OpenGTS
Installation and Configuration
Manual

Manual Revision HIstory			
Rev	**Date**	**Changed**	**Author**
0.1.1	2007/11/30	Added additional note on using 'dbAdmin.pl' to add missing table columns. Minor text changes made.	MDF
0.1.2	2008/02/06	Added comment to prerequisites regarding the installation of Tomcat on Linux.	MDF
0.1.3	2008/02/17	Added section on Internationalization/Localization. Added information regarding the implementation of device communication servers.	MDF
0.1.4	2008/02/20	Added additional comments regardin configuration of Tomcat on Debian/Ubuntu	MDF
0.1.5	2008/03/28	Java SDK 5.0 is now required to build OpenGTS.	MDF
0.1.6	2008/04/11	Added a section on installation testing	MDF
0.1.7	2008/05/14	Updated "Compile" section to include "ant all". Added additional comments to Localization section..	MDF
0.1.8	2008/06/20	Updated location of 'private.xml' file. Updated the download link information for various modules.	MDF
0.1.9	2008/07/08	Added additional comments regarding building the example 'template' server.	MDF
0.1.10	2008/07/27	Added note regarding 'bin\dbConfig.bat' command available for Windows users. Added information regarding customization of map Pushpins..	MDF
0.1.11	2008/10/16	Various minor changes	MDF
0.1.12	2008/12/01	Updated the 'Database Administration' section	MDF
0.1.13	2008/12/16	Update Apache Tomcat download URL	MDF
0.1.14	2009/02/01	Miscellaneous updates	MDF
0.2.0	2009/04/02	Added Mac OS X configuration information. Rearranged chapters. Added additional 'device communication server' start/stop information.	MDF
0.2.1	2009/05/24	Added section on installing MotoDMTP	MDF
0.3.0	2009/07/01	Added notes regarding "events.war" support for exporting GPX formatted events. Added section on "Creating/Modifying Reports". Added notes on validating LocalStrings files.	MDF
0.3.1	2009/08/02	Added additional comments regarding "events.war" use for Google Earth map updates, note regarding customizing the loog-and-feel, etc.	MDF
0.3.2	2009/08/23	Added comments regarding running "runserver.bat" on Windows. Updated Tomcat version to 5.5.28.	MDF
0.3.3	2009/10/30	Added '-sendMail' option to 'checkInstall' command.	MDF
0.3.4	2010/06/18	Updated informaton regarding 'runserver', starting/stopping device communication servers, and using 'psjava'. Added information regarding the device communication servers runtime configuration "dcservers.xml" file.	MDF
0.3.5	2010/07/08	Added GTS system architecture section. Updated various pre-requisite links. Updated starting/stopping DCS.	MDF
0.3.6	2010/09/10	Added additional "Device" admin options. Added "Optional Table Columns"	MDF
0.4.1	2011/03/08	Placed Prerequisite and OpenGTS installations into separate chapters. Renumbered chapters. Added additional information on the "gprmc" http-based device communication server.	MDF
0.4.2	2011/04/01	Added Trackstick CSV data import information (v2.3.2). Fixed minor typos, etc.	MDF
0.4.3	2011/08/21	Misc changes. Added JSON file format to "events.war" description. Update JavaMail download Version/URL. Added information on starting the TK10x DCS module.	MDF
0.4.4	2012/12/27	Included table optional field names.	MDF
0.4.5	11/20/13	Update links.	MDF
0.4.6	09/01/21	Update links.	MDF
0.5.1	2025/07/14	Updated	

OpenGTS Installation/Configuration

Contents:

OpenGTS Installation/Configuration

Contents: (continued)

1) Introduction

OpenGTS (**Open** Source **G**PS **T**racking **S**ystem) is intended to provide a generic back-end web-based service for querying and viewing GPS related data. It is desgined to operate independently of any specific GPS tracking device or protocol, but comes with support for several device protocol formats.

It is specifically designed for use in small to medium sized commercial enterprises wishing to take advantage of GPS tracking for "fleets" of vehicles. However, **OpenGTS** is highly configurable and scalable to larger enterprises as well.

On the server side, **OpenGTS** is designed to be device and protocol independent. In order to use the features of **OpenGTS**, a specific device/protocol communication server (which we call a Device Communication Server, or "DCS") will need to be implemented to communicate with the remote device and place the data in the SQL database. **OpenGTS** ships with support for a few supported DCS modules, such as the "tk10x", "aspicore", and a few others. A custom device communication server can also be implemented using the included example server source code. See the chapter titled "Creating Your Own Device Communication Server" for more information.

On the web-interface side, the user presentation is easily customizable to fit the individual desired motif. Menu options and features are also easily customizable to fit specific requirements.

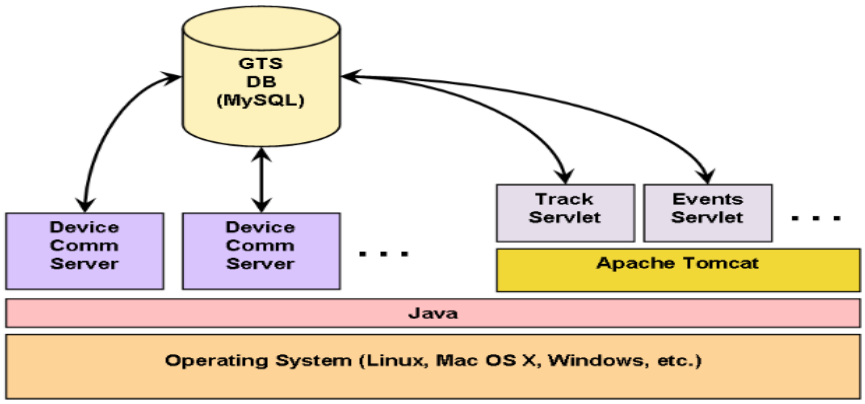
The source code for the OpenGTS project may be downloaded from SourceForge at the following link:
https://sourceforge.net/projects/opengts/files/
(Licensed under the Apache License Version 2: http://www.apache.org/licenses/LICENSE-2.0)

DISCLAIMER:
OpenGTS must not be used for any illegal activities. The providers of this project assume no responsibity for any illegal activities that may be conducted by users of this software.

1.1) Supported Platforms

OpenGTS is completely implemented in Java and should run fine on any system that fully supports the Java Runtime Environment. However, this implementation does require an SQL database server, and is therefore also limited to systems on which your chosen SQL database runs. See the respective SQL database support website for their supported systems (ie. for MySQL see "http://www.mysql.org" – which has been tested with **OpenGTS** on Linux, Mac OS X, FreeBSD, OpenBSD, and Windows-XP/Vista/20xx platforms).

1.2) System Architecture



This graphic describes the basic system architecture of the **OpenGTS** system. The various device communication servers (the modules which listen for incoming data from the remote GPS tracking devices) run as separate processes on top of Java. The Track servlet (ie. The web-interface), as well as other servlets (including any http-based device communication server), run within a Servlet Container, such as Apache Tomcat.

1.3) Planned Enhancements

OpenGTS is always evolving, and new features are continually being made available. Here are some general categories of the features that are in the planning or implementation stage:

- Additonal map features, and report.
- AI Integration.

GTS Enterprise was built on **OpenGTS** and has several additional features available as well (more information regarding the GTS Enterprise can be found at "http://www.geotelematic.com/gts.html"):

- Support for several “commercial use” mapping service providers.
- Support for many additional remote GPS tracking devices.
- Additional reporting options.
- Simple 'Rules' engine to send notifications based on criteria from incoming events (enhanced Event Notification Rules Engine is also available).

Contact us regarding the availability of these and other features at "opengts@geotelematic.com".

1.4) Document Conventions

In order to provide a generic installation/configuration document that covers various systems types (ie. Windows XP, Mac OS X, and the various Linux distributions), and the various versions of the **OpenGTS** system, the following conventions and assumptions have been adopted within this document:

1) This document will assume that the target operating system is Linux-based. For other operating system types, the appropriate path separators and directory specifications will need to be used that match the requirements of your specific operating system. Environment variable specification may also vary between operating systems. For instance, to de-reference the **JAVA_HOME** environment vairable, "`%JAVA_HOME%`" would be specified on a Windows system, while "`$JAVA_HOME`" is specified on Linux and Mac OS X.

2) This document will assume that the directory in which **OpenGTS** will be installed is "`/usr/local/`". If you will be installing **OpenGTS** in a different directory, you will need to replace the directory references in this document to the directory in which **OpenGTS** was installed.

3) **OpenGTS** has a frequent release schedule. For consistency, this document will assume that the version of **OpenGTS** to be installed is "`OpenGTS_1.2.3`". So references to "`OpenGTS_1.2.3`" within this document should be replaced with the actual name and version of **OpenGTS** that you will be installing.

4) On various command line examples, you may see the directory specification "*`/zzz`*". This specification is simply a placeholder name representing some current directory on your system, and not a literal directory name existing on your system.

5) In various locations within this document, command-line options are specified as "`-argName=value`", where "`argName`" is the name of a command-line argument, and "`value`" is the value to be assigned to the command-ine argument. When entering commands in Windows at a DOS command prompt (such as when using the "`.bat`" version of the commands), command arguments such as **`-rootUser=root`** **MUST** either be enclosed in quotes, as in **`"-rootUser=root"`**, or be specified with a colon instead of an equal-sign, as in **`-rootUser:root`** (preferred).

6) In various locations within this document, the displayed command-line options may include example values that are to be replaced with values specific to your requirements. For example, a command-line option indicating an account may be specified as "`-account=`*`myaccount`*" or as "`-account=<myaccount>`". In this case the argument name "`-account=`" may be taken literally, while "*`myaccount`*" or "`<myaccount>`" indicates a value that should be replaced with a specific value matching your requirements.

7) In various locations within this document, the displayed command-line options may command-line parameters which are optional (they only need to be included to for some applications, or to change the default behavior). These optional parameters will be displayed within square-brackets '[' ... ']' (eg. "`[-dir=/tmp/gts]`") These square-bracket will indicate that the parameter specified within the brackets is optional, depending on the command requirements. The square-brackets themselves are not to be included in the entered command.

2) Loading the Prerequisite Packages

Important Note:
Installation of the **OpenGTS** prerequisite modules does require at least an intermediate knowledge of how to install and configure systems services such as the Java compiler, Apache Ant, MySQL (or other SQL database server), Apache Tomcat (or other servlet container), and other related technologies.

Compiling **OpenGTS** requires that the following packages or applications be installed, configured, and running on the local system:

2.1) Java Compiler

Package: Java SE JDK 21
Download: https://adoptium.net/
Notes: *download the JDK (the JRE is already included in the JDK)*

Note:
OpenGTS will work on Java version 8+, however Java version 21 is recommended.

Note:
Make sure that the Java SDK installation '`bin`' directory has been added to the **PATH** environment variable (see below). Failing to do so may result in compiler errors.

IMPORTANT NOTE:
Only install the JDK version of Java, do not install the separate JRE version. The JDK already contains a JRE component, and the JDK is needed to compile the Java code. Having both the JDK and JRE installed on your system, may cause some confusion when it comes to compiling and running the application.

2.1.c) Windows users:
(Note: The instructions for Windows installation has not been updated for Java-21, so i may be out of date)
The Java installation process on Windows may go ahead and install both the JDK and JRE directories (even if the JRE wasn't selected during the installation process). The default installation will install the following directories:

```
C:\Program Files\Java\jdk1.8.0_40\
C:\Program Files\Java\jre1.8.0_40\
```

Apache "ant" and other Java programs may not work well with a **`JAVA_HOME`** environment variable that contains embedded spaces (such as "`C:\Program Files\java\jdk1.8.0_40`"). To get around this Oracle has created a subdirectory at "`C:\ProgramData\Oracle\Java\javapath\`" to contain links to various Java executables without any embedded spaces, however it has for some reason omitted the JDK compiler from this list. To correct for this, we need to create an additional directory symbolic link to the JDK installation. Go to the "Start" menu, then select "All Programs", then "Accessories", then right-click on "Command Prompt" and select "Run as Administrator". This will provide the proper privileges to create the symbolic link, using the following commands:

```
cd C:\ProgramData\Oracle\Java\javapath\
mklink /D JDK "C:\Program Files\Java\jdk1.8.0_40"
```

The **`JAVA_HOME`** environment System variable can then be set to the following (with no embedded spaces):

```
JAVA_HOME ==> C:\ProgramData\Oracle\Java\javapath\JDK
```

The Environment Variable editor can be accessed from the "Start" menu by clicking on "Control Panel", then "System", then "Advanced System Settings", then "Environment Variables".

The **`PATH`** environment System variable should then be prefixed with the following

```
%JAVA_HOME%\bin;
```

The various required library jar files (ie. "`javax.mail.jar`", etc) will need to be copied to **BOTH** the extended library directories in the JDK and JRE installations. These directories are listed below:

```
C:\Program Files\Java\jdk1.8.0_40\jre\lib\ext\
C:\Program Files\Java\jre1.8.0_40\lib\ext\
```
NOTE: The additional library jar files **MUST** be copied using drag/drop from the Windows console. Attemping to copy them from a command-prompt windows may appear like they have copied successfully, however these library jar files may still not be accessible by the Java runtime.

2.2) Apache "Ant" Build Tool

Package: Ant v1.10.x+ (or later version)
Download: http://ant.apache.org/

Note:
Make sure that the Ant installation '`bin`' directory has been added to the **PATH** environment variable (see below).

2.3.a) Linux users:
The Linux installed Apache "Ant" can be installed and used instead of downloading the "Ant" installation directly from Apache. Examine the documentation of your Linux distributes for information on downloading the latest avalable Apache "Ant" package.

2.3) Apache "Tomcat" Servlet Container

Package: Apache Tomcat v8.5 servlet container, or later
URL: http://tomcat.apache.org

Notes:
- If your version of Linux distribution already comes with a version of Tomcat installed, it is still **highly recommended** that you start with a version of Tomcat downloaded directly from the Apache website. This will eliminate or reduce any permissions or classpath problems that may be encountered. Then when everything is up and working properly, if you choose to do so, you can go back and install **OpenGTS** in the version of Apache Tomcat that was shipped/installed with your Linux distribution.

2.3.a) Linux users installing the downloaded Apache Tomcat:
It is recommended that the manual Tomcat installation be installed in the "`/usr/local/`" directory ('root' access will be required to install into this directory).

The **`CATALINA_HOME`** environment variable should then be set to the following:
```
$ export CATALINA_HOME=/usr/local/apache-tomcat-x.xx.xx
```
Where "`/usr/local/apache-tomcat-x.xx.xx`" matches the name of the Tomcat installation directory.

It is recommended to also create a "`tomcat`" symbolic link in the "`/usr/local/`" directory which points to the Tomcat installation, as follows:
```
#  cd /usr/local
#  ln -s $CATALINA_HOME tomcat
```

Within the Tomcat installation 'bin' directory (ie. "`$CATALINA_HOME/bin`"), make sure the execute permissions bit is set on all "`.sh`" files. If not set, the following command will set the execution bit:
```
$ cd $CATALINA_HOME/bin
$ chmod a+x *.sh
```
If the execute bit is not set on these files, Tomcat "`startup.sh`" and "`shutdown.sh`" commands may not be able to execute.

If you are using Java-9+ (Java-21 recommended), you will need to add an additional directory to the Tomcat startup classpath that included the required dependent jar files that your running servlets will need. This is done by creating a "setenv.sh" file in the Tomcat "bin/" directory, that contains the following:

2.3.a) Linux: Adding "setenv.sh" to Tomcat:
Assuming that the **`$CATALINA_HOME`** environment variable points to your Tomcat installation.
Start editing the new "setenv.sh" file with the following "vi" command (or other text editor):

```
$ vi ${CATALINA_HOME}/bin/setenv.sh
```

The enter the following:

```
export CLASSPATH="${CLASSPATH}:${GTS_HOME}/.Dependencies/*"
```

Then save this file.

2.3.b) Windows: Adding "setenv.sh" to Tomcat:
Assuming that the **`%CATALINA_HOME%`** environment variable points to your Tomcat installation.
Edit a new "**`%CATALINA_HOME%\bin\setenv.sh`**" file with a text editor, and enter the following:

```
set CLASSPATH=%CLASSPATH%;%GTS_HOME%\.Dependencies\*"
```

Then save this file.

2.4) MySQL Database Provider

Package: MySQL v5.7.X, or MySQL v8.x.x
URL: http://dev.mysql.com/downloads/mysql/
URL: https://downloads.mysql.com/archives/community/

2.5.a) Windows users:
On Windows, download/install the following file:

```
mysql-essential-5.7.XX-m2-win64.msi
```

Where "`mysql-essential-5.7.XX-m2-win64.msi`" is the name of the latest MySQL installation for Windows.

2.5.b) Linux users:
The Linux "MySQL" can be installed and used instead of downloading the "MySQL" installation directly from MySQL. See your Linux distribution documentation for downloading/installing MySQL.

Note:
Make sure that the MySQL installation "`bin`" directory has been added to the **PATH** environment variable.

3) Installing/Compiling the OpenGTS Source

Important Note:
Installation of the supported **OpenGTS** features does require at least an intermediate knowledge of how to install and configure systems services such as the Java compiler, Apache Ant, MySQL (or other SQL database server), Apache Tomcat (or other servlet container), and other related technologies.

3.1) Unzipping/Installing the OpenGTS Source

On Linux systems, it is recommended that the **OpenGTS** zip file be unzipped and installed in the "`/usr/local/`" directory. On Windows, it can be installed in any convenient directory, such as in the root partition of "`C:\`".

For the purposes of this document, we will assume that the target operating system is Linux-based, and that the location where **OpenGTS** will be installed/unzipped is "`/usr/local/OpenGTS_1.2.3/`" (Note: you may need to choose another installation directory if you do not have 'root' access on the target system). Adjust the file/path separators and commands as necessary for your particular operating system, and chosen installation directory.

Install the OpenGTS source code:
Unzip the **OpenGTS** package in "`/usr/local/`" (this will need to be done as the "`root`" user), or other convenient directory (on Windows, choose a directory where you would like this package to be placed – preferrably a path which does not contain any embedded spaces). For instance, if the **OpenGTS** package to be installed is "`OpenGTS_1.2.3.zip`", then the command to unzip the package would be:

```
/zzz> cd /usr/local
/usr/local> su root
/usr/local# unzip /tmp/OpenGTS_1.2.3.zip
/usr/local# chown -R user:group OpenGTS_1.2.3
/usr/local# exit
/usr/local> export GTS_HOME=/usr/local/OpenGTS_1.2.3
```

(the above assumes that `OpenGTS_1.2.3.zip` was downloaded to "`/tmp/`". If the **OpenGTS** zip file was downloaded into a different directory, modify the above directory location and downloaded file name accordingly. Also, replace the user name "`user`", and group name "`group`", above with the name of the `user:group` that you wish to have own the **OpenGTS** installation).

(Also note that the "*`/zzz`*" directory name above is just a placeholder name which represents any current directory that may be in effect before the "`cd`" command is issued).

3.2) Setting the Environment Variables

The following environment variable should be set to the installation directory of the corresponding package or application:

- **`JAVA_HOME`** – The Java JDK (**NOT** the JRE) installation directory.
- **`ANT_HOME`** – The Apache Ant installation directory.
- **`CATALINA_HOME`** – The Apache Tomcat installation directory.
- **`GTS_HOME`** – The OpenGTS installation directory.

3.2.a) Windows users:
The location of the installation '`bin`' directories for the Java SDK, Ant, and SQL database server installations, needs to be added to the command execution **PATH** environment variable (if the installation process has not already added them to the **PATH** variable).

Environment variables can be set manually in a command-prompt with the "set" command, as in the following example:

```
C:\> set GTS_HOME=C:\OpenGTS_1.2.3
```

(When setting environment variables, quotes should not be used to enclose an installation directory, even if the directory contains embedded spaces)

Environment variables are referenced by enclosing them in '`%`'. For instance, after setting the environment variable **`JAVA_HOME`** to point to your JDK installation directory, this environment variable would be dereferenced as "**`%JAVA_HOME%`**".

The file path separator is the back-slash character "`\`". So, while on Linux a file/directory could be referenced as "**`$JAVA_HOME/jre/lib/ext/.`**", on Windows this same directory would be referenced as "**`%JAVA_HOME%\jre\lib\ext\.`**"

The above environment variables can be set to be automatically defined when starting a command-prompt through the "**System Properties**" window as follows:

- Right-click on "**My Computer**" and select "**Properties**", the "**System Properties**" window will display. Select the "**Advanced**" tab, then press the "**Environment Variables**" button.
- In the "**System Variables***"* section, add the following variables:
  - Variable Name: **JAVA_HOME** *(required for building OpenGTS, and running Tomcat)*
    Value: *(The location of your JAVA **SDK** Installation Folder, **NOT** the JRE)*
  - Variable Name: **ANT_HOME**
    Value: *(The location of your Ant Installation Folder)*
  - Variable Name: **CATALINA_HOME** *(required for building OpenGTS)*
    Value: *(The location of your Apach Tomcat Installation Folder)*
  - Variable Name: **GTS_HOME**
    Value: *(The location of your OpenGTS Installation Folder)*

  *(Quotes should **NOT** be used to enclose an installation directory for these environment variable specifications, even if the directory contains embedded spaces)*
- Prefix the following to the "**Path**" environment variable in the "**System Variables***"* section (create a new "**Path**" variable if one does not already exist):

  ```
  .;%JAVA_HOME%\bin;%MYSQL_HOME%\bin;%ANT_HOME%\bin;
  ```

  (Quotes may be added to the PATH variable if necessary)
- Click "**OK**" on the "**Environment Variable**" window.

3.2.b) Linux users:
It is recommended that the following symbolic links be created within the "`/usr/local/`" directory which point to their corresponding 'home' directories (skip a given symbolic link if it has already been created):

```
#  cd /usr/local
#  ln -s $JAVA_HOME java
#  ln -s $CATALINA_HOME tomcat
#  ln -s $GTS_HOME gts
```

3.3) Installing the Maven build tool and downloading Jar file dependencies

Package: Maven 3.9.x
URL: https://maven.apache.org/download.cgi

Maven is used in this project just for its library management features. The "pom.xml" file that is shipped with OpenGTS and GTS Enterprise contains a list of all jar file library depencies to compile and run the OpenGTS software.

The config file "Dependencies.conf", included with the OpenGTS package, specifies the following property specification:

```
DependencyDir=${GTS_HOME}/.Dependencies
```

This specifies the default location where Maven will place the list of jar file dependencies.

The command to instruct Maven to read the "pom.xml" file and place the list of dependencies in the above file, it "mvn_dependencies.sh". Use the "-help" option to get more information about the command:

```
cd $GTS_HOME
bin/mvn_dependencies.sh -help
```

The command to instruct Maven to copy the dependencies is the following:

```
cd $GTS_HOME
bin/mvn_dependencies.sh -copy
```

The command will display information regarding where it is about to copy the dependencies, when wait for a confirmation. Simply hit the return key to continue.

3.4) Compiling the Supporting GTS Library Files.

> **3.3a) Precompiled Versions of GTS:**
> If you have received a pre-compiled version of the GTS package, this section may be skipped (however you may rebuild the various servlets and jar files if you wish to make any changes to the runtime configuration before deployment.

Compile the OpenGTS library ".jar "and servlet ".war" files:
'`cd`' into the **OpenGTS** installation directory and compile the jar files, and servlet war files, using the supplied Ant "`build.xml`" script:

```
/usr/local> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> ant all
```

This will build several jar files, and war files, in the "`$GTS_HOME/build/`" directory, including:

- "`lib/gtsutils.jar`" – This jar contains the base utilities and db access tools.
- "`lib/gtsdb.jar`" – This jar contains the database access utilities and table definition.
- "`lib/tools.jar`" – This jar contains miscellaneous system check and administrative tools.
- "`track.war`" – This "war" file (web-archive) contains the web-interface 'Track' servlet.
- "`events.war`" – This "war" file contains the web accessible EventData access servlet.

(Note: this is only a partial list. Other modules will be created as well).

The build should complete normally. There may be some warnings displayed, however if the warning or error can be ignored, there will also be a message indicating this next to the warning/error (or on a line just below the warning/error).

4) Initialization and Installation Testing

Before using **OpenGTS**, it must first be initialized. This section describes the steps required for initialization and testing.

4.1) Initializing the SQL Database Tables

Before storing data in the SQL database, it must first be initialized with the tables used by **OpenGTS**. This can be accomplished with the "`bin/init.sh`" command as follows:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> bin/initdb.sh -rootUser=<rootUser> -rootPass=<rootPass>
```

Where *<rootUser>* is the user with root access to the SQL server, and *<rootPass>* is the root user password (may be optional depending on the configuration of your SQL server). [NOTE: This is not the same as the Linux "root" user]

4.1.a) Important note regarding ".sh" and ".bat" command files:
Commands ending with "`.sh`" or "`.bat`" **MUST** be executed from the **OpenGTS** installation directory. Attempting to execute these commands from another directory may result in a "ClassNotFoundException" or "NoClassDefFoundError" error, or similar. (This means that you must cd to **$GTS_HOME**, then execute the command as "`bin/<command>`") Windows users may wish to install a Perl interpreter on their machine in order to use the Perl versions ("`.pl`") of the command-line scripts which do not require that they be executed from the **OpenGTS** installation directory. More information on possible Perl distributions available on Win32 platforms may be found at this location: "http://win32.perl.org/wiki/index.php?title=Win32_Distributions"

4.1.b) Important note for Windows users:
When using the "`.bat`" version of the commands in a DOS window, command arguments such as **`-rootUser=root`** must either be enclosed in quotes, as in **`"-rootUser=root"`**, or be specified with a colon instead of an equal-sign, as in **`-rootUser:root`**. Thus, on Windows, the command is:

```
bin\initdb.bat "-rootUser:userName" "-rootPass:userPass"
```

Where '*userName*' and '*userPass*' should be replaced with the appropriate root user and password.

The "`initdb.sh`" command performs the following functions when initializing the **OpenGTS** database:

- Creates a database called "**gts**".
- Creates/Grants user "**gts**" with password "**opengts**" with access to the "**gts**" database.
- Creates the following tables in the "**gts**" database (this is only a partial list):
  - **Account** - Account owner table
  - **User** - User table
  - **UserAcl** - User Access-Control-List table
  - **Device** - Device information table
  - **EventData** - Received Event data
  - **Geozone** - Geozone/Geofence definitions
  - **EventTemplate** - Custom event packet templates (DMTP only)
  - **PendingPacket** - Packets pending transmission to device (DMTP only)
  - etc.

The "`initdb.sh`" command performs the same functions as the following sequence of commands:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -createdb -user=<rootUser>
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -grant -user=<rootUser>
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -tables=ca
```

Note for Windows Users:
"`bin/dbAdmin.pl`" is only available for Linux users, and Windows users which are running within a Cygwin environment. "`bin\dbConfig.bat`" provides a subset of the features available in "`bin/dbAdmin.pl`" which will run from a Windows command prompt.

4.2) Testing the Installation

> **4.2.a) Important note regarding ".sh" and ".bat" command files:**
> Commands ending with "`.sh`" or "`.bat`" **MUST** be executed from the **OpenGTS** installation directory. Attempting to execute these commands from another directory may result in a "ClassNotFoundException" or "NoClassDefFoundError" error, or similar. (This means that you must cd to **$GTS_HOME**, then execute the command as "`bin/<command>`")

The following command has been included to assist in checking the installation of the system and displaying any inconsistencies that might cause problems at runtime:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> bin/checkInstall.sh
```

Or, on Windows:

```
C:\> cd %GTS_HOME%
C:\OpenGTS_1.2.3> bin\checkInstall.bat
```

This command will display various configured directories and environment variables. If any errors are displayed, they should be corrected (or at least understood) before continuing system deployment.

SMTP configuration is required to support features such as sending forgotten email notifications, emailing reports, etc. The properties required for SMTP can be configured in one of the "`.conf`" runtime configuration files (typically "`custom.conf`"). If you wish to test your SMTP email configuration, you can add the option "`-sendMail <emailAddress>`", which will attempt to send a test email to the specified email address:

```
/usr/local/OpenGTS_1.2.3> bin/checkInstall.sh -sendMail myemailaddress@example.com
```

(note that there are is a space between the '`-sendMail`' option and the email address)

Or, on Windows:

```
C:\OpenGTS_1.2.3> bin\checkInstall.bat -sendMail:myemailaddress@example.com
```

(note that there are is a ':' between the '`-sendMail`' option and the email address for the Windows version of the command)

Replace "`myemailaddress@example.com`" with the email address you wish to have receive the test email.

4.3) Loading the Sample Data

Some sample data has been provided with the **OpenGTS** installation which can be loaded and viewed within the web-interface. Please refer to the document at "`sampleData/README.txt`" within the **OpenGTS** installation directory for information regarding how to load the sample data.

4.4) Creating the "sysadmin" Account

When logging in to the "`sysadmin`" account a new menu 'tab' will be available, with new web-page selections, that allow the creating of new accounts. The following command can be used to create the "`sysadmin`" account:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh Account -account=sysadmin -pass=password -create
```

Or, on Windows:

```
C:\OpenGTS_1.2.3> bin\admin.bat Account -account:sysadmin -pass:password -create
```

Replace the above "`password`" specification with a secure password.

You should then be able to log in to the "`sysadmin`" account to see the new "System Admin" tab.

5) Installing “track.war”

The "track.war" (**W**eb-**AR**chive) runs in a Java Servlet container and works with the SQL DB datastore to provide a full-featured web interface to the GPS location data captured in the SQL database from remote devices. The mapping support currently uses OpenLayers/OpenStreetMap, but can be configured to use other commercial mapping service providers.

5.1) Configuring the "webapp.conf" file

The default runtime configuration file "webapp.conf" includes the file "common.conf", which in-turn includes "system.conf" and "custom.conf". For most installations, the default values specified in this file can be left as-is. However, some items, such as your SMTP server specifications, should be configured to fit your system requirements.

5.2) Configuring the available reports

Various detail and summary reports can be defined using the file "reports.xml", which can be found at "$GTS_HOME/reports.xml".

More detailed information can be found in the above "reports.xml" file, and elsewhere in this document. Here is a summary of the available features in the "reports.xml" file (experience in the general format and editing of XML files will be necessary).

"ReportLayout" defines the expected record format and the available columns for the defined report. The specific report layout is defined by the specified Java class, and 2 report layouts have been provided:

- **org.opengts.war.report.event.EventDataLayout** - This layout expects to display EventData records and specifies available columns based on the fields available in the EventData record.

- **org.opengts.war.report.field.FieldLayout** - This layout expects to display generic "FieldData" records and specifies various available columns type which can be used to display pertinent data.

The "Report" tag specifies a Java class which is bound to a specific ReportLayout. The "Report" also specifies how it is to be presented to the user (ie. the menu option), report title, displayed columns, and report selection criteria.

5.3) Configuring the Private Label look & feel

The configuration and customization of the web user interface can be specified in the file "$GTS_HOME/private.xml" (or "$GTS_HOME/private/private_common.xml" for the GTS Enterprise). This file controls the following options that are available on a 'Domain' basis (the domain name of the reference URL visiting the server):

- The Date/Time formats, and displayed TimeZones.
- The MapProvider used (ie. Google Maps, Microsoft Virtual Earth, Mapstraction, OpenLayers, etc). Including what pushpin icons are to be displayed on the map.
- The ReverseGeocodeProviders used to convert latitude/longitude values into a street address (Geonames, etc.).
- The GeocodeProvider used to convert street addresses into a latitude/longitude (if available).
- Available menu options, webpages, and customizing JSP files.
- Available report options.
- Access-Control-List (ACL) definitions.
- And much, much more ...

Please refer to the comments contained within the "private.xml" (or "private_common.xml") file for more information.

OpenGTS includes mapping support for OpenLayers/OpenStreetMap, Google Maps, Microsoft Virtual Earth, and Leaflet (which can support several other mapping service providers as well). Contact us regarding support for other commercial mapping service providers. If you will be using Google Maps for your map provider, you must also register for a Google Map key (make sure you comply with their terms of service) and place the returned key in the "`private.xml`" (or "`private_common.xml`") file at the location indicated (ie. replace "`*** Place Google Maps Key Here ***`" with your quoted key). To change the default displayed map pushpins, you can create your own 'Pushpins' section within your chosen MapProvider. See the '`private.xml`' file Pushpins section (in the "`openLayers`" MapProvider section) for more information regarding customizing pushpin icons.

Consult the contents of the provided private-label file at "`$GTS_HOME/private.xml`" for more information on specific customizations (experiance in the general format, editing, and syntax of XML files will be necessary).

> **5.3.a) Maintain proper XML syntax when modifying "`private.xml`" or "`reports.xml`"**
> Make sure that any changes to the '`private.xml`' file still comply with proper XML syntax. XML is very particular about proper syntax, and introducing an XML syntax error often results in an error message similar to the following when attempting to view the login page in a web browser:
>
> ```
> Invalid 'private.xml' configuration, please contact the System Administrator
> ```
>
> Run "`bin/checkInstall.sh`" to help diagnose any XML syntax errors that may have been introduced.

The general look-and-feel of the web-interface can also be changed by modifying the JSP file "`$GTS_HOME/war/track/jsp/loginSession.jsp`" and the various CSS files in the directory "`$GTS_HOME/war/track/css/`". Look for the "`WebPages`" tag section in the "`private.xml`" file for additional information regarding the customization of the "`loginSession.jsp`" file.

> **5.3.b) IMPORTANT: Redeploy all servlets after modifying any runtime configuration file**
> Changes to any of "`private.xml`", "`reports.xml`", "`webapp.conf`", "`common.conf`", "`system.conf`", or "`custom.conf`" files (or other "`.xml`" or "`.conf`" file) will require that the "`track.war`" (as well as the other servlets) file be re-built and re-deployed.

5.4) Compiling/Installing the "track.war" Java Servlet

To build the "`track.war`" file, run the Ant build command as follows:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> ant track
```

(note, the "`ant all`" performed above also builds the "`track.war`" file)

The target "`track`" is a wrapper for ant targets "`track.compile`" and "`track.war`". The target "`track.compile`" compiles all necessary classes and configuration files into the build directory "`$GTS_HOME/build/track`". The target "`track.war`" then creates the 'web archive' file "`$GTS_HOME/build/track.war`". If any of the runtime configuration files have changed, such as "`private.xml`", "`reports.xml`", "`webapp.conf`", or "`common.conf`" files (or possibly any other "`*.conf`" of "`*.xml`" file), then the "`track.war`" file must be rebuilt and redeployed. A shortcut to rebuilding the "`track.war`" file, if all source modules have already been compiled, is to issue the following command:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> ant track.war
```

This will simply repackage the "`track.war`" file from the pre-built source modules, and changed runtime configuration files. If everything has already been compiled, this command typically takes only a few seconds to complete.

Install the created "track.war" file per the Apache Tomcat installation/configuration instructions. Typically, this means copying the "track.war" file to the directory "$CATALINA_HOME/webapps/.":

```
/usr/local/OpenGTS_1.2.3> cp build/track.war $CATALINA_HOME/webapps/.
```

Or, the following "ant" target may also be used:

```
/usr/local/OpenGTS_1.2.3> ant track.deploy
```

The above method for deployment assumes that Tomcat is set for 'autoDeploy="true"'. If your changes to not appear after rebuilding and redeploying the "track.war" file, then it may be necessary to force Tomcat to update the "track.war" servlet by following these steps:

- Stop Tomcat (ie. "$CATALINA_HOME/bin/shutdown.sh")
- Delete the existing "track" servlet (ie. "rm -rf $CATALINA_HOME/webapps/track*")
- Deploy the new "track" servlet (ie. "cp $GTS_HOME/build/track.war $CATALINA_HOME/webapps/.")
- Restart Tomcat (ie. "$CATALINA_HOME/bin/startup.sh")

5.5) Testing the Installation

5.5.a) Secure web access:
Configuration and use of '**https**' (ie. SSL) is highly recommended as the URL includes the account password and will be encrypted via 'https', but will be sent in the clear if plain 'http' is used. Instructions for configuring Tomcat to support SSL can be found on the Apache Tomcat website.

After building/deploying 'track.war', you should be able to view the login page with a URL similar to the following:

```
http://localhost:8080/track/Track
```

(replace "'localhost:8080" with your own domain name where 'track.war' was installed.)
Note that the specification for the URL directory "/track/Track" is case sensitive.

Support for reverse-geocoding (turning a latitude/longitude into an address), using services such as Geonames (http://geonames.org) and Google, has also been included. Look for the "**ReverseGeocodeProvider**" tags in the 'private.xml' file for more information.

5.5.b) Browser Compatibility:
The GPS tracking map page in the web interface makes heavy use of JavaScript and HTML formatting. Firefox v3.X.X, Chrome 8.0.X, and Safari 5.0.X, are the platforms targeted, but it also appears to work fine (with some minor differences) on Microsoft IE 6.0/7.0/8.0 (some visual anomalies have been reported with earlier versions of IE). Other browsers have not been fully tested.

5.6) Installing Multiple Versions of "track.war"

The URL for accessing the login page is normally as follows:

```
http://localhost:8080/track/Track
```

The name "**track**" listed above derives it's name from the name for the war file, in this case "**track**.war". This means that you can install multiple/different copies of the "**track**.war" file, as long as the name of the war file is changed during the copy. For instance, if you copy the "**track**.war" file to Tomcat as follows:

```
/usr/local/OpenGTS_1.2.3> cp build/track.war $CATALINA_HOME/webapps/track1.war
```

Then you could access this installed version with the following URL:

```
http://localhost:8080/track1/Track
```

6) Installing “events.war”

The "events.war" (**W**eb-**AR**chive) runs in a Java Servlet container and works with the SQL DB datastore to allow downloading selected portions of a sequence of events over the web. This can be used with web-based mapping applications to provide near real-time tracking of a vehicle or person. The "'events.war" servlet currently supports data retrieval in KML, XML, CSV, TXT, GPX, or JSON file formats, and can be used in mapping programs such as Google Earth, or MS MapPoints.

6.1) Configuring the "webapp.conf" File

The default runtime configuration file "webapp.conf" includes the file "common.conf", which in-turn includes "system.conf" and "custom.conf". For most installations, the default values specified in this file can be left as-is. However, some items, such as your SMTP server specifications, should be configured to fit your system requirements.

Should you wish to customize the "webapp.conf" file specifically for the "events.war" servlet, copy this file to the directory "$GTS_HOME/war/events/WEB-INF/" and modify this copy.

6.2) Compiling/Installing the "events.war" Java Servlet

To build the "events.war" file, run the Ant build command as follows:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> ant events
```

(note, the "ant all" performed above also builds the "events.war" file)

The target "events" is a wrapper for ant targets "events.compile" and "events.war". The target "events.compile" compiles all necessary classes and configuration files into the build directory "$GTS_HOME/build/events". The target "events.war" then creates the 'web archive' file "$GTS_HOME/build/events.war".

Install the "events.war" file per the Apache Tomcat installation/configuration instructions. Typically, this simply involves copying the "events.war" file to the directory "$CATALINA_HOME/webapps/.". (The above method for deployment assumes that Tomcat is set for 'autoDeploy="true"')

6.3) Testing the Installation

Access the data stored in the SQL DB via the web with the following constructed URL:

```
http[s]://localhost:8080/events/<file>.{kml|xml|csv|txt|gpx|json}?
      a[ccount]=<account>    - the account name
     &u[ser]=<user>          - the user name
     &p[assword]=<password>  - the account/user password
     &d[evice]=<device>      - the device name
     &g[roup]=<group>        - the device group name (optional)
     [&rf=<fromTime>]        - optional 'from' data range.
     [&rt=<toTime>]          - optional 'to' data range.
     [&l[imit]=<limit>]      - optional 'limit' number of returned events.
```

Where "localhost:8080" should be replaced with the actual domain name and port used to access the Apache Tomcat web server. [Note: above items placed in square-brackets are optional. The options placed in curly braces indicate that one of the options within the curly braces should be selected].

Note: The '`rf`' and '`rt`' date ranges may be specified in 'Unix Epoch' time format (number of seconds since midnight Jan 1 1970) or in "`yyyy/mm/dd/HH:MM:SS`" format. If not specified, the last 100 events will be returned.

> **6.3.a) Note regarding secure web access:**
> Configuration and use of '**`https`**' (ie. SSL) is highly recommended as the URL includes the account password and will be encrypted via '`https`', but will be sent in the clear if plain '`http`' is used. Instructions for configuring Tomcat to support SSL can be found on the Apache Tomcat website.

Some examples:

- `https://localhost:8080/events/data.csv?a=myaccount&p=mypass&d=mobile`
  Return a CSV formatted data file ('`data.csv`') containing the last 100 event record for the device '**myaccount**'/'**mobile**'. The data is returned via an http SSL connection. (Note: replace '**mypass**' with the proper password)

- `http://localhost:8080/events/data.json?a=demo&p=mypass&d=demo`
  Return a JSON formatted data file ('`data.json`') containing the last 100 event record for the device '**demo**'/'**demo**'. (Note: replace '**mypass**' with the proper password)

- `http://localhost:8080/events/data.kml?a=gts&p=mypass&d=dev&rf=1145776000&rt=1145777000`
  Return a KML (XML) fomatted data file ('`data.kml`') with the first 100 events within the specified range for the device "gts/dev".

- `http://localhost:8080/events/data.gpx?a=gts&p=mypass&d=dev&rf=1145776000&rt=1145777000`
  Return a GPX (XML) fomatted data file ('`data.gpx`') with the first 100 events within the specified range for the device "gts/dev" (see "http://www.topografix.com/gpx.asp" for information regarding the GPX data format).

Google Earth has the capability of automatically polling data from this URL at specified intervals. To configure Google Earth to read event data points from the server, click on "Add" on the main menu bar, then select "Network Link". Add the KML retrieval URL to the server and click "Refresh Parameters" to be able to enter periodic refresh times. To always display the most recent events within Google Earth, omit the date range option ("`rf`" and "`rt`") and instead specify the option "`limit`" to cause the returned list to always include the latest set of events.

- `http://localhost:8080/events/data.kml?a=gts&p=mypass&d=dev&limit=100`
  Return a KML (XML) fomatted data file ('`data.kml`') with the last 100 available events for the device "gts/dev".

- `http://localhost:8080/events/data.kml?a=gts&p=mypass&d=dev&limit=1`
  Return a KML (XML) fomatted data file ('`data.kml`') with only the last (most recent) event for the device "gts/dev".

7) Database Administration

7.a) Important note regarding ".sh" and ".bat" command files:
Commands ending with ".sh" or ".bat" **MUST** be executed from the **OpenGTS** installation directory. Attempting to execute these commands from another directory may result in a "ClassNotFoundException" or "NoClassDefFoundError" error, or similar. (This means that you must cd to **$GTS_HOME**, then execute the command as "`bin/<command>`")

7.b) Important note for Windows users:
When using the "`.bat`" version of the commands in a DOS window, command arguments such as **`-rootUser=root`** must either be enclosed in quotes, as in **`"-rootUser=root"`**, or be specified with a colon instead of an equal sign, as in **`-rootUser:root`** .

Most database administration (Account, User, and Device, etc) can be performed through either the command-line utilities or through the web-interface. The example Account/User/Device editing examples shown below describe only a few of the possible fields in each of these tables. The file '`SCHEMA.txt`', included with the **OpenGTS** package, contains a list of the current tables, and the fields in each of the available tables.

A list of the currently defined tables and fields can also be generated with the following command:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -schema
```

Or, on Windows:

```
C:\zzz> cd %GTS_HOME%
C:\OpenGTS_1.2.3> bin\dbAConfig.bat -schema
```

7.1) Creating/Editing Accounts

The command "`bin/admin.sh Account`" supports many administrative function which act on the SQL "`Account`" table. Here are a few of the functions that can be performed using the "`bin/admin.sh Account`" command:

Creating an Account:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh Account -account=<acct> -create
```

This creates the specified Account with default values (replace "`<acct>`" with the account id you wish to create).

Editing an Account:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh Account -account=<acct> -edit
```

This command displays a command-line Account field editor, similar to the following:

(NOTE: The following is only an example. Your implementation will contain addtional field definitions. Please review the file 'SCHEMA.txt' in the OpenGTS package for a list of possible field definitions.)

```
----------------------------------------
Key: exampletable
----------------------------------------
 0) Password                          : "demo"
 1) Description                       : "Example Account"
 2) Is Active                         : "true"
 3) Contact Name                      : ""
 4) Contact Phone                     : ""
 5) Contact EMail Address             : ""
 6) Time Zone                         : "US/Hawaii"
 7) Speed Units                       : "0"
 8) Distance Units                    : "0"
 9) Geocoder mode                     : "0"
10) PrivateLabel Name                 : "*"
Enter field number [or 'save','exit']:
```

To select a field value to change, enter the field number, then hit enter. After changing the value of the field, hit enter again. Save your changes by finally entering "save".

Here is a description of a few of the Account fields *(please see '`SCHEMA.txt`' for a description of other possible field definitions)*:

Password – The Account login password. When logging in, if the user "admin" exists, then the "admin" password will be used, instead of this password, to authenticate the user.
Description – The Account description (used on reports, etc).
Is Active – This value is "true" if the Account is still considered in-service. If "false", then all connections by all owned devices will be refused.
Contact Name – The name of the contact person for the Account.
Contact Phone – The contact person's phone number.
Contact Email Address – The contact person's email addres.
Time Zone – The preferred timezone for the Account.
Speed Units – The preferred speed units for the Account. Valid values are: 0=mph, 1=kph, 2=knots.
Distance Units – The preferred distance units for the Account. Value values are: 0=Miles, 1=Kilometers, 2=Knots.
Geocoder mode – This is the reverse-geocoding mode used for this Account. Valid values are: 0=No reverse-geocoding performed, 1=Geozone lookup only, 2=Reverse-geocoding for high-priority status codes only, 3=Reverse-geocode everything (an available reverse-geocoding service is required).
PrivateLabel Name – This is the name of the '`Domain`' in the '`private.xml`' file to which this account should be assigned. If there is more than one '`Domain`' defined in the '`private.xml`' file, then this allows for using different reverse-geocoding, and mapping resources for different accounts.

Listing existing Accounts:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh Account -list
```

This lists all Accounts and owned Devices.

7.2) Creating/Editing Users

The command "`bin/admin.sh User`" supports several administrative functions which act on the SQL "`User`" table. Here are a few of the functions that can be performed using the "`bin/admin.sh User`" command:

Creating a User:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh User -account=<acct> -user=<user> -create
```

This creates the specified User with default values (replace "`<user>`" with the user id you wish to create). The user name "`admin`" is reserved for use by the Account administrator. When the Account administrator logs in (by leaving the user name field blank on the log in screen), then the log in process will check to see if the user "`admin`" exists. If this user name does exist, then the password and access-control assigned to the "admin" user will be used for the Account administrator (Note: the default login user can be changed on the Account Admin web page, or on the Account command-line edit).

Editing a User:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh User -account=<acct> -user=<user> -edit
```

This command displays a command-line User field editor, similar to the following:

(NOTE: The following is only an example. Your implementation will contain addtional field definitions. Please review the file '`SCHEMA.txt`' in the OpenGTS package for a list of possible field definitions.)

```
----------------------------------------
Key: account,user
----------------------------------------
 0) Password                          : ""
 1) Description                       : "New User"
 2) Is Active                         : "true"
 3) Contact Name                      : ""
 4) Contact Phone                     : ""
 5) Contact EMail Address             : ""
 6) Time Zone                         : ""
Enter field number [or 'save','exit']
```

To select a field value to change, enter the field number, then hit enter. After changing the value of the field, hit enter again. Save your changes by finally entering "save".

Here is a detailed description of a few of the User fields *(please see "`SCHEMA.txt`" for a description of other possible field definitions)*:

Password – The User login password. Leaving the password file empty will prevent the user from logging in. If you wish to allow the user to log in without having to enter a password, then you must set the password field to the test "**`*blank*`**" (case insensitive, and without the quotes of course).
Description – The User description (used on reports, etc).
Is Active – This value is "true" if the User is still considered in-service. If "false", then all login attempts by this User will be refused.
Contact Name – The name of the contact person for the User.
Contact Phone – The contact person's phone number.
Contact Email Address – The contact person's email addres.
Time Zone – The preferred timezone for the User

7.3) Creating/Editing Devices

The command "`bin/admin.sh Device`" supports many administrative functions which act on the SQL "`Device`" table. Here are a few of the functions that can be performed using the "`bin/admin.sh Device`" command:

Creating a Device:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh Device -account=<acct> -device=<dev> -create
```

This creates the specified Device with default values (replace "`<dev>`" with the device id you wish to create).

Editing a Device:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh Device -account=<acct> -device=<dev> -edit
```

This command displays a command-line User field editor, similar to the following:

(NOTE: The following is only an example. Your implementation will contain addtional field definitions. Please review the file "SCHEMA.txt" in the OpenGTS package for a list of possible field definitions.)

```
----------------------------------------
Key: myaccount,mobile
----------------------------------------
 0) Description                        : "New Device"
 1) Is Active                          : "true"
 2) Valid IP Addresses                 : ""
 3) ... etc.
Enter field number [or 'save','exit']:
```

To select a field value to change, enter the field number, then hit enter. After changing the value of the field, hit enter again. Save your changes by finally entering "save".

Retrieving Device Events through the command-line:

```
/usr/local/OpenGTS_1.2.3> bin/admin.sh Device -account=<acct> -device=<dev> -events
```

This command returns the last few events in CSV format.

```
Date,Time,Code,Latitude,Longitude,Speed,Heading,Altitude,Address
2007/03/07,23:13:21,InMotion,29.57241,-142.78869,103.9,178.3,287.0,""
2007/03/07,23:15:23,InMotion,29.57241,-142.78869,103.9,178.3,287.0,""
2007/03/07,23:17:23,InMotion,29.57241,-142.78869,103.9,178.3,287.0,""
2007/03/07,23:19:23,InMotion,29.57241,-142.78869,103.9,178.3,287.0,""
2007/03/07,23:21:25,InMotion,29.57241,-142.78869,103.9,178.3,287.0,""
2007/03/07,23:23:27,InMotion,29.57241,-142.78869,103.9,178.3,287.0,""
```

Deleting "Future" Events for a given Device:

Occasionally a GPS tracking device will emit a GPS date/time which is in the future, sometimes a long time into the future. To display the number of current events which are more than 5 minutes into the future, enter the following command:

```
...> bin/admin.sh Device -account=<acct> -device=<dev> -countFutureEvents=300
```

To delete these 'future' events:

```
...> bin/admin.sh Device -account=<acct> -device=<dev> -deleteFutureEvents=300
```

Deleting "old" Events for a given Device:

As events collect in the database, it may be handy at some point to trim old events from the system. The following command will count the number of events prior to 90 day ago:

```
...> bin/admin.sh Device -account=<acct> -device=<dev> -countOldEvents=-90d
```

To delete these "old" events:

```
...> bin/admin.sh Device -account=<acct> -device=<dev> -deleteOldEvents=-90d
```

The arguement value for the "`countOldEvents`" and "`deleteOldEvents`" can be specified using a relative time, such as "`-120d`", indicating events older than 120 days ago, or with a specific time specification, such as "`2010/03/12,13:15:00,GMT`" indicating events existing before March 12, 2010 1:15pm GMT.

WARNING: Deleting old events from the EventData table is final. Once the events have been deleted, they cannot be recovered.

7.4) General Database Administrative Functions

The command "`dbAdmin.pl`" (only available as a Perl script) can perform various administrative functions on the SQL database (Note: the command '`bin\dbConfig.bat`' is provided for Windows users, and performs a subset of the operations available to the "`dbAdmin.pl`" command). Here are a few of the functions that can be performed using the "`dbAdmin.pl`" command:

Verify/Update table columns:

```
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -tables
```

This command will check the column configuration of all **OpenGTS** tables and report on any missing columns, or other anomolies. If an **OpenGTS** table does not exist, it will be created.

```
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -tables=c
```

When upgrading to a newer version of **OpenGTS** in which new columns have been added to various tables, the above command will issue the required "`ALTER TABLE`" commands to the tables as required in order to add any new table columns. If a specific column 'type' has changed, using "`-tables=ca`" will cause column types to be altered.

```
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -tables=ca
```

Or, on Windows:

```
C:\OpenGTS_1.2.3> bin\dbConfig.bat -tables:ca
```

This command should be used whenever upgrading to a newer version of **OpenGTS**.

Dump tables to disk:

```
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -dump -dir=/tmp/gts
```

This command will dump all **OpenGTS** tables to the directory specified by the "`-dir`" argument ("`/tmp/gts`" is the default destination if the "`-dir`" option is not specified). If required by your MySQL installation, you may also need to specify the database root user (ie. as in "`-rootUser=<user>`"). Individual tables can then later be reloaded with the "`-load=<table>`" option.

(Note for Linux users: On some versions of Linux which employ SELinux, you may receive an error indicating that MySQL is not allowed to write into the specified directory. In these cases, you may need to either specify a directory that MySQL is allowed to write to, or change the SELinux security settings to provide MySQL with authorization to write to the "`/tmp`" directory.)

Load tables from Disk:

```
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -load=<table> -dir=/tmp/gts
```

This command will load the specified table from the file previously created by the "`-dump`" argument. During the table load, columns are matched where possible. If a column is present in the 'dumped' file, but has been removed in the current GTS table, a warning will be generated that the column has been dropped. This command is useful when small table changes need to be made in the column structure. [Note: the square brackets specified above indicate that the option within the brackets is optional. The square brackets should not be specified literally on the command-line if the optional argument within the brackets is used].

8) Installing/Starting the TK10x, and Aspicore DCS Modules

This section describes how to start and stop the "tk10x", and "aspicore" device communication server (DCS) modules, however these instructions are also applicable to other device communication servers which may also be installed or implemented.

The "tk10x" DCS module supports most common TK102/TK103 protocol compliant devices (Note: some manufacturers producing a TK102/TK103 device may be using their own custom protocol variant that is not compatible with the common TK102/TK103 protocol).

Aspicore provides client phone software for tracking various Nokia, Samsung, and Sony Ericsson phones. The Aspicore DCS within the OpenGTS package is designed to work with the TCP or UDP data transport method which can be configured within the Aspicore client phone application. For more information on their supported phones, and to obtain their client software, vistit their website at "http://www.aspicore.com/en/tuotteet_tracker.asp?tab=2&sub=1".

8.1) Configuring the "dcservers.xml" File

The file "dcservers.xml" contains a few configurable properties that effect the execution of the **Aspicore** ("aspicore") servers, and others. Most of the properties values should be left as their default value, but the following properties values can be set to those appropriate to your operating environment:

TK10X:
- tcpPort="31272"
- udpPort="31272"

Aspicore:
- tcpPort="31265"
- udpPort="31265"

These ports are specified on the "ListenPorts" tag for their respective "DCServer", and are the default ports on which these servers listen for incoming connections from the remote devices. You can change this port by changing the value on this tag attribute. You can also indicate multiple ports by specifying them with comma separators. (ie. 'tcpPort="31000,31100"').

8.2) Starting the Device Communication Server

The "tk10x", or "aspicore" server can be started as follows:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> bin/runserver.sh -s server
```

Where "***server***" should be replaced with the specific name of the server to start (ie. "tk10x", or "aspicore").

8.2a) Important note regarding ".sh" and ".bat" command files:
Commands ending with "`.sh`" or "`.bat`" MUST be executed from the **OpenGTS** installation directory. Attempting to execute these commands from another directory may result in a "ClassNotFoundException" or "NoClassDefFoundError" error, or similar. (This means that you must cd to **$GTS_HOME**, then execute the command as "`bin/<command>`")

Or, the Perl version of this command can be used without needing to be in the **OpenGTS** installation directory:

```
/zzz> $GTS_HOME/bin/runserver.pl -s server
```

On Windows, the command can omit the "`-s`" and can be entered as follows:

```
C:\> cd \OpenGTS_1.2.3
C:\OpenGTS_1.2.3\> \bin\runserver.bat server
```

The server will initialize and start listening on the port(s) specified by the "`ListenPorts`" tag in the "`dcservers.xml`" file for the specific named DCServer. To change the listen port on the command line, you can add a "`-port`" argument as follows:

```
/zzz> $GTS_HOME/bin/runserver.pl -s server -port 31123
```

Or on Windows:

```
C:\OpenGTS_1.2.3\> \bin\runserver.bat server -port:31123
```

To set listening on port "`31123`".

You can also add the command-line option "`-debugMode`" to enable debug-level logging.

While running in "background" mode, the output logs are stored in the file "`$GTS_HOME/logs/server.log`". (The file "`server.out`" is also created by "`runserver.pl`" to capture output to stdout/stderr, but will typically remain empty).

When testing/debugging, you may also start a server "interactively". That is, the server is run in the foreground (ie. not 'backgrounded'), and all logging output is sent to the console instead of the log file. To start a server "interactively", add the option "`-i`" to the command line, as follows:

```
/zzz> $GTS_HOME/bin/runserver.pl -s server -i
```

(To stop the server in this mode, simply press 'Control-C')

8.3) Stopping the Device Communication Server

When started in "background" mode (ie. Without the "`-i`" option), a "Process ID" (PID) file is created at "`$GTS_HOME/logs/server.pid`", which contains the process-id of the backgrounded task. This file is used to stop the server with the command:

```
/zzz> $GTS_HOME/bin/runserver.pl -s server -kill
```

Where "***server***" should be replaced with the specific name of the server to start (ie. s"`tk10x`", or "`aspicore`").

On Linux systems, the Perl command "psjava", included with **OpenGTS**, can be used to display information regarding which Device Communication Servers are currently runningr:

```
/zzz> $GTS_HOME/bin/psjava

  PID  Parent  L User     Java class/jar
------ ------  - -------- ------------------------------------------------------------
   215(     1) 1 opengts  org.apache.catalina.startup.Bootstrap
  8120(     1) 1 opengts  /usr/local/OpenGTS_1.2.3/build/lib/tk10x.jar
  8123(     1) 1 opengts  /usr/local/OpenGTS_1.2.3/build/lib/aspicore.jar
  8129(     1) 1 opengts  /usr/local/OpenGTS_1.2.3/build/lib/template.jar
```

This command lists all known running Java processes, with their associated PID (process-id), parent PID, user, and Java class or jar file which is currently running. The "org.apache.catalina.startup.Bootstrap" process indicates that Apache Tomcat is running with the indicated PID. "tk10x", "aspicore", and "template" indicate their respective running Device Communication Servers.

8.4) Adding a New Device Record

Once the proper device communication server (DCS) is running, login to an account you wish to have own the device/phone you will be tracking and add a new Device/Vehicle record on the "Device Admin" page.

TK10x:
TK102/TK103 devices typically report their unique identifying information using the IMEI number of the modem. On the "**Device Admin**" page, set the "**Unique ID**" field to the IMEI value in the format [**tk_12345679012345**], where "**123456789012345**" is the IMEI number of the phone (The "**tk_**" prefix is used to help identify the type of device communication server used to parse the data – in this case TK102/TK103).

Aspicore:
The Aspicore client software reports the unique identifying information using the IMEI number of the phone. On the "**Device Admin**" page, set the "**Unique ID**" field to the IMEI value in the format [**s60_12345679012345**], where "**123456789012345**" is the IMEI number of the phone (The "**s60_**" prefix is used to help identify the type of device communication server used to parse the data – in this case Aspicore).

9) Creating your own Device Communication Server

In order for **OpenGTS** to receive data from a device, a customized "Device Communication Server" will need to be implemented that understands the protocol used to communicate with the remote device, and insert received events into the SQL database. This section provides a brief description of the example 'template' servers provided with **OpenGTS** for implementing your own remote device communication server.

The method used by remote devices to transport events to the server varies greatly with the manufacturer of the device. Some transport data to a server via SMS messages, some use an SMTP email transport to send data to a server, some use an HTTP-based protocol which encode data in the request to the server, and many use some form of raw-socket based communication (via TCP/UDP) to connect to a listener on the server to transmit data. In order to create a device communication server that is able to parse incoming data from a device, an intimate understanding of the specifics of the protocol used by the device manufacturer is required. **OpenGTS** includes examples for HTTP-based servers, and raw-socket based servers (supporting both TCP and UDP), however, since each device manufacturer typically has their own protocol and way of transporting data, these example servers cannot be assumed to just-work with any/every device. Depending on the particular device chosen, there may be a significant and substantial amount of work necessary in order to support the chosen hardware device.

The first, and most important, step when starting to implement a device communication server for a chosen hardware device is to obtain and fully understand the protocol documentation from the manufacturer of the device. While http-based communication can often be reverse-engineered from observing the data arriving at the server, attempting to reverse-engineer a raw-socket based protocol can prove extremely difficult, if not impossible, without proper protocol documentation.

9.a) Important note regarding the implemention of a device communication server:
Implementing a device communication server for a given device may take a significant and substantial amount of programming work to accomplish, depending on the device protocol. To implement a server, you will likely need an in-depth understanding of TCP/UDP based communication, and a good understanding of Java programming techniques, including socket communication, multi-threading, and possibly bitwise manipulation techniques for extracting bit fields from binary data (including knowing whether the tracking device sends data in big-endian or little-endian formats). If using an http-based protocol, you may also need an understanding of how servlets are built, and how they operate within a Servlet container, such as Apache Tomcat.

9.1) HTTP Based Device Communication Servers (using the "gprmc" servlet)

[Skills required: Java, Servlets, HTTP based protocols]

HTTP-based communication is typically the easiest to implement. If your remote tracking device sends messages to the server using an HTTP-base communication protocol, then the example server '`gprmc`' can be modified to parse received data and insert it into the SQL database. The sources for the '`gprmc`' server (named after the $GPRMC record in the NMEA-0183 protocol) can be found in the **OpenGTS** source directory "`src/org/opengts/war/gprmc`" and runs as a Servlet in a servlet container such as Apache Tomcat (and uses the same access ports configured for Tomcat).

9.1.1) Configuring the "gprmc" Servlet:

This module has many built-in configurable options and may only need special customization properties which can be specified in the "`webapp.conf`" file. The following is a list of configurable properties available for the "`gprmc`" http-based device communication server:

Data format/value Property Keys:

- **`gprmc.logName`**
  Logging output name.
  Default value: **`gprmc`**
- **`gprmc.uniquePrefix`**
  UniqueID prefix, used when lookup up Device.
  Defaults value: *(same as "**`gprmc.logName`**" property value)*
- **`gprmc.defaultAccountID`**
  Default account ID.
  Default value: **`gprmc`**
- **`gprmc.minimumSpeedKPH`**
  Minimum acceptable speed (speeds less than this value will be set to '0')
  Default value: **`0.0`**
- **`gprmc.estimateOdometer`**
  If true, a GPS based odometer will be calculated.
  Default value: **`false`**
- **`gprmc.simulateGeozones`**
  If true, a geozone arrival/departures will be detected, and the appropriate additional events will be created.
  Default value: **`false`**
- **`gprmc.dateFormat`**
  Date format for 'date' parameter.
  Valid values: **`NONE`**, **`EPOCH`**, **`YMD`**, **`DMY`**, **`MDY`**
  Default value: **`YMD`**
- **`gprmc.response.ok`**
  Response returned to device on on successful data receipt.
  Default value: *(nothing returned)*
- **`gprmc.response.error`**
  Response on returned to device on on unsuccessful (error) data receipt.
  Default value: *(nothing returned)*

URL Parameter Property Keys:

- **gprmc.parm.unique**
  URL Parameter Key: Unique-ID
  Default value: **id**
- **gprmc.parm.account**
  URL Parameter Key: Account-ID
  Default value: **acct**
- **gprmc.parm.device**
  URL Parameter Key: Device-ID
  Default value: dev
- **gprmc.parm.auth**
  URL Parameter Key: Auth/Password (not used)
  Default value: *(blank)*
- **gprmc.parm.status**
  URL Parameter Key: StatusCode
  Default value: **code**
- **gprmc.parm.gprmc**
  URL Parameter Key: GPRMC (the NMEA-0183 $GPRMC record is expected for this value of this parameter)
  Default value: **gprmc**
- **gprmc.parm.date**
  URL Parameter Key: Date (ignored if '**gprmc.parm.gprmc**' is specified)
  Default value: **date**
- **gprmc.parm.time**
  URL Parameter Key: Time (ignored if '**gprmc.parm.gprmc**' is specified)
  Default value: **time**
- **gprmc.parm.latitude**
  URL Parameter Key: Latitude (ignored if '**gprmc.parm.gprmc**' is specified)
  Default value: **lat**
- **gprmc.parm.longitude**
  URL Parameter Key: Longitude (ignored if '**gprmc.parm.gprmc**' is specified)
  Default value: **lon**
- **gprmc.parm.speed**
  URL Parameter Key: Speed(kph) parameter key (ignored if '**gprmc.parm.gprmc**' is specified)
  Default value: **speed**
- **gprmc.parm.heading**
  URL Parameter Key: Heading(degrees) (ignored if '**gprmc.parm.gprmc**' is specified)
  Default value: **head**
- **gprmc.parm.altitude**
  URL Parameter Key: Altitude(meters)
  Default value: **alt**
- **gprmc.parm.odometer**
  URL Parameter Key: Odometer(kilometers)
  Default value: **odom**
- **gprmc.parm.address**
  URL Parameter Key: Reverse-Geocoded Address
  Default value: **addr**
- **gprmc.parm.driver**
  URL Parameter Key: Driver ID
  Default value: **drv**
- **gprmc.parm.message**
  URL Parameter Key: Message
  Default value: **msg**

9.1.2) Default "gprmc" Configuration:

The default "gprmc" cofiguration (with no special configuration changes made in the "webapp.conf" file) expects a http-based URL formatted as follows: (the URL below displays the various fields on separate lines, but is normally sent to the server as a single URL all on a single line):

```
http://example.com:8080/gprmc/Data?
   id=123456789012345&
   code=0xF020&
   gprmc=$GPRMC,080701.00,A,3128.7540,N,14257.6714,W,000.0,000.0,180707,,,A*1C
```

Where

`http://example.com:8080/gprmc/Data?`
Represents the host:port ("example.com:8080") and "gprmc.war" servlet location ("/gprmc/Data?")

`123456789012345`
Represents the unique mobile ID of the device (such as the IMEI#). (This value should be entered into the "Device Admin" page "Unique ID" field in the format "gprmc_123456789012345").

`0xF020`
Represents the status code used for identifying the reason for the event. The status code used here should match the hex, or decimal value of a status code listed in the "Status Codes and Descriptions" manual at the following location:
http://www.geotelematic.com/docs/StatusCodes.pdf
In addition to one of the above hex/decimal status codes, any of the following text code names may also be used, which will automatically be converted into it's corresponding hex-based status code when it is entered into the EventData table: GPS, PANIC, WAYMARK, etc, (see gprmc source module "Data.java" for a current list).

`$GPRMC,080701.00,A,3128.7540,N,14257.6714,W,000.0,000.0,180707,,,A*1C`
Represents the NMEA-0183 $GPRMC record straight from the GPS receiver.

9.1.3) Building the "gprmc" Servlet:

The "gprmc.war" file is automatically built by the OpenGTS "ant all" command, but can also be built specifically with the following command:

```
ant gprmc
```

After making any configuration changes to the above properties, the "gprmc.war" file must be rebuilt and redeployed in order for the configuration changes to take effect.

If you need to make changes to this source module to support your device, it is recommended that you copy the source files to an new directory (remember to change the Java package name, and also copy/modify the "web.xml" information found at "war/gprmc/WEB-INF/web.xml"), then modify the copied sources with the specific changes required by your tracking device.

This module will need to be configured and installed in a servlet container, similar to the installation process used for other servlets in this documentation.

Consult the 'gprmc' server source code directly for additional information. (Source code for this module can be found in the directory "src/org/opengts/war/gprmc/").

9.2) Raw Socket Based Device Communication Server

[Skills required: Java, TCP/UDP socket communication, multi-threading, bitwise manipulation, general client/server protocols]

If your remote tracking device sends messages to the server using a socket-based communication protocol, then the example '`template`' server can be modified to parse received data and insert it into the SQL database. The sources for the '`template`' server can be found in the **OpenGTS** source directory "`src/org/opengts/servers/template`". This server type runs as a separate process listening on a selected socket port for incoming TCP/UDP connections.

You will likely need to make some significant changes to the source code to support your particular device. It is recommended that you copy the source files to an new directory (remember to change the Java package name), then modify the copied sources with the specific changes required by your tracking device.

Here are some of the main attributes of the protocol that need to be determined before starting to implement a TCP/UDP socket based device communication server:

- Are the protocol packets transmitted in ASCII, Binary, or both?
- How is the actual length of a client packet determined (this is extremely important)?
- What are the various types and content of packets received from the client?
- For binary packets, are integer fields encoded in Big-Endian or Little-Endian format?
- What response packets, if any, is the client expecting to receive from the server?

The example 'template' server contains the following source modules:

`Constants.java` – This module constains most of the customized configurable options used to support your specific remote device protocol. Such as timeouts, minimum/maximum packet lengths, ASCII/Binary encoding, etc. The basic information regarding the type of protocol should be set in this module.

`Main.java` – This is the main entry point for the server and will not likely need to be modified.

`TrackClientPacketHandler.java` – The purpose of this module is to understands the specific characteristics of the communication protocol for the remote device, and will require most of the significant customization required to support your remote device. This is where incoming client packets are identified, data is parsed and inserted into the tables, and any required responses are returned to the client device.

`TrackServer.java` – This is a wrapper/handler for a TCP/UDP session and delegates most control to the `TrackClientPacketHandler` class.

This server environment/framework handles listening for incoming connections and multi-threading for you (this server can handle multiple simultaneous incoming connections), as well as most error handling.

When using the example '`template`' server as the basis for your own device communication server, it is recommended that you copy the '`template`' files into a new folder (and thus a new Java package) named after your chosen tracking/telematic device (ie. such as "`acme1000`"). The "`build.xml`" file should also be modified to include a 'target' for your chosen server name. To build the example '`template`' server, the Ant target '`template`' can be used as follows:

```
> ant template
```

This '`template`' target in '`build.xml`' can be copied to create your own specific ant build target for your chosen device communication server name.

Consult the '`template`' server source code directly for additional customization information.

The server "`icare`" is also available (in the source directory "`src/org/opengts/servers/icare/`"), which supports the ICare G3300 device. This device communication server module can also be examined for help in creating your own customized device communication server.

9.2.1) Starting the Device Communication Server

Once your server jar file has been created, you can use the "`bin/runserver.sh`" command (or "`runserver.bat`" on Windows) to start your device communication server as follows:

```
/zzz> cd $GTS_HOME
/usr/local/OpenGTS_1.2.3> bin/runserver.sh -s template
```

> **9.2.1.a) Important note regarding ".sh" and ".bat" command files:**
> Commands ending with ".sh" or ".bat" MUST be executed from the **OpenGTS** installation directory. Attempting to execute these commands from another directory may result in a "ClassNotFoundException" or "NoClassDefFoundError" error, or similar. (This means that you must cd to **$GTS_HOME**, then execute the command as "`bin/<command>`")

Or, the Perl version of this command can be used without needing to be in the **OpenGTS** installation directory:

```
/zzz> $GTS_HOME/bin/runserver.pl -s template
```

On Windows, the command can omit the "`-s`" and can be entered as follows:

```
C:\> cd \OpenGTS_1.2.3
C:\OpenGTS_1.2.3\> \bin\runserver.bat template
```

The server will initialize and start listening on the port(s) specified in the "`dcservers.xml`" file for the "`template`" server. The default port is `31200`. To change the listen port on the command line, a "`-port`" can be added as follows:

```
/zzz> $GTS_HOME/bin/runserver.pl -s template -port 31123
```

Or, on Windows:

```
C:\OpenGTS_1.2.3\> \bin\runserver.bat template -port:31123
```

To set listening on port "`31123`" (for example).

You can also add the command-line option "`-debugMode`" to enable debug-level logging.

While running in "background" mode, the output logs are stored in the file "`$GTS_HOME/logs/template.log`". (The file "`template.out`" is also created in this directory to catch any stdout/stderr messages which are displayed within the server that don't use the 'Print' logging feature built into OpenGTS. Typically, this file should alway be empty).

When testing/debugging, you may also start a server "interactively". That is, the server is run in the foreground (ie. not 'backgrounded'), and all logging output is sent to the console instead of the log file. To start a server "interactively", add the option "-i" to the command line, as follows:

```
/zzz> $GTS_HOME/bin/runserver.pl -s template -i
```

(To stop the server in this mode, simply press 'Control-C')

9.2.2) Stopping the Device Communication Server

When started in "background" mode (ie. Without the "`-i`" option), a "Process ID" (PID) file is created at "`$GTS_HOME/logs/template.pid`", which contains the process-id of the backgrounded task. This file is used to stop the server with the command:

```
/zzz> $GTS_HOME/bin/runserver.pl -s template -kill
```

On Linux systems, the Perl command "`psjava`", included with **OpenGTS**, can be used to display information regarding which Device Communication Servers are currently running, including the "`template`" server:

```
/zzz> $GTS_HOME/bin/psjava

  PID  Parent  L User     Java class/jar
------ ------  - -------- ------------------------------------------------------
   215(     1) 1 opengts  org.apache.catalina.startup.Bootstrap
  8115(     1) 1 opengts  /usr/local/OpenGTS_1.2.3/build/lib/gtsdmtp.jar
  8129(     1) 1 opengts  /usr/local/OpenGTS_1.2.3/build/lib/template.jar
```

This command lists all known running Java processes, with their associated PID (process-id), parent PID, user, and Java class or jar file which is currently running. The "`org.apache.catalina.startup.Bootstrap`" process indicates that Apache Tomcat is running with the indicated PID. "`gtsdmtp`" and "`template`" indicate their respective running Device Communication Servers.

9.3) Runtime XML Configuration File.

The raw socket-based device communication servers (such as the example 'template' server) support the runtime configuration using the file "`dcservers.xml`". An example format of the "`dcservers.xml`" is as follows:

```
<DCServerConfig
    bindAddress=""
    backlog=""
    portOffset="0"
    includeDir="dcservers"
    >

    <Include file="dcserver_template.xml" optional="true"/>

    <DCServer name="icare">
       ...
    </DCServer>
    ...

</DCServerConfig>
```

The attributes for the `DCServerConfig` tag include the following:

`bindAddress` : This attribute specifies the local IP address or host name to which the server will bind. This is useful when the local server has more than one IP address, and needs to send UDP packets back to a client device. If left blank, the server will bind to the default local IP address.

`backlog` : The maximum queue length for incoming connection indications (a request to connect). If a connection indication arrives when the queue is full, the connection is refused. If left blank, or is 0 or less, then the default backlog value will be used. See the class "`java.net.ServerSocket`" for more information.

`portOffset` : This value is added to any port specification. Unless otherwise needed for specific system requirements, this value should remain "0".

`includeDir` : If the "`DCServerConfig`" tag contains any "`Include`" sub-tags, this is the directory that will be search for the included files.

An example "Include" tag format is as follows:

```
<Include file="dcserver_template.xml" optional="true"/>
```

The attributes for the Include tag include the following:

file : This attribute specifies the name of the file to include. The included file must also be a properly formatted DCServerConfig XML file. All device communication servers defined within this included file (as defined by the "DCServer" tags) will be added to the device communication servers defined elswhere in this XML file. Recursive Include directives are not allowed.

optional : This attribute specifies whether the include file is required to exist. If this value is "true" and the include file does not exist, and error will be displayed. If this value is "false" and the include file does not exist, then the Include directory is quietly ignored.

An example "DCServer" tag format is as follows:

```
<DCServer name="template">

    <Description><![CDATA[
        Example Template Server
        ]]></Description>

    <UniqueIDPrefix><![CDATA[
        template_
        imei_
        *
        ]]></UniqueIDPrefix>

    <ListenPorts
        tcpPort="31200"
        udpPort="31200"
        />

    <Properties>
        <Property key="minimumSpeedKPH">4.0</Property>
        <Property key="estimateOdometer">true</Property>
        <Property key="simulateGeozones">true</Property>
        </Properties>

</DCServer>
```

The attribute for the DCServer tag are as follows:

name : This attribute is required and specifies the name of the device communication server. The specified name should be unique among all loaded device communication servers. If a name of a device communication server is encountered that has already been defined, the subsequent named DCServer entry will be ignored.

"Description" sub-tag:
This tag specifies the optional description of the device communication server.

"UniqueIDPrefix" sub-tag:
This tag specifies the optional "Unique-ID" prefixes that will be used when looking up the device mobile-id in the Device table. In the order specified, the specified prefix is prepended to the mobile-id then the resulting ID is looked-up in the Device table "uniqueID" field. If not found, then the next prefix will be used. The prefix specification "*" means that the mobile-id will be used as-is (without any prepended prefix).

"ListenPorts" sub-tag:

This tag specifies the ports on which the device communication server will listen for incoming connections from remote devices. The attribute "tcpPort" specifies the port on which a TCP listener will be started. The attribute "udpPort" specified the port on which a UDP listener will be started. If either "tcpPort" or "udpPort" is blank, or not specified, the the corresponding "listener" will not be started.

"Properties" sub-tag:

This tag includes "Property" sub-tags which specify runtime properties which can be used to further specify the behavior of the device communication server at runtime. The standard properties that most device communication server recognize are as follows:

minimumSpeedKPH : (Double) This property specifies the minimum acceptable GPS-based speed. A speed value beow the value specified by this property will be considered a speed of '0'. This is used to mitigate GPS speed values which can indicate motion, even when the GPS receiver is stationary.

estimateOdometer : (Boolean) This property specifies the whether a GPS-based odometer value should be automatically calculated from the valid GPS locations reported by the incoming event. The odometer value of the current event is calculated by determining the distance from the previous event location to the current event location, then adding this distance to the previous odometer value.

simulateGeozones : (Boolean) This property specifies whether incoming events should be checked for Geozone arrive/depart occurances. If the current event was found to have arrive, or have departed, from a Geozone (as listed in the Geozone table), then the appropriate event, with the arrive/depart status code, will be generated and inserted into the EventData table.

Specific device communication servers may also support other property specifications.

10) Internationalization/Localization

The **OpenGTS** source code is i18n ("internationalization") compliant, which means that it can adapted to various lanugages without requiring any code changes (ie. no fixed hardcoded text that cannot be changed at display time). Localization ("L10n") is the process of applying language specific text for a given Locale.

10.1) Supporting a New Language

All English text which is displayable within the **OpenGTS** web-interface can be found in the resource property files called "`LocalStrings_en.properties`" found within the various source file directories. These files contain a listing of the default English text which will be displayed if no specific locale has been chosen. The format of this file is "`textKey=value`", where the "`textKey`" is a unique id for the specific text "`value`".

10.1.a) Download the latest Locale files for the next release:
You can download a zip file containing the "`LocalStrings_xx.properties`" files for the next release of OpenGTS at the following URL:
`http://www.geotelematic.com/download.html#LocalStrings`

To support a new language/locale, search for every occurance of the file "`LocalStrings_en.properties`" and create a copy named "`LocalStrings_`***`xx`***`.properties`" in the same directory, where "***`xx`***" in this case is the **ISO-639** 2-letter language code. Then in the new file, remove the prefixing "`#`" comment character on each key=value line, and modify the text value to match your specific language syntax and meaning. Make sure to leave the text key as-is. The contents of the "`LocalStrings_`***`xx`***`.properties`" files **must** be written using only **ISO-8859-1** encoded characters (per "`java.util.Properties`" and "`java.util.ResourceBundle`" restrictions). Characters that cannot be directly represented in **ISO-8859-1** can be included using **Unicode** escapes by specifying the format '`\u`***`xxxx`***', where '***`xxxx`***' is the hex representation of the unicode character. The Java JDK installation includes the tool '`native2ascii`' to assist with converting unicode text to unicode escaped text.

If a "`LocalStrings_XX.properties`" file does not contain proper **Unicode** encoding (ie. an invalid "`\u`***`xxxx`***" specification), then Java may be unable to load the properties file, and the specific LocalStrings file may not be used. To verify that no invalid "`\u`***`xxxx`***" specifications are present, the following CheckInstall command will validate the "`LocalStrings_`***`xx`***`.properties`" files:

```
/zzz> $GTS_HOME/bin/checkInstall.sh -- -localStrings=$GTS_HOME/src
```

This command will search for all "`LocalStrings_`***`xx`***`.properties`" files in the "`$GTS_HOME/src`" directory and examine their contents for invalid "`\u`***`xxxx`***" specifications. If any LocalStrings file contains invalid "`\u`***`xxxx`***" specifications, the error message "Malformed \uxxxx encoding" will be displayed for the particular file. This command will also check to see that only **ISO-8859-1** characters are used in the LocalStrings files.

The specific language locale displayed on the web-interface is controlled by the "`locale`" attribute on the "`Domain`" tag in the "`private.xml`" file.

After making any changes to the "`private.xml`" file, or any of the "`LocalStrings_XX.properties`" files, make sure you rebuild and redeploy the "`track.war`" file.

10.2) Changing the Displayed Language

Setting the displayed language can be configured in the '`private.xml`' file. On the "`Domain`" tag, change the "`locale`" to the desired 2-letter country code. For example, to change the language to German, set the Domain "`locale`" attribute to '`locale="de"`'. You can also add a pull-down language selection menu to the login page by setting the Property "`accountLogin.showLocaleSelection`" to "`true`". The language selections are specified in the "`SupportedLocales`" tag.
(Rebuild/redeploy the '`track.war`' file after making any changes to the '`private.xml`' file.)

11) Creating/Modifying Reports

OpenGTS comes with a very simple and configurable report generation engine. Reports are comprised of 3 main components: the report layout, the report data iterator, and the report specification XML.

The report specification XML specifies a report data iterator, reporting constraints, and the columns which are to appear on the report. The report data iterator constructs the data which will be included in the report based on the reporting constraints. The report data layout then iterates through the report data and generates a report based on the column formatting information provided by the report specification XML.

The "Report Layout" and "Report Data Iterator" components must be implemented in Java code by a Java programmer, and should be configurable for a general use. The "Report Specification XML" is a report configuration text file that specifies the type, columns, and contraints for a specific report. Provided the report layout and data iterator are implemented for general use, many different kinds of reports may be created that utilize the same layout and data iterator.

11.1) Report Layout

The Report Layout is a Java module that defines what columns are available for a given report, and their respective formatting options.

A report layout must extend the abstract Java class "`org.opengts.war.report.ReportLayout`" and must define a "`DataRow`" subclass that understands how to parse report columna/fields from report row objects provided by the report data iterator.

The class "`org.opengts.war.report.event.EventDataLayout`" is an example ReportLayout subclass that defines the available columns and formatting options for the Event Detail and Summary reports.

11.2) Report Data Iterator

The Report Data Iterator is a Java module that constructs the list of records that are to be included in the report based on the constraints specified in the report specification XML.

A report data iterator must extend the abstract Java class "`org.opengts.war.report.ReportData`" and provide implementations for the "`getBodyDataIterator`" and "`getTotalDataIterator`" methods. It must also bind to a specific ReportLayout by providing an implementation for the "`getReportLayout`" method.

The class "`org.opengts.war.report.event.EventDetailReport`" is an example ReportData subclass that generates the Event Detail report.

11.3) Report Definition XML

The file "`report.xml`" defines the html style used for a column defined in a ReportLayout. If also defines specific reports by specifying which ReportData iterator, and which columns will be included in a given report. It also specifies the constraints that are to be applied to the data which the report will contain.

Here is an example report definition from the '`report.xml`' file for the "Event Detail" report:

```
<!--
 === The 'name' provides a name for the report, referenced in 'private.xml'
 === The 'type' provides a report group name, referenced in 'private.xml'
 === The 'class' specifies the report data iterator used to generate the report
 -->
<Report name="EventDetail" type="device.detail"
    class="org.opengts.war.report.event.EventDetailReport">

   <!-- The description of the report display on the reporting menu -->
   <MenuDescription i18n="ReportsXML.eventDetail.menu">
      Event Detail
   </MenuDescription>

   <!-- The title displayed above the report -->
   <Title i18n="ReportsXML.eventDetail.title">
      Event Detail
   </Title>

   <!-- The subtitle displayed above the report -->
   <Subtitle i18n="ReportsXML.eventDetail.subtitle">
      ${deviceDesc} [${deviceId}]\n${dateRange}
   </Subtitle>

   <!-- The columns included in the report -->
   <Columns>
      <Column name="index"                      />
      <Column name="date"                       />
      <Column name="time"                       />
      <Column name="statusDesc"                 />
      <Column name="latitude"           arg="5" />
      <Column name="longitude"          arg="5" />
      <Column name="speedH"             arg="1" />
      <Column name="altitude"                   />
      <Column name="odometer"           arg="0" />
      <Column name="address"                    />
   </Columns>

   <!-- The report data constraints -->
   <Constraints>
      <SelectionLimit type="first">1000</SelectionLimit>
      <ReportLimit>1000</ReportLimit>
      <OrderAscending>true</OrderAscending>
   </Constraints>

   <!-- the map icon selector (if map display is enabled) -->
   <MapIconSelector ruleFactoryName="CustomRulesEngine">
      <!-- this section requires an installed "RuleFactory" implementation -->
      <![CDATA[ ((mph<4)?"reddot":(speed<15)?"yellow":"heading") ]]>
   </MapIconSelector>

</Report>
```

11.4) Available Report Specifications

Once a report has been defined in the "`report.xml`" file, it can be made available for user selection in the web-interface by referencing the report name in the "`private.xml`" file in the "`Reports`" tag.

Here is an example report specification from the "`private.xml`" file:

```
<!-- Defined reports
 === All reports referenced here must be predefined in 'reports.xml'
 -->
<Reports>
   <Report name="EventDetail">
      <AclName>acl.report.eventDetail</AclName>
      </Report>
   <Report name="EventSummary">
      <AclName>acl.report.eventSummary</AclName>
      </Report>
   <!-- ... -->
</Reports>
```

Appendix)

A) Support for Microsoft SQL Server

Initial support for Microsoft SQL Server has been included with the standard **OpenGTS**. Microsoft SQL Server 2005 can be downloaded/installed from the following Microsoft webpage(s):

```
http://www.microsoft.com/sqlserver/2005/en/us/express.aspx
http://www.microsoft.com/Sqlserver/2005/en/us/express-down.aspx
```

To enable support for SQL Server, modify the file "`common.conf`" accordingly to turn off support for MySQL, and turn on support for Microsoft SQL Server:

```
# --- Microsoft SQL Server
db.sql.provider=sqlserver
db.sql.host=localhost
db.sql.port=3193
db.sql.dbname=gts
db.sql.user=gts
db.sql.password=opengts
db.sql.url=jdbc:sqlserver://${db.sql.host}:${db.sql.port}
db.sql.url.db=${db.sql.url};databaseName=${db.sql.dbname}
```

The JDBC driver support for SQL Server will also need to be downloaded/installed into the Java runtime environments described below.

Download:

```
http://msdn.microsoft.com/en-us/data/ff658549
```

Install:

%JAVA_HOME%`\jre\lib\ext\.`

*(where **%JAVA_HOME%** is the location of your Java installation)*

Once "`common.conf`" have been modified, and the SQL Server JDBC driver has been installed, recompile the **OpenGTS** code and initialize the database, and install the components, as described above in this document.

Please contact us with any issues you encounter, or suggestions you may have regarding support for Microsoft SQL Server.

Appendix)

B) Optional Table Columns

Various tables within the OpenGTS provide for additional table columns which can be used for special application requirements. These table columns can be enabled by setting specific propery values within one of the available "`.conf`" files (ie. "`config.conf`", etc). The following section describes the various optional table columns, and the property name that can be used to enable these columns.

After adding the specified property to the chosen "`.conf`" file, the tables will need to be updated with the newly added columns. To update the columns within the various database tables, run the "`dbAdmin.pl`" command as follows:

```
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -tables=ca
```

Or, on Windows:

```
C:\OpenGTS_1.2.3> bin\dbConfig.bat -tables:ca
```

The following command will display the columns defined within each of the tables used by OpenGTS (note: the above "`dbAdmin.pl`" or "`dbConfig.bat`" commands are still required to ensure that the defined table columns are also added to the actual database table):

```
/usr/local/OpenGTS_1.2.3> bin/dbAdmin.pl -schema=TABLE_NAME
```

Or, on Windows:

```
C:\OpenGTS_1.2.3> bin\dbConfig.bat -schema:TABLE_NAME
```

Where "**TABLE_NAME**" is replaced with the name of the table for which you wish to display the defined columns. If "**TABLE_NAME**" is omitted, then defined columns for all tables will be displayed.

> **C.a) IMPORTANT: Redeploy all servlets after modifying any runtime configuration file**
> Changes to any of "`private.xml`", "`reports.xml`", "`webapp.conf`", "`common.conf`", "`system.conf`", or "`custom.conf`" files (or other "`.xml`" or "`.conf`" file) will require that the "`track.war`" (as well as the other servlets) file be re-built and re-deployed.

Note: The fields described below may only contain a partial listing of the fields which may be available in the various table optional fields. Please consult the table source module for a definitive list of included fields.

B.1) Optional Account Table Columns:

AddressFieldInfo
Config Property: `startupInit.Account.AddressFieldInfo=true`
These fields are used to store additional Address information, as specified by application requirements.

```
addressLine1
addressLine2
addressLine3
addressCity
addressState
addressPostalCode
addressCountry
```

MapLegendFieldInfo
Config Property: `startupInit.Account.MapLegendFieldInfo=true`
These fields are used to store custom map legend information.

```
mapLegendDevice
mapLegendGroup
```

AccountManagerInfo
Config Property: `startupInit.Account.AccountManagerInfo=true`
These fields are used to store Account Manager configuration information.

```
isAccountManager
managerID
```

DataPushInfo
Config Property: `startupInit.Account.DataPushInfo=true`
These fields are used to store the state of certain account/event data 'pushed' to an alternate server:

```
requestPassCode
requestIPAddress
dataPushURL
lastDataRequestTime
lastDataPushTime
```

B.2) Optional Device Table Columns:

NotificationFieldInfo
Config Property: `startupInit.Device.NotificationFieldInfo=true`
These fields are used by the installed "`RuleFactory`" implementation, or possibly the extended Event Notification Rules Engine (ENRE) module. On the open-source OpenGTS version, this can be the `RuleFactoryExample.java` module. When enabled, the "Device Admin" page will be automatically adjusted to display some of these fields.

```
allowNotify
lastNotifyTime
lastNotifyCode
notifyEmail
notifySelector
notifyAction
notifyDescription
notifySubject
notifyText
notifyUseWrapper
notifyPriority
parkedLatitude
parkedLongitude
parkedRadius
```

LinkFieldInfo
Config Property: `startupInit.Device.LinkFieldInfo=true`
These fields are used to store URL link information for displaying on the Device map page (see "`trackMap.showDeviceLink`" property in "`private.xml`"), and possibly on email notifications (if configured).

```
linkURL
linkDescription
```

DataPushInfo
Config Property: `startupInit.Device.DataPushInfo=true`
These fields are used to store information regarding the last time EventData records were sent to another system.

```
lastDataPushTime
lastEventCreateMillis
```

FixedLocationFieldInfo
Config Property: `startupInit.Device.FixedLocationFieldInfo=true`
These fields are used for special applications where the 'telematic' device does not have a GPS receiver and where the asset being monitored does not move (ie. Such as when tracking water tank levels in stationary tanks, etc).

```
fixedLatitude
fixedLongitude
fixedAddress
fixedContactPhone
fixedServiceTime
```

BorderCrossingFieldInfo
Config Property: `startupInit.Device.BorderCrossingFieldInfo=true`
These fields are used for storing state-line border crossing information. (requires a module that can detect stateline border crossings - currently not used in the open-source OpenGTS system):

```
borderCrossing
lastBorderCrossTime
```

GeoCorridorFieldInfo
Config Property: `startupInit.Device.GeoCorridorFieldInfo=true`
These fields are used for storing the active GeoCorridor information (currently not used in the open-source OpenGTS system).

```
activeCorridor
```

MaintOdometerFieldInfo
Config Property: `startupInit.Device.MaintOdometerFieldInfo=true`
These fields are used for tracking periodic maintenance information (currently not used in the open-source OpenGTS system).

```
maintIntervalKM0
maintOdometerKM0
maintIntervalKM1
maintOdometerKM1
maintIntervalHR0
maintEngHoursHR0
maintNotes
```

WorkOrderInfo
Config Property: `startupInit.Device.WorkOrderInfo=true`
These fields are used for storing WorkOrder information and miscellaneous custom field information (currently not used in the open-source OpenGTS system).

```
workOrderID
customAttributes
```

B.3) Optional User Table Columns:

AddressFieldInfo
Config Property: `startupInit.User.AddressFieldInfo=true`
These fields are used to store additional Address information, as specified by application requirements.

```
addressLine1
addressLine2
addressLine3
addressCity
addressState
addressPostalCode
addressCountry
```

B.4) Optional EventData Table Columns:

AddressFieldInfo
Config Property: `startupInit.EventData.AddressFieldInfo=true`
These fields are used to store additional reverse-geocoded Address information, as specified by application requirements. In order for these columns to be filled in, the active reverse-geocode provider must be able to support these field:

```
streetAddress
city
stateProvince
postalCode
subdivision
speedLimitKPH
isTollRoad
```

GPSFieldInfo
Config Property: `startupInit.EventData.GPSFieldInfo=true`
These fields are used to store additional GPS and modem attributes, such as accuracy, dilution of precision, signal strength, etc.

```
gpsFixType
horzAccuracy
vertAccuracy
HDOP
satelliteCount
batteryLevel
batteryVolts
signalStrength
```

CustomFieldInfo
Config Property: `startupInit.EventData.CustomFieldInfo=true`
These fields are used to store custom miscellaneous event data fields, such as driver-id, generic analog values, etc.

```
entityID
driverID
driverStatus
driverMessage
emailRecipient
sensorLow
sensorHigh
costCenter
jobNumber
rfidTag
attachType
attachData
```

GarminFieldInfo
Config Property: `startupInit.EventData.GarminFieldInfo=true`
These fields are used tocollect ETA/Stop data from a Garmin Personal Navigation Device (PND) (a device capabile of sending this information to the server is required). (currently not used in the open-source OpenGTS system):

```
etaTimestamp
etaUniqueID
etaDistanceKM
etaLatitude
etaLongitude
stopID
stopStatus
stopIndex
```

CANBUSFieldInfo (previously J1708FieldInfo, as of v2.3.2**)**
Config Property: `startupInit.EventData.CANBUSFieldInfo=true`
These fields are used to store engine diagnostic information received from the on-board engine diagnostic computer. The engine diagnostic information is usually obtained from the vehicle J1708, J1939, OBD-II, or CANBUS interface. This optional list includes fields such as "`engineRPM`", "`engineHours`", "`coolantLevel`", "`coolantTemp`", "`oilLevel`", "`oilTemp`", "`fuelLevel`", "`fuelTotal`", "`fuelIdle`", and many more (see the "`CANBUSFieldInfo`" section in the EventData.java source module for a complete list of supported fields).

```
fuelTotal
engineRpm
engineHours
engineLoad
engineTorque
idleHours
workHours
transOilTemp
coolantLevel
coolantTemp
intakeTemp
brakeGForce
acceleration
oilPressure
oilLevel
oilTemp
airPressure
airFilterPressure
turboPressure
ptoEngaged
ptoHours
throttlePos
brakePos
j1708Fault
faultCode
malfunctionLamp
fuelLevel
fuelIdle
fuelPTO
vBatteryVolts
fuelPressure
fuelUsage
fuelTemp
fuelEconomy
brakePressure
tirePressure
tireTemp
```

AtmosphereFieldInfo
Config Property: `startupInit.EventData.AtmosphereFieldInfo=true`
These fields are used to store atmospheric data, such as temperature and barometric pressure.

```
barometer
ambientTemp
cabinTemp
```

ThermoFieldInfo
Config Property: `startupInit.EventData.ThermoFieldInfo=true`
These fields are used to store temperature information typically received from temperature monitors placed in the cargo being transported.

```
thermoAverage0
thermoAverage1
thermoAverage2
thermoAverage3
```

AnalogFieldInfo
Config Property: `startupInit.EventData.AnalogFieldInfo=true`
These fields are used for storing generic analog information.

```
analoge0
analoge1
analoge2
analoge3
```

AutoIncrementIndex
Config Property: `startupInit.EventData.AutoIncrementIndex=true`
These fields are used for auto-indexing the EventData records. Unless absolutely required, these fields should remain disabled.

```
autoIndex
```

EndOfDaySummary
Config Property: `startupInit.EventData.EndOfDaySummary=true`
These fields are used to store the daily summary data which may be provided by the remote GPS tracking device. Most devices will not be able to provide this information (currently used only for Antx device communication server).

```
dayEngineStarts
dayIdleHours
dayFuelIdle
dayWorkHours
dayFuelWork
dayFuelPTO
dayDistanceKM
dayFuelTotal
```

ServingCellTowerData
Config Property: `startupInit.EventData.ServingCellTowerData=true`
These fields are used to store the serving cell-tower information. This may be useful for providing a cell-tower based approximate location (a service which can provide cell-tower latitude/longitude location is required).

```
cellTowerID
mobileCountryCode
mobileNetworkCode
cellTimingAdvance
locationAreaCode
cellServingInfo
cellLatitude
cellLongitude
cellAccuracy
```

NeighborCellTowerData
Config Property: `startupInit.EventData.NeighborCellTowerData=true`
These fields are used to store the neighboring cell-tower information. This may be useful for providing a cell-tower based approximate location (a service which can provide cell-tower latitude/longitude location is required).

```
cellNeighborInfo0
cellNeighborInfo1
cellNeighborInfo2
cellNeighborInfo3
cellNeighborInfo4
cellNeighborInfo5
```

WorkZoneGridData
Config Property: `startupInit.EventData.WorkZoneGridData=true`
These fields are used to WorkZone data sampling information. (not used in the open-source OpenGTS, and in most GTS Enterprise installations):

```
sampleIndex
sampleID
```

CreationTimeMillisecond
Config Property: `startupInit.EventData.CreationTimeMillisecond=true`
These fields are used to EventData record creation time with millisecond resolution. This also creates a "`dataPush`" field which can be used with the Device "`DataPushInfo`" optional fields.

```
dataPush
creationMillis
```