DigitalEdge™

Alerts API Guide

Version 1.3

September 2014

leidos

DISCLAIMER OF WARRANTY AND LIMITATION OF LIABILITY

The Software accompanying this Documentation is provided with the Limited Warranty contained in the License Agreement for that Software. Leidos, its affiliates and suppliers, disclaim all warranties that the Software will perform as expected or desired on any machine or in any environment. Leidos, its affiliates and suppliers, further disclaim any warranties that this Documentation is complete, accurate, or error-free. Both the Software and the Documentation are subject to updates or changes at Leidos' sole discretion. LEIDOS, ITS LICENSORS AND SUPPLIERS MAKE NO OTHER WARRANTIES, WRITTEN OR ORAL, EXPRESS OR IMPLIED RELATING TO THE PRODUCTS, SOFTWARE, AND DOCUMENTATION. LEIDOS, ITS LICENSORS AND SUPPLIERS DISCLAIM ALL IMPLIED WARRANTIES, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, USE, TITLE, AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. In no event shall Leidos, its affiliates or suppliers, be liable to the End User for any consequential, incidental, indirect, exemplary, punitive, or special damages (including lost profits, lost data, or cost of substitute goods or services) related to or arising out of the use of this Software and Documentation however caused and whether such damages are based in tort (including negligence), contract, or otherwise, and regardless of whether Leidos, its affiliates or suppliers, has been advised of the possibility of such damages in advance. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAWS, END USER ACKNOWLEDGES AND AGREES THAT LEIDOS AND ITS AFFILIATES AND SUPPLIERS IN NO EVENT SHALL BE RESPONSIBLE OR LIABLE TO THE END USER FOR ANY AMOUNTS IN EXCESS OF THE FEES PAID BY THE END USER TO LEIDOS. LEIDOS SHALL NOT BE RESPONSIBLE FOR ANY MATTER BEYOND ITS REASONABLE CONTROL.

LEIDOS PROPRIETARY INFORMATION

This document contains Leidos Proprietary Information. It may be used by recipient only for the purpose for which it was transmitted and will be returned or destroyed upon request or when no longer needed by recipient. It may not be copied or communicated without the advance written consent of Leidos. This document contains trade secrets and commercial or financial information which are privileged and confidential and exempt from disclosure under the Freedom of Information Act, 5 U.S.C. § 552.

TRADEMARKS AND ACKNOWLEDGMENTS

Private installations of DigitalEdge are powered by Eucalyptus®.

Public cloud installations of DigitalEdge are powered by Amazon Web Services™.

The following list includes all trademarks that are referenced throughout the DigitalEdge documentation suite.

Adobe, Flash, PDF, and Shockwave are either registered trademarks or trademarks of Adobe Systems Incorporate in the United States and/or other countries.

Amazon Web Services, AWS, Amazon Elastic Compute Cloud, Amazon EC2, EC2, Amazon Simple Storage Service, Amazon S3, Amazon VPC, Amazon DynamoDB, Amazon Route 53, the "Powered by Amazon Web Services" logo, are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries.

Apache, Archiva, Cassandra, Hadoop, Hive, HBase, Hue, Lucene, Maven, Apache Phoenix, Solr, Zoie, ActiveMQ are all trademarks of The Apache Software Foundation.

ArcSight is a registered trademark of ArcSight, Inc.

CAS is copyright 2007, JA-SIG, Inc.

CentOS is a trademark of the CentOS Project.

Cloudera is a registered trademark of Cloudera, Inc.

CloudShield is a registered trademark of CloudShield Technologies, Inc. in the U.S. and/or other countries.

CTools are open-source tools produced and managed by Webdetails Consulting Company in Portugal.

Drupal is a registered trademark of Dries Buytaert.

Elasticsearch is a trademark of Elasticsearch BV, registered in the U.S. and in other countries.

Eucalyptus and Walrus are registered trademarks of Eucalyptus Systems, Inc.

CONTACT INFORMATION

Leidos Franklin Center
6841 Benjamin Franklin Drive
Columbia, Maryland 21046

Email: DigitalEdgeSupport@Leidos.com

DigitalEdge Technical Support: 443-367-7770

DigitalEdge Sales Support: 443-367-7800

To submit ideas or feedback: https://www9.v1ideas.com/digitaledge/welcome

Contents

Introduction

Most DigitalEdge systems need an alerting engine to identify anomalous situations such as cybersecurity breaches or financial fraud. Implementing alerts involves:

- Identifying the business rules that define a potential threat, and translating those rules into data filters that will trigger an alert
- Identifying the people who should be notified of an alert situation
- Writing the messages that should be sent out as alert notifications
- Specifying the type of alerting messages to issue (email, JMS messages, etc.)
- Configuring an alerting data sink to filter processed records against the alerting criteria

The DigitalEdge alert model is subscription oriented, so that each user can subscribe to specific alerts.

The Alerts API provides a means for DigitalEdge to send real time alerts based on criteria specified from the content of ingested data. The Alerts API also controls how, to whom, and with what content the alerts are sent.

You can access the Alerts API at: `http://default.<system_name>/alertsapi`

Product documentation

DigitalEdge is a complex big data platform. The system comes with a complete set of documentation in PDF and HTML5 formats to help you master DigitalEdge:

Document	Use	Audience
Overview Guide	Basic information about the DigitalEdge platform, including architecture, concepts, and terminology; a must-read before working with any aspect of DigitalEdge	Anyone working with DigitalEdge in any capacity
Configuration Guide	Instructions for defining data models and building processing pipelines	Data Specialists, DigitalEdge Administrators
Operations Guide	Daily administration information, covering monitoring, managing, and modifying the platform	DigitalEdge Administrators
Cookbook	Guidelines and procedures for many common tasks in a DigitalEdge system	DigitalEdge Administrators
DigitalEdge SDK Guide	Reference for building custom plug-in components	Developers
Alerts API Guide	Reference for specifying data triggers and notifications for an alerting capability	Developers

Document	Use	Audience
Search API Guide	Reference for providing search services on a Lucene data sink node	Developers

Typographical conventions

The following style conventions are used throughout this documentation:

Type of Information	Style in Documentation
Code, commands, filenames	`code`
Cross references	Click to see this topic
Emphasis	*important point*
Hyperlinks	Click to go to this site
Notes, warnings, tips	★
References to other documents	*Document Title*
Sample code blocks	`code`
Troubleshooting issue or problem	?
Troubleshooting solution	✓
User input	*Italics*
User interface labels and controls	**Bold**
Variables	<*change-this-name*>

Creating a Filter Definition

A filter definition defines the conditions and values that trigger an alert. The definition is dependent on the data model that you use in the filter; the fields in the definition must match fields in the filter's data model. All of the following examples are based on the rectangle data model:

```
{"dimensions":{"width":<Int>,"height":<Int>},"color":"<String>"}
```

Syntax

The examples in this API use a rectangle data model:

```
{"dimensions":{"width":<Int>,"height":<Int>},"color":"<String>"}
```

where variables in angled brackets < > should be replaced with actual values.

To create a valid definition, you must first understand the syntax. Definitions are written in JSON format. This format does not have to exactly match the format of the data model being used, but the fields in the definition must be actual fields in the data model. For example, a trigger that alerts on a rectangle with a height of 6 would be defined as:

```
{"dimensions":{"height":"=6"}}
```

Notice that not all the fields are included, only the one that will trigger the alert.

Notice the notation used to declare the trigger expression (the field `"=6"`). Not only do you declare the value for the trigger, but you also explicitly declare the operator needed by the trigger. If you exclude the operator, the expression will be incomplete and the system cannot accurately match the trigger. This field must be surrounded by double quotes.

When specifying the value to use in a trigger, notice the value's type in the data model. In the example above, the value of `dimensions.height` is an integer. So the value of `6` is not surrounded by quotes but the whole expression is. If the triggering field is defined is a string or date string, you must use single quotes around the value. For example, a trigger that alerts on a rectangle of the color BLUE would be defined as:

```
{"color":"='BLUE'"}
```

The trigger value of `BLUE` is enclosed in single quotes, indicating that it is a string. Also, the value is in all upper case, since the handling of strings is case sensitive.

Numerical operators

Definitions can trigger alerts with an AND or an OR condition, depending on how the definition is configured:

- An AND condition occurs if trigger X AND trigger Y are both present
- An OR condition occurs if trigger X OR trigger Y is present

To configure an **OR** definition, specify an array of possible triggers. Note that when using an array, the entire array must be enclosed in [ ] brackets, and each possible trigger must be separated by commas. For example, to define an alert that triggers if the width is 5 OR the height is 6, create an array of definitions that match these cases:

```
{"dimensions":["width":"=5","height":"=6"]}
```

The filter includes two separate JSON strings.

To create an **AND** condition, combine these two JSON strings into one string. For example, to trigger when the width is 5 AND the height is 6:

```
{"dimensions":{"width":"=5","height":"=6"}}
```

If you want to alert on rectangles with a width of 5, 7, OR 9, it would be advantageous to use a notation that is closer in format to that of the data model:

```
{"dimensions":[{"width":"=5"},{"width":"=7"},{"width":"=9"}]}
```

This example uses the array notation to indicate an OR situation, but it is only referencing one specific field in the model.

Besides equivalence, there are **other numeric operators** to define a filter:

Operator	Meaning
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
!=	Not equal to

For example, to find rectangles with a width greater than 5:

```
{"dimensions":{"width":">5"}}
```

String operators

In addition to the numerical operators, there are operators for strings:

- LIKE
- IN
- @

Use the **LIKE** operator to match a regular expression. For example, to find a rectangle that matches all the colors that begin with BLUE: BLUE, BLUE-GREEN, BLUE-ISH, BLUETIFUL, use the LIKE operator:

```
{"color":"LIKE 'BLUE*'"}
```

Notice the syntax: a space follows the LIKE operator, and the wildcard character is within the single quotes of the string value.

The **IN** operator triggers an alert if a value matches any value in a list of trigger specifications. This is another way to create an OR situation. For example, to alert on rectangles that are only primary colors, you could use an OR array, or you could use the IN operator:

```
{"color":"IN ('RED','BLUE','YELLOW')"}
```

Notice the syntax: a space follows the IN operator, the list of possible values is comma-separated and enclosed in parentheses, and each possible string is enclosed in single quotes. However, remember that a number does *not* require single quotes.

The **@** operator specifies other fields within the data model as values in the expression. Use the @ operator to compare values in two fields of the same data type. For example, to alert when the rectangle is a square, you would define a trigger when the height = width:

```
{"dimensions":{"height":"='@dimensions.width'"}}
```

The @ operator will plug in the data value from the `dimensions.width` field and test it against the `dimensions.height` field. Notice the syntax: the combination of the @ symbol and the field name must be included in single quotes, even if the value of the resulting substitution is *not* a string.

Wildcard characters

Use wildcard characters to match on one or more unspecified characters. There are two wildcards available:

- ? = single character wildcard
- * = multiple characters wildcard

For example:

```
{"color":"='BLU*'"}
```

Examples

Here are more sample definitions from other data models:

```
"[{"networkData":{"source":{"dshield":"LIKE '*'"}}},
{"networkData":{"destination": {"dshield":"LIKE '*'"}}}]">

"{"arcsightHeader":{"severity":" IN ('High','Very-High')"}}"

"{"order":{"orderPrice":">350000.0"},"D_supplierNation":
{"nationName":"IN ('IRAQ','IRAN')"}}"

"{"D_eventType":{"eventType":"= 'Flight'"},"geoList":
[{"geoCity":"LIKE 'Kahului*'"}]}"

"{"order":{"clerk":"='Clerk#000000042'"}}"
```

```
"{"D_customer":{"customerName":"LIKE 'Customer#0000000*'"}}"
```

```
"{"BranchID":"!= '@LastTransaction.BranchID'"}"
```

```
"[{"networkData":{"source":{"network":{"network":"='Unregistered
Network'"}}}}, {"networkData":{"destination":{"network":
{"network":"='Unregistered Network'"}}}}]"
```

```
"{"eventType":"='Checkpoint'"}"
```

```
"{"Teller": {"Name":"='@Account.Name'"} }"
```

Alerting Components

Alerting is configured with a collection of application database tables in the gateway node.

NAMED_FILTER table

The NAMED_FILTER table defines the name of an alerting filter, the data model the filter looks at, and the definition of what to alert on, based on data model fields. Additionally, this table contains the Subject and Message of email notifications when an alert is triggered.

NAMED_FILTER Table				
Column	**Type**	**Size**	**Nullable?**	**Description**
Key	Decimal		No	Unique key assigned to the filter for identification
Name	Varchar	125	No	Name of the filter
Model	Varchar	100	No	Data model that the filter alerts on
Definition	Varchar	4000	No	Definition of what to trigger an alert on. See "Creating a Filter Definition" on page 3.
Email_Subject	Varchar	100	Yes	Subject of the email that is sent when an alert is triggered
Email_Message	Varchar	4000	Yes	Message that is sent in an email when an alert is triggered. The message can include information from the triggering data by surrounding the data model field with @ symbols. For example: using the rectangle data model such as: `{"dimensions":{"width":5,"height":6}}` the message could display the triggering data's width by including `@dimensions.width@` in the message.

NAMED_FILTER_USERS table

The NAMED_FILTER_USERS table defines the recipients of email alerts. This table contains the Username and Password of each user who will receive alerts, and a column to define the email addresses.

NAMED_FILTER_USERS Table				
Column	**Type**	**Size**	**Nullable?**	**Description**
Key	Decimal		No	Unique key assigned to the user for identification
Username	Varchar	25	No	Username of the email recipient
Password	Varchar	125	No	Password of the email recipient
Email_List	Varchar	4000	Yes	Email address(es) for the user who will receive an

NAMED_FILTER_USERS Table				
Column	Type	Size	Nullable?	Description
				alerts email message. Specify multiple email addresses in a comma-separated list.

NAMED_FILTER_WATCHLIST table

The NAMED_FILTER_WATCHLIST table links users to specific alert filters. The table includes a column that can be toggled Y or N to indicate if the user should receive an email notification, making it easier to switch alerts on or off for a user without adding or removing names in the table.

NAMED_FILTER_WATCHLIST Table				
Column	Type	Size	Nullable?	Description
User_ Key	Decimal		No	Key for a user defined in the NAMED_FILTER_USERS table
Filter_ Key	Decimal		No	Key for a filter defined in the NAMED_FILTER table
Email	Char	1	Yes	Toggle Y or N to indicate if the user should receive an alert when triggered by the specified filter. This makes it easy to switch alerts on or off for a user. A value of NULL is equivalent to N.

Using the API

The API is accessible via REST calls from externally defined REST clients. These REST services are how alerting is configured, rather than with direct access to the database tables. The following REST services are available via the API.

ConfigService

The `ConfigService` provides a means to create, modify, delete, and generally access/move the configurations required for AlertViz graph settings. Configurations are intended to be used by AlertViz application, and could be used by other applications if they needed configuration support.

FilterService & FiltersService

The `FilterService` provides an interface to the **NAMED_FILTER** table of the database. It is used to create, delete, and generally modify and access specific filters in the database. The FiltersService is used to access more than one alert filter at a time.

UserService & UsersService

The `UserService` provides an interface to the **NAMED_FILTER_USERS** table of the database. It is used to create, delete, and generally modify and access specific users in the database. The `UsersService` is used to access more than one alert filter at a time.

WatchListService

The `WatchListService` provides an interface to the **NAMED_FILTER_WATCHLIST** table of the database. It is used to define and access which users are mapped to specific alert filters.

HistoricalAlertsService

The `HistoricalAlertsService` provides a way to access alerts that are stored in memory by the `HistoricalAlertManager`. This feature stores alerts in addition to sending alerts in real time. `HistoricalAlerts` are currently specific to use by AlertViz.

Sample Code

The sample code is an example of an externally defined REST Client that can be used to access the Alerts API User REST Service.

AlertsApiUserRestClient.java

```java
package <your package>;

import javax.ws.rs.core.MediaType;
import net.sf.json.JSONObject;
import org.apache.commons.configuration.ConversionException;
import org.apache.log4j.Logger;
import com.deleidos.rtws.commons.config.RtwsConfig;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.representation.Form;

public class AlertsApiUserRestClient {
        private Logger logger = Logger.getLogger(AlertsApiUserRestClient.class);
        private static Client REST_CLIENT_INSTANCE;
        private static String ALERTSAPI_BASE_URL;

        public AlertsApiUserRestClient () {
                if (ALERTSAPI_BASE_URL == null) {
                        // Set the url of the Alerts API so the REST call knows where the API is
                        try {
                                RtwsConfig conf = RtwsConfig.getInstance();
                                ALERTSAPI_BASE_URL = conf.getString("webapp.alertsapi.url.path");
                        } catch (ConversionException e) {
                                logger.fatal("Load property [webapp.alertsapi.url.path] failed. Message: " +
                                        e.getMessage());
                        }
                }
                configureClient();
        }

        private void configureClient() {
                ClientConfig config = new DefaultClientConfig();
                config.getProperties().put(ClientConfig.PROPERTY_CONNECT_TIMEOUT, 10000);
                config.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 30000);
                REST_CLIENT_INSTANCE = Client.create(config);
        }

        public Client getClient() {
                return REST_CLIENT_INSTANCE;
        }

        public String[] getEmailList(String username) throws RestClientException {
                // Create the resource to access the REST Service based on the Alerts API url
                // and REST Service path
                WebResource resource = getClient().resource(ALERTSAPI_BASE_URL);
                resource = resource.path("user/retrieve/email").path(username);

                // Execute the rest call
                String response = resource.get(String.class);

                // Process the results of getting the email List as a comma separated String of emails
                String[] emailList = response.split(",");
                for(int i = 0; i < emailList.length; i++) {
                        emailList[i] = emailList[i].trim();
                }

                return emailList;
        }

        public void updateEmailList(String username, String emailList) throws RestClientException {
                // Create the resource to access the REST Service based on the Alerts API url
                // and REST Service path
                WebResource resource = getClient().resource(ALERTSAPI_BASE_URL);
                resource = resource.path("user/update/email").path(username);

                Form f = new Form();
                f.add("emailList", emailList);


                // Execute the rest call
                String response = resource.type(MediaType.APPLICATION_FORM_URLENCODED_TYPE)
                                .put(String.class, f);
                // Process if the response was a success
                JSONObject jsonResponse = JSONObject.fromObject(response);
                JSONObject standardHeader = jsonResponse.getJSONObject("standardHeader");
                if (standardHeader.getInt("code") != 200) {
                        String message = jsonResponse.getString("responseBody");
                        throw new RestClientException(message);
                }

        }
}
```

RestClientException.java

```
package <your package>;

public class RestClientException extends Exception {
        private static final long serialVersionUID = -5669274968174191493L;

        public RestClientException() {
                super();
        }

        public RestClientException(String message, Throwable cause) {
                super(message, cause);
        }

        public RestClientException(String message) {
                super(message);
        }

        public RestClientException(Throwable cause) {
                super(cause);
        }
}
```

How to Run the Samples

To execute a client that invokes the Alerts API locally:

1. Open the `AlertsApiUserRestClient.java` file
2. You must provide a `main` function in the sample file to run the Java application.
3. Right-click the file and select **Run As/Java Application**.

   Or, use the following commands to run the sample using Java:

   ```
   cd <my-project-path>
   ```

   Build your project which contains the `AlertsApiUserRestClient.java` file.

   ```
   java -cp .;<project-jar-path> <package>.AlertsApiUserRestClient
   ```