

GTS Enterprise Rules Engine Event Notification and Icon Selection

Copyright © 2008-2010 GeoTelematic Solutions, Inc.
All rights reserved

projects@geotelematic.com
<http://www.geotelematic.com>

Manual Revision History			
Rev	Date	Changed	Author
0.1.0	2008/12/14	Initial Release	MDF
0.1.1	2009/04/26	Added new rule identifiers and functions. Added additional information regarding email recipients. Added information on new command-line "ruleTest.sh" command.	MDF
0.1.2	2009/05/09	Misc Changes.	MDF
0.1.3	2009/05/24	Added comments regarding additional EventData table lookups for various functions.	MDF
0.1.4	2009/08/12	Added Event \${speedLimit} replacement variable. Added information on creating "RuleListener" action targets.	MDF
0.1.5	2009/11/01	Added periodic maintenance \$MAINTKM function definition.	MDF
0.1.6	2010/04/12	Added \$WORKHOURS, \$INDEX, \$INCCORR function definition.	MDF
0.1.7	2010/05/??	Added several new "Global" functions. Added info on creating custom functions. Added information on "Alert Monitor" panel configuration.	MDF

GTS Enterprise Rules Engine

Contents:

- 1 Introduction**
 - 1.1 Event Notification**
 - 1.2 Icon Selection**
- 2 Rule Specification**
 - 2.1 Rule Syntax**
 - 2.2 Rule Evaluation**
 - 2.3 Rule Definitions**
- 3 Event Notification**
 - 3.1 Action Specification**
 - 3.2 Example Rule Specifications**
 - 3.3 How It Works**
- 4 Icon Selection**
 - 4.1 Example Rule Specifications**
- 5 "Rule Administration" Web Interface**
 - 5.1 Rule Fields**
 - 5.2 'private.xml' EMail Template**
- 6 "Alert Monitor" Panel**
 - 6.1 Configuring Alert Rules**
 - 6.2 "Device Alerts" Panel**

Appendix:

- A) Internal Rule Variable Definitions**
- B) Internal Rule Function Definitions**
- C) Adding Your Own Custom Selector Functions**
- D) Event Email Subject/Body Variables**

1) Introduction

The **GTS Enterprise** Rules Engine is an add-on to the basic **GTS Enterprise** platform that allows specifying "rules" which are evaluated against incoming events to perform some notification action or icon selection. For instance, A simple rule can be set up to send an email when a vehicle's door is opened, has exceed a preset speed, or has arrived at a specific geographic location. A rule can also be used to select a specific icon which is to be displayed on a map to represent a specific pushpin.

To determine if your system has the Event notification rules engine installed, look in the OpenGTS/GTS installation directory ("build/lib/" or "lib/*/") for either the 'rulefact.jar' or 'optdb.jar' files.

This manual is intended for system administrators and developers that wish to understand the workings of the Event notification rules engine, and wish to set their own rule specifications.

1.1) Event Notification

Applying the Rules Engine to Event Notification allows an email to be sent to an individual or group if an incoming Event triggers a rule action. The predefined action can be specified to email one, or more, of the Account 'contact' email address, Device 'contact' email address, or another address specified on the rule definition.

When a remote client sends an event to the server, the rules engine analyzes the event against variable pre-defined rules, and if a rule is triggered, the rule action is performed. An evaluated rules is deemed 'triggered' if it returns a non-zero value.

1.2) Icon Selection

Applying the Rules Engine to Icon Selection allows displaying a specific pushpin icon on a map based on the returned value from the evaluated rule. The icon selector rule is constructed such that it returns a 'String' value representing the name of the icon to display for the specific pushpin. These icon "names" are defined in the 'private.xml' file.

2) Rule Specification

A rule specification consists of a logical series of operators and operands. Certain pre-defined variables can be used which will be substituted with their appropriate value from the current analyzed incoming event. Several functions are also available to perform other operations and can return boolean, numeric, or even String values.

2.1) Rule Syntax

The Rule specification syntax closely follows the 'C' language expression syntax. If you are familiar with the 'C' language expression syntax, then the rule specification syntax should look familiar. The following operators are available for use within a rule specification.

Operator	Type	Description
&&	binary	Logical "AND"
	binary	Logical "OR"
^^	binary	Logical "XOR"
!	unary	Logical "NOT"
==	binary	Logical "EQUALS"
>=	binary	Logical "GREATER-THAN-OR-EQUALS"
<=	binary	Logical "LESS-THAN-OR-EQUALS"
>	binary	Logical "GREATER-THAN"
<	binary	Logical "LESS-THAN"
!=	binary	Logical "NOT-EQUALS"
+	binary	Arithmetic "ADD", or String "CONCATENATION"
-	binary/unary	Arithmetic "SUBTRACT" (binary), or "NEGATION" (unary)
*	binary	Arithmetic "MULTIPLY"
/	binary	Arithmetic "DIVIDE"
%	binary	Arithmetic "MODULO"
**	binary	Arithmetic "POWER"
&	binary	Bitwise "AND"
	binary	Bitwise "OR"
^	binary	Bitwise "XOR"
~	unary	Bitwise "NOT" (ie. Complement)
>>	binary	Bitwise right shift
<<	binary	Bitwise left shift
? :	ternary	Conditional value (ternary)
()		Forced association
"..."		String constant
\$FTN(...)		A function specification with arguments. (see Appendix A)
var		A variable specification (see Appendix B)

Notes:

- Constants can be specified as integer, floating-point, or quoted Strings.
- Pre-defined variables (defined below) can also be specified which will be replaced with their current value when the rule is evaluated.
- Operand association is left to right and follows the standard 'C' language precedence rules. Associations can be forced with the use of parenthesis.
- When binary operations occur on mismatched types, Longs are promoted to Double. For some binary operators Longs and Double are promoted to Strings (such as for comparison operators, String concatenation using '+', logical OR '||', and logical AND '&&').
- The boolean value resulting from Logical operators is a Long value, either '0' for False, or '1' for True.

2.2) Rule Evaluation

The result of an evaluated rule may be a Long (64-bit Integer), Double (64-bit floating point), or a String value. The result is considered **True** for boolean purposes if the returned value is non-zero (in the case of a Long or Double result) or a non-empty String (in the case of a String result).

If a rule specification contains a syntax error, then an error will be displayed to the capturing log file during rule evaluation. Rule selectors which contain syntax errors always return an evaluation of false.

Here are a few simple example rule specifications:

(100.0 / 5.0)	returns 20.0
(2 + 3 * 10)	returns 32
(2 + 3 * 10.0)	returns 32.0
((4 % 2)? "apple" : "pear")	returns "pear"
(34 == 23)	returns 0
((2 + 3) == 5)	returns 1
((34 == 23) && ((2 + 3) == 5))	returns 0
((34 == 23) ((2 + 3) == 5))	returns 1
("pear" + 42)	returns "pear42"
(2==1)? "A" : (2==2)? "B" : "C"	returns "B"
\$MAX(1.3, 5, -23)	returns 5.0
\$SQRT(3)	returns 1.7320508075688772
\$POW(4, 0.5)	returns 2.0
\$BIT(0x0010, 4)	returns 1
\$BIT(0x0010, 3)	returns 0
\$TOD(12, 30)	returns 750
\$ROUND(1.52)	returns 2
\$DOUBLE("1.2345", 2.3456)	returns 1.2345
\$DOUBLE("XXXX", 2.3456)	returns 2.3456
("apple" "pear")	returns "apple"
("" "pear")	returns "pear"

A command-line rule evaluation testing tool is provided in the GTS installation 'bin' directory. This command must be executed from the GTS installation directory itself (\$GTS_HOME):

```
/zzz> cd $GTS_HOME
/usr/local/GTS_X.X.X> bin/ruleTest.sh -account=myacct -device=mydev -gps=37.123/-142.123
```

The prompt "Expression>" will be displayed, allowing you to enter the various rule selectors described in this document. Hitting carriage-return will display the results of the rule evaluation, then the "Expression>" prompt will again display waiting for another entry. Entering "exit" will cause the command line tool to exit.

The "-account", "-device", and "-gps" arguments allow specifying values for a simulated event upon which the evaluated rule selectors will be based. (Run the command with the option "-help" to display any other options which may also be available).

Here is an example session should how the "bin/ruleTest.sh" utility might be used:

```
/usr/local/GTS_X.X.X> bin/ruleTest.sh -account=myacct -device=mydev -gps=37.123/-142.123
```

```
-----
Rule Evaluation Testing Utility (v1.1.5)
Copyright 2007-2009, GeoTelematic Solutions, Inc.
(subject to licensing terms/conditions which accompany this software product)
-----
```

```
Account : myacct
Device  : mydev
GPS     : 37.12300/-142.12300
Speed   : 0.0 km/h (0.0 mph)
-----
```

```
(Enter 'exit' and carriage-return to exit this command-line utility)
```

(Simple arithmetic expression)

```
Expression> (2 + (3 * 10.0))
==> value: 32.0
==> type : Double
==> match: true
```

(9 to the ½ power – same as the square-root of 9)

```
Expression> $POW(9, 0.5)
==> value: 3.0
==> type : Double
==> match: true
```

(Test for event GPS location inside "home" Geozone)

```
Expression> $INZONE("home")
==> value: 0
==> type : Long
==> match: false
```

(Calculate number of days in 2008 – which was a leap year)

```
Expression> $DAY(1,1,2009)-$DAY(1,1,2008)
==> value: 366
==> type : Long
==> match: true
```

2.3) Rule Definitions

Rules may be defined in the "Rule" table, and assigned to specific Devices and Status codes in the "RuleList" table. Alternatively, each Device has the ability to define a single Rule specification and action in the Device table record itself.

The Device table contains the following fields to allow specifying a default notification rule (defined in the "NotificationFieldInfo" optional fields section):

allowNotify	- True to enable notification, False to disable.
notifyEmail	- The email address to which email notifications are sent for the Device
notifySelector	- The rule specification.
notifyAction	- The triggered action.
notifyDescription	- The description of the rule specification.
notifyPriority	- The notification priority (may be optional).

The Rule table contains the following fields to define a specific Rule:

accountID	- The Account ID.
ruleID	- The Rule ID.
description	- The Rule Description.
isActive	- True to enable this Rule, False to disable.
selector	- The Rule specification
actionMask	- The Action(s) to perform if this Rule is Triggered
priority	- The notification priority (may be optional).
notifyEmail	- The notification email address
emailSubject	- The email subject line
emailText	- The email body

Rules are assigned to a given Device and specific StatusCode within the RuleList table:

accountID	- The Account ID.
deviceID	- The Device ID. ("*" for all Devices)
statusCode	- The Event StatusCode.
ruleID	- The Rule ID.

The Rule 'Action' is a bitmask, currently defined (RuleFactory.java) as follows:

```
// Notification email recipient
ACTION_NOTIFY_MASK          0x000000FF
ACTION_NOTIFY_ACCOUNT       0x00000001 // send email to Account 'notifyEmail'
ACTION_NOTIFY_DEVICE        0x00000002 // send email to Device 'notifyEmail'
ACTION_NOTIFY_RULE          0x00000004 // send email to Rule 'notifyEmail'

// Notification method
ACTION_VIA_MASK             0x0000FF00
ACTION_VIA_EMAIL            0x00000100 // notify via SendMail (default)
ACTION_VIA_QUEUE            0x00000200 // notify via Notify Queue
ACTION_VIA_LISTENER         0x00000400 // notify via Listener
ACTION_SAVE_LAST            0x00010000 // save last notification
```

The action values placed in the 'actionMask' (Rule table), and 'notifyAction' (Device table) must be a bitwise 'OR' of the bitmask items above. For instance, to specify an action that sends email to the Account owner, and Rule email address, the action mask should be hex "0x00000105". Currently, if the specified action mask is "0x00000000", then the actual action mask will default to "0x00010507" (send email to the Account, Device, and Rule email addresses, send rule trigger to callback method, save last notification in Device record).

If "ACTION_VIA_QUEUE" is included in the action mask, then the notification will be added to the "NotifyQueue" table. The notification written to this table is similar to an email (including a sender, recipient list, subject, and message body), but allows other applications to query and pull notification messages from this table at a later time.

If "ACTION_VIA_LISTENER" is included in the action mask, then the list of classes which implement the "RuleListener" interface, and have been added to the "EventRuleAction" list of rule listeners, will be called. This allows customized types of actions to be performed that are specific to a system installation.

The "org.opengts.rule.RuleListener" interface supports the following method:

```
public interface RuleListener
{
    /**
     *** Callback to handle a rule notification trigger
     *** @param account    The Account for which the rule was triggered
     *** @param device     The Device for which the rule was triggered (may be null)
     *** @param event      The Event which triggered the rule (may be null)
     *** @param selector   The rule selector
     *** @param rule       The rule that was triggered (may be null)
     **/
    public void handleRuleNotification(Account account, Device device,
        EventData event, String selector, Rule rule);
}
```

To install your own custom RuleListener notification handler, create and compile a Java class which implements the above interface, then add the following line to the "common.conf" runtime configuration file:

```
rule.ruleListenerClass=my.custom.ruleListener
```

Where "my.custom.ruleListener" is the name of your class that implements the "RuleListener" interface.

At startup initialization time, an instance of your class will be instantiated using the default constructor (if any errors occur during initialization, they will be displayed in the log files). Then as event rules are triggered, a call will be made to "handleRuleNotification" with the various appropriate arguments which triggered the rule.

3) Event Notification

Incoming events can be analyzed against a list of rule specifications. If an evaluated rule returns True, then an action can be triggered, such as sending an email to the interested parties.

To fully enable the various features provided by the Event Notification Rules Engine, there are additional fields within the Device table which should be enabled. To enable these features, the following properties should be specified in the "custom.conf", or "custom_gts.conf":

```
startupInit.Device.NotificationFieldInfo=true
```

Enables the rule trigger notification flags and email address features, as well as enabling support for the "Device Alerts" and "Alert Panel".

```
startupInit.Device.MaintOdometerFieldInfo=true
```

Enables support for the periodic maintenance notification support (uses the daily "Cron" tasks to monitor for periodic maintenance on a daily basis).

```
startupInit.Device.GeoCorridorFieldInfo=true
```

Enables the "activeCorridor" feature in the Device table.

3.1) Action Specifications

An action is typically specified as an email to be sent to the Account owner, Device owner, or even to an email address specified by the Rule itself. See "Rule Definitions" above for additional possible rule actions that may be specified.

3.2) Example Rule Specifications

Here are a few rule specifications that can be used to trigger an 'Event Notification' action based on the current Event:

```
($WEEKEND || !$TODRANGE(0730,1730))
```

Returns True if the current Event timestamp represents the weekend, or is outside of working hours (before 7:30am, or after 5:30pm).

```
$DIN(0,1)
```

Assuming that Digital Input #0 is attached to a door in the vehicle, this rule would return True if the current Event represents a digital input with the specified door currently open.

```
($ARRIVE("home") || $DEPART("home"))
```

Returns True if the current Event represents either an arrival or departure from the pre-defined Geozone ID "home" (a Geozone with the ID "home" must already exist).

```
($ARRIVE || $DEPART)
```

Returns True if the current Event represents either an arrival or departure from any pre-defined Geozone.

```
(( $DOW == MONDAY ) && $INZONE("home"))
```

Return True if the current Event timestamp represents Monday, and the GPS location is currently inside Geozone ID "home" (a Geozone with the ID "home" must already exist).

3.3) How It Works

These are the steps followed in the rule evaluation and notification process when a new event is received from the remote tracking device:

1. Check to see if notification is allowed for the current device. If not, then no further rule checking is performed.
2. If specified, execute the rule selector specified in the Device 'notifySelector' field. If triggered, notification is performed based on the action specified in the 'notifyAction' field in the Device record. The EMail composition specified in the Device record will be used to generate the notification email. Depending on the rule action, the recipients of the email are comprise of email addresses found in the notification email section of both the Account and Device records.
3. Retrieve all Rule specification for the current Device and Event statusCode, from the RuleList table. Execute each retrieved rule specification and perform the specified action for each triggered rule. The EMail composition specified in the triggered Rule record will be used to generate the notification email. Depending on the rule action, the recipients of the email are comprise of email addresses found in the notification email section of the Account, Device, and triggered Rule records.

4) Icon Selection

Customized Icon Selectors may be used to display specific pushpin icons on a map, based on the resulting value from an icon selector rule specification.

The returned result from an icon selector rule specification should be the name of the pushpin to display for a specific event on the map. The "name" of the icon should be defined in the 'private.xml' file, in the Pushpins tag section under the active MapProvider.

The IconSelector rule specification should also be specified in the 'private.xml' file as a Property in the MapProvider tag section. This IconSelector indicates that for the Device map, display the "red" pushpin if the speed is less than 5 mph, otherwise display the "green" pushpin:

```
<Property key="iconSelector"><![CDATA[ (mph<5.0)?"red":"green"] ]></Property>
```

This IconSelector indicates that for the Fleet map, always display the "blue" pushpin:

```
<Property key="iconSelector.fleet"><![CDATA[ ("blue") ] ]></Property>
```

The 'StatusCode' table, which is used for custom statusCode descriptions, also has an 'iconSelector' column that can be used to specify an icon selector rule specification for specific status code types.

4.1) Example Rule Specifications

Here are a few example icon selector rule specifications:

```
($DIN?"orange":(speed<5)?"red":(speed<32)?"yellow":"green")
```

If the current Event represents a digital input event, display the "orange" pushpin, else if the speed is less than 5 km/h, then display the "red" pushpin, else if the speed is less than 32 km/h, then display the "yellow" pushpin, else display the "green" pushpin (when the speed \geq 32).

```
($DIN(1,1)?"green":"red")
```

If digital input #1 is True, display the "green" pushpin, else display the "red" pushpin.

```
($INZONE?"yellow":"green")
```

If inside any defined Geozone, display the "yellow" pushpin, else display the "green" pushpin.

5) "Rule Administration" Web Interface

The "Rule Administration" web interface page is included in the "Rules Engine" package and allows defining and naming rules and assigning them to event status codes (to be evaluated when an event with a given status code arrives at the server).

5.1) Rule Fields

The "Rule Administration" web interface allows the following fields to be assigned to a specific rule:

- **Rule ID**
The name/id assigned to a specific rule. This should be a short name, such as "arriveHome" or "depart", and can be referenced in other rule selectors using the \$RULE function, such as \$RULE("depart").
- **Rule Description**
The description of the rule. This is only used for information purposes when viewing a list of defined rules.
- **Rule Selector**
The actual rule selector. This contains the rule selector syntax defined above.
- **Status Code**
Currently, the "Rule Administration" interface automatically creates a single entry in the "RuleList" table assigning the current rule to this specified status code.
- **Email Address**
The EMail address to which a notification will be sent. This email address will be combined with the notification email addresses found in the Account and Device records for the particular event.
- **Email Subject**
The EMail subject. This value may contain event variable specifications outlined in Appendix C below. The resolved subject line (with variables replaced with their current event value) may again be referenced as the variable \${subject} in the email template defined in 'private.xml' (see below).
- **Email Message**
The EMail message. This value may contain event variable specifications outlined in Appendix C below. The resolved message (with variables replaced with their current event value) may again be referenced as the variable \${message} in the email template defined in 'private.xml' (see below).
- **Notify Use Wrapper**
If true, the EMail template defined in 'private.xml' will be used to compose EMail notifications. The EMail Subject/Message defined above will be accessible via email variables \${subject} and \${message} within this template (See "EventNotificationEMail" in the 'private.xml' file). If false, the EMail template will not be used and the EMail will be composed entirely from the EMail Subject/Message specified above.

5.2) 'private.xml' EMail Template.

The "private.xml" file defines an EMail **template** in the XML tag section "EventNotificationEMail". This section contains the sub-tags "Subject" and "Body" which specify the actual formatted email that will be sent to the various notification email addresses. In addition to the event variables defined in Appendix C, the variables `${subject}` and `${message}` may also be used (which are resolved from the subject/message fields in the "Rule" table).

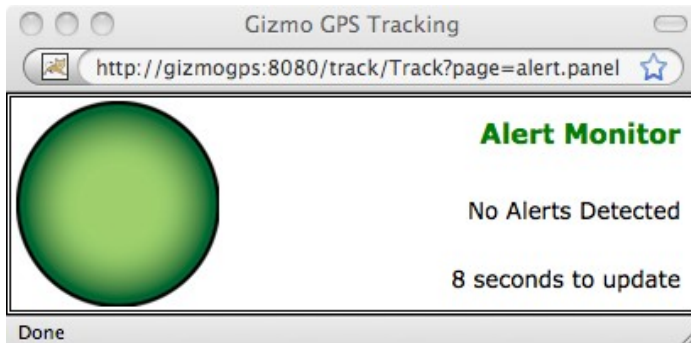
The following is an example "EventNotificationEMail" tag entry that may be specified in the 'private.xml' file:

```
<EventNotificationEMail from="notify@example.com">
  <Subject><![CDATA[
    Event Notification from '${device}' [${deviceid}]
  ]]></Subject>
  <Body><![CDATA[
    '${device}' generated the following notification:
    Account   : ${account} [${accountid}]
    Device    : ${device} [${deviceid}]
    Date/Time : ${datetime}
    Status    : [${statuscode}] ${status}
    Location  : ${geopoint}
    Address   : ${address}
    Speed     : ${speed} ${direction}
    Altitude  : ${altitude}
    Odometer  : ${odometer}
    Distance  : ${distance}
    Message   : ${message}
  ]]></Body>
</EventNotificationEMail>
```

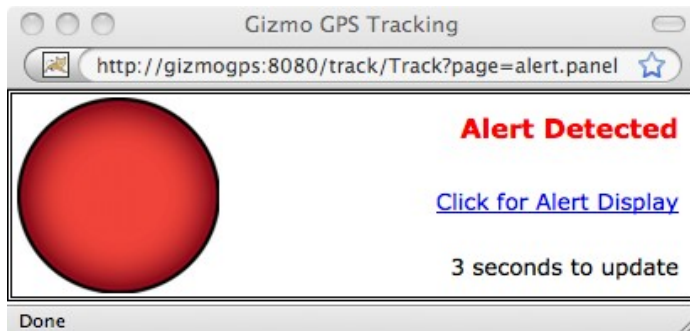
6) "Alert Monitor" Panel

The "Alert Monitor" feature allows setting up a specific rule which can be monitored through a visual interface when an administrator is logged-in to the system. For instance, if the administrator wishes to monitor for an exception such as "temperature out of range", an alert rule can be created which watches for this condition, then changes the display of the "Alert Monitor" to indicate that this alert has occurred.

Here is an example "Alert Monitor" showing that there are currently no active alerts:



When an alert becomes active, the "Alert Monitor" changes to the following:



Clicking on the Red button, or on the displayed link, will bring up the alert display in the main browser window showing the device which caused the alert to trigger.

6.1) Configuring Alert Rules

(Note: The Device table optional fields "NotificationFieldInfo" must be enabled for the "Alert Monitor" panel to function properly)

The "Alert Monitor" panel can be configured in the "private_common.xml" file (or "private.xml" file, depending on the installation configuration). The "Alert Monitor" panel is defined in the "WebPages" tag section, and typically appears as follows:

```

<Page class="org.opengts.rulewar.track.page.AlertPanel" cssDir="extra/css"
  jsp="/extra/jsp/loginSession_panel.jsp"
  optional="true">
  <AclName>acl.admin.alertPanel</AclName>
  <Property key="ruleName"           >alert</Property>
  <Property key="pollInterval"       >10</Property>
  <Property key="alertText.off"      >Alert Monitor</Property>
  <Property key="alertText.on"       >Alert Detected</Property>
  <Property key="actionText.off"     >No Alerts Detected</Property>
  <Property key="actionText.on"     >Click for Alert Display</Property>
  <Property key="alertPageName"      >dev.alerts</Property>
  <Property key="bindToParentWindow">false</Property>
</Page>

```

The meaning of the various properties are as follows:

`ruleName`

Then name of the rule which will be periodically checked for the specific rule trigger. A rule with this name must exist on the "Rule Admin" page, otherwise this "Alert Monitor" page cannot properly identify an alert rule trigger condition.

`pollInterval`

This is the interval, in seconds, between successive rule trigger tests. The value entered here will be seen in the count-down timer on the "Alert Monitor" panel.

`alertText.off`

This is the text which is displayed in the "Alert Monitor" panel during a non-Alert condition.

`alertText.on`

This is the text which is displayed in the "Alert Monitor" panel during an Alert condition.

`alertPageName`

This is the page name used for the "Alert Monitor" panel within the GTS system.

`bindToParentWindow`

If true, indicates that the "Alert Monitor" windows should be bound to the parent (creating) browser window, and should be closed if the parent browser window is closed.

6.2) "Device Alerts" Panel

The "Device Alerts" panel shows all devices with active alerts. An active alert can be selected, and the "View Map" button can be pressed to immediately go to the Device Map for the selected device, showing the Device/Vehicle current location.

This page also allows clearing a selected Device alert, or clearing all active alerts for all Devices.

Appendices)

Appendix A) Internal Rule Variable Definitions

The Event notification rules engine pre-defines a set of constants/variables that can be used during the evaluation of a rule specification. Most of these variables obtain their value from the current Event. The following describes the available pre-defined constants and variables.

Notes:

- Boolean values represent either a Long '0' (False), or Long '1' (True).
- Variable names are case insensitive.
- Contact us regarding how to add new custom variables.

Name: account, accountid
Type: String
Description: Event account-id [as returned by "<EventData>getAccountID ()"]

Name: device, deviceid
Type: String
Description: Event device-id [as returned by "<EventData>getDeviceID ()"]

Name: statusCode, code
Type: Long
Description: Event statusCode [as returned by "<EventData>getStatusCode ()"]

Name: timestamp, time
Type: Long
Description: Event Epoch Timestamp [as returned by "<EventData>getTimestamp ()"]

Name: latitude, lat
Type: Double
Description: Event GPS Latitude [as returned by "<EventData>getLatitude ()"]

Name: longitude, lon
Type: Double
Description: Event GPS Longitude [as returned by "<EventData>getLongitude ()"]

Name: gpsAge, age
Type: Long
Description: Event GPS age (in seconds) [as returned by "<EventData>.getGpsAge ()"]

Name: speed, kph, speedKPH
Type: Double
Description: Event GPS speed (in km/h) [as returned by "<EventData>.getSpeedKPH ()"]

Name: speedmph, mph
Type: Double
Description: Event GPS speed (in mph)

Name: heading, head
Type: Double
Description: Event GPS heading (in degrees) [as returned by "<EventData>.getHeading ()"]

Name: altitude
Type: Double
Description: Event GPS altitude (in meters) [as returned by "<EventData>.getAltitude()"]

Name: distance, distkm
Type: Double
Description: Event distance (in Kilometers) [as returned by "<EventData>.getDistanceKM()"]

Name: odometer, odomkm
Type: Double
Description: Event odometer (in Kilometers) [as returned by "<EventData>.getOdometerKM()"]

Name: geozoneindex, zoneindex
Type: Long
Description: Event Geozone client-id [as returned by "<EventData>.getGeozoneIndex()"]

Name: geozone, geozoneid, zone, zoneid
Type: String
Description: Event Geozone ID [as returned by "<EventData>.getGeozoneID()"]

Name: j1708fault
Type: Long
Description: Event J1708 fault [from EventData field 'j1708Fault', if present]

Name: driver
Type: String
Description: Event 'Driver' [from EventData field 'driver', if present].

Name: trailer, entity
Type: String
Description: Event 'Entity' [as returned by "<EventData>.getEntity()"].

Name: subdivision, state
Type: String
Description: Event subdivision/state [as returned by "<EventData>.getSubdivision()"].

Name: input, inputMask
Type: Long
Description: Event input mask [as returned by "<EventData>.getInputMask()"].

Name: lastEvent
Type: Boolean
Description: True if this the last event in a list [as returned by "<EventData>.getIsLastEvent()"].

Name: pi
Type: Double
Description: The constant PI (3.14159...).

Name: true
Type: Boolean
Description: The constant True ('1').

Name: false
Type: Boolean
Description: The constant False ('0').

Name: sunday, sun
Type: Long
Description: The constant Day-Of-Week value '0'

Name: monday, mon
Type: Long
Description: The constant Day-Of-Week value '1'

Name: tuesday, tue
Type: Long
Description: The constant Day-Of-Week value '2'

Name: wednesday, wed
Type: Long
Description: The constant Day-Of-Week value '3'

Name: thursday, thu
Type: Long
Description: The constant Day-Of-Week value '4'

Name: friday, fri
Type: Long
Description: The constant Day-Of-Week value '5'

Name: saturday, sat
Type: Long
Description: The constant Day-Of-Week value '6'

Appendix B) Internal Rule Function Definitions

The Event notification rules engine pre-defines a set of functions that can be executed during the evaluation of a rule specification. The following describes the available pre-defined function.

Notes:

- Square brackets in the argument list indicate optional arguments.
- Timestamp are Long values which represent a Unix/Epoch time.
- Day-Numbers represent the number of days since the start of the Gregorian Calendar.
- Time-of-Day values represent the number of minutes since the previous Midnight.
- Timezones are based on the specified value for the current Account.
- Boolean values represent either a Long '0' (False), or Long '1' (True).
- If a function specifies no arguments, then the parenthesis "()" are optional.
- Function names are case insensitive.
- Any function may have at most 4 arguments.
- Contact us regarding how to add new custom functions.

Boolean Functions:

True:

Boolean \$TRUE ()
Return True (Long '1').

False:

Boolean \$FALSE ()
Return False (Long '0').

String Comparison Functions:

Case Insensitive String Comparison:

Boolean \$EQ(String A, String B)
Return True is String A is equals to B, ignoring case.

Case Insensitive String 'StartsWith':

Boolean \$SW(String A, String B)
Return True is String A starts with String B, ignoring case.

Case Insensitive String 'EndsWith':

Boolean \$EW(String A, String B)
Return True is String A ends with String B, ignoring case.

Case Insensitive String 'Contains':

Boolean \$CONTAINS(String A, String B)
Return True is String A contains String B, ignoring case.

Type Conversion Functions:

Double Conversion:

Double \$DOUBLE(String A, Double D)

Return A converted to a Double value, or return D if A cannot be converted to a Double.

Long Conversion:

Long \$LONG(String A, Long L)

Return A converted to a Long value, or return L if A cannot be converted to a Long.

Convert Long to Hex String:

String \$HEX(Long A)

Converts the A to a String representing the hex value (ie. 1234 ==> "0x04D2")

Mathematical Functions:

Minimum Value:

Double \$MIN(Double A, Double B [, Double C])

Return the smallest (minimum) value of the specified arguments.

Maximum Value:

Double \$MAX(Double A, Double B [, Double C])

Return the largest (maximum) value of the specified arguments.

Power:

Double \$POW(Double A, Double B)

Return the value of A raised to the B power

Square-Root:

Double \$SQRT(Double A)

Return the square-root of A.

Floating Point Rounding:

Long \$ROUND(Double A)

Return the value of A rounded to the nearest Long.

Floating Point Ceiling:

Long \$CEIL(Double A)

Return the value of A rounded up to the nearest Long.

Floating Point Floor:

Long \$FLOOR(Double A)

Return the value of A rounded down to the nearest Long.

Floating Point Truncation:

Long \$TRUNC(Double A)

Return the value of A truncated to a Long.

Random Number Generation:

Double \$RANDOM()

Return a random Double value between 0.0 and 1.0.

Double \$RANDOM(Double A)

Return a random Double value between 0.0 and A.

Range Check:

Boolean \$RANGE(Double A, Double B, Double C)
Return True iff A is between values B/C (inclusive).

Bitwise checking:

Boolean \$BIT(Long A, Long B)
Return true iff bit B in value A is set. [Equivalent to ((A & (1 << B)) != 0)]

Indexed Value:

Object \$INDEX(Long I, Object A0, Object A1 [,Object A2])
Returns A0 if I==0, or A1 if I==1, or A2 if I==2, etc.

Time Functions:

Current Time:

Timestamp \$NOW()
Return the current Timestamp (Long) value.

Convert Time to String Format:

String \$FMETIME(Long EPOCH [, String TIME_ZONE])
Converts and returns the specified EPOCH time value, with the optional TIME_ZONE specification, as a String value.

Time-Of-Day:

Long \$TOD()
Return the time-of-day (in minutes from midnight) for the current Event.
Long \$TOD(Timestamp T)
Return the time-of-day for the specified Timestamp T.
Long \$TOD(Long HOUR, Long MINUTE)
Return the time-of-day for the specified HOUR, MINUTE.

Time-Of-Day Range Check:

Boolean \$TODRANGE(Long FROM_HHMM, Long TO_HHMM)
Return True iff the current Event time is between the specified FROM_HHMM, TO_HHMM values.

Day-Of-Week:

Long \$DOW()
Return the day-of-week for the current Event (0=Sunday, ... 6=Saturday).
Long \$DOW(Timestamp T)
Return the day-of-week for the specified Timestamp T (0=Sunday, ... 6=Saturday).
Long \$DOW(Long DAY, Long MONTH [, Long YEAR])
Return the day-of-week for the specified DAY, MONTH, and YEAR values. If YEAR is not specified, the current event year will be used.

Day-Number:

Long \$DAY()
Return the day-number for the current Event.
Long \$DAY(Timestamp T)
Return the day-number for the specified Timestamp T.
Long \$DAY(Long DAY, Long MONTH [, Long YEAR])
Return the day-number for the specified DAY, MONTH, and YEAR values. If YEAR is not specified, the current event year will be used.

Weekday Check:**Boolean \$WEEKDAY ()**

Return True iff the current Event represents a Weekday (Monday through Friday).

Boolean \$WEEKDAY (Timestamp T)Return True iff the specified Timestamp **T** represents a Weekday (Monday through Friday).**Boolean \$WEEKDAY (Long DAY, Long MONTH [, Long YEAR])**Return True iff the specified **DAY**, **MONTH**, and **YEAR** is a Weekday (Monday through Friday). If **YEAR** is not specified, the current event year will be used.**Weekend Check:****Boolean \$WEEKEND ()**

Return True iff the current Event represents a Weekend (Saturday/Sunday).

Boolean \$WEEKEND (Timestamp T)Return True iff the specified Timestamp **T** represents a Weekend (Saturday/Sunday).**Boolean \$WEEKEND (Long DAY, Long MONTH [, Long YEAR])**Return True iff the specified **DAY**, **MONTH**, and **YEAR** is a Weekend (Saturday/Sunday). If **YEAR** is not specified, the current event year will be used.**WorkHours Check:****Boolean \$WORKHOURS ()**

Return True iff the current Event occurred during working hours. Working hours is defined by the runtime configuration properties "rule.workHours.XXX", where "XXX" represents the 3 letter day abbreviation. A missing day specification represents a non-working day. For example, to represent normal 9am to 5pm working hours, the following property specifications can be added to a runtime configuration file:

```
run.workHours.mon=09:00-17:00
run.workHours.tue=09:00-17:00
run.workHours.wed=09:00-17:00
run.workHours.thu=09:00-17:00
run.workHours.fri=09:00-17:00
```

Event Digital Input Functions:**Digital Input Check:****Boolean \$DIN ()**

Return True iff the current Event represents a digital-input status code.

Boolean \$DIN (Long INDEX [, Boolean STATE])Return True iff the current Event represents digital-input **INDEX**, with state **STATE** (if specified).

GeoZone Functions:

Geozone "Existence" Check:

Boolean \$ZONE(String GEOZONE_ID, Long SORT_ID)

Return True iff the specified Geozone GEOZONE_ID exists, with the specified SORT_ID.

Boolean \$ZONE(String GEOZONE_ID)

Return True iff the specified Geozone GEOZONE_ID exists (with any SORT_ID).

Geozone "Inclusion" Check:

Boolean \$INZONE()

Return True iff the current Event is inside any defined Geozone.

Boolean \$INZONE(String GEOZONE_ID)

Return True iff the current Event is inside the specified GEOZONE_ID. Returns False if the specified GEOZONE_ID does not exist.

Boolean \$INZONE(Double LATITUDE, Double LONGITUDE, Double RADIUS)

Return True iff the current Event is inside the specified LATITUDE, LONGITUDE, RADIUS.

Geozone Arrival:

Boolean \$ARRIVE()

Return True iff the current Event represents an arrival at any defined 'arrival' Geozone.

Boolean \$ARRIVE(String GEOZONE_ID)

Return True iff the current Event represents an arrival at the specified GEOZONE_ID.

Note: The "isArrivalZone" flag is ignored for an explicitly specified Geozone-ID.

Boolean \$ARRIVE(Double LATITUDE, Double LONGITUDE, Double RADIUS)

Return True iff the current Event represents an arrival at the specified LATITUDE, LONGITUDE, and RADIUS (in meters).

Notes:

This function first attempts to compare the last valid GPS location stored in the Device record to the GPS location of the current event, which avoids an additional EventData lookup. Otherwise, if the last valid GPS location is not set in the Device record, the EventData record previous to the current event will be read from the database, requiring another database lookup.

Geozone Departure:

Boolean \$DEPART()

Return True iff the current Event represents a departure from any defined 'departure' Geozone

Boolean \$DEPART(String GEOZONE_ID)

Return True iff the current Event represents a departure from the specified GEOZONE_ID.

Note: The "isDepartureZone" flag is ignored for an explicitly specified Geozone-ID.

Boolean \$DEPART(Double LATITUDE, Double LONGITUDE, Double RADIUS)

Return True iff the current Event represents a departure from the specified LATITUDE, LONGITUDE, and RADIUS (in meters).

Notes:

This function first attempts to compare the last valid GPS location stored in the Device record to the GPS location of the current event, which avoids an additional EventData lookup. Otherwise, if the last valid GPS location is not set in the Device record, the EventData record previous to the current event will be read from the database, requiring another database lookup.

Proximity:

Boolean \$NEAR(String DEVICE_ID, Double RADIUS)

Return True iff the Device represented by the current Event is within **RADIUS** meters of the specified **DEVICE_ID**.

Boolean \$NEAR(Double LATITUDE, Double LONGITUDE, Double RADIUS)

Return True iff the Device represented by the current Event is within **RADIUS** meters of the specified **LATITUDE, LONGITUDE**.

Notes:

If **DEVICE_ID** is specified, this function first attempts to use the last valid GPS location stored in the Device record for the specified **DEVICE_ID**. If the last valid location was not set for this device, then the EventData table is used to determine the last location of the specified device, requiring another database lookup.

GeoCorridor "Inclusion" Check:

Boolean \$INCORR()

Return True iff the current Event is inside the Device "Active" GeoCorridor.

Boolean \$INCORR(String GEOCORR_ID)

Return True iff the current Event is inside the specified **GEOCORR_ID**. Returns False if the specified **GEOCORR_ID** does not exist.

Distance:

Double \$DISTANCE(Double LATITUDE, Double LONGITUDE)

Return the distance (in meters) from the GPS location of the current Event, to the specified **LATITUDE, LONGITUDE**.

Double \$DISTANCE()

Return the distance (in meters) from the GPS location of the current Event, to the GPS location of the previous event (with a valid GPS location).

Event Time Functions:

Dormant Check:

Long \$DORMANT()

Return the number of seconds since the last 'Stop' event.

Boolean \$DORMANT(Long A)

Return True if the Device represented by the current Event has been dormant for at least **A** seconds.

Notes:

This function looks for a prior event with one of start, stop, or in-motion status codes, thus requiring an additional database lookup. This rule function is typically used for periodic server-side (ie 'cronjob') type analysis to determine if a vehicle has not reported for a certain length of time.

Elapsed Time:

Long \$ELAPSED([Long CODE [, Long CODE]])

Return the number of seconds since the last event represented by the specified status **CODE(s)**, or since the latest event if no **CODE** is specified.

Periodic Maintenance (distance):

Boolean \$MAINTKM(Integer INDEX [, Double INTERVAL])

Return True if the current event odometer is greater than, or equal to, the last maintenance odometer, for the specified index, plus the maintenance interval specified in the device record (or **INTERVAL** Kilometers, if specified).

Boolean \$MAINTKM()

Return True if the current event odometer is greater than, or equal to, the last maintenance odometer for any of the maintenance fields in the device record, plus the corresponding maintenance interval.

Event Speed Functions:

Exceeding the Posted Speed Limit:

Boolean \$SPEEDING()

Returns True if the current Event represents a speed that exceeds the posted speed limit. If the posted speed limit is not available, then this function returns False.

Boolean \$SPEEDING(Double DEFAULT_SPEED_KPH)

Returns True if the current Event represents a speed that exceeds the posted speed limit. If the posted speed limit is not available, then returns True if the Event speed is greater than the specified **DEFAULT_SPEED_KPH** speed value. If the posted speed limit is not available and **DEFAULT_SPEED_KPH** is '0', then this function returns False.

Boolean \$SPEEDING(Double DEFAULT_SPEED_KPH, Double OFFSET_SPEED_KPH)

Returns True if the current Event represents a speed that exceeds the posted speed limit plus the specified **OFFSET_SPEED_KPH**. If the posted speed limit is not available, then returns True if the Event speed is greater than the specified **DEFAULT_SPEED_KPH** speed value plus **OFFSET_SPEED_KPH**. If the posted speed limit is not available and **DEFAULT_SPEED_KPH** is '0', then this function returns False.

Notes:

To utilize this function effectively requires that the reverse-geocoding service used to provide address information also provide posted-speed-limit information as well.

Event OBD Functions:

J1708 MID/PID Check:

Boolean \$J1708PID(Long MID, Long PID [, Long FMI])

Return True iff the current Event represents a J1708 fault code with the specified **MID**, **PID** (and **FMI**, if specified).

J1708 MID/PID Check:

Boolean \$J1708SID(Long MID, Long SID [, Long FMI])

Return True iff the current Event represents a J1708 fault code with the specified **MID**, **SID** (and **FMI**, if specified).

Event Temperature Functions:

Temperature Check:

Boolean \$THERMO(Long INDEX)

Return True iff the temperature value at the specified **INDEX**, in the current Event, is valid.

Boolean \$THERMO(Long INDEX, Double LOW_C, Double HIGH_C)

Return True iff the current Event temperature value at index **INDEX** is within the specified range **LOW_C**, **HIGH_C** (in degrees Celsius)

Icon/Pushpin Functions:

Device Pushpin Icon ID:

String \$DEVICON()

Return the defined Pushpin Icon ID for the current Device, or an empty String if no Pushpin Icon ID has been defined for the current Device.

String \$DEVICON(String DEFAULT_ID)

Return the defined Pushpin Icon ID for the current Device, or the specified **DEFAULT_ID** if no Pushpin Icon ID has been defined for the current Device.

StatusCode Pushpin Icon ID:

String \$CODEICON()

Return the Pushpin Icon ID evaluated from the current StatusCode icon selector, or an empty String if no icon selector has been defined for the current StatusCode.

String \$CODEICON(String DEFAULT_ID)

Return the Pushpin Icon ID evaluated from the current StatusCode icon selector, or the specified **DEFAULT_ID** if no icon selector has been defined for the current StatusCode.

StatusCode, or Device, Pushpin Icon ID:

String \$ICON()

First attempt to return the Pushpin Icon ID evaluated from the current StatusCode icon selector. If no icon selector has been defined for the current StatusCode, then attempt to return the defined Pushpin Icon ID for the current Device. Otherwise return an empty String if no StatusCode icon selector, and no Device pushpin ID have been defined.

String \$ICON(String DEFAULT_ID)

First attempts to return the Pushpin Icon ID evaluated from the current StatusCode icon selector. If no icon selector has been defined for the current StatusCode, then attempt to return the defined Pushpin Icon ID for the current Device. Otherwise return the specified **DEFAULT_ID** if no StatusCode icon selector, and no Device pushpin ID have been defined.

Device Functions:

In Device Group:

Boolean \$INGROUP(String GROUP)

Return True if the Device represented by the current Event is contained by the specified **GROUP**.

Check Device 'lastNotifyTime':

Boolean \$DEVNOTIFY([Long AGE_SEC])

Return True if any device has a 'lastNotifyTime' within the last **AGE_SEC** seconds. If **AGE_SEC** is not specified, then return True if any device has a 'lastNotifyTime' greater than 0.

Account Functions:

Elapsed time since Last Login:

Long \$LASTLOGIN()

Return the elapsed number of seconds since a user has logged in to this Account.

Boolean \$LASTLOGIN(Long ELAPSED_SEC)

Return True if any user has logged-in to this Account within the last ELAPSED_SEC seconds..

Elapsed time since last Device Communication Server activity:

Long \$LASTDCS([String DCS_ID])

Return the elapsed number of seconds since the last communication was received for the specified DCS name, or for any DCS if the DCS_ID is not specified. This function checks all devices for the current account and returns the timestamp of the last communication as recorded by the "lastTotalConnectTime" field in the Device table. This function is useful in periodic "cron" checks for possible periods of device communication server inactivity for the current account.

Global System Functions:

These global system functions are intended to be called only within "Cron" based rules, and against the "sysadmin" account, for purposes of monitoring system-level flags and attributes.

GTS Installation Directory String:

String \$GTS_HOME()

Return a String representing the "\$GTS_HOME" environment variable, which should be pointing to the current GTS Installation directory. This function will return an empty String if not run against the SystemAdmin user.

Elapsed time since last Device Communication Server activity:

Long \$G_LASTDCS([String DCS_ID])

Return the elapsed number of seconds since the last communication was received for the specified DCS name, or for any DCS if the DCS_ID is not specified. This function checks all devices for all accounts and returns the timestamp of the last communication as recorded by the "lastTotalConnectTime" field in the Device table. If specified, the DCS_ID check for Device records which match on the "deviceCode" field (which indicates which Device Communication Server this device utilizes). This function is useful in periodic "Cron" checks for possible periods of Device Communication Server inactivity. This function will return False if not run in "Cron" mode, or if not run against the SystemAdmin user.

The Amount of Total Space (in Megabytes) on the Named Disk Partition:

Double \$G_TOTALDISK([String DIR])

Return the total amount of space available (in Megabytes) on the named disk partition (ie. "/tmp", "/usr/local", etc). If the directory DIR is not specified, then total-space of the partition on which the GTS installation resides will be returned (ie. "\$G_TOTALDISK(\$GTS_HOME)"). For the purpose of this function, a Megabyte is defined as 1024². This function will return False if not run in "Cron" mode, or if not run against the SystemAdmin user.

The Amount of Free Space (in Megabytes) on the Named Disk Partition:

Double \$G_FREEDISK([String DIR])

Return the amount of free-space available (in Megabytes) on the named disk partition (ie. "/tmp", "/usr/local", etc). If the directory DIR is not specified, then free-space of the partition on which the GTS installation resides will be returned (ie. "\$G_FREEDISK(\$GTS_HOME)"). For the purpose of this function, a Megabyte is defined as 1024². This function is useful for periodically checking for dwindling disk-space in a log file directory, etc. For instance, a selector which triggers when free-space on the "\$GTS_HOME" partition falls below 700Mb might be expressed as "(\$G_FREEDISK < 700.0)". To check when more than 80% of a disk partition has been consumed, the selector might be expressed as "(\$G_FREEDISK/\$G_TOTALDISK > 0.80)". This function will return False if not run in "Cron" mode, or if not run against the SystemAdmin user.

Check Host:Port Accessibility:

Boolean `$G_SOCKET(String HOST, Integer PORT [, Integer TIMEOUT_MS])`

Return True if a socket can be opened on the named host and port. If `TIMEOUT_MS` is not specified, a default timeout of 3000 milliseconds is assumed. The value for `TIMEOUT_MS` may not be larger than 7000 milliseconds. Note that this function only checks that a socket can successfully be opened on the named host and port. It does not check that any meaningful communication can be made over this host:port. This function will return False if not run in 'Cron' mode, or if not run against the SystemAdmin user.

Rule Evaluation Functions:

Evaluate Selector:

Object `$EVAL(String SELECTOR [, Object DEFAULT_VAL [, Boolean PRIOR_EVENT]])`

Evaluate the specified selector and returns the Object result. If unable to evaluate the specified selector (due to syntax errors, etc), then `DEFAULT_VAL` will be returned, or '0' if `DEFAULT_VAL` is not specified. If '`PRIOR_EVENT`' is specified and True, then the selector will be evaluated relative to the prior Event. (The returned value type may be Long, Double, or String).

Recursive Rule Evaluation:

Object `$RULE(String RULE_ID)`

Execute and return the Object result of the specified `RULE_ID`. (The returned value type may be Long, Double, or String).

Evaluate Selector from Runtime Property Value:

Object `$RTEVAL(String PROP_ID [, Object DEFAULT_VAL])`

Evaluate the specified selector found at the specified Runtime property, and returns the Object result. The Properties defined in the account specified Domain in the 'private.xml' file are checked first for a matching `PROP_ID` property key, if not found then the runtime 'default.conf' or 'webapp.conf' files are checked. If the property is still not found, then `DEFAULT_VAL` will be returned, if specified, otherwise '0' (false) will be returned.

Runtime Property Value:

String `$RTPROP(String PROP_ID [, String DEFAULT_VAL])`

Returns the Runtime property value of the specified `PROP_ID`. The Properties defined in the account specified Domain in the 'private.xml' file are checked first for a matching `PROP_ID` property key, if not found, then the runtime 'default.conf' or 'webapp.conf' files are checked. If the property is still not found, then `DEFAULT_VAL` will be returned, if specified, otherwise an empty String will be returned.

EventData Field Value:

Object `$EFLD(String FIELD_NAME [, Boolean PRIOR_EVENT])`

Returns the value of the specified EventData field. The return type may be one of Double, Long, or String. If the specified field does not exist, then an evaluation error will occur. If '`PRIOR_EVENT`' is specified and is True, then the field value of the prior Event will be returned.

Appendix C) Adding Your Own Custom Selector Functions

You can add your own custom functions to those already defined in the Appendix B above. The easiest way to accomplish this is to add a few lines to the "StartupInit.java" module, starting with some additional "import" statements:

```
import org.opengts.rule.selector.FunctionMap;
import org.opengts.rule.EventRuleFactory;
import org.opengts.rule.event.EventFunctionMap;
import org.opengts.rule.event.EventFunctionHandler;
```

The following adds a simple new function which takes the average of the 2 arguments. This code example should be added to the "addTableFactories" in the "StartupInit.java" module.

```
RuleFactory ruleFactory = Device.getRuleFactory();
if (ruleFactory instanceof EventRuleFactory) {
    EventRuleFactory erf = (EventRuleFactory)ruleFactory;
    // Add function "$AVG"
    erf.addFunction(new EventFunctionHandler("$AVG") {
        public String getUsage() { return "Double $AVG(Double A, Double B)"; }
        public String getDescription() { return "Average of A and B"; }
        public Object invokeFunction(FunctionMap fm, Object args[]) {
            // check args
            if ((args.length != 2) && !HasNumericArgs(args)) {
                return null; // invalid arguments
            }
            // return results
            double A = DoubleValue(args[0]);
            double B = DoubleValue(args[1]);
            return new Double((A + B) / 2.0);
        }
    });
    // Add other functions below
}
```

The example added function below show how you might check for the "age" of a GPS fixe being greater that the specified function argument:

```
erf.addFunction(new EventFunctionHandler("$GPSAGE") {
    public String getUsage() { return "Boolean $GPSAGE(Long A)"; }
    public String getDescription() { return "True if GPS age > A"; }
    public Object invokeFunction(FunctionMap fm, Object args[]) {
        // check args
        if ((args.length != 1) || !HasNumericArgs(args)) {
            return null; // invalid arguments
        }
        // get EventData
        EventData ed = ((EventFunctionMap) fm).getEventData();
        if (ed == null) {
            return LONG_FALSE; // Boolean false
        }
        // return results
        long A = LongValue(args[0]);
        return (A > ed.getGpsAge())? LONG_TRUE : LONG_FALSE;
    }
});
```

When defining a function within the context of the "EventHandler" method, there are several utility methods which may be useful:

```
protected static boolean IsNumericArg(Object arg)
```

Returns true if the specified "arg" is either instances of a Long or Double class.

```
protected static boolean HasNumericArgs(Object arg[])
```

Returns true if all arguments specified in the "args" array are either instances of a Long or Double class. This method returns True if "args" is null.

```
protected static double DoubleValue(Object arg)
```

Returns the "double" value of the specified numeric "arg", or "0.0", if "arg" is non-numeric.

```
protected static long LongValue(Object arg)
```

Returns the "long" value of the specified numeric "arg", or "0", if "arg" is non-numeric.

```
protected static int IntValue(Object arg)
```

Returns the "int" value of the specified numeric "arg", or "0", if "arg" is non-numeric.

```
protected static boolean BooleanValue(Object arg)
```

Returns the "boolean" value of the specified numeric "arg", or "0", if "arg" is non-numeric.

```
protected static boolean IsStringArg(Object arg)
```

Returns true if the specified "arg" is either instances of a String class.

```
protected static boolean HasStringArgs(Object arg[])
```

Returns true if all arguments specified in the "args" array are either instances of a String class. This method returns True if "args" is null.

```
protected static String StringValue(Object arg)
```

Returns the "String" value of the specified "arg".

The following constants are also provided within the "EventHandler" method context:

```
public static final Long LONG_TRUE = new Long(1L);
public static final Long LONG_FALSE = new Long(0L);
public static final Long LONG_ZERO = new Long(0L);
public static final Double DOUBLE_ZERO = new Double(0.0);
```

The "EventFunctionMap" instance also provides the following methods:

```
public EventData getEventData()  
    Returns the current "EventData" instance (if any).  
  
public Account getAccount()  
    Returns the current "Account" instance (if any).  
  
public String getAccountID()  
    Returns the current "AccountID" instance.  
  
public Account getDevice()  
    Returns the current "Device" instance (if any).  
  
public String getDeviceID()  
    Returns the current "DeviceID" instance.  
  
public BasicPrivateLabel getPrivateLabel()  
    Returns the current "BasicPrivateLabel" instance.  
  
public BasicPrivateLabel getPrivateLabel()  
    Returns the current "BasicPrivateLabel" instance.
```

Appendix D) Event EMail Subject/Body Variables

When using the "Rule Admin" page to define a rule, the notification email subject and message body can contain certain variables which will be replaced in the final message sent to the user. The following describes the available email subject/body variables that can be specified in the email message.

Notes:

- All variable specifications must be inclosed in `${...}`.
- Variable names are case insensitive.
- Returned values are based on the current Event. Obtaining returned values based on the Event prior to the current Event is possible by specifying the keyname preceded by a '#' character, as in "`#{#fullAddress}`".

Example Email Subject:

```
Arrival at terminal ${geozoneID}
```

Example Email Message Body:

```
Truck ${device} has arrived at terminal ${geozoneID}
```

Name:	<code>\${account}</code>
Description:	Event account description.
Example:	Acme Incorporated
Name:	<code>\${device}</code>
Description:	Event device description.
Example:	Taxi #123
Name:	<code>\${dateTime}, \${date}</code>
Description:	Event date/time displayed in the preferred format.
Example:	2008/11/01 12:53:01 PDT
Name:	<code>\${dateYear}, \${year}</code>
Description:	Event date year
Example:	2008
Name:	<code>\${dateMonth}, \${month}</code>
Description:	Event date month
Example:	11
Name:	<code>\${dateDay}, \${day}</code>
Description:	Event date day-of-month
Example:	17
Name:	<code>\${dateDow}, \${dayOfWeek}</code>
Description:	Event date day-of-week.
Example:	Sunday
Name:	<code>\${time}</code>
Description:	Event date time.
Example:	12:15:47
Name:	<code>\${status}</code>
Description:	Event status code description.
Example:	InMotion

Name: `${entityID}`
Description: Event tracked 'Entity' ID.
Example: trailer123

Name: `${entity}, ${entityDesc}`
Description: Event tracked 'Entity' description.
Example: T123

Name: `${deviceEntities}, ${deviceTrailers}`
Description: List of entities/trailers currently attached to the Event device.
Example: T123, T234

Name: `${geopoint}`
Description: Event GPS location.
Example: 39.12345, -142.12345

Name: `${latitude}`
Description: Event GPS latitude (full resolution)
Example: 39.123456789

Name: `${longitude}`
Description: Event GPS longitude (full resolution)
Example: -142.123456789

Name: `${speed}`
Description: Event speed in the account preferred units.
Example: 45.6 mph

Name: `${speedLimit}`
Description: Reverse-Geocoded Event posted speed limit in the account preferred units (if available).
Example: 65.0 mph

Name: `${direction}`
Description: Event direction (bearing) abbreviation.
Example: SE

Name: `${bearing}, ${heading}`
Description: Event bearing.
Example: 123.4

Name: `${odometer}`
Description: Event odometer value in account preferred units
Example: 12776.3 miles

Name: `${distance}`
Description: Event distance/tripometer value in account preferred units
Example: 976.3 miles

Name: `${altitude}, ${alt}`
Description: Event altitude value in account preferred units
Example: 1329 feet

Name: `${address}, ${fullAddress}`
Description: Reverse-Geocoded Event full address
Example: 123 Somewhere St, Anywhere, CA 12345 USA
Notes: May return an empty value if there is no active ReverseGeocodeProvider, or the ReverseGeocodeProvider does not support this field, or if the active ReverseGeocodeProvider is not a 'FastOperation' (ie. "isFastOperation()" returns "false"), which is likely the case if it needs to obtain an address using an Internet based service. In this case the rule triggered email is processed long before the address is returned by the reverse-geocoding service. To make sure that this address variable returns a non-blank value, make sure you use a reverse-geocode provider where "isFastOperation()" returns "true", which typically requires that the reverse geocoded address can be resolved using a locally available database (on the same computer as the server).

Name: `${street}, ${streetAddress}`
Description: Reverse-Geocoded Event street address (if available)
Example: 123 Somewhere St
Notes: May return an empty value if there is no active ReverseGeocodeProvider, or the ReverseGeocodeProvider does not support this field, or if the active ReverseGeocodeProvider is not a 'FastOperation' (ie. "isFastOperation()" returns "false"), which is likely the case if it needs to obtain an address using an Internet based service. In this case the rule triggered email is processed long before the address is returned by the reverse-geocoding service. To make sure that this address variable returns a non-blank value, make sure you use a reverse-geocode provider where "isFastOperation()" returns "true", which typically requires that the reverse geocoded address can be resolved using a locally available database (on the same computer as the server).

Name: `${city}`
Description: Reverse-Geocoded Event city address (if available)
Example: Anywhere
Notes: May return an empty value if there is no active ReverseGeocodeProvider, or the ReverseGeocodeProvider does not support this field, or if the active ReverseGeocodeProvider is not a 'FastOperation' (ie. "isFastOperation()" returns "false"), which is likely the case if it needs to obtain an address using an Internet based service. In this case the rule triggered email is processed long before the address is returned by the reverse-geocoding service. To make sure that this address variable returns a non-blank value, make sure you use a reverse-geocode provider where "isFastOperation()" returns "true", which typically requires that the reverse geocoded address can be resolved using a locally available database (on the same computer as the server).

Name: `${state}, ${province}`
Description: Reverse-Geocoded Event state/province address (if available)
Example: CA
Notes: May return an empty value if there is no active ReverseGeocodeProvider, or the ReverseGeocodeProvider does not support this field, or if the active ReverseGeocodeProvider is not a 'FastOperation' (ie. "isFastOperation()" returns "false"), which is likely the case if it needs to obtain an address using an Internet based service. In this case the rule triggered email is processed long before the address is returned by the reverse-geocoding service. To make sure that this address variable returns a non-blank value, make sure you use a reverse-geocode provider where "isFastOperation()" returns "true", which typically requires that the reverse geocoded address can be resolved using a locally available database (on the same computer as the server).

Name: `${postalCode}, ${zipCode}`
Description: Reverse-Geocoded Event postal code address (if available)
Example: 12345
Notes: May return an empty value if there is no active ReverseGeocodeProvider, or the ReverseGeocodeProvider does not support this field, or if the active ReverseGeocodeProvider is not a 'FastOperation' (ie. "isFastOperation()" returns "false"), which is likely the case if it needs to obtain an address using an Internet based service. In this case the rule triggered email is processed long before the address is returned by the reverse-geocoding service. To make sure that this address variable returns a non-blank value, make sure you use a reverse-geocode provider where "isFastOperation()" returns "true", which typically requires that the reverse geocoded address can be resolved using a locally available database (on the same computer as the server).

Name: `${subdivision}, ${subdiv}`
Description: Reverse-Geocoded Event country/state address (if available)
Example: US/CA
Notes: May return an empty value if there is no active ReverseGeocodeProvider, or the ReverseGeocodeProvider does not support this field, or if the active ReverseGeocodeProvider is not a 'FastOperation' (ie. "isFastOperation()" returns "false"), which is likely the case if it needs to obtain an address using an Internet based service. In this case the rule triggered email is processed long before the address is returned by the reverse-geocoding service. To make sure that this address variable returns a non-blank value, make sure you use a reverse-geocode provider where "isFastOperation()" returns "true", which typically requires that the reverse geocoded address can be resolved using a locally available database (on the same computer as the server).

Name: `${geozoneID}`
Description: Event arrival/departure Geozone ID (if any)
Example: home

Name: `${geozone}`
Description: Event arrival/departure Geozone Description (if any)
Example: Home Base

Name: `${batteryLevel}`
Description: Event battery level (if available)
Example: .12

Name: `${priorGeozoneID}`
Description: GeozoneID of previous event. Same as "\${#geozoneID}".
Example: home

Name: `${priorFullAddress}`
Description: Full address of previous event. Same as "\${#fullAddress}".
Example: 123 Somewhere St, Anywhere, CA 12345 USA