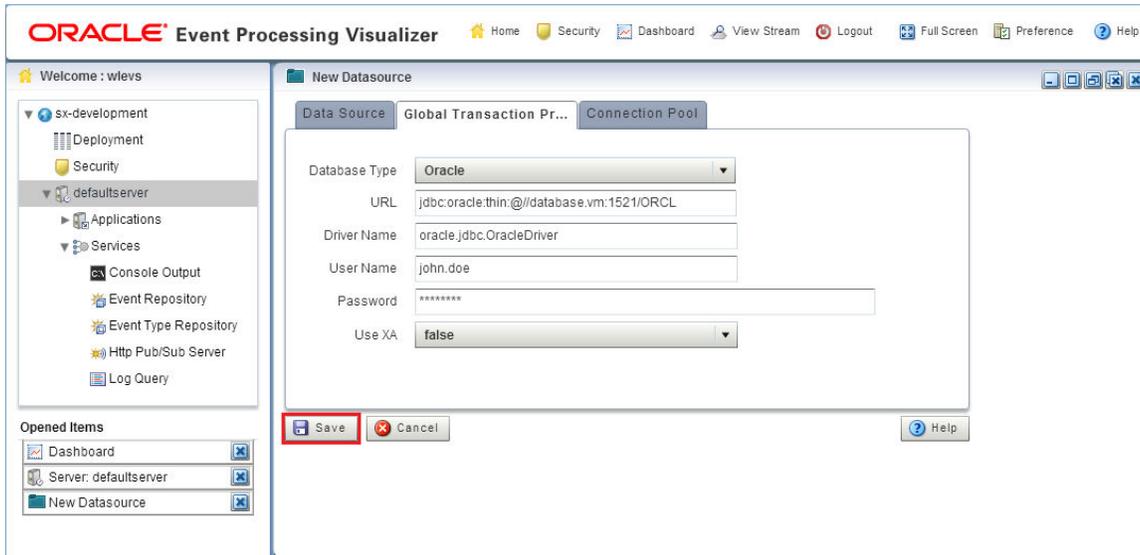


Figure 9. Entering the JDBC connection details of the database.

Enter the requested JDBC connections details of the database that contains the two tables needed for this implementation. Regardless of which type of database is being used, set the value of the Use XA field to false. Once the configuration of the JDBC connection details of your database is finished, click in the Save button to finish the creation of the database connection pool. Figure 9 shows an example of this configuration.

With the database connection pool properly created, the development of the artifacts can start. Access the Oracle Stream Explorer application using the following URL: <http://host:port/sx>. Those are the same host and port used to access the Oracle Event Processing Visualizer, and the same credentials that were used before should work here. Figure 10 shows the login screen of the Oracle Stream Explorer.

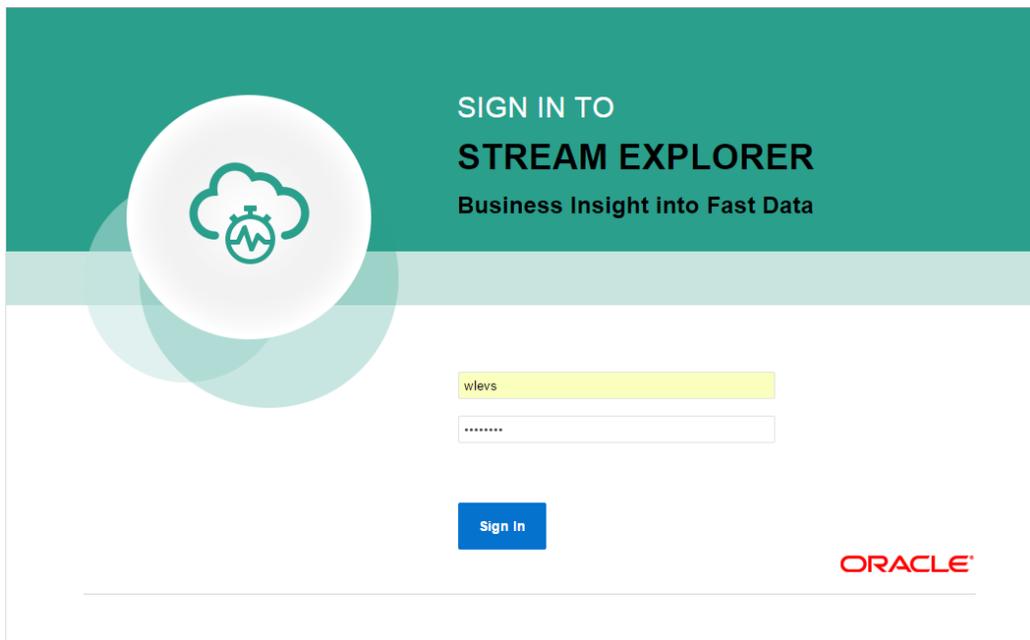


Figure 10. Login screen of the Oracle Stream Explorer.

After getting authenticated to Oracle Stream Explorer, the home screen will be displayed. The home screen provides a welcome message and displays a set of fancy images, each one inside a square that represents the common business domains where event processing applications can be used. Clicking into one of these images sets the tag of the business domain and also drives you directly to the catalog screen. The catalog screen can also be accessed without setting a business domain tag, via the [Catalog](#) button, found on the top right corner of the home screen.

The catalog screen is where artifacts in Oracle Stream Explorer are created and maintained. It displays all artifacts previously created in the main table on the center of the screen, allowing the filtering of specific artifacts such as explorations, streams and references. To create artifacts, you must request the creation of a new one in the [Create New Item](#) combo box, just as shown in figure 11.

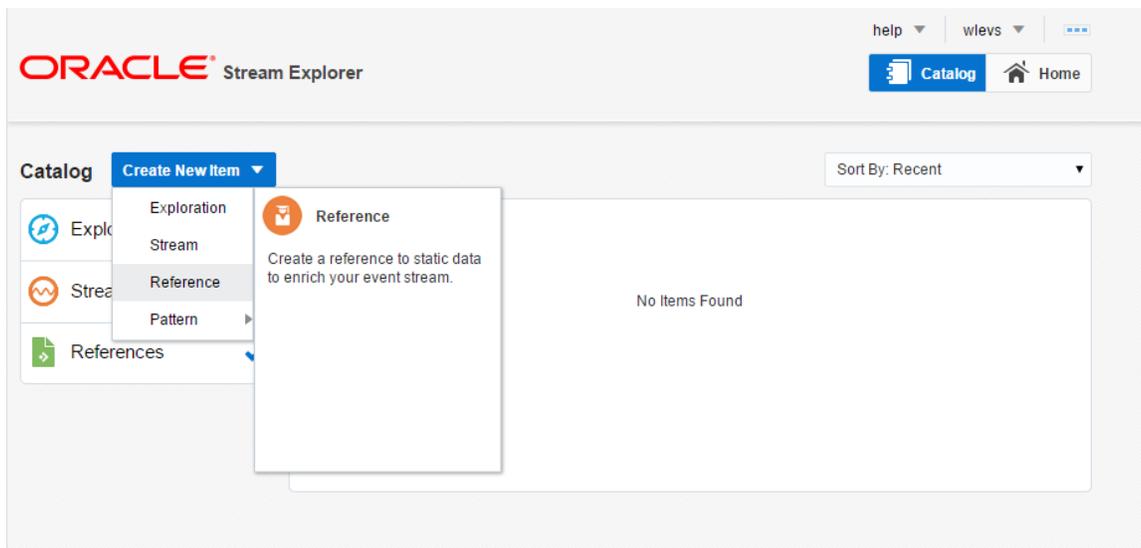


Figure 11. Requesting the creation of new artifacts in Oracle Stream Explorer.

The first artifact that will be created is the customer reference. Request the creation of a new reference as shown in figure 11, selecting the option "Reference" in the list. The new reference creation wizard will then start. In the [Name](#) field enter the value "CustomerData". Provide the tag "customer" in the [Tags](#) field and from the [Source Type](#) combo box, choose the option "Database Table". Figure 12 shows the first step of this wizard.

Create Reference [X]

◀ Back Source Details Type Properties Shape Next ▶

* Name: CustomerData

Description: [Empty text area]

Tags: customer ✖ Enter tag

* Source Type: Database Table ▼

Cancel Create

Figure 12. First step of the new reference creation wizard.

Click in the Next button to proceed to the second step of the wizard. The wizard will ask for the name of the database connection pool that will be used to connect to the database server. In the Data Source Name combo box, choose the option "jdbc/minorityReport" as shown in figure 13. Click in the Next button to proceed to the third step.

Create Reference [X]

◀ Back Source Details Type Properties Shape Next ▶

Type: Database Table

* Data source name: Select Data source name ▼

Select Data source name

jdbc/minorityReport

Cancel Create

Figure 13. Second step of the new reference creation wizard.

In the third and last step, the wizard will ask for the name of the database table that the reference should point to. In the Select Shape combo box, choose the option "CUSTOMER_DATA" as shown in figure 14. Click in the Create button to finish the new reference creation wizard.

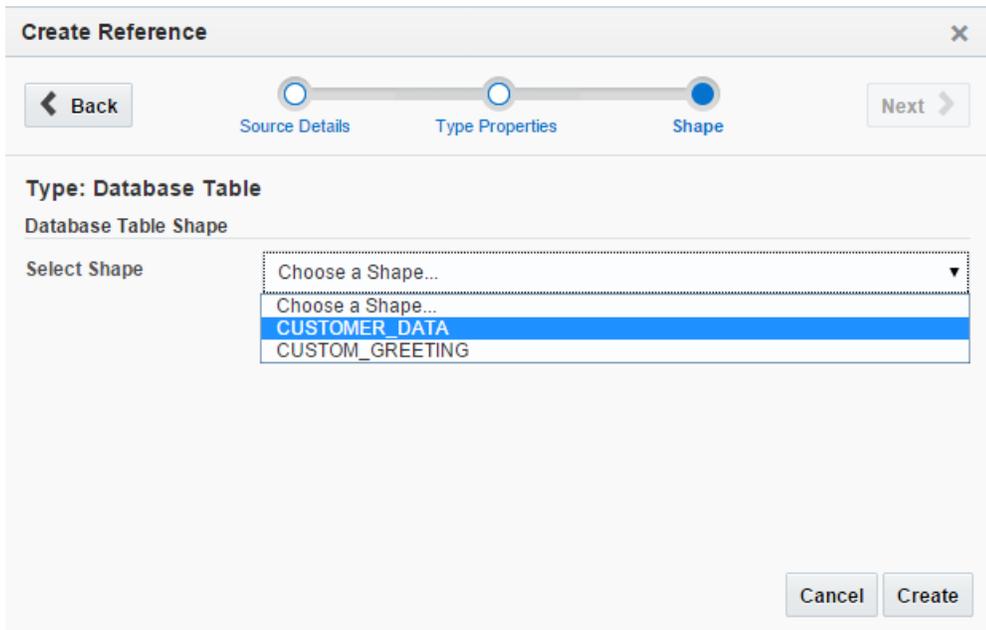


Figure 14. Third step of the new reference creation wizard.

After the completion of the wizard, a new shape named "CustomerData" should be listed in the table in the center of the screen. Now, the same steps must be followed to create the second reference. During the creation of the second reference, set the Name field to "CustomGreeting" and the shape selected must be the "CUSTOM_GREETING" database table. Figure 15 shows what should be exhibited in the catalog screen after the creation of these two references.

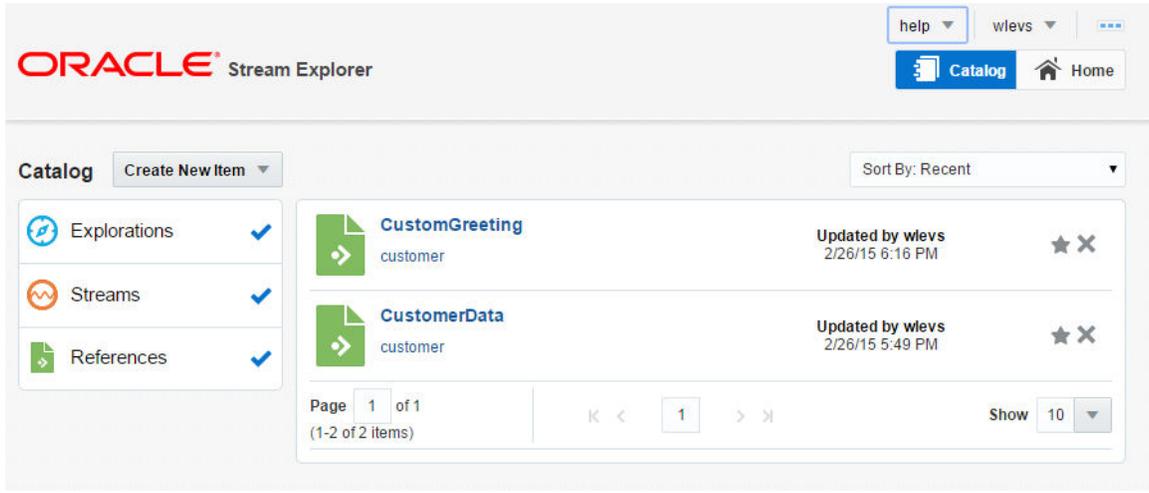


Figure 15. Oracle Stream Explorer showing the two created references.

Now it is necessary the creation of a third source, but this time it will be a stream instead of a reference. This stream represents the incoming flow of eye scans containing the person's retina reading and its location. Request the creation of a new stream as shown in figure 11, selecting the option "Stream" in the list. The new stream creation wizard will then start.

In the Name field enter the value "EyeScanStream". Provide the tag "customer" in the Tags field and from the Source Type combo box, select the option "CSV File". Leave the Create Exploration with this Source checkbox unchecked, since all the explorations are going to be created in the sequence. Click in the Next button to proceed to the second step of the wizard. Since the selected source type was set to CSV, the wizard will then ask for the source path of the CSV file. Click in the Upload File button to select and upload the CSV file to Oracle Stream Explorer.

Click the Next button to proceed to the third step. In the Name field enter the value "EyeScanStream" again. During the file upload process, the wizard parses the CSV file trying to detect the attribute names and their data types, in order to be able to present them in the third step of the wizard. For this reason, make sure if the Manual Mapping field is selected, and if the attributes shown matches with what is shown in figure 16. Click in the Create button to finish the new stream creation wizard.

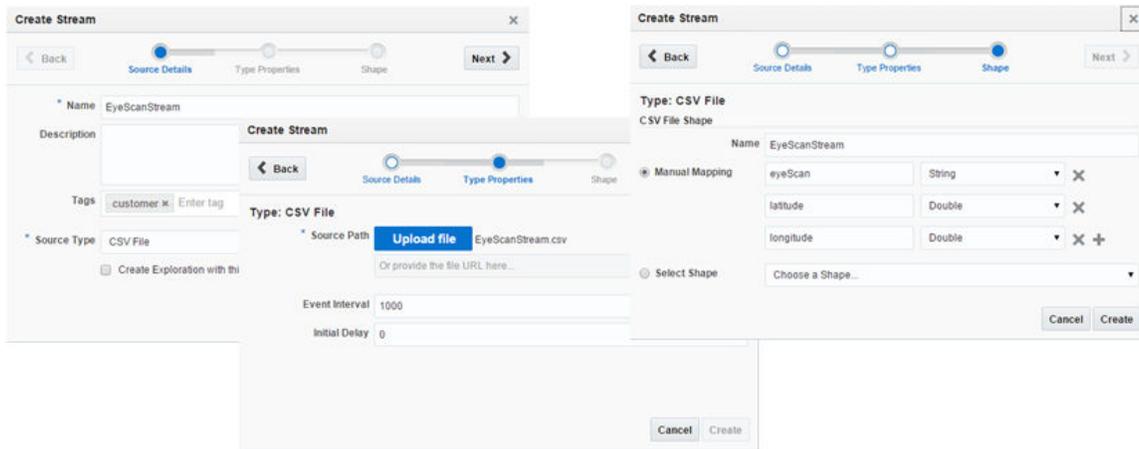


Figure 16. All the three steps from the new stream creation wizard.

All sources needed to start the implementation of the case study are now created. Now it is time to start the creation of the explorations, the artifacts that will provide the expected behavior. The first exploration to be built is the one that detects when some person is walking near the store, also identifying them against the customer reference. This exploration will be named "All Customers Near the Store". Request the creation of a new exploration as shown in figure 11, selecting the option "Exploration" in the list. The new exploration creation wizard will then start.

In the Name field enter the value "All Customers Near the Store". Provide the tag "customer" in the Tags field and from the Source combo box, choose the option "EyeScanStream", just as shown in figure 17. Click in the Create button to finish the new exploration creation wizard. The exploration editor will then open with the newly created exploration as shown in figure 18.

Create Exploration ✕

*** Name**

Description

Tags

*** Source**

EyeScanStream

Figure 17. Setting the source in the new exploration creation wizard.

The first thing noticed in the exploration editor is that the Live Output Stream and the Charts sections shows in real-time the results coming from the sources, which in this case is the eye scan stream. This is interesting because as the exploration is changed, these sections are also updated, in real-time, helping the user to have a better view of what the output result looks like. Figure 18 shows the newly created exploration presenting in real-time the data coming from the sources.

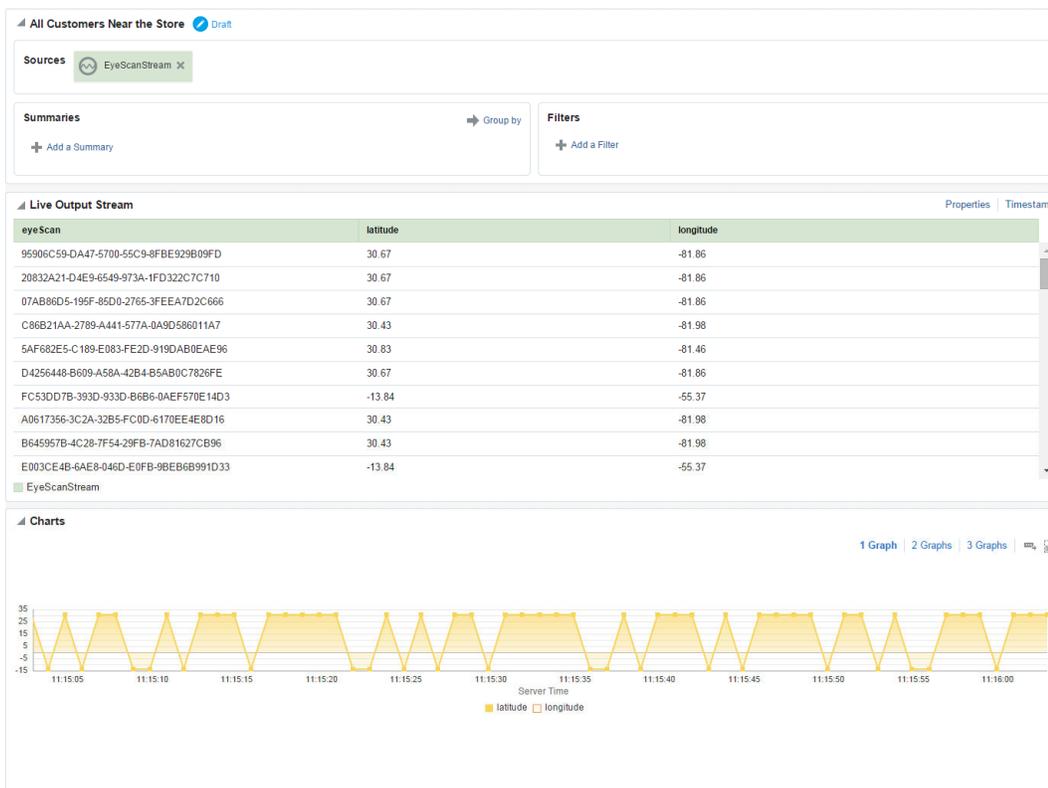


Figure 18. Newly created exploration presenting in real-time the data coming from the sources.

Since the exploration needs to detect only persons that are near the store, filters to pick up only the ones of interest need to be created. In the [Filters](#) section, click in the [Add a Filter](#) link. A new filter entry will be created where you can choose which attribute to handle, which operator to use and set which value restricts the output result that you are looking for. For this exploration, the filters will guarantee that only persons in which its location has the latitude attribute greater than or equals to "20.00" and the longitude attribute is less than or equals to "-60.00". Figure 19 shows these two filters being created in Oracle Stream Explorer.

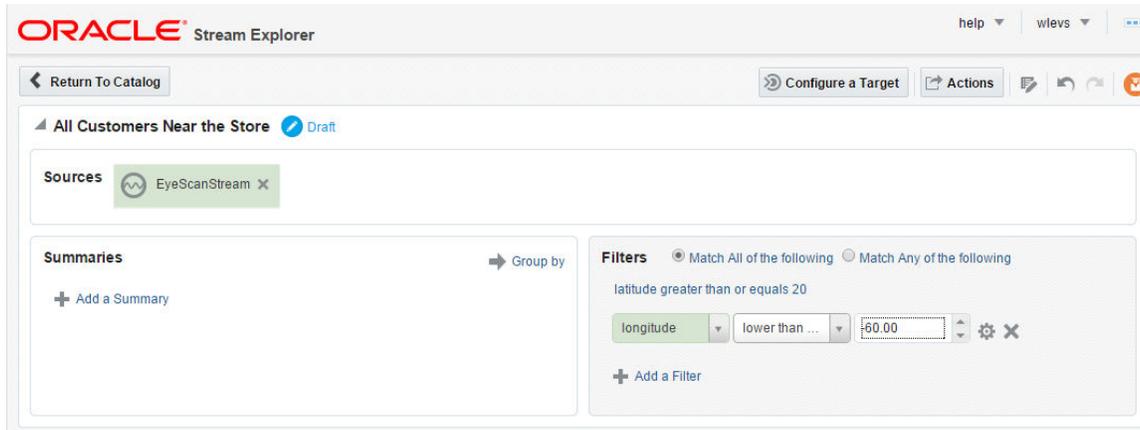


Figure 19. Filtering the person's location in Oracle Stream Explorer.

After setting these filters, the [Live Output Stream](#) section will automatically discard any event that does not satisfies the criteria, and the [Charts](#) section will then start to show a more flat graph, since the live output stream now presents only approximate values for the latitude and longitude attributes.

In order to identify who each person is, it is necessary that every event from the eye scan stream be correlated with the customer reference that holds the details of the customer. The first step is to change the [Sources](#) section of the exploration to include another shape. Click on the [Sources](#) section and select the "CustomerData" reference from the drop down list. When two or more shapes are used as sources in an exploration, a new section called [Correlations](#) starts to appear in the exploration editor.

Click in the [Add a Correlation](#) link to create a new correlation between the sources. From the left side choose the "eyeScan" attribute and from the right side choose the "scan_entry" attribute. Once the correlation is configured, the [Live Output Stream](#) section will show attributes from both sources, as shown in figure 20.

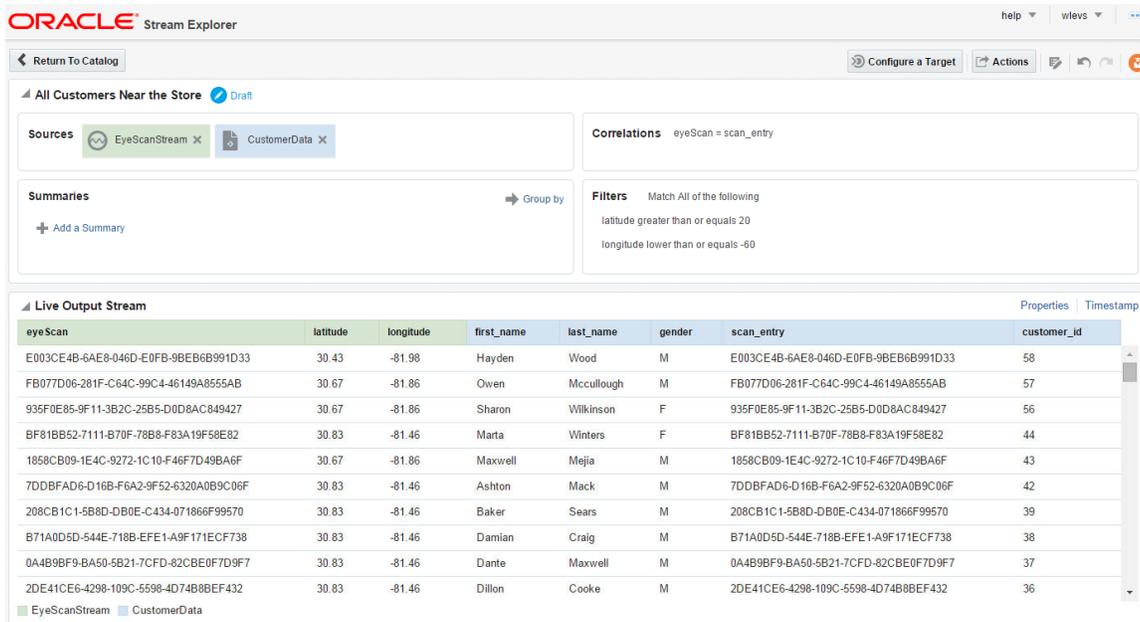


Figure 20. Result of the event correlation between the eye scan stream and the customer reference.

In order to be considered complete, explorations need to provide the expected output result since once published, they may be used in the list of sources of other explorations. Oracle Stream Explorer allows the modification of the output result via the [Properties](#) link. Access the [Properties](#) link and leave selected only the attributes "last_name" and "gender", also re-ordering them to be the first and second attributes of the properties section. After this, double-click the "last_name" attribute to provide a custom display name for it, setting its value to "customerName", as shown in figure 21.

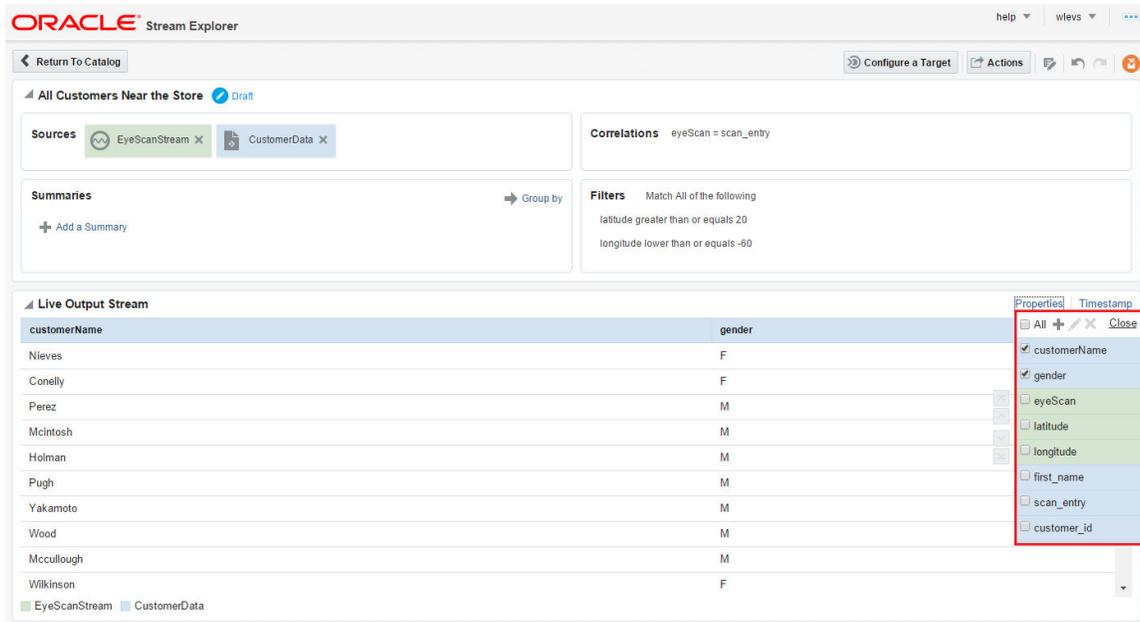


Figure 21. Customizing the output result of an exploration before publishing it.

The exploration is now ready to be published. In the top right corner of the exploration editor there is a button labeled [Actions](#) that once clicked opens a suspended menu with other buttons, each one of them related to a possible action that can be performed in the exploration. Click in the [Publish](#) button just as shown in figure 22.

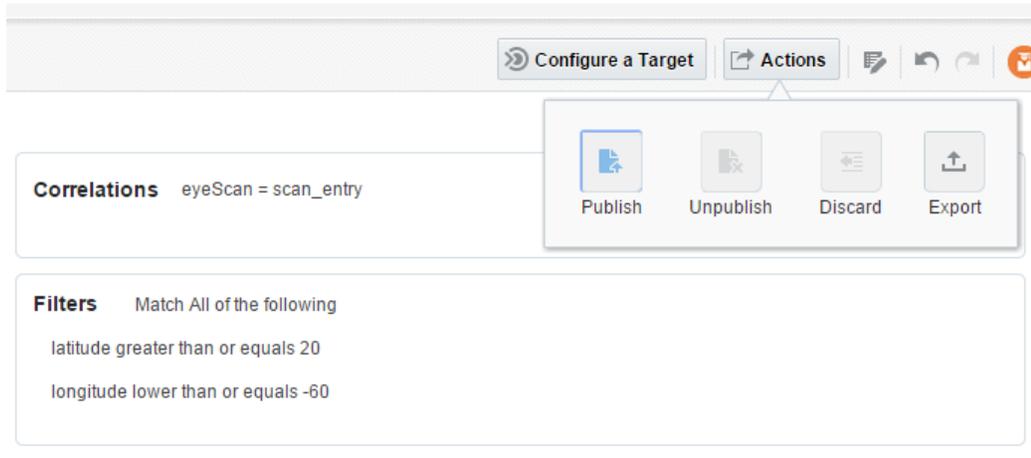


Figure 22. Publishing the exploration to be available as a stream.

With the first exploration properly published, the second exploration can start to be built. The second exploration will create greetings for each customer near the store, using the person gender to detect which appropriate greeting to use. This exploration will be named "Greetings for All Customers Near". Request the creation of a new exploration as shown in figure 11, selecting the option "Exploration" in the list. The new exploration creation wizard will then start.

In the [Name](#) field enter the value "Greetings for All Customers Near". Provide the tag "customer" in the [Tags](#) field and from the [Source](#) combo box, choose the option "All Customers Near the Store". Click the [Create](#) button to finish the new exploration creation wizard. The exploration editor will then open with the newly created exploration, showing in the [Live Output Stream](#) section the output result from the first exploration. Click in the [Sources](#) section and select the "CustomGreeting" reference from the drop down list. Once the [Correlations](#) section starts to appear in the exploration editor, click on the [Add a Correlation](#) link to create a new correlation between the sources. From the left side choose the "gender" attribute and from the right side choose the "gender" attribute too. Once the correlation is configured, the [Live Output Stream](#) section will show attributes from both sources, as shown in figure 23.

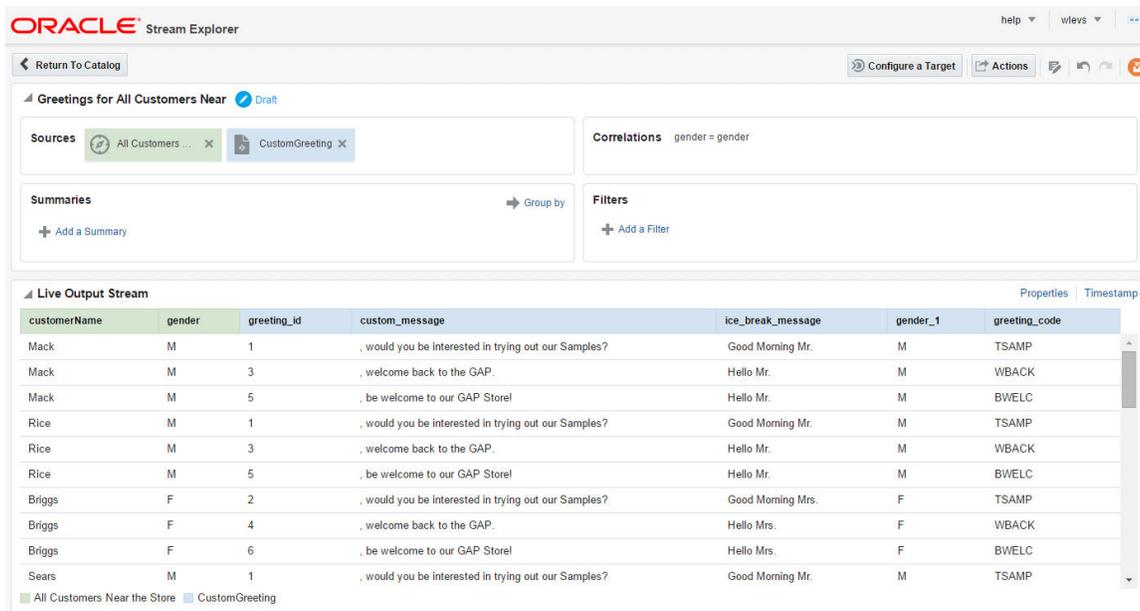


Figure 23. Second exploration with the correlation already in place.

To avoid unnecessary greetings for the customers, a filter needs to be created to establish that only the "Welcome back to the GAP" greeting is used. In the [Filters](#) section, click in the [Add a Filter](#) link to create a new filter entry, setting the value of the "greeting_code" attribute to be equals to "WBACK".

Just like it was done with the first exploration, the output result of this second exploration needs to be customized. Access the [Properties](#) link and leave selected only the attributes "ice_break_message", "customerName" and the "custom_message", also re-ordering them to be the first, second and the third attributes of the properties section. Still in the [Properties](#) link, provide custom display names for each one of these three attributes, setting the first attribute to "iceBreakMessage", the second attribute to "customerName" and the third attribute to "customMessage", just as shown in figure 24.

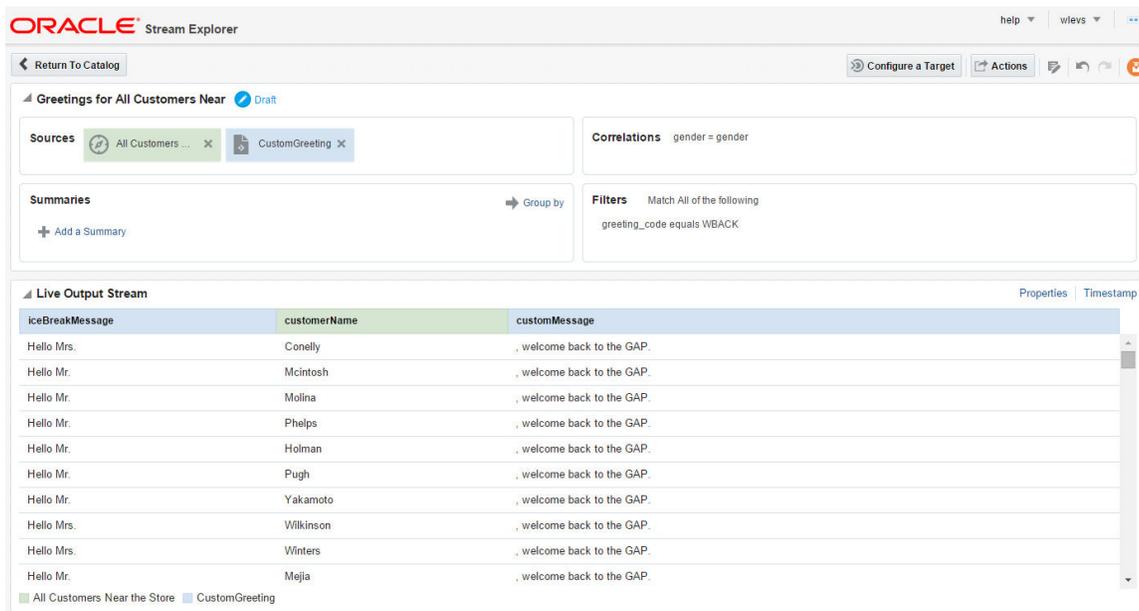


Figure 24. Second exploration with its output result properly customized.

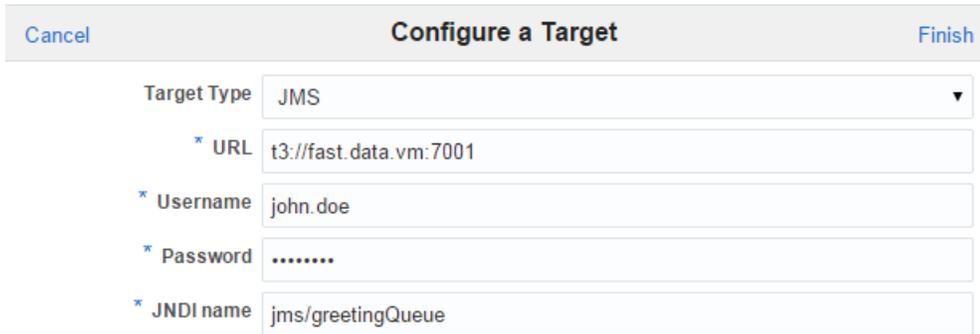
Publish this second exploration as shown in figure 22. The case study is now completely implemented, with Oracle Stream Explorer detecting in real-time when someone is near the store and greeting them with a custom message. By using the person's gender to differentiate if it is a man or a woman, the automated approach also uses English salutations such as "Mr." and "Mrs." before saying their names.

However, while it is interesting to have the output result correctly shown in the Oracle Stream Explorer exploration editor, this can be considered useless if this output result could not be sent to an external system that handles the greeting notification. Fortunately, Oracle Stream Explorer allows that output results from explorations to be sent to external systems via the [Configure a Target](#) button. The following targets are currently supported in Oracle Stream Explorer:

- » **CSV File:** Write the output results to a CSV file, allowing the contents to be appended.
- » **HTTP Publisher:** Publishes the output results to an outbound HTTP channel from OEP.
- » **Event-Driven Network:** Send the output results to SOA Suite EDN-enabled subscriber.
- » **Java Message Service:** Creates a `javax.jms.MapMessage` and send it to a JMS destination.
- » **REST Endpoints:** Performs a HTTP POST with the output results into a REST endpoint.

It is important to note that regardless of the chosen target type, the generated output results are sent continuously to the configured target. This means that once a new output result is available in Oracle Stream Explorer, it is sent immediately without any delays. This approach allows that actions can be taken in the exact moment in which the situation became relevant, allowing a truly event-driven approach.

For instance, imagine that a voice-enabled system implemented in Java listen greeting requests via JMS, using a MDB (Message-Driven Bean) to consume messages from the JMS destination and execute the greetings speech via the [Java Speech API](#). The output results from the exploration could be easily sent via JMS by the configuration of a target of this type, as shown in figure 25.



Configure a Target	
Target Type	JMS
* URL	t3://fast.data.vm:7001
* Username	john.doe
* Password
* JNDI name	jms/greetingQueue

Figure 25. Configuration of a JMS-based target to send the output results.

The end-to-end implementation of a voice-enabled system using Java is definitely out of the scope of this paper, which focuses on explaining the fundamentals of Oracle Stream Explorer and how to build event processing applications. However, the Appendix B of this paper contains an implementation of a MDB that leverages the Java Speech API to synthesize the greeting using a human voice.

Conclusion

Can you imagine yourself driving your car by only looking in the rear-mirror? This is how many enterprises run their business today, trying to get insight using traditional data warehouse and business intelligence technologies. Looking at past information does not always provide the ability to react appropriately; this in turn leads to missing important opportunities or letting threats take advantage of our lack of awareness.

Oracle Stream Explorer is a web-based application that leverages the capabilities found in Oracle Event Processing to provide a tool for business users to analyze streams of events in real-time, empowering them to gain insight and take appropriate actions when needed. This paper explained the fundamentals of Oracle Stream Explorer and how to develop event processing-based applications through a step-by-step implementation of a case study.

Appendix A: Scripts and Samples used in the Case Study

In order to be able to implement the case study presented in this paper, two database tables need to be created. The script shown in the listing 1 creates these two database tables and also populates it with sample data. The script was successfully tested against the Oracle database, but it may work well in other databases. If necessary, adjustments in the script can be performed, as long as the structure of the tables and the column names remains unchanged.

```

-----
-- DDL for the table CUSTOMER_DATA
-----

CREATE TABLE "CUSTOMER_DATA"
(
  "CUSTOMER_ID" INTEGER NOT NULL PRIMARY KEY,
  "SCAN_ENTRY" VARCHAR2(255) NOT NULL,
  "FIRST_NAME" VARCHAR2(20) NOT NULL,
  "LAST_NAME" VARCHAR2(20) NOT NULL,
  "GENDER" VARCHAR2(1) NOT NULL
);

```

```
-----
-- DDL for the table CUSTOM_GREETING
-----
```

```
CREATE TABLE "CUSTOM_GREETING"
(
  "GREETING_ID" INTEGER NOT NULL PRIMARY KEY,
  "GREETING_CODE" VARCHAR2(5) NOT NULL,
  "GENDER" VARCHAR2(1) NOT NULL,
  "ICE_BREAK_MESSAGE" VARCHAR2(50) NOT NULL,
  "CUSTOM_MESSAGE" VARCHAR2(100) NOT NULL
);
```

```
-----
-- Loading data into the table CUSTOMER_DATA
-----
```

```
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(1,'CB281A82-EBF9-247F-8D4C-632F2CD4DC9E','Gregory','Berger','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(2,'B7C841D6-DC7D-0C32-1402-58E0E06F0C74','George','Griffith','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(3,'0028CF68-2264-D591-C3F8-EECF78E3635F','Damian','Key','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(4,'B5C9DA94-6AF7-9BFF-19F5-9EECC4797417','Scott','Bridges','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(5,'1EB943F4-C7B3-29D8-D0D3-042507CCDD50','Colton','Kinney','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(6,'B71E107F-DD89-76EC-497B-9F3BB81CC790','Brody','Goodman','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(7,'AF25D32D-0870-7C79-5ECE-A8D88E63A099','Simon','Park','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(8,'EA712817-9B59-042A-F952-4BD9B2621FE7','Addison','Medina','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(9,'27AFC1DC-74D6-D988-9E7E-35522FD65CAC','Charles','Sutton','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(10,'45D8A1B8-72DB-A6EE-7725-4AA8F379FA2E','Vaughan','Lawson','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(11,'3FAB4B01-AA8C-CA14-03B7-69F75670D0C2','Felix','Haley','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(12,'F6C6D0BC-5772-97BA-2EB3-E0A6DE91D64B','Mason','Mclean','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(13,'096458CC-3F58-0CFC-08B9-08E2134AFCB7','Brent','Faulkner','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(14,'95005563-5DFB-2F6D-5B12-1BB0DEE71492','Kasimir','Fitzgerald','F');
```

```

Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(15,'F812B9FD-0EC3-30FE-3D3D-37ECB7A5B012','Zahir','Savage','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(16,'5F8F72F2-48DF-2507-6D34-AA2E6FD663B8','Jin','Wilcox','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(17,'C54C5779-7BB8-4B02-D78E-153879C9876A','Chadwick','Snyder','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(18,'D1EB489E-3FA6-C4CC-F923-CFB6005447F6','Chaim','Moses','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(19,'50035120-B48A-4904-5498-591F6F15A1FD','Lester','Fitzgerald','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(20,'F35FB6B7-6B68-4E9E-5D1F-7EE852A11F8A','Otto','Dixon','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(21,'DBA752B4-5ABE-BEB5-360C-572E52F02B4B','Reese','Gross','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(22,'23E5896B-9F6C-1E18-E671-3BABD8E41E05','Griffith','Fields','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(23,'480ED76A-D8DC-3C45-1304-EC7EECD18B25','Erasmus','Cobb','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(24,'FF7A36A3-C8B4-4305-9911-64F61FEC0CA7','Cooper','Barrett','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(25,'E083DDBE-282B-2CCF-7D8A-0C1D77D93E5A','Stephen','Rivas','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(26,'B7E4A6AD-6331-09F1-43D8-CA849A2FD796','Steven','Kramer','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(27,'3BB44589-FD5E-8CAB-5E76-7640897ACCBE','Kirsten','Malone','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(28,'D929DC13-0616-A3EC-0191-8CE9B725F37F','Perry','Bender','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(29,'B441636D-CDE4-A840-E848-B650635129C3','Alfonso','Velez','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(30,'B8D4A245-67F0-9386-7AAE-11F0E30C582C','Brody','Craft','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(31,'BFD9B89D-08E3-6DB9-8F76-A9499A2B50D8','Trevor','Hinton','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(32,'FC1E3D87-3955-7F7C-F865-EDD637D0558C','Nasim','Allison','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(33,'60473650-8AB0-12F4-9C91-84B9E3B86DE7','Ali','Solomon','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(34,'F44E0BF6-2F46-29D4-D9BC-4D35009C0D3B','Kennan','Schneider','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(35,'77F7D210-C655-21DA-B8EC-869ED54F820D','Andrea','Santos','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(36,'2DE41CE6-4298-109C-5598-4D74B8BEF432','Dillon','Cooke','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(37,'0A4B9BF9-BA50-5B21-7CFD-82CBE0F7D9F7','Dante','Maxwell','M');

```

```

Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(38,'B71A0D5D-544E-718B-EFE1-A9F171ECF738','Damian','Craig','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(39,'208CB1C1-5B8D-DB0E-C434-071866F99570','Baker','Sears','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(40,'8873F976-89FC-24A3-71B5-A1AB6FE9D30E','Kelly','Briggs','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(41,'3E4F0B65-7445-1CD8-8A31-B5BAEED5CFA7','Eric','Rice','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(42,'7DDBFAD6-D16B-F6A2-9F52-6320A0B9C06F','Ashton','Mack','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(43,'1858CB09-1E4C-9272-1C10-F46F7D49BA6F','Maxwell','Mejia','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(44,'BF81BB52-7111-B70F-78B8-F83A19F58E82','Marta','Winters','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(45,'CA2582D4-40A0-C718-0F34-12AD26AAEB3B','Harlan','Wilcox','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(46,'0D333238-D10C-8FF9-B2EB-BCC1B1876767','Drew','Lynch','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(47,'656D6A68-7C70-02BC-32A8-240E5B48332D','Felix','Cash','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(48,'7158556F-CF03-05EA-D43F-AF2B2DA28BEE','Clayton','Chambers','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(49,'EA620BA5-CD44-2487-87EA-9CD352172BCC','Keaton','Woodward','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(50,'DD5FF2F5-ADE0-18F6-1E86-5E98C039D9B5','Ricardo','Ferreira','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(51,'7DBF9ABB-4979-6BE6-2BBE-A782512D6AF5','Oliver','Hurst','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(52,'F83662D2-BC5A-1E00-4910-53AE7FFB85C4','Channing','Hubbard','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(53,'4BA48031-B489-21BE-3EAD-C2E245AEDC21','Peter','Frazier','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(54,'CBD8BCD1-0385-C7F0-6D94-DAF8674087FA','Kieran','Farley','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(55,'617E92EB-DDA0-3200-C13C-CBAC5523526C','Monica','Atkins','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(56,'935F0E85-9F11-3B2C-25B5-D0D8AC849427','Sharon','Wilkinson','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(57,'FB077D06-281F-C64C-99C4-46149A8555AB','Owen','McCullough','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(58,'E003CE4B-6AE8-046D-E0FB-9BEB6B991D33','Hayden','Wood','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(59,'B645957B-4C28-7F54-29FB-7AD81627CB96','Mitsuko','Yakamoto','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(60,'A0617356-3C2A-32B5-FC0D-6170EE4E8D16','Garrison','Pugh','M');

```

```

Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(61,'FC53DD7B-393D-933D-B6B6-0AEF570E14D3','Salvador','Holman','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(62,'D4256448-B609-A58A-42B4-B5AB0C7826FE','Tyrone','Phelps','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(63,'5AF682E5-C189-E083-FE2D-919DAB0EAE96','Ryan','Molina','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(64,'C86B21AA-2789-A441-577A-0A9D586011A7','Paul','Mcintosh','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(65,'07AB86D5-195F-85D0-2765-3FEEA7D2C666','Ralph','Perez','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(66,'20832A21-D4E9-6549-973A-1FD322C7C710','Jennifer','Conelly','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(67,'EC4EFB21-AFC6-A079-4FAE-9B6BE9EA62AC','Raymond','Gould','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(68,'12A4FECC-3BA8-1D1C-2B35-BB478BDAD662','George','Morse','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(69,'BF2D39C3-6030-6938-EEEE-4863600E7519','Matthew','Cole','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(70,'13E4712C-463F-BAA7-8537-A657BEB01D28','Blake','Benson','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(71,'830373DE-038B-95C5-467C-2573DFE51586','Debbie','Howell','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(72,'0D562E79-16B8-B81D-6144-913CA61DA166','Nataly','Shaffer','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(73,'BE643FCC-5F3B-E220-3670-FA99F7A1E507','Chandler','Dillon','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(74,'FD4483E9-928B-91C9-BDB1-A1A62FE2CA24','Hamilton','Rodriquez','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(75,'18555BC9-0B99-2F86-8FBB-9ED4B5FBF1FC','Jeff','Mcdaniel','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(76,'C32D18A2-85AA-F3A3-5FD1-8520D7892D40','Alfonso','Salazar','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(77,'EAB26AAA-DD16-0B56-4A6B-7BA332BCFB52','Calvin','Underwood','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(78,'375FC126-B176-AC8D-D31D-A589AA9B40EC','Maria','Nieves','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(79,'45BB2865-CAA5-29E4-3E9B-BE4010A3F9E2','Jonas','Sawyer','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(80,'B4D9B9DE-5912-CE87-235C-6CB165AD0FA0','Devin','Harper','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(81,'B1EBF0B6-F7C1-3DC4-3214-35E7F5238C47','Aquila','Hatfield','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(82,'5B69FE1B-2B9E-3919-F988-10E4298CF4F4','Derek','Richard','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(83,'FCFC70E9-6334-8B6C-D538-7161AFA80739','Chaney','Pratt','M');

```

```

Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(84,'471AF640-42B5-2D28-F057-9F3CFE2072A2','Malik','Beard','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(85,'332BF106-9B6B-C30C-1426-349909637024','Nathaniel','Cox','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(86,'6345109E-26FD-198B-F11F-410C93E52651','Elaine','Vargas','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(87,'8B111703-04FD-D5B6-1F9B-A1C5A5077241','Nissim','Macias','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(88,'1C7FBB69-266F-1F8D-902A-4F987115CABC','Hu','Stein','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(89,'95906C59-DA47-5700-55C9-8FBE929B09FD','Pamela','Hatfield','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(90,'BFBE734E-C80D-35C5-B441-139B7296712E','Cyrus','Gay','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(91,'B577A64A-C1ED-18C4-01CF-8F72EC65FA8C','Fletcher','Wiley','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(92,'FAFCE8D2-1500-8712-38A6-710BDE770A4A','Fitzgerald','Hughes','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(93,'4ED62A82-FDCE-445C-F12F-335745870917','Diane','Reese','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(94,'C89BD428-8B10-933B-CC58-5B01DDA827EA','Ingrid','Harrell','F');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(95,'1B5C67BF-33F1-9F7B-D94A-11560C15617C','Kenyon','Boyer','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(96,'1829958B-0CC7-0B93-B804-8F023188DC2F','Emerson','Mcgowan','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(97,'ECA02944-0A5F-023B-A292-223016BA2B3C','Berk','Simon','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(98,'4E8DEBDD-2E3E-B59E-D785-234767DC8522','Chancellor','Sears','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(99,'0519C7FC-3292-0B00-BB97-E0464F7D64ED','Grant','Williamson','M');
Insert into CUSTOMER_DATA (CUSTOMER_ID,SCAN_ENTRY,FIRST_NAME,LAST_NAME,GENDER) values
(100,'47CCF80B-46A0-66C8-BA1A-2DA5FB862AEE','Judah','Salazar','M');

```

```

-----
-- Loading data into the table CUSTOMER_DATA
-----

```

```

Insert into CUSTOM_GREETING
(GREETING_ID,GREETING_CODE,GENDER,ICE_BREAK_MESSAGE,CUSTOM_MESSAGE) values
(1,'TSAMP','M','Good Morning Mr. ',',', 'would you be interested in trying out our
Samples?');
Insert into CUSTOM_GREETING
(GREETING_ID,GREETING_CODE,GENDER,ICE_BREAK_MESSAGE,CUSTOM_MESSAGE) values

```

```

(2,'TSAMP','F','Good Morning Mrs. ',', would you be interested in trying out our
Samples?');
Insert into CUSTOM_GREETING
(GREETING_ID,GREETING_CODE,GENDER,ICE_BREAK_MESSAGE,CUSTOM_MESSAGE) values
(3,'WBACK','M','Hello Mr. ',', welcome back to the GAP. ');
Insert into CUSTOM_GREETING
(GREETING_ID,GREETING_CODE,GENDER,ICE_BREAK_MESSAGE,CUSTOM_MESSAGE) values
(4,'WBACK','F','Hello Mrs. ',', welcome back to the GAP. ');
Insert into CUSTOM_GREETING
(GREETING_ID,GREETING_CODE,GENDER,ICE_BREAK_MESSAGE,CUSTOM_MESSAGE) values
(5,'BWELC','M','Hello Mr. ',', be welcome to our GAP Store! ');
Insert into CUSTOM_GREETING
(GREETING_ID,GREETING_CODE,GENDER,ICE_BREAK_MESSAGE,CUSTOM_MESSAGE) values
(6,'BWELC','F','Hello Mrs. ',', be welcome to our GAP Store! ');

```

Listing 1. SQL script to create and populate the database tables.

For the implementation of the eye scan stream, a CSV file with sample data need to be used. The listing 2 shows an example of this file, containing only the first ten rows. The CSV file is structured having each row with three fields, the person's retina reading, the latitude and the longitude from its location.

```

eyeScan,latitude,longitude
C86B21AA-2789-A441-577A-0A9D586011A7,30.831086,-81.460571
CB281A82-EBF9-247F-8D4C-632F2CD4DC9E,30.425764,-81.975556
0A4B9BF9-BA50-5B21-7CFD-82CBE0F7D9F7,30.425764,-81.975556
E003CE4B-6AE8-046D-E0FB-9BEB6B991D33,-13.843414,-55.371094
B645957B-4C28-7F54-29FB-7AD81627CB96,-13.843414,-55.371094
B7C841D6-DC7D-0C32-1402-58E0E06F0C74,30.674122,-81.862946
0028CF68-2264-D591-C3F8-EECF78E3635F,30.674122,-81.862946
5AF682E5-C189-E083-FE2D-919DAB0EAE96,-13.843414,-55.371094
C86B21AA-2789-A441-577A-0A9D586011A7,-13.843414,-55.371094
1B5C67BF-33F1-9F7B-D94A-11560C15617C,30.674122,-81.862946

```

Listing 2. Example of the CSV file used as eye scan stream.

It is highly recommended that during the tests using Oracle Stream Explorer, the complete version of the file is used instead of the example shown in listing 2. The complete version of the CSV file can be downloaded in the following URL: <http://www.ateam-oracle.com/wp-content/uploads/2015/02/EyeScanStream.csv>.

Appendix B: Creating a Message-Driven Bean that Greets

Considering that Oracle Stream Explorer allows sending the output results from explorations to external systems, it would be interesting to leverage this capability to come up with solutions able to sense and respond in a contextual manner. For instance, the case study implemented in this paper shown that it is possible to create custom greetings to persons walking around the store. But what if the output results from the exploration were really transformed into a greeting action?

Thanks to the [Java Speech API](#), create voice-based systems capable of speaking from text messages became possible. Originally developed by Sun Microsystems and in collaboration with companies like Apple, AT&T, IBM and others, the first version of the Java Speech API specification was released on October 26, 1998. From the technical point of view the Java Speech API is not part of the JDK implementation, it must be acquired from third-party speech



vendors that provide their own implementations. One of the most popular implementations is the [FreeTTS](#), an open-source speech synthesizer written entirely in Java.

To demonstrate how the FreeTTS implementation can be used to create a voice-based system that listen greeting requests via JMS, listing 3 shows an example of a MDB that leverages the Java Speech API to synthesize the greeting using a human voice.

```
package com.oracle.fmw.ateam.fastdata;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJBException;
import javax.ejb.MessageDriven;
import javax.jms.MapMessage;
import javax.jms.Message;
import javax.jms.MessageListener;

import com.sun.speech.freetts.Voice;
import com.sun.speech.freetts.VoiceManager;

@MessageDriven(activationConfig = {
    @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(
        propertyName = "connectionFactoryJndiName", propertyValue = "jms/connFact"),
    @ActivationConfigProperty(
        propertyName = "destinationJndiName", propertyValue = "jms/greetingQueue"))

public class GreetingListener implements MessageListener {

    private Voice voice;

    @PostConstruct
    private void allocateVoice() {

        VoiceManager voiceManager = VoiceManager.getInstance();
        voice = voiceManager.getVoice("kevin16");
        voice.allocate();

    }

    @PreDestroy
    private void deallocateVoice() {

        if (voice != null) {
```



```
        voice.deallocate();
    }
}

@Override
public void onMessage(Message message) {

    if (message instanceof MapMessage) {

        MapMessage mapMessage = null;
        String iceBreakMessage = null;
        String customerName = null;
        String customMessage = null;
        String greeting = null;

        try {

            mapMessage = (MapMessage) message;

            iceBreakMessage = mapMessage.getString("iceBreakMessage");
            customerName = mapMessage.getString("customerName");
            customMessage = mapMessage.getString("customMessage");

            greeting = iceBreakMessage + customerName + customMessage;
            voice.speak(greeting);

        } catch (Exception ex) {

            throw new EJBException(ex);

        }

    }

}
```

Listing 3. Implementation of the MDB using the Java Speech API.

In the `onMessage()` method implementation shown in listing 3, the received message is transformed into a `javax.jms.MapMessage`, because this is the type of message that Oracle Stream Explorer sends the output results when the JMS-based target is used. Also, the attributes obtained from the message matches with the ones generated by the exploration, as shown in figure 24.



From the compilation perspective, the only library from the FreeTTS implementation that needs to be available in the classpath is the `$FREETTS/lib/freetts.jar` library. But from the deployment perspective, some other libraries from the FreeTTS implementation also need to be deployed, either together with the MDB (In case of packaging it using an EAR) or installed directly into the Java EE application server classpath. The libraries that need to be considered for deployment are:

- » `$FREETTS/lib/cmulex.jar`
- » `$FREETTS/lib/cmu_us_kal.jar`
- » `$FREETTS/lib/en_us.jar`
- » `$FREETTS/lib/freetts.jar`



Oracle Corporation, World Headquarters

500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries

Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

-  blogs.oracle.com/oracle
-  facebook.com/oracle
-  twitter.com/oracle
-  oracle.com

Hardware and Software, Engineered to Work Together

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.0115

Introduction to the Oracle Stream Explorer
March 2015
Author: Ricardo Ferreira
Reviewers: Peter Farkas, Prabhu Thukkaram