# GTS Enterprise
# Event Notification Rules Engine
# and
# Icon Selection

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, $INCORR function definition. | MDF |
| 0.1.7 | 2010/05/30 | Added several new "Global" functions. Added info on creating custom functions. Added information on "Alert Monitor" panel configuration. | MDF |
| 0.1.8 | 2010/11/29 | Added information regarding "Cron" rules. | MDF |
| 0.1.9 | 2010/12/31 | Updated "$SPEEDING" function | MDF |
| 0.1.10 | 2011/04/01 | Added "$OILDROP", "$CDEPART", "$LASTCONNECT" functions. Updated "Cron" task rule information. Updated "Rule Admin" field information. | MDF |
| 0.1.11 | 2011/05/15 | Added "$CARRIVE", "$IGNITION", "ignition", "lastIgnition". | MDF |
| 0.1.12 | 2011/06/23 | Added "$INPUT" function. Additional alias "$SCINP" added to "$DIN". Added "$SC" function. | MDF |
| 0.1.13 | 2011/10/03 | Added "$MAINHR" function description. Added "fuelLevel" variable description. Added Device var "lastInput" and "dcs". | MDF |
| 0.1.14 | 2011/12/31 | Added some example notification email messages | MDF |
| 0.1.15 | 2012/09/03 | Added "temperature" replacement field. Update "Alert Panel" info. Added "$IGNITIONON" and "$IGNITIONOFF" functions. Added "$ARRDEP" function. Added From/To time-of-day options to "$WORKHOURS". Added functions "$DEVRULE" and "$THERMALPROFILE". | MDF |
| 0.1.16 | 2013/01/02 | Updated "<RuleListener>.handleRuleNotification" definition. Changed "$PARKEDVIO" to "$PARKVIO". Added "eventJobNumber", "deviceJobNumber", "eventCostCenter". Added "$DFIELD(FIELD_NAME)". "$HASJOBNUMBER", "$JOBNUMBER". Added "$REMINDER" function and usage information. | MDF |
| 0.1.17 | 2013/09/30 | Added "$DISTANCE(String DEV_ID)", "$EXPIRE(Long A, Long B)", "$DRIVEREXPIRED", "lastIgnitionOnTime", "lastIgnitionOffTime". Added additional EventData field descriptions. Added "$ARRIVEP"/"$DEPARTP", "$FRAC", "$G_DISKUSAGE". | MDF |
| 0.1.18 | 2014/02/03 | Added "equipmentType", "equipmentStatus". Added "Predefined Actions" information (Appendix E) | MDF |
| 0.1.19 | 2014/09/23 | Added "$TripOdom", "$TripOdomTOD". Changed definition of "$InCorr", Added "$InActiveCorr", "driverLicExpire", "driverInsExpire", "deviceExpireTime", "$IdleSec", "$Idle", "temp0".."temp7" | MDF |
| 0.1.20 | 2014/12/26 | Added replacement variable note for adding a Google link. Added "$RegistrationExpire", "$Miles", "$Kilometers", "$Celsius", "$Fahrenheit", "coolantTempF", "oilTempF". | MDF |
| 0.1.21 | 2015/05/15 | Added "$PostedSpeeding". SECONDS_AGO option added to "$FuelDrop" and "$FuelDelta". Added "$G_LastDCSLog", "$Gforce", "$MetersPerSecondSquared" | MDF |
| 0.1.22 | 2016/??/?? | Added "$PasswordExpired(USER)", "$TcpSessionTime", "malfunctionLamp", "hasFaultCode", "$HasDriver" | MDF |

# GTS Enterprise Rules Engine

**Contents:**

# 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 'ruledb.jar' or 'rulewar.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 Long, Double, or 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 | Arithmentic "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 contatenation 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 sytax 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 areithmetic 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/geofence)
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 specifiaction 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. |
| `notifyDescrption` | - 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 thid 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 which triggered the rule
    *** @param device     The device which triggered the rule (may be null)
    *** @param event      The event which triggered the rule (may be null)
    *** @param isCronMode True if executed from the 'Cron' task,
    ***                   False is executed from a normal incoming event analysis.
    *** @param selector   The rule selector that triggered this notification
    *** @param rule       The rule record from which the selector was obtained. May
    ***                   be null if the selector was not provided by a Rule record.
    **/
    public void handleRuleNotification(
        Account account, Device device, EventData event,
        boolean isCronMode, 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 returns 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/geofence ID "home" (a geozone/geofence 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/geofence.

`(($DOW == MONDAY) && $INZONE("home"))`
Return True if the current Event timestamp represents Monday, and the GPS location is currently inside geozone/geofence ID "home"  (a geozone/geofence 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 that 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 >= 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/geofence, display the "yellow" pushpin,
else display the "green" pushpin.

# 5) "Rule Administration" Web Interface

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

*Note: Rules defined when logged in as the "System Administrator" (ie "sysadmin") will be available to other Accounts/Users when defining rules. These "System Administrator" pre-defined rules will available in a pull-down menu on the "Rule Admin" page in a field called "System Rule". If the user selected a predefined "System Rule", they do not need to enter a rule selector or email subject/message, instead the pre-defined values will be used.*

## 5.1) "Rule Admin" Fields

The "Rule Admin" 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 or id, such as "arriveHome" or "depart", and can be referenced in in other rule selectors using the $RULE function, such as $RULE("depart").
- **Notification EMail**
  The EMail address to which an email 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. (this field may be left blank).
- **System Rule**
  If displayed, this will include a list of rules which have been pre-defined by the System Administrator. Predefined rules will already specify the rule selector and email subject/message so that they do not need to be entered by the user on this rule definition page.
- **Active**
  Specifies whether this rule is active or inactive. An inactive rule will not be tested.
- **Is Cron Rule**
  This specifies whether a rule is to only be tested by a running "Cron" task (see "Cron" rule information below). A rule "tag" can be selected which indicates to the "Cron" task handler when the "Cron" rule should be checked (ie. "Hourly", "Daily", etc). A "No" selection indicates that this is not a "Cron" taks rule, and this rule can be used to check events as they are received from remote tracking devices.
- **Description**
  The description of the rule. This is only used for information purposes when viewing a list of defined rules. (this field should not be left blank).
- **Rule Selector**
  The actual rule sector. This contains the rule selector syntax defined elsewhere in this document. (this field should not be left blank).
- **Trigger Action**
  The displayed checkboxes allow selecting what type of action to perform when a rule has 'triggered' (ie. returned true) for a specific event. If "**Email**" is checked, then an email notification should be sent. If "**Listener**" is checked, then a callback to an installed custom rule trigger handler should be called. If "**Save/Alert**" is checked, then the rule trigger information is stored in the Device record and can then be used by the "View Alerts" page and "Alert Monitor" window. If "**Queue**" is checked, then the rule trigger information will be written to the "NotifyQueue" table (to be used by other legacy systems as needed).
- **Predefined Actions**
  If displayed, this contains a list of pre-defined actions (this field is still experimental and may not be fully implemented). (this field may be left blank).

- **EMail Subject**
  This is the subject line of the email which will be sent to the notification email addresses (combined from the account, device, and rule notification email addresses). This value may contain event variable specifications outlined in the "Event Email Subject/Body Variables" Appendix described below. The resolved subject line (with variables replaced with their current event value) will be used as the Email subject line.
- **EMail Message**
  This is the body/message of the email which will be sent to the notification email addresses (combined from the account, device, and rule notification email addresses). This value may contain event variable specifications outlined in the "Event Email Subject/Body Variables" Appendix described below. The resolved body/message (with variables replaced with their current event value) will be used as the Email body/message.
- **Notify Use Wrapper**
  *(if this field is displayed, selecting "false" is recommended)* If "true" is selected, 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.
- **Device ID (or "Vehicle ID")**
  This should specify the device to which this rule should apply. To apply to all devices, select "All Devices". A selection of "No" indicates that it should not apply to any device, and is typically reserved for special rules that may be run from a "Cron" task.
- **Status Code**
  This should specify the status code to which this rule should apply (only incoming events which match this selected status code will be applied to this rule). To apply to all status-codes, select "All Codes".

**5.2) 'private.xml' EMail Template.**

*(this section should only be used for special installations where an email template/wrapper is required. In general, it is highly recommended that the "Notify Use Wrapper" described above, and the template/wrapper "EventNotificationEMail" described below not be used).*

The "`private.xml`" (or "`private/private_common.xml`") file defines an EMail *template/wrapper* in the XML tag section "`EventNotificationEMail`" which will be used when the "Notify Use Wrapper" is selected (see section above). 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 replacement variables "`${subject}`" and "`${message}`" may also be used (which are resolved from the email subject/message fields specified on the the "Rule Admin" page).

The following is an example "`EventNotificationEMail`" tag entry that may be specified in the "`private.xml`" (or "`private/private_common.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:



The Alert Monitor periodically checks a rule named "alert" (as indicated in parenthesis after the "Alert Monitor" title).  When the rule selector defined for this rule triggers (ie. returns true), the alert becomes active and the "Alert Monitor" windows changes to red, as in the following:



Clicking on the displayed link ("Click for Alert Display"), will bring up the "View Alerts" page in the main browser window showing the device which caused the alert to trigger.  The "View Alerts" page will then also allow displaying map for the selected device causing the alert, or clearing the alert once it has been handled.


### 6.1) Configuring Alert Rules

The Device table optional fields "`NotificationFieldInfo`" must be enabled for the "Alert Monitor" panel to function properly.  These fields can be enabled in the "`config.conf`" (or other ".`conf`" file if this file does not exist) file by setting the following property value to true:

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

The "Alert Monitor" panel, and "Device Alerts" page, can be enabled by uncommenting/setting the following properties in the "`config.conf`" file:

```
Domain.WebPages.DeviceAlerts=true
Domain.WebPages.AlertPanel=true
```

The above configuration will be enough to display the "Alert Panel" and "Device Alerts" with their default configuration.  Additional configuration can be performed on the "Alert Panel" as necessary in the "`WebPages`" tag section of the "`private/private_common.xml`" file, 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
> This is the 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. To work with the "Device Alerts" panel, the rule selector for this rule should specify the value "($DEVNOTIFY)". This indicates that this rule evaluation should return true if any devices have an active alert (see "Device Alerts" below for more information on "active" alerts). This rule should also not be assigned to any devices (ie. The "Device ID" pull-down should specify "No Devices"). The default rule name is "alert".

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. This value should remain as "dev.alerts".

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) "View Device Alerts" Panel

The "View 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.

An "active" alert is one that has the "Save" trigger action selected on the "Rule Admin" page. The "Save" action indicates to the rule trigger processing that if the rule is triggered it should save the time of the trigger in the Device record. The "($DEVNOTIFY)" rule selector used in the "Alert Monitor" screen would then return true, indicating that a Device has received an active alert.

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

**6.3) Overview of the Device Alert Process**

With the above configuration, the device alert process then can work as follows:

- A Rule is defined that checks for a specific condition, and has the "**Save/Alert**" checkbox checked. The rule is then assigned to all devices in the account.
- An event arrives for the device that meets the specific condition checked by the rule. The rule is triggered and saves the alert condition in the Device record.
- The "**Alert Panel**" periodically checks the "**alert**" rule, which has the selector "`($DEVNOTIFY)`". The "`$DEVNOTIFY`" rule function checks for any devices which have their Alert status set.
- Having now found a Device that has its Alert status set, the Alert Panel now turns Red.
- The User/Operator can then click on the "Click for Alert Display" link to view any/all devices which have their alert status set. The User/Operator then has the option to view a map for the specific Device.
- The User/Operator can then clear the alert status for the device.
- The "**Alert Panel**" will then return to Green when the device alerts have been cleared.

# 7)  Running "Cron" Rule Checks

The GTS Enterprise includes a "Cron" (derived from the Greek word "chronos", meaning "time") rule-check scheduler which allows setting up rules which are checked periodically.  These rule checks can be used for checking conditions such as not having received communication from a specific device, or a specific device communication server, or checks for disk usage.

## 7.1) Creating a "Cron" scheduler configuration file

The Cron schedular requires a configuration file which tells it what types of jobs (tasks) to perform, and how often they should be performed.  This file is created as an XML file which contains directives to the "cron" scheduler. You can find some examples in the GTS installation "crontab" directory.  Here is an example showing the basic format of the file:

```
<Crontab
    timeZone="US/Pacific"              - Timezone
    interval="60"                      - Test for job tasks every 60 seconds
    autoReload="false"                 - Reload this config file if changed
    threadPoolSize="1">                - Number of allowed concurrent jobs
  <Job name="PeriodicMaintenance"      - Name of Job (Task)
    active="true">                     - This Job (Task) is active
    <Class>org.opengts.rule.RuleCron</Class>   - Name of Java class which runs Job
    <Method>cron</Method>              - Name of the Java class method to run
    <Arg>-debugMode</Arg>              - Run with 'debug' mode logging
    <Arg>-account=ALL</Arg>            - Scan through all accounts
    <Arg>-device=ALL</Arg>             - Scan through all devices in account
    <Arg>-rule=ALL</Arg>               - Test all "Cron" rules
    <Arg>-tag=daily</Arg>              - Test "Cron" rules with "daily" tag
    <When hour="1" minute="05"/>       - Test rule at 1:05 AM every day
  </Job>
</Crontab>
```

The above Cron shedular configuration indicates that the class "`org.opengts.rule.RuleCron`", method "`cron`", should be executed once every hour so that it may perform its assigned task.  This "`cron`" method then examines the specified arguments to determine which type of rule check to perform.  In this case, all "Cron" rules will be tested which match the tag "`daily`", and will be run against all devices for all accounts.  Multiple "`Job`" tags may be specified in the same "`Crontab`" configuration file.

The "`When`" tag indicates how often this rule-check should be performed.  The above example indicates that the rule-check should be performed when the hour of the day is "1", and the minute is "0" (ie. 1 AM).  The specific timezone is indicated on the "`Crontab`" tag "`timeZone`" attribute.

The "`When`" tag can also indicate that certain jobs/tasks should be executed on a more frequent interval.  Here are some examples of other possible interval specifications:

Run this task every 15 mintues of every hour:
```
<When hour="*" minute="*/15"/>
```

Run this task at 2 AM on the first day of every month:
```
<When monthDay="1" hour="2" minute="0"/>
```

Run this task at 3 AM every Sunday:
```
<When weekDay="sun" hour="3" minute="0"/>
```

The **Event Notification Rules Engine** comes with a pre-defined Cron "Job" tasks in the file "`crontab/crontab.xml`".  This file defines the following "`tag`" Job tasks:

- 30min (name "30MinuteCron") : Checks all cron rules which are defined with tag "30min", every 30 minutes.
- hourly (name "HourlyCron"):  Checks all cron rules which are defined with tag "hourly", once per hour.
- daily (name "DailyCron"):  Checks all cron rules which are defined with tag "daily", once per day.
- weekly (name "WeeklyCron"):  Checks all cron rules which are defined with tag "weekly", once per week.
- monthly (name "MonthlyCron"):  Checks all cron rules which are defined with tag "monthly", once per month.

## 7.2) Starting the "Cron" service

The "`cron`" service is started in much the same manner that is used to start other device communication servers, using the command "`runserver.pl`", with an additional argument indicating which "crontab" configuration file to load:

```
> cd $GTS_HOME
> bin/runserver.pl -s cron -- -crontab=./crontab/crontab.xml
```

Where "`./crontab/crontab.xml`" is the location and name of the Crontab configuration file to load and execute.   Change this crontab file to the name of the file that contains the Cron configuration you wish to run.

The logging output will be written to the file "`$GTS_HOME/logs/cron.log`".  Check this log file for information regarding executed rule checks, etc.

The following command can be used to see that the "Cron" process is running:

```
> cd $GTS_HOME
> bin/psjava

  PID  Parent  L Java class/jar
------ ------  - ----------------------------------------------------
 98096(     1) 1 /usr/local/GTS_2.2.8-B12/build/lib/cron.jar
```

This running cron process can be stopped with the following command:

```
> cd $GTS_HOME
> bin/runserver.pl -s cron -kill

Killing 'cron' PID:  98096
```

The "`cron`" scheduler can also be set to automatically start by adding the following line to the "`bin/serverList`" file (also requires operating system auto-start support):

```
execServer "Crontab" "cron" "${option}" "-crontab=${GTS_HOME}/crontab/crontab.xml"
```

Where "`${GTS_HOME}/crontab/crontab.xml`" is the location and name of the Crontab configuration file which will be loaded and executed.

### 7.3) Creating "Cron" rules

Only "Cron" rules will be tested within the "RuleCron" process.  To indicate that a rule is designated for "Cron" checking, set the "Is Cron Rule" field, on the Rule Admin page, to one of the defined tag names (ie. "30Min", "Hourly", "Daily", etc.).   Then set the rule selector and other fields to their appropriate values for your periodically checked rule.

---

**7.3.a) Note:**
If the "Is Cron Rule" field is not displayed on the Rule Admin page, then this field may be disabled in the "`private/private_common.xml`" file.   To enable this field, set the following property to true:
```
  <Property key="ruleInfo.showCronRules">true</Property>
```
Then rebuild and redeploy the "`track.war`" file.

---

When testing a "Cron" rule, any rule selector variable or function reference to an event will be referring to the last event received by the current Device.

Some of the available rule selector functions are only available for use in a "Cron" rule-checking context.  For instance, the "`$G_SOCKET`" function (which checks socket connectivity) will immediately return false if not executed within a "Cron" task.

Just as the rule actions are perform during incoming event rule testing, they will also be performed for periodic "Cron" rule testing.  So if an "Email" rule action has been selected for a "Cron" rule, the email will be sent as specified.


### 7.4) Setting Up an Example Periodic Maintenance Rule

A Periodic Maintenance notification rule can be set up to monitor a service interval set up for a device.   To determine when a periodic maintenance is due, the rule checks the current vehicle odometer against the odometer of the previous service, plus the periodic maintenance interval setting (ie. 4000 miles).  This section describes the steps required to set up a periodic maintenance due notification.

**Enabling the Device record "`MaintOdometerFieldInfo`" fields:**

To enable Vehicle periodic maintenance checking will first require that the optional "`MaintOdometerFieldInfo`" fields be added to the Device table (this is where the last maintenance service odometer and interval values are stored).   To enable the periodic maintenance fields, uncomment and set the following property to "`true`" in the "`config.conf`" file:

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

Then run the following command to update the tables with the new fields:

```
    %  cd $GTS_HOME
    %  bin/dbAdmin.pl -tables=ca
```

The "`track.war`" must also be rebuilt and redeployed, and any running DCS must be restarted after changing the runtime configuration.

*Note: If the optional "`MaintOdometerFieldInfo`" fields have already been added to your Device record, you may not see any indication of any new fields being added to the Device record.*

**Setting the Device record periodic maintenance values:**

After the addition of the Periodic Maintenance fields in the Device record, there will now be some additional fields displayed on the "Device Admin" (or "Vehicle Admin") page for setting the  vehicle's "Maintenance Interval".  Set this value to the distance interval (miles, kilometers, etc) that should be used for periodici maintenance due notification.

The "Reset Service" box can also be checked on this page to reset any periodic maintenance due notification.

**Creating the Periodic Maintenance rule:**

The "$MAINTKM" rule function checks a Device's odometer and periodic maintenance interval to determine is periodic maintenance is due.

For each account that should have periodic maintenance checking enabled, create a rule (named "maintenance", for example) that has a "($MAINTKM)" rule selector, and is designated with "is Cron Rule" tag of "Daily".  The "Daily" tag indicates to the running "Cron" task handler that this rule should be check daily (at night), and the periodic maintenance due notification should be sent at that time.  Set the email Notify Subject/Message to indicate to the recipients that periodic maintenance is due.

*Note: The rule can be created in the System Administrator account ("sysadmin") with the "($MAINTKM)" rule selector and email subject/message.  This pre-defined rule will then be available in a pull-down on the individual account Rule Admin page to select so the user does not need to enter the rule selector or email subject/message themselves.*

When a vehicle's periodic maintenance service becomes due (ie. the vehicle's odometer value has passed the periodic service interval), an email will be sent nightly until the periodic service has been reset on the "Device Admin" (or "Vehicle Admin") page.

**Creating a Periodic Maintenance "Reminder" rule:**

The "$REMINDER" rule function checks a Device's reminder interval to see if a general reminder message is due to be sent.  The Device "Reminder Interval" and "Reminder Message" are set on the Device/Vehicle Admin page.

The following steps will enable the ability to send periodic reminder messages:
- Make sure that the Device record "MaintOdometerFieldInfo" columns are enabled (see above).
- Enable the "Reminder" fields on the Device/Vehicle Admin and Rule Admin pages (in "config.conf"):
    Domain.Properties.deviceInfo.showReminderMessage=true
    Domain.Properties.ruleInfo.showPredefinedActions=true
- Make sure the "Cron" service is running (see above)..
- Create a reminder rule with the following attributes:

```
Selector           : $REMINDER
Is Cron Rule       : Hourly
Trigger Action     : Email
Prefedined Actions : resetReminder
Email subj/body    : ${reminderMessage}
```
  Through the "Cron" process, the above rule will check the "$REMINDER" function every hour.  When "$REMINDER" returns 'true', the email subj/body message will be sent (the "${reminderMessage}" variable in the email message/body will be replaced with the reminder message entered on the Device/Vehicle Admin page).  The "resetReminder" predefined action will then reset the reminder for the next interval.  (Note: without the "resetReminder" predefined-action, the reminder notification will continue to be sent every hour)
- Set the "**Reminder Interval**" and "**Reminder Message**" values on the Device/Vehicle Admin page.
  Acceptable values for the "**Reminder Interval**" include the following:
    Specific Date – "date:2012/12/25"
    Day of week abbrev – "sun", "mon", "tue", "wed", "thu", "fri", "sat"
    Month abbrev – "jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec"

# 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.


### Event Variables - General:

**Event Account-ID**
**Name:**        `account, accountid`
**Type:**        `String`
**Note:**        *As returned by "*`<EventData>.getAccountID()`*"*

**Event Device-ID**
**Name:**        `device, deviceid`
**Type:**        `String`
**Note:**        *As returned by "*`<EventData>.getDeviceID()`*"*

**Event Epoch Timestamp (Epoch)**
**Name:**        `timestamp, time`
**Type:**        `Long`
**Note:**        *As returned by "*`<EventData>.getTimestamp()`*"*

**Event StatusCode (see Status Code Description manual)**
**Name:**        `statusCode, code`
**Type:**        `Long`
**Note:**        *As returned by "*`<EventData>.getStatusCode()`*"*

**Event GPS Latitude (degrees)**
**Name:**        `latitude, lat`
**Type:**        `Double`
**Note:**        *As returned by "*`<EventData>.getLatitude()`*"*

**Event GPS Longitude (degrees)**
**Name:**        `longitude, lon`
**Type:**        `Double`
**Note:**        *As returned by "*`<EventData>.getLongitude()`*"*

**Event Valid GPS (boolean)**
**Name:**        `validgps`
**Type:**        `Boolean`
**Note:**        *As returned by "*`<EventData>.isValidGeoPoints()`*"*

**Event GPS Age, (seconds)**
**Name:**        `gpsAge, age`
**Type:**        `Long`
**Note:**        *As returned by "*`<EventData>.getGpsAge()`*"*

***Event GPS Speed, (km/h)***
***Name:*** **speed, kph, speedKPH**
***Type:*** **Double**
***Note:*** *As returned by "*`<EventData>.getSpeedKPH()`*"*

***Event GPS Speed, (mph)***
***Name:*** **speedmph, mph**
***Type:*** **Double**
***Note:*** *As return by "*`<EventData>.getSpeedKPH()`*", converted to mph.*

***Event GPS Bearing/Heading, (degrees)***
***Name:*** **heading, head**
***Type:*** **Double**
***Note:*** *As returned by "*`<EventData>.getHeading()`*"*

***Event GPS Altitude, (meters)***
***Name:*** **altitude**
***Type:*** **Double**
***Note:*** *As returned by "*`<EventData>.getAltitude()`*"*

***Event Input Mask***
***Name:*** **input, inputMask**
***Type:*** **Long**
***Note:*** *As returned by "*`<EventData>.getInputMask()`*"*

***Event Ignition State ("0" if Off, "1" if On, "-1" if unknown)*** *(Since v2.3.3-B19)*
***Name:*** ignition, ignitionState
***Type:*** **Long**
***Note:*** *Based on the Device ignition input specification. If Device ignition specification is set to look for Ignition On/Off status codes, this may cause another read on the EventData table to locate the previous ignition state change event.*

***Event Address***
***Name:*** **address**
***Type:*** **String**
***Note:*** *As returned by "*`<EventData>.getAddress()`*"*

***Event Geozone/Geofence client-id***
***Name:*** **geozoneindex, zoneindex**
***Type:*** **Long**
***Note:*** *As returned by "*`<EventData>.getGeozoneIndex()`*"*

***Event Geozone/Geofence ID***
***Name:*** **geozone, geozoneid, zone, zoneid**
***Type:*** **String**
***Note:*** *As returned by "*`<EventData>.getGeozoneID()`*"*

***Event Creation Age, (seconds)*** *(Since v2.4.3-B06)*
***Name:*** **creationAge, createAge**
***Type:*** **Long**
***Note:*** *As returned by "*<EventData>.getCreationAge()*"*

***Event Creation Time (Epoch)***
***Name:*** **creationTime**
***Type:*** **Long**
***Note:*** *As returned by "*<EventData>.getCreationTime()*"*

***Last Event   state (True if last in a list)***
***Name:*** **lastEvent, isLast, isLastEvent**
***Type:*** **Boolean**
***Note:*** *As returned by "*<EventData>.getIsLastEvent()*"*

***First Event state (True if first in a list)*** *(Since v2.3.3-B19)*
***Name:*** **firstEvent, isFirst, isFirstEvent**
***Type:*** **Boolean**
***Note:*** *As returned by "*<EventData>.getIsFirstEvent()*"*

## Event Variables – Odometer Fields:

***Event Odometer, (Kilometers)***
***Name:*** `odometer, odomkm`
***Type:*** `Double`
***Note:*** *As returned by "*`<EventData>.getOdometerKM()`*"*

***Event Odometer, (Miles)*** *(Since v2.5.5-B04)*
***Name:*** `odometerMiles, odomMiles`
***Type:*** `Double`
***Note:*** *As returned by "*`<EventData>.getOdometerKM() * GeoPoint.MILES_PER_KILOMETER`*"*

***Event Odometer Offset, (Kilometers)***
***Name:*** `odometerOffsetKM`
***Type:*** `Double`
***Note:*** *As returned by "*`<EventData>.getOdometerOffsetKM(null)`*"*

***Event Odometer Offset, (Miles)*** *(Since v2.5.6-B24)*
***Name:*** `odometerOffsetMiles`
***Type:*** `Double`

***Reported Odometer (including offset), (Kilometers)*** *(Since v2.5.6-B24)*
***Name:*** `reportOdometerKM`
***Type:*** `Double`

***Reported Odometer (including offset), (Miles)*** *(Since v2.5.6-B24)*
***Name:*** `reportOdometerMiles`
***Type:*** `Double`

***Event Distance, (Kilometers)***
***Name:*** `distance, distkm`
***Type:*** `Double`
***Note:*** *As returned by "*`<EventData>.getDistanceKM()`*"*

## Event Variables – Temperature Sensor Fields (if enabled):

***Event Temperature Sensor #0 (degrees Celcius)***
**Name:**          `temp0, temperature0`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getThermoAverage0()`".*

***Event Temperature Sensor #1(degrees Celcius)***
**Name:**          `temp1, temperature1`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getThermoAverage1()`".*

***Event Temperature Sensor #2 (degrees Celcius)***
**Name:**          `temp2, temperature2`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getThermoAverage2()`".*

***Event Temperature Sensor #3 (degrees Celcius)***
**Name:**          `temp3, temperature3`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getThermoAverage3()`".*

***Event Temperature Sensor #4 (degrees Celcius)***
**Name:**          `temp4, temperature4`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getThermoAverage4()`".*

***Event Temperature Sensor #5 (degrees Celcius)***
**Name:**          `temp5, temperature5`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getThermoAverage5()`".*

***Event Temperature Sensor #6 (degrees Celcius)***
**Name:**          `temp6, temperature6`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getThermoAverage6()`".*

***Event Temperature Sensor #7 (degrees Celcius)***
**Name:**          `temp7, temperature7`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getThermoAverage7()`".*

***Event Analog Sensor #0***
**Name:**          `analog0`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getAnalog0()`".*

***Event Analog Sensor #1***
**Name:**          `analog1`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getAnalog1()`".*

***Event Analog Sensor #2***
**Name:**          `analog2`
**Type:**          `Double`
**Note:**          *As returned by "`<EventData>.getAnalog2()`".*

***Event Analog Sensor #3***
***Name:*** **analog3**
***Type:*** **Double**
***Note:*** *As returned by "*<EventData>.getAnalog3()*".*

***Event Pulse Count***
***Name:*** **pulseCount**
***Type:*** **Double**
***Note:*** *As returned by "*<EventData>.getPulseCount()*".*

***Event Frequency Hz*** *(Since v2.6.2-B03)*
***Name:*** **frequencyHz, freqHz**
***Type:*** **Double**
***Note:*** *As returned by "*<EventData>.getFrequencyHz()*".*

***Event Frequency kHz*** *(Since v2.6.2-B03)*
***Name:*** **frequencyKHz, freqKHz**
***Type:*** **Double**
***Note:*** *As returned by "*<EventData>.getFrequencyKHz()*".*

***Event Frequency MHz*** *(Since v2.6.2-B03)*
***Name:*** **frequencyMHz, freqMHz**
***Type:*** **Double**
***Note:*** *As returned by "*<EventData>.getFrequencyMHz()*".*

***Event Frequency GHz*** *(Since v2.6.2-B03)*
***Name:*** **frequencyGHz, freqGHz**
***Type:*** **Double**
***Note:*** *As returned by "*<EventData>.getFrequencyGHz()*".*

## Event Variables – Extra Address Fields (if enabled):

*Event Street Address*
*Name:*        `streetAddress`
*Type:*        `String`
*Note:*        *As returned by "*`<EventData>.getStreetAddress()`*"*

*Event City*
*Name:*        `city`
*Type:*        `String`
*Note:*        *As returned by "*`<EventData>.getCity()`*"*

*Event State/Province*
*Name:*        `stateProvince`
*Type:*        `String`
*Note:*        *As returned by "*`<EventData>.getStateProvince()`*"*

*Event Postal Code*
*Name:*        `postalCode`
*Type:*        `String`
*Note:*        *As returned by "*`<EventData>.getPostalCode()`*"*

*Event Country*
*Name:*        `country`
*Type:*        `String`
*Note:*        *As returned by "*`<EventData>.getCountry()`*"*

*Event Subdivision/State*
*Name:*        `subdivision`, `state`
*Type:*        `String`
*Note:*        *As returned by "*`<EventData>.getSubdivision()`*"*

## Event Variables – GPS Info Fields (if enabled):

*Event Horizontal Accuracy*
*Name:*        horzAccuracy
*Type:*        Double
*Note:*        *As returned by "*<EventData>.getHorzAccuracy()*"*

*Event Vertical Accuracy*
*Name:*        vertAccuracy
*Type:*        Double
*Note:*        *As returned by "*<EventData>.getVertAccuracy()*"*

*Event Horizontal Dilution of Precision*
*Name:*        HDOP
*Type:*        Double
*Note:*        *As returned by "*<EventData>.getHDOP()*"*

*Event Satellite Count*
*Name:*        satelliteCount
*Type:*        Integer
*Note:*        *As returned by "*<EventData>.getSatelliteCount()*"*

*Event Battery Level (% 0.0..1.0)*
*Name:*        batteryLevel
*Type:*        Double
*Note:*        *As returned by "*<EventData>.getBatteryLevel()*"*

*Event Battery Voltage (Volts)*
*Name:*        batteryVolts
*Type:*        Double
*Note:*        *As returned by "*<EventData>.getBatteryVolts()*"*

*Event Signal Strength*
*Name:*        signalStrength
*Type:*        Double
*Note:*        *As returned by "*<EventData>.getSignalStrength()*"*


## Event Variables – Custom Info Fields (if enabled):

*Event Priority*
*Name:*        priority
*Type:*        Integer
*Note:*        *As returned by "*<EventData>.getPriority()*"*

*Event 'DriverID'*
*Name:*        driver, driverID
*Type:*        String
*Note:*        *As returned by "*<EventData>.getDriverID()*"*

*Event 'Driver' Description/Name*
*Name:*        driverName, driverDesc
*Type:*        String
*Note:*        *As returned by "*<EventData>.getDriver(true).getDescription()*"*

***Event 'Driver' badge ID*** *(Since v2.4.2-B15)*
**Name:**      `driverBadge`, `driverBadgeID`
**Type:**      `String`
**Note:**      *As returned by "*`<EventData>.getDriver(true).getBadgeID()`*"*


***Event 'Driver' license*** *(Since v2.4.2-B15)*
**Name:**      `driverLicense`
**Type:**      `String`
**Note:**      *As returned by "*`<EventData>.getDriver(true).getLicenseNumber()`*"*


***Event 'Driver' license expiration date*** *(Since v2.4.2-B15)*
**Name:**      `driverLicExpire`
**Type:**      `Long`
**Note:**      *As returned by "*`<EventData>.getDriver(true).getLicenseExpire()`*"*


***Event 'Entity' ID***
**Name:**      `trailer`, `entity`
**Type:**      `String`
**Note:**      *As returned by "*`<EventData>.getEntity()`*"*


***Event RFID Tag***
**Name:**      `rfidTag`
**Type:**      `String`
**Note:**      *As returned by "*`<EventData>.getRfidTag()`*".*


***Event Job Number*** *(Since v2.4.6-B29)*
**Name:**      `eventJobNumber`, `eventJobID`
**Type:**      `String`
**Note:**      *As returned by "*`<EventData>.getJobNumber()`*". (see also "*`deviceJobID`*")*


***Event Cost Center*** *(Since v2.4.6-B29)*
**Name:**      `costCenter`, `eventCostCenter`
**Type:**      `Long`
**Note:**      *As returned by "*`<EventData>.getCostCenter()`*".*


## Event Variables – OBD/J1939/CANBUS Fields (if enabled):


***Event Fuel Total (LItres)***
**Name:**      `fuelTotal`
**Type:**      `Double`
**Note:**      *As returned by "*`<EventData>.getFuelTotal()`*".*


***Event Engine RPM***
**Name:**      `engineRpm`
**Type:**      `Long`
**Note:**      *As returned by "*`<EventData>.getEngineRpm()`*".*


***Event Engine Hours (hours)***
**Name:**      `engineHours`
**Type:**      `Double`
**Note:**      *As returned by "*`<EventData>.getEngineHours()`*".*

***Event Engine Load (%: 0.0..1.0)***
*Name:*  **engineLoad**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getEngineLoad()`".


***Event Idle Hours (hours)***
*Name:*  **idleHours**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getIdleHours()`".


***Event Work Hours (hours)***
*Name:*  **workHours**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getWorkHours()`".


***Event Transmission Oil Temperature (degrees Celsius)***
*Name:*  **transOilTemp**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getTransOilTemp()`".


***Event Coolant Level (%: 0.0..1.0)***
*Name:*  **coolantLevel**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getCoolantLevel()`".


***Event Coolant Temperature (degrees Celsius)***
*Name:*  **coolantTemp**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getCoolantTemp()`".


***Event Coolant Temperature (degrees Fahrenheit)*** *(Since v2.5.8-B41)*
*Name:*  **coolantTempF**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getCoolantTemp()`", and converted to Fahrenheit.


***Event Brake G-Force (G-Force)***
*Name:*  **brakeGForce**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getBrakeGForce()`".


***Event Acceleration (Meters per Second per Second)***
*Name:*  **acceleration**
*Type:*  **Double**
*Note:*  As returned by "`<EventData>.getAcceleration()`".

### Event Oil Pressure (kPa)
*Name:*           **oilPressure**
*Type:*           **Double**
*Note:*           As returned by "*<EventData>.getOilPressure()*".

### Event Oil Level (%: 0.0..1.0)
*Name:*           **oilLevel**
*Type:*           **Double**
*Note:*           As returned by "*<EventData>.getOilLevel()*".

### Event Oil Temperature (degrees Celsius)
*Name:*           **oilTemp**
*Type:*           **Double**
*Note:*           As returned by "*<EventData>.getOilTemperature()*".

### Event Oil Temperature (degrees Fahrenheit) *(Since v2.5.8-B41)*
*Name:*           **oilTempF**
*Type:*           **Double**
*Note:*           As returned by "*<EventData>.getOilTemperature()*", *and converted to Fahrenheit.*

### Event PTO Engaged (boolean)
*Name:*           **ptoEngaged**
*Type:*           **Boolean**
*Note:*           As returned by "*<EventData>.getPtoEngaged()*".

### Event PTO Hours (Hours)
*Name:*           **ptoHours**
*Type:*           **Double**
*Note:*           As returned by "*<EventData>.getPtoHours()*".

### Event Throttle Position (%: 0.0..1.0)
*Name:*           **throttlePos**
*Type:*           **Double**
*Note:*           As returned by "*<EventData>.getThrottlePos()*".

### Event Brake Position (%: 0.0..1.0)
*Name:*           **brakePos**
*Type:*           **Double**
*Note:*           As returned by "*<EventData>.getBrakePos()*".

### Event J1708 fault
*Name:*           **j1708fault**
*Type:*           **String**
*Note:*           From EventData field "j1708Fault", *if present.*

### Event J1708/OBDII fault
*Name:*           **faultCode**
*Type:*           **String**
*Note:*           From EventData field "faultCode", *if present.*

### Event J1708/OBDII fault  *(Since v2.6.3-B55)*
*Name:*           **hasFaultCode**
*Type:*           **Boolean**
*Note:*           True if EventData record contains a fault-code.

### Event Malfunction Indicator Lamp (MIL)   *(Since v2.6.3-B55)*
*Name:*        **malfunctionLamp**
*Type:*        **Boolean**
*Note:*        As returned by "*`<EventData>.getMalfunctionLamp()`*".


### Event Vehicle Battery Volts (Volts)
*Name:*        **vBatteryVolts**
*Type:*        **Double**
*Note:*        As returned by "*`<EventData>.getVBatteryVolts()`*".


### Event Fuel Level – tank #1 (%: 0.0..1.0)
*Name:*        **fuelLevel**
*Type:*        **Double**
*Note:*        As returned by "*`<EventData>.getFuelLevel()`*".


### Event Fuel Level – tank #2 (%: 0.0..1.0) *(Since v2.6.1-B40)*
*Name:*        **fuelLevel2**
*Type:*        **Double**
*Note:*        As returned by "*`<EventData>.getFuelLevel2()`*".


### Event Fuel Pressure (kPa)
*Name:*        **fuelPressure**
*Type:*        **Double**
*Note:*        As returned by "*`<EventData>.getFuelPressure()`*".


### Event Fuel Temperature (degrees Celcius)
*Name:*        **fuelTemp**
*Type:*        **Double**
*Note:*        As returned by "*`<EventData>.getFuelTemp()`*".


### Event Brake Pressure (kPa)
*Name:*        **brakePressure**
*Type:*        **Double**
*Note:*        As returned by "*`<EventData>.getBrakePressure()`*".


### Event Mass Air Flow Rate (Grams/Second)
*Name:*        **massAirFlowRate**
*Type:*        **Double**
*Note:*        As returned by "*`<EventData>.getMassAirFlowRate()`*".

## Event Variables – Acceleration Fields:

***Acceleration (meters/second/second)*** *(Since v2.5.9-B48)*
| | |
|---|---|
| ***Name:*** | `acceleration` |
| ***Type:*** | `Double` |
| ***Note:*** | As returned by "`<EventData>.getAcceleration()`". |

***Impact Magnitude (meters/second/second)*** *(Since v2.5.9-B48)*
| | |
|---|---|
| ***Name:*** | `impactMagnitude` |
| ***Type:*** | `Double` |
| ***Note:*** | As returned by "`<EventData>.getImpactMagnitude()`".   Checks each of the following acceleration data items and returns the first defined magnitude value:  Maximum magnitude of impact data returned by "`<EventData>.getImpactData()`", magnitude of Accelerometer returned by "`<EventData>.getAccelerometer()`", value of Acceleration as returned by "`<EventData>.getAcceleration()`". Returns 0.0 if no magnitude value is defined or is invalid. |

***Accelerometer Magnitude (meters/second/second)*** *(Since v2.5.9-B48)*
| | |
|---|---|
| ***Name:*** | `accelerometerMagnitude, accelMagnitude` |
| ***Type:*** | `Double` |
| ***Note:*** | Returns the magnitude of the Accelerometer value as returned by "`<EventData>.getAccelerometer()`".  Returns 0.0 if no Accelerometer value is defined or is invalid. |

### Event Variables – Garmin/GFMI Fields (if enabled):

***Event ETA Timestamp (Epoch)***
***Name:***          `etaTimestamp`
***Type:***           `Long`
***Note:***          *As returned by "`<EventData>.getEtaTimestamp()`".*

***Event ETA ID***
***Name:***          `etaUniqueID`
***Type:***           `String`
***Note:***          *As returned by "`<EventData>.getEtaUniqueID()`".*

***Event ETA Distance KM (Kilometers)***
***Name:***          `etaDistanceKM`
***Type:***           `Double`
***Note:***          *As returned by "`<EventData>.getEtaDistanceKM()`".*

***Event ETA Latitude (degrees)***
***Name:***          `etaLatitude`
***Type:***           `Double`
***Note:***          *As returned by "`<EventData>.getEtaLatitude()`".*

***Event ETA Longitude (degrees)***
***Name:***          `etaLongitude`
***Type:***           `Double`
***Note:***          *As returned by "`<EventData>.getEtaLongitude()`".*

***Event Stop ID***
***Name:***          `stopID`
***Type:***           `String`
***Note:***          *As returned by "`<EventData>.getStopID()`".*

***Event Stop Status***
***Name:***          `stopStatus`
***Type:***           `Integer`
***Note:***          *As returned by "`<EventData>.getStopStatus()`".*

***Event Stop Index***
***Name:***          `stopIndex`
***Type:***           `Integer`
***Note:***          *As returned by "`<EventData>.getStopIndex()`".*

## Device Variables:

***Latest/Current Event Input Mask*** *(Since v2.3.8-B53)*
**Name:**     `lastInput, lastInputMask`
**Type:**     `Long`
**Note:**     *As returned by "*`<Device>.getLastInputState()`*"*

***Latest/Current Ignition State*** *(Since v2.3.3-B19)*
**Name:**     `lastIgnition, lastIgnitionState, currentIgnition`
**Type:**     `Long`
**Note:**     *As returned by "*`<Device>.getCurrentIgnitionState()`*"*

***Latest Ignition-On Epoch Time*** *(Since v2.5.1-B21)*
**Name:**     `lastIgnitionOnTime`
**Type:**     `Long`
**Note:**     *As returned by "*`<Device>.getLastIgnitionOnTime()`*"*

***Latest Ignition-Off Epoch Time*** *(Since v2.5.1-B21)*
**Name:**     `lastIgnitionOffTime`
**Type:**     `Long`
**Note:**     *As returned by "*`<Device>.getLastIgnitionOffTime()`*"*

***Device Communication Server Name*** *(Since v2.3.8-B53)*
**Name:**     `dcs, dcserver`
**Type:**     `String`
**Note:**     *As returned by "*`<Device>.getDeviceCode()`*"*

***Device Job Number*** *(Since v2.4.6-B29)*
**Name:**     `deviceJobNumber, deviceJobID`
**Type:**     `String`
**Note:**     *As returned by "*`<Device>.getJobNumber()`*". (see also "*`eventJobID`*")*

***Device Equipment Type*** *(Since v2.5.3-B03)*
**Name:**     `equipmentType, equipType`
**Type:**     `String`
**Note:**     *As returned by "*`<Device>.getEquipmentType()`*".*

***Device Equipment Status*** *(Since v2.5.3-B03)*
**Name:**     `equipmentStatus, equipStat`
**Type:**     `String`
**Note:**     *As returned by "*`<Device>.getEquipmentStatus()`*".*

***Device License Expiration Day Number*** *(Since v2.5.4-B29)*
**Name:**     `deviceLicExpire, deviceLicExpiration`
**Type:**     `Long`
**Note:**     *As returned by "*`<Device>.getLicenseExpire()`*".*

***Device Insurance Expiration Day Number*** *(Since v2.5.4-B29)*
**Name:**     `deviceInsExpire, deviceInsExpiration`
**Type:**     `Long`
**Note:**     *As returned by "*`<Device>.getInsuranceExpire()`*".*

### Device Expiration Epoch Timestamp *(Since v2.5.4-B48)*

*Name:*     **deviceExpireTime**
*Type:*     **Long**
*Note:*     *As returned by "*`<Device>.getExpirationTime()`*".*

### Device Command State Mask *(Since v2.5.4-B15)*

*Name:*     **commandStateMask, cmdState**
*Type:*     **Long**
*Note:*     *As returned by "*`<Device>.getCommandStateMask()`*".*

### Device Last Service Day Number *(Since v2.5.5-B14)*

*Name:*     **lastServiceDay, lastService**
*Type:*     **Long**
*Note:*     *As returned by "*`<Device>.getLastServiceDayNumber()`*".*

### Device Last Service Timestamp (Epoch) *(Since v2.5.5-B14)*

*Name:*     **lastServiceTime**
*Type:*     **Long**
*Note:*     *As returned by "*`<Device>.getLastServiceTime()`*".*

### Device Next Service Day Number *(Since v2.5.5-B14)*

*Name:*     **nextServiceDay, nexttService**
*Type:*     **Long**
*Note:*     *As returned by "*`<Device>.getNextServiceDayNumber()`*".*

### Device Next Service Timestamp (Epoch) *(Since v2.5.5-B14)*

*Name:*     **nexttServiceTime**
*Type:*     **Long**
*Note:*     *As returned by "*`<Device>.getNextServiceTime()`*".*

## Constants (Fixed Value):

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

***The constant True ('1')***
***Name:***      `true`
***Type:***      `Boolean`

***The constant False ('0')***
***Name:***      `false`
***Type:***      `Boolean`

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

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

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

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

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

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

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

# 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.
– The term "iff" use in the description of the rule functions means "if and only if".

## Boolean Functions:

*False:*
**Boolean $FALSE()**
   Return False (Long '0').

*True:*
**Boolean $TRUE()**
   Return True (Long '1').

## String Comparison Functions:

*Case Insensitive String 'Contains':*
**Boolean $Contains(String A, String B)**
   Return True iff String **A** contains String **B**, ignoring case.

*Case Insensitive String Comparison:*
**Boolean $EQ(String A, String B)**
   Return True iff String **A** is equals to **B**, ignoring case.

*Case Insensitive String 'EndsWith':*
**Boolean $EW(String A, String B)**
   Return True iff String **A** ends with String **B**, ignoring case.

*Case Insensitive String 'StartsWith':*
**Boolean $SW(String A, String B)**
   Return True iff String **A** starts with 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.

*Convert Long to Hex String:*
```
String $Hex(Long A)
```
    Converts the **A** to a String representing the hex value (ie. 1234 ==> "0x04D2")

*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.

## Mathematical Functions:

*Bitwise checking:*
```
Boolean $Bit(Long A, Long B)
```
    Returns true iff bit **B** in value **A** is set.  [Equivalent to (($A$ & (1 << $B$)) != 0)]

*Floating Point Ceiling:*
```
Long $Ceil(Double A)
```
    Returns the value of **A** rounded up to the nearest Long.

*Floating Point Floor:*
```
Long $Floor(Double A)
```
    Returns the value of **A** rounded down to the nearest Long.

*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.

*Minimum Value:*
```
Double $Min(Double A, Double B [, Double C])
```
    Returns the smallest (minimum) value of the specified arguments.

*Maximum Value:*
```
Double $Max(Double A, Double B [, Double C])
```
    Returns the largest (maximum) value of the specified arguments.

*Expired:*   *(Since v2.4.9-B10)*
```
Double $Expired(Long A, Long B)
```
    Returns True if ((A > 0) && (A < B)), False otherwise.  Used for comparing expiration dates or times against a fixed time, as in "**$EXPIRED(driverLicExpire,$DAY)**", or "**$EXPIRED(deviceExpireTime,$NOW)**".

*Power:*
```
Double $POW(Double A, Double B)
```
    Returns the value of **A** raised to the **B** power.

*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).


*Floating Point Rounding:*
`Long $Round(Double A)`
    Return the value of **A** rounded to the nearest Long.


*Square-Root:*
`Double $SQRT(Double A)`
    Return the square-root of **A**.


*Floating Point Truncation:*
`Long $Trunc(Double A)`
    Return the integer portion of **A** (ie. the value of **A** truncated to a Long).


*Floating Point Franction:* *(Since v2.5.1-B28)*
`Double $Frac(Double A)`
    Return the decimal portion of **A**.


*Convert G-Force to Meters/Second/Second:* *(Since v2.5.9-B48)*
`Double $MetersPerSecondSquared(Double G)`
    Returns the value of G-Force **G** converted to Meters/Second/Second.


*Convert Meters/Second/Second to G-Force:* *(Since v2.5.9-B48)*
`Double $GForce(Double MSS)`
    Returns the value of Meters/Second/Second **MSS** converted to G-Force.


*Convert Miles to Kilometers:* *(Since v2.5.8-B41)*
`Double $Kilometers(Double M)`
    Returns the value of Miles **M** converted to Kilometers.


*Convert Kilometers to Miles:* *(Since v2.5.8-B41)*
`Double $Miles(Double KM, Double B)`
    Returns the value of kilometers **KM** converted to Miles.


*Convert Fahrenheit to Celsius:* *(Since v2.5.8-B41)*
`Double $Celsius(Double F)`
    Returns the value of fahrenheit **F** converted to Celsius.


*Convert Celsius  to Fahrenheit:* *(Since v2.5.8-B41)*
`Double $Fahrenheit(Double C)`
    Returns the value of Celsius **C** converted to Fahrenheit.

## Time Functions:

### Day-Number:
```
Long $Day()
```
    Return the day-number for the current Event.
```
Long $Day(Timestamp T)
```
    Return the day-number for the specified Timestamp(Long) **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.

### 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(Long) **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.

### Convert Time to String Format:
```
String $FMTTIME(Long EPOCH [, String TIME_ZONE])
```
    Converts and returns the specified **EPOCH** time value, with the optional **TIME_ZONE** specification, as a String value.

### Current Time:
```
Timestamp $Now()
```
    Return the current Timestamp(Long) 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(Long) **T**.
```
Long $TOD(Long HOUR, Long MINUTE)
```
    Return the time-of-day for the specified **HOUR**, **MINUTE**.
```
Long $TOD(String HOUR_MINUTE)
```
    Return the time-of-day for the specified **HOUR_MINUTE** String, in the format "hh:mm" (as in "08:30").

### 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.

### *Weekday Check:*

`Boolean $Weekday()`

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

`Boolean $Weekday(Timestamp T)`

Return True iff the specified Timestamp(Long) `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 curent Event represents a Weekend (Saturday/Sunday).

`Boolean $Weekend(Timestamp T)`

Return True iff the specified Timestamp(Long) `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 curent Event occured 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.sun=
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
run.workHours.sat=
```

`Boolean $WorkHours(Long FROM_HHMM, Long TO_HHMM)` *(Since v2.4.3-B03)*

Return True iff the curent Event occured during the work hours as specified in the function arguments, excluding week-ends. This function performs the same operation as the following selector definition:

```
(!$WEEKEND && $TODRANGE(FROM_HHMM, TO_HHMM))
```

Which is defined as "if not the weekend, and the event time is within the specified time range". For example, to specified a 9am to 5pm workday, Monday through Friday, excluding weekends, the selector would be:

```
($WORKHOURS(0900,1700))
```

## Event General Functions:

***EventData Field Value:***
`Object $EField(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.

***Previous EventData Field Value:*** *(Since v2.5.9-B12)*
`Object $PEField(String FIELD_NAME)`
    Returns the value of the specified EventData field for the most recent previous event.  The return type may be one of Double, Long, or String.  If the specified field does not exist, then an evaluation error will occur.
`Object $PEField(String FIELD_NAME, Long SECONDS_AGO [, Boolean AFTER_TS [, Boolean`
    `VALID_GPS]])`
    Returns the value of the specified EventData field for the previous event that occurred <= `SECONDS_AGO` if `AFTER_TS` is False, or >= `SECONDS_AGO` if `AFTER_TS` is True, relative to the current event.  If `SECONDS_AGO` is <= 0, then the most recent previous event will be used.  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 `VALID_GPS` is specified and is True, then the retrieved previous event will also have a valid GPS location.


## Event Digital Input Functions:

***Digital Input "Status Code" Check:*** *(Updated v2.3.4-B39, was "$DIN")*
`Boolean $SCInp()`
    Return True iff the current Event represents a digital-input status code.
`Boolean $SCInp(Long INDEX [,Boolean STATE])`
    Return True iff the current Event represents a digital-input `INDEX` (0-based) status code, with state `STATE` (if specified).  See "StatusCodes.pdf" for a list of digital input status codes.

***Digital Input "Input Mask" Check:*** *(Since v2.3.4-B39)*
`Boolean $Input()`
    Return True iff any "inputMask" digital-input bit is set (true) for the current Event.
`Boolean $Input(Long INDEX)`
    Return True iff the specified "inputMask" digital-input bit `INDEX` (0-based) is set (true) for the current Event.

***Ignition-On "Status Code" Check:*** *(Since v2.5.9-B14)*
`Boolean $SCIgnOn()`
    Return True iff the current Event represents a ignition-on status code.

***Ignition-Off "Status Code" Check:*** *(Since v2.5.9-B14)*
`Boolean $SCIgnOff()`
    Return True iff the current Event represents a ignition-off status code.

***Ignition State Check:***
`Long $Ignition()`
    Return "0" if the latest ignition state is "OFF", "1" if the latest ignition state is "ON", or "-1" if the latest ignition state is unknown..
`Long $Ignition(Boolean LAST)`
    Return "`$IGNITION()`" if `LAST` is True, otherwise attempt to determine the ignition state at the time of the current event (may cause another EventData table read if the Device ignition is specified to use Ignition On/Off status codes).

*Elapsed Ignition-ON Time Check:* *(Since v2.4.3-B02)*
`Boolean $IgnitionOn(Long ELAPSED_SECONDS)`
Return True if the ignition state has been ON for at least `ELAPSED_SECONDS`.

*Elapsed Ignition-OFF Time Check:* *(Since v2.4.3-B02)*
`Boolean $IgnitionOff(Long ELAPSED_SECONDS)`
Return True if the ignition state has been OFF for at least `ELAPSED_SECONDS`.

## Event StatusCode Functions:

*"Status Code" List Match:* *(Since v2.3.5-B09)*
`Boolean $SC(Long A [, Long B [, Long C [, Long D]]])`
Return True iff the current Event represents a status code that matches one of the codes specified within the argument list.

## Event Geozone/GeoCorridor/Geofence Functions:

### Geozone/Geofence "Existence" Check:
`Boolean $Zone(String ZONE_ID, Long SORT_ID)`
   Return True iff the specified geozone/geofence `ZONE_ID` exists, with the specified `SORT_ID`.
`Boolean $ZONE(String ZONE_ID)`
   Return True iff the specified geozone/geofence `ZONE_ID` exists (with any `SORT_ID`).

### Geozone/Geofence Arrival:
`Boolean $Arrive()`
   Return True iff the current Event represents an arrival at any defined 'arrival' geozone/geofence.
`Boolean $Arrive(String ZONE_ID)`
   Return True iff the current Event represents an arrival at the specified `ZONE_ID`.
   Note: The "isArrivalZone" flag is ignored for an explicitly specified geozone/geofence 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.
   - If the current event contains status code `STATUS_GEOFENCE_DEPART`, this function will return false.  Thie prevents having arrival rule notifications for departure events.  *(Since v2.5.8-B60)*

### Geozone/Geofence Departure:
`Boolean $Depart()`
   Return True iff the current Event represents a departure from any defined 'departure' geozone/geofence.
`Boolean $Depart(String ZONE_ID)`
   Return True iff the current Event represents a departure from the specified `ZONE_ID`.
   Note: The "isDepartureZone" flag is ignored for an explicitly specified geozone/geofence 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.
   - If the current event contains status code `STATUS_GEOFENCE_ARRIVE`, this function will return false.  Thie prevents having departure rule notifications for arrival events.  (Since v2.5.8-B60)

### Geozone/Geofence Arrival/Departure: *(Since v2.4.3-B03)*
**Boolean $ArrDep()**
> Return True iff the current Event represents an arrival or departure from any defined 'departure' geozone/geofence.

**Boolean $ArrDep(String ZONE_ID)**
> Return True iff the current Event represents an arrival or departure from the specified **GZONE_ID**.
> Note: The "isArrivalZone"/"isDepartureZone" flags are ignored for an explicitly specified geozone/geofence ID.

**Boolean $ArrDep(Double LATITUDE, Double LONGITUDE, Double RADIUS)**
> Return True iff the current Event represents an arrival or 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.
> - If the current event contains status code **STATUS_GEOFENCE_ARRIVE**, and the current GeoZone was a depart, or conversely, if the current event contains status code **STATUS_GEOFENCE_DEPART**, and the current GeoZone was an arrive, this function will return false.  Thie prevents having departure/arrival rule notifications for arrival/departure events.  (Since v2.5.8-B60)

### Geozone/Geofence Arrival, by Geozone Purpose: *(Since v2.5.1-B26)*
**Boolean $ArriveP(String PURPOSE_ID)**
> Return True iff the current Event represents an arrival at a zone with the specified **PURPOSE_ID**.

**Notes:**
> - The "isArrivalZone" flag is ignored for this function.
> - If the current event contains status code **STATUS_GEOFENCE_DEPART**, this function will return false.  Thie prevents having arrival rule notifications for departure events. *(Since v2.5.8-B60)*

### Geozone/Geofence Departure, by Geozone Purpose: *(Since v2.5.1-B26)*
**Boolean $DepartP(String PURPOSE_ID)**
> Return True iff the current Event represents an departure from a zone with the specified **PURPOSE_ID**.

**Notes:**
> - The "isDepartureZone" flag is ignored for this function.
> - If the current event contains status code **STATUS_GEOFENCE_ARRIVE**, this function will return false.  Thie prevents having departure rule notifications for arrival events. *(Since v2.5.8-B60)*

### Geozone/Geofence "Inclusion" Check:
**Boolean $InZone()**
> Return True iff the current Event is inside any defined geozone/geofence.

**Boolean $InZone(String ZONE_ID)**
> Return True iff the current Event is inside the specified **ZONE_ID**.  Returns False if the specified **ZONE_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**.

### GeoCorridor/Geofence "Inclusion" Check:
**Boolean $InCorr()**
> Return True iff the current Event location is inside any geocorridor, false otherwise.

**Boolean $InCorr(String CORR_ID)**
> Return True iff the current Event location is inside the specified **CORR_ID**.  Returns False if the specified **CORR_ID** does not exist.

### GeoCorridor/Geofence "Inclusion" Check:
**Boolean $InActiveCorr()**
> Return True iff the current Event location is inside the Device "Active" geocorridor, false otherwise.

### GeoCorridor/Geofence "Arrive" Check:
**Boolean $CArrive()**

Return True if the current Event location is inside the Device "Active" geocorridor/geofence, and the previous event location was outside the Device "Active" geocorridor/geofence. (producing a rule 'trigger' only upon the transition of the vehicle travelling from outside the geocorridor/geofence to inside the geocorridor/geofence).

**Boolean $CArrive(String CORR_ID)**

Return True iff the current Event location is inside the specified **CORR_ID**, and the previous event location was outside the specified **CORR_ID**. Returns False if the specified **CORR_ID** does not exist. (producing a rule 'trigger' only upon the transition of the vehicle travelling from outside the named geocorridor/geofence to inside the named geocorridor/geofence).

### GeoCorridor/Geofence "Depart" Check:
**Boolean $CDepart()**

Return True if the current Event location is outside the Device "Active" geocorridor/geofence, and the previous event location was inside the Device "Active" geocorridor/geofence. (producing a rule 'trigger' only upon the transition of the vehicle travelling from inside the geocorridor/geofence to outside the geocorridor/geofence).

**Boolean $CDepart(String CORR_ID)**

Return True iff the current Event location is outside the specified **CORR_ID**, and the previous event location was inside the specified **CORR_ID**. Returns False if the specified **CORR_ID** does not exist. (producing a rule 'trigger' only upon the transition of the vehicle travelling from inside the named geocorridor/geofence to outside the named geocorridor/geofence).

### Vehicle "Parked" Check:
**Boolean $IsParked()**

Return True if the current Device has a valid "`parkedLatitude`", "`parkedLongitude`", and "`parkedRadius`" value. These fields are typically set by the Predefined-Actions "`park`" command to indicate that a temporary geozone/geofence should be placed around the current vehicles location. The Predefined-Action "`unpark`" clears the "parked" state. To enable the "Predefined Actions" field on the "Rule Admin" page, uncomment/set the property "`Domain.Properties.ruleInfo.showPredefinedActions=true`" in the "`config.conf`" file.

### Vehicle "Parked" Violation:
**Boolean $ParkVio()**

Return True if the current Device is "parked" (as defined above) and the current event indicates a location that is outside the vehicles parked "radius". A typical used of this function may include setting the "parked" state when the ignition is turned off (and "unparked" state when the ignition is turned on), and using this function to determine that the vehicle has been moved/towed while the ignition was still turned off.

## Event Distance Functions:

### *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).

`Double $Distance(String DEVICE_ID)`    *(Since v2.4.8-B31)*

    Return the distance (in meters) from the GPS location of the current Event, to the last known location for the specified `DEVICE_ID`.

### *Trip distance travelled in the last specified number of seconds:*    *(Since v2.5.4-B24)*
`Double $TripOdom(Long SECONDS_AGO)`

    Return the distance travelled (in kilometers) in the last specified number of seconds.  The EventData records for the current device must have a valid Odometer value.

### *Trip distance travelled since previous time-of-day (TOD):*    *(Since v2.5.4-B24)*
`Double $TripOdomTOD(String HOUR_MINUTE)`

    Return the distance travelled (in kilometers) since the specified time-of-day on the previous day.  For example, to calculate the distance travelled since 8am the previous day, you could specify `($TripOdomTOD("08:00"))`. The EventData records for the current device must have a valid Odometer value.

### *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.

## 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 (based on the vehicle "Odometer"):
`Boolean $MaintKM(Integer INDEX [, Double INTERVAL])`
> Return True if the current event odometer value is greater than, or equal to, the last maintenance odometer value, for the specified index, plus the maintenance interval distance specified in the device record (or `INTERVAL` Kilometers, if specified).

`Boolean $MaintKM()`
> Return True if the current event odometer value is greater than, or equal to, the last maintenance odometer value for any of the maintenance fields in the device record, plus the corresponding maintenance interval distance.

### Kilometers Until Next Periodic Maintenance (based on the vehicle "Odometer"): *(Since v2.5..5-B14)*
`Double $NextMaintKM(Integer INDEX)`
> Return the number of kilometers until the next odometer maintenance is due, for the specified index. Returns zero if the odometer-based mantenance is past due.

`Double $NextMaintKM()`
> Return the number of kilometers until the next/nearest odometer maintenance is due (the smallest pending odometer-based maintenance). Returns zero if any odometer-based mantenance is past due.

### Days Until the Next Service is Due (based on the vehicle "Next Service Day"): *(Since v2.5..5-B14)*
`Long $NextService()`
> Return the number of days until the next service is due. Returns zero if the Servicee is past due.

### Next Service is Now Due (based on the vehicle "Next Service Day"): *(Since v2.5..5-B14)*
`Long $ServiceExpired()`
> Return true if the next service is due., false otherwise.

### Stopped for a specified amount of time: *(Since v2.5.2)*
`Boolean $Stopped(Long SEC)`
> Return True if the vehicle has been stopped for at least the specified number of seconds.

`Boolean $Stopped()`
> Return True if the vehicle is currently stopped.

### Elapsed stopped time: *(Since v2.5.2)*
`Long $StopSec()`
> Return the number of seconds that the vehicle has been stopped, or "0" if the vehicle is not stopped.

### Idle for a specified amount of time: *(Since v2.5.7-B36)*
`Boolean $Idle(Long SEC)`
> Return True if the vehicle has been idle (ignition-on and stopped) for at least the specified number of seconds. Requires ability to detect ignition and motion state.

`Boolean $Idle()`
> Return True if the vehicle is currently idle. Requires ability to detect ignition and motion state.

### Elapsed idle time: *(Since v2.5.7-B36)*
`Long $IdleSec()`
> Return the number of seconds that the vehicle has been idle (ignition-on and stopped), or "0" if the vehicle is not idle. Requires ability to detect ignition and motion state.

### Periodic Maintenance (based on the vehicle "Engine Hours"):
**`Boolean $MaintHR(Integer INDEX [, Double INTERVAL])`**

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

**`Boolean $MaintHR()`**

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

### Periodic Maintenance Reminder: (Since v2.4.3-B18)
**`Boolean $Reminder()`**

Return True if the current device has specified a reminder interval that has expired..

## Event Speed Functions:

*To utilize these functions effectively may require that the reverse-geocoding service used to provide address information also provide posted-speed-limit information as well.*

### Exceeding the Speed Limits:
**`Boolean $Speeding()`**

Returns True if the current Event represents a speed that exceeds the set Device maximum speed, or exceeds the posted speed limit (if available). If the maximum Device speed has not been set, and if the posted speed limit is not available, then this function returns False.

**`Boolean $Speeding(Double DefaultSpeedKPH)`**

Returns True if the current Event represents a speed that exceeds the set Device maximum speed, or exceeds the posted speed limit (if available). If the posted speed limit is not available, then returns True if the Event speed is greater than the specified **`DefaultSpeedKPH`** speed value. If the maximum Device speed has not been set, and if the posted speed limit is not available and **`DefaultSpeedKPH`** is '0', then this function returns False.

**`Boolean $Speeding(Double DefaultSpeedKPH, Double OffsetSpeedKPH)`**

Returns True if the current Event represents a speed that exceeds the set Device maximum speed, or exceeds the posted speed limit plus the specified **`OffsetSpeedKPH`**. If the posted speed limit is not available, then returns True if the Event speed is greater than the specified **`DefaultSpeedKPH`** speed value plus **`OffsetSpeedKPH`**. If the maximum Device speed has not been set, and if the posted speed limit is not available and **`DefaultSpeedKPH`** is '0', then this function returns False.

**`Boolean $Speeding(Double DefaultSpeedKPH, Double OffsetSpeedKPH, Boolean CheckZones)`**

Returns True if the current Event represents a speed that exceeds the set Device maximum speed, or exceeds the posted speed limit plus the specified **`OffsetSpeedKPH`**. If the posted speed limit is not available, then returns True if the Event speed is greater than the specified **`DefaultSpeedKPH`** speed value plus **`OffsetSpeedKPH`**. If **`CheckZones`** is True, and the vehicle is curently in a geozone/geofence, then the speed limit of the current geozone/geofence speed limit will also be checked. If the maximum Device speed has not been set, and if the posted speed limit is not available, and **`DefaultSpeedKPH`** is '0', and **`CheckZones`** is false or the vehicle is not in a geozone/geofence, then this function returns False.

### Exceeding the Posted Speed Limit:
**`Boolean $PostedSpeeding()`**

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

**`Boolean $PostedSpeeding(Double OffsetSpeedKPH)`**

Returns True if the current Event represents a speed that exceeds the posted speed limit (if available) by more than the specified km/h offset. If the posted speed limit is not available, then this function returns False.

## Event OBD Functions:

*These functions require that the remote GPS tracking/telematic device supports sending J1708 fault codes to the device communication server.*

### 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:

*These functions require that the remote GPS tracking/telematic device supports sending auxiliary temperature information to the device communication server.*

### 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)`

    Return True iff the current Event temperature value at index **INDEX** is above, or equal-to **LOW_C** (in degrees Celsius)

`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)

### Temperature Profile Check: *(Since v2.4.5-B19)*

`Boolean $ThermalProfile([Boolean DEFAULT_VAL])`

    Return True if the available event temperature values are within the specified temperature ranges found in the Device record "**thermalProfile**" field.

**Notes:**

– The Device "**thermalProfile**" field is part of the optional "**NotificationFieldInfo**" Device table field set.  See the GTS FAQ page for more information on enabling optional fields.  The "**thermalProfile**" value is specified as a space separated list of values of the form "**T#=LOWC,HIGHC**", where "**#**" represents the temperature probe index (0..7), and "**LOWC**" and "**HIGHC**" prepresent the low and high temperature range values respecively.  For instance, the following "**thermalProfile**" value string specified the temperature profile information for temperature probe index #0 and #1: ""**T0=-20,-10 T1=30,40**"

– The EventData "**thermoAverage#**" fields are part of the optional "**ThermoFieldInfo**" EventData table field set.  See the GTS FAQ page for more information on enabling optional fields.

– If the Device record does not define a thermal profile, or if otherwise unable to check the temperature profile, then "**DEFAULT_VAL**" will be return if specified (if "**DEFAULT_VAL**" is not specified, then the value "1" (true) will be returned under these same conditions).

## Event Fuel/Oil Level Functions:
*These functions require that the remote GPS tracking/telematic device supports sending fuel/oil level information to the device communication server.*

### *Fuel Level Change Percent:*
`Double $FuelDelta()`
> Returns the percent change in the fuel level since the last received event. The returned value ranges from `-1.00` (ie. Descreased by 100%) to `1.00` (increased by 100%). For instance, a returned value of `0.15` indicates that the fuel level has decreased by 15%.

`Double $FuelDelta(Long SECONDS_AGO)`
> Returns the percent change in the fuel level since the event received at most `SECONDS_AGO` seconds ago. The returned value ranges from `-1.00` (ie. Descreased by 100%) to `1.00` (increased by 100%). For instance, a returned value of `0.15` indicates that the fuel level has decreased by 15%.

### *Excessive Fuel Level Drop:*
`Boolean $FuelDrop([Double MAX_STOPPED_PERCENT [, Double MAX_MOVING_PERCENT]])`
> Returns True if the change in fuel level percent since the last received event exceeds `MAX_STOPPED_PERCENT` while currently stopped, or `MAX_MOVING_PERCENT` while currently moving (ie. Odometer value has increased). The percent value should be expressed in a floating point number between `0.0` and `1.0`. If not specified, the default value for `MAX_MOVING_PERCENT` is `0.50`, and the default value for `MAX_STOPPED_PERCENT` is `0.03`.

`Boolean $FuelDrop([Double MAX_STOPPED_PERCENT [, Double MAX_MOVING_PERCENT [, Long SECONDS_AGO]]])`
> Returns True if the change in fuel level percent since the event received at most `SECONDS_AGO` seconds ago exceeds `MAX_STOPPED_PERCENT` while currently stopped, or `MAX_MOVING_PERCENT` while currently moving (ie. Odometer value has increased). The percent value should be expressed in a floating point number between `0.0` and `1.0`. If not specified, the default value for `MAX_MOVING_PERCENT` is `0.50`, and the default value for `MAX_STOPPED_PERCENT` is `0.03`.

### *Excessive Oil Level Drop:*
`Boolean $OilDrop([Double MAX_STOPPED_PERCENT [, Double MAX_MOVING_PERCENT]])`
> Returns True if the change in oil level percent exceeds `MAX_STOPPED_PERCENT` while stopped, or `MAX_MOVING_PERCENT` while moving (ie. Odometer value has increased). The percent value should be expressed in a floating point number between `0.0` and `1.0`. If not specified, the default value for `MAX_MOVING_PERCENT` is `0.50`, and the default value for `MAX_STOPPED_PERCENT` is `0.02`.

## Icon/Pushpin Functions:

### *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.

### *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, 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.

## Event Job Functions:

### *Event/Device Job Number Defined: (since v2.4.6-B29)*
**Boolean $HasJobNumber()**
> Returns true is either the current EventData record, or the parent Device record, have a non-blank "**jobNumber**" value..

**Boolean $HasJobNumber(Boolean EVENT_JOB)**
> If **EVENT_JOB** is **true**, returns true if the current EventData record has a non-blank "**jobNumber**".
> If **EVENT_JOB** is **false**, returns true if the parent Device record has a non-blank "**jobNumber**".

### *Event/Device Job Number: (since v2.4.6-B29)*
**Boolean $JobNumber()**
> Returns the current EventData "**jobNumber**" if non-blank, otherwise it returns the parent Device record **jobNumber**. Returns blank if both the EventData and Device record "**jobNumber**" values are blank.

**Boolean $JobNumber(Boolean EVENT_JOB)**
> If **EVENT_JOB** is **true**, returns the current EventData record "**jobNumber**" value.
> If **EVENT_JOB** is **false**, returns the parent Device record "**jobNumber**" value.

## Driver Functions:

### *Driver record exists for Event specified "driverID":* *(since v2.6.3-B63)*
`Boolean $HasDriver()`
> Return True if the "driverID" specified in the current event exists in the Driver table. Returns false if the current Event does not have a defined "driverID".

`Boolean $HasDriver(String DRIVER_ID)`
> Return True if the specified **DRIVER_ID** exists in the Driver table..

### *Driver License Expired:*
`Boolean $DriverExpired()`
> Return True if the license for the current vehicle driver-id has expired (as specified by the "licenseExpire" field in the Driver table record), False otherwise. Return False if the current vehicle driver-id is not defined.

`Boolean $DriverExpired(String DRIVER_ID)`
> Return True if the license for the specified **DRIVER_ID** has expired (as specified by the "licenseExpire" field in the Driver table record), False otherwise. Return False if the specified **DRIVER_ID** is blank, or not found.

### *Driver Fatigue Check:* *(since v2.6.1-B39)*
`Long $DriverFatigue([String NAME_ID])`
> Calls the Device method at "`<Device>.checkDriverFatigue(...)`" which returns > 0 if the driver fatigue condition check fails. If **NAME_ID** is specified, the "`checkDriverFatigue`" will try to run the check for the specifically named driver fatigue conditions. If the **NAME_ID** is not found, the default conditions will be checked. The following describes the default conditions:
>  1) Within the last 24-hours: Stopped at least 7-contiguous-hours
>  2) Within the last 24-hours: Stopped at least a total of 9-hours (minimum 30-contiguous-min)
>  3) Within the last 6-hours: Stopped at least 30-contiguous-minutes
>
> If one of the conditions fail, the "**$DriverFatigue**" function will return a value > 0 (which represents the index of the condition that failed), and the replacement variable indicated by **NAME_ID** (or "DriverFatigue" if **NAME_ID** is not specified) can be used in the response notification email to indicate the specific condition that failed (for example "${DriverFatigue}").

### Device General Functions:

***Device Field Value:*** *(since v2.4.6-B29)*
`Object $DField(String FIELD_NAME)`
>    Returns the value of the specified Device 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.

***Elapsed time since Last Connection:***
`Long $LastConnect()`
>    Returns the number of elapsed seconds since the current Device has last connected with the server.

`Boolean $LastConnect(Long ELAPSED_SEC)`
>    Return True if the number of elapsed seconds since the current Device has last connected with the server is within the last **ELAPSED_SEC** seconds..

**Notes:**
>    This function tests the value of the Device record "lastTotalConnectTime" field (the applicable Device Communication Server is expected to set this field's value).  A 'connection' does not necessarily mean that the Device has sent an event at this time.  It may have been a simple "keep-alive" packet, which did not contain any event information.  This rule function is typically used within a "cron" task to report on Devices which appear to have not been heard from for an extended period of time.

***Timestamp of current TCP session:***
`Timestamp $TcpSessionTime()`
>    If the current Device has a active TCP connection/session with its associated DCS module, then the Epoch Timstamp(Long) of the initial connection will be returned.  If the device does not have an active TCP session with a DCS, then this function will return "0".  (will return zero for devices which communication via UDP).

**Notes:**
>    This function works by attempting to send a command-socket message to the device associated DCS module with a request to have it return information on the active TCP session.   If the associated DCS cannot be contacted, or if the DCS cannot find an active TCP session, then this function will return zero.

***In Device Group:***
`Boolean $InGroup(String GROUP)`
>    Return True if the Device represented by the current Event is contained by the specified **GROUP**.

***Device/Vehicle Registration Expired:*** *(since v2.5.8-B41)*
`Boolean $RegistrationExpired()`
>    Return True if the registration/license for the current vehicle has expired (as specified by the "licenseExpire" field in the Device table record), False otherwise.  Return False if the current vehicle is not defined.

***Device Command State Bit:*** *(since v2.5.4-B15)*
`Boolean $CmdState(Long BIT_INDEX)`
>    Return True iff the specified bit in the Device command state mask (see "commandStateMask") is true, False otherwise.

### Account General Functions:

***Check all Device's 'lastNotifyTime' within the current Event Account:***
**Boolean $DevNotify([Long AGE_SEC])**
    Return True if any device for the current Event Account has a 'lastNotifyTime' within the last **AGE_SEC** seconds.  If **AGE_SEC** is not specified, then return True if any device for the current Event Account has a 'lastNotifyTime' greater than 0. Make sure that the following 'config.conf' properties are set to include "NotificationFieldInfo" fields in the Device table:

        startupInit.Device.NotificationFieldInfo=true      *(required)*
        Device.keyedLastNotifyTime=true           *(recommended)*

***Elapsed time since last Device Communication Server activity:***
**Long $LastDCS([String DCS_ID])**
    Return the elapsed number of seconds since the last communication was received for the specified DCS name, or for any DCS if the **DCS_ID** is not specified.  This function checks all devices for the current account and returns the elapsed time since the last communication, as recorded by the "lastTotalConnectTime" field in the Device table.  This function is useful in periodic "cron" checks for possible periods of device communication server inactivity for the current account. (see also "**$G_LASTDCS**").

***Elapsed time since Last Login:***
**Long $LastLogin()**
    Return the number of elapsed 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..

***Account Expired:*** *(since v2.4.8-B32)*
**Boolean $AccountExpired()**
    Return True if the current account is past its expiration date.
**Boolean $AccountExpired(Long EXPIRE_SEC)**
    Return True if the current account will expire within **EXPIRE_SEC** seconds.

***Account Field Value:*** *(since v2.6.1-B40)*
**Object $AField(String FIELD_NAME)**
    Returns the value of the specified Account 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.

***Account/User Password Expired:*** *(since v2.4.8-B33)*
**Boolean $PasswordExpired()**
    Returns true if the password for the current account has expired.
**Boolean $PasswordExpired(String USER)**    *(since v2.6.3-B03)*
    Returns true if the password for the specified user in the current account has expired.

### 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.

### Returns the current average number of events received per second for all Accounts: *(Since v2.5.7-B23)*
`Double $G_EventsPerSecond()`
> Returns the average number of received events per second for all Accounts and all Devices.

### Check all Device's 'lastNotifyTime' for all Accounts: *(Since v2.3.1-B24)*
`Boolean $G_DevNotify([Long AGE_SEC])`
> Return True if any device, for any account, has a 'lastNotifyTime' within the last `AGE_SEC` seconds. If `AGE_SEC` is not specified, then return True if any device, for any account, has a 'lastNotifyTime' greater than 0. This function will return false if not run against the SystemAdmin user. Make sure that the following 'config.conf' properties are set to include "`NotificationFieldInfo`" fields in the Device table:
> 
> | | | |
> |---|---|---|
> | `startupInit.Device.NotificationFieldInfo=true` | | *(required)* |
> | `Device.keyedLastNotifyTime=true` | | *(recommended)* |

### The Amount of Free Space (in Megabytes) on the Named Disk Partition:
`Double $G_FreeDisk([String DIR])`
> Return the amount of free-space available (in Megabytes) on the named disk partion (ie. "`/tmp`", "`/usr/local`", etc). If the directory `DIR` is not specified, then free-space of the partition on which the GTS installation resides will be returned (ie. "`$G_FREEDISK($GTS_HOME)`"). For the purpose of this function, a Megabyte is defined as $1024^2$. 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 less than 15% of free space is left on a disk partion, the selector might be expressed as "`($G_FREEDISK/$G_TOTALDISK < 0.15)`". This function will return "`0.0`" if not run in "Cron" mode, or if not run against the SystemAdmin user, or if the specified directory does not exist.

### The Amount of Disk Utilization, as a Percent, on the Named Disk Partition: *(Since v2.5.1-B28)*
`Double $G_DiskUsage([String DIR])`
> Return the amount of disk utilization, as a percentage, on the named disk partion (ie. "`/tmp`", "`/usr/local`", etc). If the directory `DIR` is not specified, then disk-utilization for the partition on which the GTS installation resides will be returned (ie. "`$G_DISKUSAGE($GTS_HOME)`"). This function is useful for periodically checking for excessive disk-space usage in a log file directory, etc. For instance, a selector which triggers when disk utilization on the "`$GTS_HOME`" partition exceeds 90% might be expressed as "`($G_DISKUSAGE > 0.90)`". This function will return "`0.0`" if not run in "Cron" mode, or if not run against the SystemAdmin user, or if the specified directory does not exist.

### 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 partion (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 "**`0.0`**" if not run in "Cron" mode, or if not run against the SystemAdmin user, or if the specified directory does not exist.

### Elapsed time since last Device Communication Server activity:
**`Long $G_LastDCS([String DCS_ID])`**

Returns 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 indicaes 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 the current time (ie. maximum elapsed time) if not run in "Cron" mode, or if not run against the SystemAdmin user.

### Elapsed time since last Device Communication Server log file modified time: *[Since 2.5.9-B29]*
**`Long $G_LastDCSLog(String DCS_ID)`**

Returns the elapsed number of seconds since the last modified time for the specified DCS log file (**`DCS_ID`** parameter is required). This function is useful in periodic "Cron" checks for possible periods of Device Communication Server inactivity (ie. No log file activity). This function will return the current time (ie. maximum elapsed time) 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])`**

Returns 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 ("sysadmin") user.

### Check URL Accessibility: *[Since 2.5.3-B40]*
**`Boolean $G_ReadURL(String URL, String STARTS_WITH [, Integer TIMEOUT_MS])`**

Returns True if the specified URL can be opened and read. If the URL can be opened and read and if **`STARTS_WITH`** is non-blank, then it also checks to see if the returned response from the URL also starts with the value specified by **`STARTS_WITH`**. 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. This function will return False if not run in 'Cron' mode, or if not run against the SystemAdmin ("sysadmin") 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).

### *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.

### *Evaluate Selector from Device record "`notifySelector`" field: (Since v2.4.5-B19)*
`Object $DEevRuleE([Object DEFAULT_VAL])`
    Evaluates the selector found in the "`notifySelector`" field of the current Device record, and returns the Object result.  If the "`notifySelector`" field of the current Device record does not contain a valid selector, 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.

### *Recursive Rule Evaluation:*
`Object $Rule(String RULE_ID)`
    Exectute and return the Object result of the specified `RULE_ID`.  (The returned value type may be Long, Double, or String).  (As of v2.5.8-B03, `RULE_ID` must be either a valid Rule-ID, or a Rule-Selector enclosed in "(..)")

### *Rule Evaluation on Prior Events:  (Since v2.5.8-B03)*
`Boolean $RetroRule(String RULE_ID, Long ElapsedSEC)`
    Exectute the specified `RULE_ID` on the current event, and all prior events up to at most `ElapsedSEC` seconds ago.  This function will return true on the first encountered event for which the specified `RULE_ID` returns true.  `RULE_ID` must be either a valid Rule-ID, or a Rule-Selector enclosed in "(..)".

## 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 "`EventFunctionHandler`" 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 "`EventFunctionHandler`" 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`" value.

`public Device getDevice()`
    Returns the current "`Device`" instance (if any).

`public String getDeviceID()`
    Returns the current "`DeviceID`" value.

`public Device getDriver()`
    Returns the current "`Deriver`" instance obtained from the EventData "`getDriverID`" value, or the Device "`getDriverID`" value.

`public String getDriverID()`
    Returns the current "`DriverID`" value obtained from the EventData "`getDriverID`" value, or the Device "`getDriverID`" value.

`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}`".
– These replacement variables are defined in the EventData/Device/Account Java source modules, in the Java method "`getKeyFieldValue(...)`".

### Example Email Subject/Body Messages:

***Geozone/Geofence Arrival Notification:***
    Subject:
```
Arrival at terminal ${geozoneID}
```
    Body:
```
Truck ${device} has arrived at terminal ${geozoneID}
```

***Periodic Maintenance Notification:***
    Subject:
```
Vehicle ${device} is due for Periodic Maintenance
```
    Body:
```
Vehicle ${device} is due for Periodic Maintenance
Odometer: ${odometer}
```

***General Notification:***
    Subject:
```
Vehicle ${device}: [${statuscode}] ${status}
```
    Body:
```
===================================================
Account   : ${account} [${accountid}]
Device    : ${device} [${deviceid}]
Date/Time : ${datetime}
Status    : [${statuscode}] ${status}
Location  : ${geopoint}  [${latitude}/${longitude}]
Address   : ${address}
Speed     : ${speed}  ${direction}
Altitude  : ${alt}
Odometer  : ${odometer}
Distance  : ${distance}
===================================================
```

Notes:
1) Since `${latitude}` and `${longitude}` are replacement variables that return the location latitude and longitude, you could also add a Google maps link into the message by adding a line similar to the following:
```
https://www.google.com/maps/search/${latitude},${longitude}
```

## Account/Device Values:

*Event account description:*
**Name:**          ${account}
**Options:**       *(Since v2.4.5-B19)*
                   ":desc" (default) includes the account description.
                   ":id" (eg. ${account:id}) includes the account id.
**Example:**       Acme Incorporated

*Current number of Device records in current Event Account: (Since v2.3.1-B24)*
**Name:**          ${deviceCount}
**Example:**       43

*Event device description:*
**Name:**          ${device}
**Options:**       *(Since v2.4.5-B19)*
                   ":desc" (default) includes the device description.
                   ":id" (eg. ${device:id}) includes the device id.
**Example:**       Taxi #123

*The number of events received in the last 24-hours for current Event Device: (Since v2.3.1-B24)*
**Name:**          ${eventCount24}
**Example:**       1017

*Event device link URL value (html encoded):*
**Name:**          ${deviceLink}, ${devLink}
**Options:**       *(Since v2.4.5-B19)*
                   ":a" (default) includes the link within an html "a" anchor tag.
                   ":url" (eg. ${devLink:url}) includes only the url.
**Example:**       <a href='http://www.example.com/link?device=DEVICEID" target='_blank'>Description</a>
**Notes:**         The value for the 'href' comes directly from the Device "linkURL" field, and the value for
                   '**Description**' comes directly from the Device '**linkDescription**' field.  These are fields
                   optional fields that must first be enabled on the Device record by setting the property
                   "**startupInit.Device.LinkFieldInfo=true**", and updating the Device table, before they
                   are available to be edited with specific values in the Device/Vehicle Admin page. (See the
                   Frequently Asked Questions page at "http://www.geotelematic.com/FAQ.html#faq_optionalFields"
                   for more information).

*Elapsed time since Event device last connected with server: (Since v2.3.1-B24)*
**Name:**          ${connectAge}, ${devConnectAge}
**Example:**       0:27:08

*Comma separated list of entities (ie. trailers) attached to the current Event device:*
**Name:**          ${deviceEntities}, ${deviceTrailers}
**Example:**       T123, T234, T789

*The current Event Device service maintenance notes (if available):*
**Name:**          ${serviceNotes}
**Example:**       Checked tires

*The active GeoCorridor  ID (if available):*
**Name:**          ${activeCorridor}, ${corridorID}
**Example:**       route5

*The active GeoCorridor  ID description (if available):*
**Name:**          ${activeCorridorDesc}, ${corridorDesc}
**Example:**       Route 5

***The Maintenance Description:***
***Name:***           `${maintDesc}`
***Example:***      Oil Change

***The Maintenance Odometer:***
***Name:***           `${maintOdometer}`
***Example:***      97123 km

***The Maintenance Interval:***
***Name:***           `${maintInterval}`
***Example:***      4000 miles

***The Last Fuel Level (tank #1):***
***Name:***           `${lastFuelLevel}`
***Example:***      47%

***The Last Fuel Level (tank #2):*** *(Since v2.6.1-B40)*
***Name:***           `${lastFuelLevel2}`
***Example:***      23%

***The Device Speed Limit:***
***Name:***           `${devSpeedLimit}`
***Example:***      100 km/h

***The Device Reminder Message (if available):***
***Name:***           `${reminderMessage}, ${reminder}`
***Example:***      Hello

## Date/Time Values:

***Event date/time displayed in the preferred format:***
***Name:***      `${dateTime}, ${date}`
***Example:***      2008/11/01 12:53:01 PDT

***Event date year :***
***Name:***      `${dateYear}, ${year}`
***Example:***      2008

***Event date month:***
***Name:***      `${dateMonth}, ${month}`
***Options:***      *(Since v2.4.5-B19)*
            "`:name`" (default) includes the month abbreviation (ie "August").
            "`:abbr`" (eg. `${month:abbr}`) includes the month abbreviation (ie "Aug").
            "`:num`" (eg. `${month:num}`) includes the month number (ie 1..12).
***Example:***      August

***Event date day-of-month:***
***Name:***      `${dateDay}, ${day}`
***Example:***      17

***Event date day-of-week:***
***Name:***      `${dateDow}, ${dayOfWeek}`
***Example:***      Sunday

***Event date time:***
***Name:***      `${time}`
***Example:***      12:15:47

## Event Values:

***Event status code description:***
***Name:***        `${status}`
***Example:***    InMotion

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

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

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

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

***Event direction (bearing) abbreviation:***
***Name:***        `${direction}`
***Options:***    *(Since v2.4.5-B19)*
            "`:desc`" (default) includes the heading short description (ie "N", "NE", "E', "SE", etc.).
            "`:deg`" (eg. `${direction:deg}`) includes the heading value in degrees (ie "127.0", etc.).
***Example:***    SE

***Event bearing:***
***Name:***        `${bearing}, ${heading}`
***Options:***    *(Since v2.4.5-B19)*
            "`:deg`" (default) includes the heading value in degrees (ie "127.0", etc.).
            "`:desc`" (eg. `${heading:desc}`) includes the short description (ie "N", "NE", "E', "SE", etc.).
***Example:***    123.4

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

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

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

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

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

***Event arrival/departure Geozone/Geofence Description (if any):***
***Name:***      `${geozone}`
***Options:***      *(Since v2.4.5-B19)*
         "`:desc`" (default) includes the geozone/geofence description.
         "`:id`" (eg. `${geozone:id}`) includes the geozone/geofence id.
***Example:***      Home Base

***Event arrival/departure Geozone/Geofence Short Name (if any):*** *(Since v2.5.0)*
***Name:***      `${geozoneName}`
***Example:***      Home

***The current event battery level:***
***Name:***      `${batteryLevel}`
***Example:***      75%

***The last available internal battery voltage:***
***Name:***      `${batteryVolts}`
***Example:***      4.3

***The current event vehicle battery voltage:*** *(Since v2.4.4-B01)*
***Name:***      `${vBatteryVolts}, ${vehicleVolts}`
***Example:***      12.6

***The current event fuel level (tank #1):***
***Name:***      `${fuelLevel}`
***Example:***      75%

***The current event fuel level (tank #2):*** *(Since v2.6.1-B40)*
***Name:***      `${fuelLevel2}`
***Example:***      32%

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

***Event tracked 'Entity' description:***
***Name:***      `${entity}, ${entityDesc}`
***Options:***      *(Since v2.4.5-B19)*
         "`:desc`" (default) includes the entity description.
         "`:id`" (eg. `${entity:id}`) includes the entity id.
***Example:***      T123

***Temperature values:*** *(Since v2.4.0-B42)*
***Name:***      `${temperature:INDEX}, ${temp:INDEX}`
***Note:***      Where "*`INDEX`*" is the number of the Temperature value to display ('0' is the first index value).
***Example:***      42.6 F

## Address Values:

***Reverse-Geocoded Event full address:***
***Name:***         ${address}, ${fullAddress}
***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).

***Reverse-Geocoded Event street address (if available):***
***Name:***         ${street}, ${streetAddress}
***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).

***Reverse-Geocoded Event city address (if available):***
***Name:***         ${city}
***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).

***Reverse-Geocoded Event state/province address (if available):***
***Name:***         ${state}, ${province}
***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).

***Reverse-Geocoded Event postal code address (if available):***

| | |
|---|---|
| ***Name:*** | `${postalCode}, ${zipCode}` |
| ***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). |

***Reverse-Geocoded Event country/state address (if available):***

| | |
|---|---|
| ***Name:*** | `${subdivision}, ${subdiv}` |
| ***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). |

## Prior Event Values:

***GeozoneID (Geofence) of previous event. Same as "${#geozoneID}":***
***Name:***           ${priorGeozoneID}
***Example:***        home

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


## System Attributes:

***System disk usage (partition used by "$GTS_HOME/logs"):***
***Name:***           ${gtsDiskUsage}
***Example:***        Dir=/ust/local/GTS_2.5.8-B07/logs  Total=20480mb  Free=13280mb  [65% free]

***System disk usage folder/directory:*** *[Since v2.5.8-B07]*
***Name:***           ${gtsDiskFolder}
***Example:***        /ust/local/GTS_2.5.8-B07/logs

***System disk total (in megabytes):*** *[Since v2.5.8-B07]*
***Name:***           ${gtsDiskTotal}
***Example:***         20480mb

***System disk free (in megabytes):*** *[Since v2.5.8-B07]*
***Name:***           ${gtsDiskFree}
***Example:***        13280mb

***System disk used (in megabytes):*** *[Since v2.5.8-B07]*
***Name:***           ${gtsDiskUsed}
***Example:***        720mb

***System disk free percent:*** *[Since v2.5.8-B07]*
***Name:***           ${gtsDiskFreePct}
***Example:***        65%

***System disk used percent:*** *[Since v2.5.8-B07]*
***Name:***           ${gtsDiskUsedPct}
***Example:***        35%

## Appendix E) Rule Trigger "Predefined Actions" field

When enabled, the "Predefined Actions" field allows entering custom actions that will be executed when the rule is triggered based on the conditions set in the rule selector.  To show the "Predefined Actions" field in the "Rule Admin" page, the following property must be enabled (preferrably in the "config.conf" file):

```
Domain.Properties.ruleInfo.showPredefinedActions=true
```

After the above property has been set, the "track.war" file will need to be reqbuilt and redeployed.

 The general syntax of actions that are placed into the new "Predefined Actions" field is as follows:

> **ACTION_ID**:**PARAMETER**

Where "**ACTION_ID**" is a currently support action name/id, and the "**PARAMETER**" is a parameter that appies to the specified action name/id.

The following is a list of currently supported Action Names/IDs:

- **resetMaint**:*MAINT_INDEX*
  This action will reset the Maintenance Odometer field in the Device record indicated by the "*MAINT_INDEX*" value parameter.
- **resetReminder**
  This action will reset the reminder interval ("reminderTime") specified in the Device record.
- **park**:*RADIUS_METERS*
  This action will set the "parked" location in the Device record to the current event location, and the specified "*RADIUS_METERS*" (in meters).
- **unpark**
  This action clear any set "parked" attributes in the Device record.
- **sendCommand**:*COMMAND_ID*
  This action will send the specified "*COMMAND_ID*" to the current device.  The specified "*COMMAND_ID*" must match one of the commands specified in the corresponding "dcservers/dcserver_***XXXXXX***.xml" files for the specific device (where "***XXXXXX***" represents the device DCS name).
- **addEvent**:*STATUS_CODE*
  This action will cause another event to be inserted into the EventData table (matching the current account/device), with the specified "*STATUS_CODE*".  The status code must be specified in its hexidecimal for (ie. "0xFF01").

The the rule is triggered for a specific device, the action defined in the "Predefined Actions" field (as indicated above) will be executed.

## Appendix F) Rule Selector Examples

**Rule Admin Triggers:**

- Trigger on Geozone Arrive or Depart after business hours (after 6pm or before 6am or on Weekends:
```
($ArrDep && ($TODRANGE(1800,0600) || $WeekEnd))
```

- Trigger on Geozone Arrive at zoneID "office" on Saturday only between 8am and 6pm:
```
($Arrive("office") && ($DOW == 6) && $TOD(0800,1800))
```

- Trigger on exceeding posted speed by more than 10 km/h over the limit (require using a reverse-geocode provider that supports returning the posted speed limit):
```
($PostedSpeed(10))
```

- Trigger if event status code is either door open, or door close:
```
($SC(0xF4D0,0xF4E0))
```

- Trigger if event indicates that the vehicle has remained inside Geozone "client" for more than 1 hour (3600 seconds):
```
($InZoneElapsed("client") > 3600)
```

- Trigger if event indicates that the vehicle has remained stopped for more than 1 hour (3600 seconds):
```
($StopSec > 3600)
```

- Trigger if event indicates that the vehicle has remained idle (stopped and ignition-on) for more than 30 minutes (1800 seconds):
```
($IdleSec > 1800)
```

- Trigger if event indicates that the device has not connected with the server in the last 30 minutes (1800 seconds):
```
($LastConnect > 1800)
```

- Trigger if event indicates that the distance maintenance interval #0 has expired:
```
($MaintKM(0))
```