Basic GTS Enterprise Device Communication Server

Atrack Technologies, Inc.

AT1(E), AT3(E), AT5(i)

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

Manual Revision HIstory			
Rev	**Date**	**Changed**	**Author**
0.1.0	2011/02/21	Initial Release	MDF
0.1.1	2011/04/11	Added Temperature and CANBUS information	MDF
0.1.2	2011/05/12	Added default user-define status code mapping	MDF
0.1.3	2011/07/01	Added "customFIelds" property	MDF

Device Communication Server – Atrack Technologies

Contents:

1) Introduction

This manual describes how to configure and run the GTS Enterprise device communication server (DCS) for the **ATrack Technologies Inc,** hardware GPS tracking/telematic devices. This server supports the following devices:

- AT1(E)
- AT3(E)
- AT5(i)

The following features are supported for the **Atrack** DCS:

- Receive incoming data packets via UDP/TCP.
- ASCII/Binary mode
- Acknowledgements
- Estimated GPS-based Odometer.
- Simulated Geozone Arrival/Departure.
- Simulated GPIO input events
- 2 Temperature inputs
- Analog input
- Driver ID

2) Configuring the Server

The following section refers to the runtime configuration file for the **Atrack** device communication server, which can be found in the GTS Installation directory at "`dcservers/dcserver_atrack.xml`".

2.1) Changing the Server "Listen" Ports.

The ports on which the **ATrack** DCS listens for incoming data packets is specified on the "`ListenPorts`" tag:

```
<ListenPorts
    tcpPort="37525"
    udpPort="37525"
    />
```

If required, the "listen" port can be changed to fit the requirements of your runtime environment. The script programmed into the **ATrack** device will also need to be configured to transmit data to the same port as the server used to listen for incoming data packets.

The "listen" ports must be open through the firewall in order for the remote device to send data to the **ATrack** server.

If packet acknowledgment is required, any acknowledgments sent by the server back to the remote device must be sent from the same IP address to which the remote device sent it's data packet. If your server responds to more than one IP address, then the **ATrack** server listener must be bound to the same IP address/interface used by the remote tracking devices. This is set in the top-level "`dcservers.xml`" file, on the "`DCServerConfig`" tag, "`bindAddress`" attribute.

2.2) Setting the "Unique-ID" Prefix Characters.

The Unique-ID prefix characters can be set in the "`UniqueIDPrefix`" tag section:

```
<UniqueIDPrefix><![CDATA[
    atr_
    atrack_
    imei_
    *
    ]]></UniqueIDPrefix>
```

These prefix characters are used to 'prepend' to the IMEI number, or other custom ID, as reported by the device to look-up the owning Account/Device record for this device. For instance, if the **ATrack** ID number is "`123456789012345`", then the system will search for the owning Device using the following Unique-ID keys, in the order specified:

```
atrack_123456789012345
imei_123456789012345
123456789012345
```

Note that the '`*`' character by itself indicates that the system should look up the **ATrack** ID number without any prefixing characters.

To bind a **ATrack** device to a specified Account/Device record, set the "Unique ID:" field on the Device Admin page to the appropriate prefixed unique-id value. For example:

```
        Unique ID:  atrack_123456789012345
```

2.3) Setting the ATrack Properties

Properties which effect the behavior of the server are set in the "`Properties`" tag section. The following properties may be set:

`<Property key="heaaderPrefix">@P</Property>`
This is the header prefix characters that the parser will expect to find ad the beginning of each event packet. This value must match the prefix characters programmed into the device, otherwise packets may be rejected and data loss may occur.

`<Property key="customFields"></Property>`
This property should specify a list of custom field defintions which will be confgiured into the devices. Currently the following list of custom fields are supported:

- **`%SA`** – Number of Satellites
- **`%BV`** – Backup Battery Voltage
- **`%MV`** – Main Power Voltage

These extra/custom fields can be configured into the **ATrack** devices using the "**`AT$FORM`**" command (please see the **ATrack** device configuration/user manual for more information), and will then be transmitted on each event sent by the device. Multiple extra/custom fields may be specified, such as "**`%MV%BV`**". It is important that the extra/custom fields defined in this property **MATCHES EXACTLY** with the configuration of the device, otherwise the server will be unable to parse the incoming records properly and data loss may occur. The packet "prefix" can also be specified to apply certain custom-fields with specific "prefixed" packet types. For instance, to apply the custom field "**`%MV%BV`**" to the "**`@V`**" prefix, you can specify the following string in the property value: "**`@V:%MV%BV`**". You can also specify multiple Prefix:CustomField defintions by separating them with commas, as in the following example: "**`@V:%MV%BV,@S:%SA`**". A custom field definition without an explicit prefix will be used as the default custom field settings to for packets not otherwise matching other defined prefixes.

`<Property key="minimumSpeedKPH">3.0</Property>`
This is the minimum acceptable speed value (in km/h), below which the device will considered not moving, and the speed will be explicityly set to "`0.0`".

`<Property key="statusLocationInMotion">true</Property>`
If "`true`", the DCS will replace an event which otherwise is defined to be a general **`STATUS_LOCATION`** status code instead with a **`STATUS_MOTION_IN_MOTION`** status code, if the indicated speed of the vehicle is greater than zero.

`<Property key="minimumMovedMeters">0</Property>`
If the specified value is greater than '0', then subsequent received events will be omitted if they are within the specified number of meters to the previous event. Useful for eliminating multiple events at the same location, when the device continues to periodically report a location even if the device is stopped.

`<Property key="estimateOdometer">true</Property>`
If "`true`", the DCS will calculate the current event odometer based on the distance traveled since the last valid GPS location.

```
<Property key="simulateGeozones">true</Property>
```
If "`true`", the DCS will check for geozone arrivals/departures and insert the appropriate arrive/depart events.

```
<Property key="simulateDigitalInputs">0xFF</Property>
```
If specified, this mask value indicates which of the device's digital inputs should be checked for state changes, and if a state change is detected, an event with the appropriate digital input state change status code will be generated. The special value "`false`" is the same as entering a mask value of "`0`", which indicates that no digital input state changes should be detected.

```
<Property key="analog.1">1.0,0.0,fuelLevel</Property>
```
This property specify the conversion values for an analog sensor attached to the device. "`analog.1`" specifies the conversion factors for the first analog sensor. The format of the property value must be specified as follows:
```
    Gain,Offset,EventDataFieldName
```
Where the analog value (`0 to 10 volts`) is converted to the stored value using the following formula:
```
    StoredValue = (AnalogValue * Gain) + Offset;
```
This converted value will then be stored into the specified EventData table field. If blank, the analog value will not be stored in the EventData table.

2.4) Changing the Default Event Code to StatusCode Mapping.

The Event Code to StatusCode mapping is specified in the "`EventCodeMap`" and "`Code`" tag sections:

```
<EventCodeMap enabled="true">

   <!-- standard events -->
   <Code key="0"  >0xF040</Code> <!-- Get Current Loc   : STATUS_QUERY              -->
   <Code key="1"  >0xF020</Code> <!-- Download Log      : STATUS_LOCATION           -->
   <Code key="2"  >0xF020</Code> <!-- Tracking          : STATUS_LOCATION           -->
   <Code key="3"  >0xF020</Code> <!-- Post A Text Msg   : STATUS_LOCATION           -->
   <Code key="10" >0xF811</Code> <!-- Driver Auth       : STATUS_LOGIN              -->
   <Code key="11" >0xF020</Code> <!-- Voice Service     : STATUS_LOCATION           -->
   <Code key="12" >0xF020</Code> <!-- FOTA Completed    : STATUS_LOCATION           -->
   <Code key="13" >0xFD39</Code> <!-- GSM Jam Detected  : STATUS_MODEM_JAMMING      -->
   <Code key="14" >0xF941</Code> <!-- Impact Detected   : STATUS_IMPACT             -->

   <!-- user defined events -->
   <Code key="101">0xF020</Code> <!-- %EG : Engine      : STATUS_LOCATION           -->
   <Code key="102">0xF11A</Code> <!-- %SD : Speeding    : STATUS_MOTION_EXCESS_SPEED -->
   <Code key="103">0xF118</Code> <!-- %DL : Idle        : STATUS_MOTION_EXCESS_IDLE  -->
   <Code key="104">0xF871</Code> <!-- %TW : Towing      : STATUS_TOWING_START       -->
   <Code key="105">0xF11C</Code> <!-- %MT : Motion      : STATUS_MOTION_MOVING      -->
   <Code key="106">0xF941</Code> <!-- %IP : Impact      : STATUS_IMPACT             -->
   <Code key="107">0xFD10</Code> <!-- %PL : Power low   : STATUS_LOW_BATTERY        -->
   <Code key="108">0xFD13</Code> <!-- %PS : Power loss  : STATUS_POWER_FAILURE      -->
   <Code key="109">0xF020</Code> <!-- %SS : GPS         : STATUS_LOCATION           -->
   <Code key="110">0xF951</Code> <!-- %FU : Fuel Sensor : STATUS_FUEL_REFILL        -->
   <Code key="111">0xF113</Code> <!-- %SP : Stop        : STATUS_MOTION_STOP        -->
   <Code key="112">0xF123</Code> <!-- %HA : Harsh accel : STATUS_MOTION_ACCELERATION -->
   <Code key="113">0xF930</Code> <!-- %HB : Harsh brake : STATUS_EXCESS_BRAKING     -->
   <Code key="114">0xF937</Code> <!-- %HC : Harsh corner: STATUS_EXCESS_CORNERING   -->
   <Code key="115">0xFD25</Code> <!-- %JD : GSM Jamming : STATUS_GPS_JAMMING        -->
   <Code key="116">0xF922</Code> <!-- %RP : Engine rev  : STATUS_OBC_RPM_RANGE      -->
   <Code key="117">0xF420</Code> <!-- %IN0: Input 0     : STATUS_INPUT_ON_01        -->
   <Code key="118">0xF421</Code> <!-- %IN1: Input 1     : STATUS_INPUT_ON_02        -->
   <Code key="119">0xF422</Code> <!-- %IN2: Input 2     : STATUS_INPUT_ON_03        -->
   <Code key="120">0xF423</Code> <!-- %IN3: Input 3     : STATUS_INPUT_ON_04        -->
   <Code key="121">0xF600</Code> <!-- %AN0: Analog 0    : STATUS_SENSOR32_0         -->
   <Code key="122">0xF601</Code> <!-- %AN1: Analog 1    : STATUS_SENSOR32_1         -->
   <Code key="123">0xF710</Code> <!-- %TP0: Temp 0      : STATUS_TEMPERATURE_0      -->
   <Code key="124">0xF711</Code> <!-- %TP1: Temp 1      : STATUS_TEMPERATURE_1      -->
   <Code key="125">0xF250</Code> <!-- %GF0: Geofence 0  : STATUS_GEOFENCE_VIOLATION  -->
   <Code key="126">0xF250</Code> <!-- %GF1: Geofence 1  : STATUS_GEOFENCE_VIOLATION  -->
   <Code key="127">0xF250</Code> <!-- %GF2: Geofence 2  : STATUS_GEOFENCE_VIOLATION  -->
   <Code key="128">0xF250</Code> <!-- %GF3: Geofence 3  : STATUS_GEOFENCE_VIOLATION  -->
   <Code key="129">0xF250</Code> <!-- %GF4: Geofence 4  : STATUS_GEOFENCE_VIOLATION  -->
   <Code key="130">0xF250</Code> <!-- %GF5: Geofence 5  : STATUS_GEOFENCE_VIOLATION  -->

</EventCodeMap>
```

The "`key`" attribute represents the ***default*** status code generated by the **ATrack** server for the indicated Alarm Code. The hex value indicated within the commented section following the "`Code`" definition is the value of the actual received Alarm Code.

The text value of the "`Code`" tag should be the status-code to which the Alarm Code should be mapped. The special value "`ignore`" can be used to cause events which specify this Alarm Code to be ignored. The special value "`default`" indicates that the status code on the generated event will be **`STATUS_LOCATION`** if the vehicle is not moving, and **`STATUS_MOTION_IN_MOTION`** if the vehicle is moving. The numeric values, specified as either decimal or hexidecimal will be used as the status code on the generated event.

If an event arrives with a Event Code which is not specified in the "`EventCodeMap`" tag section, then it will be logically "OR'ed" with the hex value `0x1000` and used as the status-code for the generated event.

Refer to the "Status Codes and Description" documentation for a list of currently defined status codes.

2.5) Supporting Temperature Sensors.

The **ATrack** DCS supports the 2 temperature sensor inputs. To enable the EventData table to capture this information in the table, the EventData "`thermoAverage#`" optional fields will need to be enabled. These fields can be enabled in the "`config.conf`" file by uncommenting and setting the following property specification to "`true`":

```
    Domain.Reports.EventThermo=true
```

Then update the EventData table columns:

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

This will add the temperature fields to the EventData table and will allow the temperature sensor information available on the data packets to be stored in the table records. The Temperature Monitor report can then be used to display this temperature information (see "`EventThermo`" report in "`reports.xml`" to configure additional temperature report columns).

2.6) Supporting CANBUS/OBDII/J1939 Data.

The **Atrack** DCS support OBDII and J1939 data fields, including Fuel Level, Engine RPM, Engine Load, Coolant Temerpature, etc. In order to have this information stored into the EventData table, the EventData table needs to have the CANBUS data columns enabled. To enable the CANBUS data columns, modify the "`config.conf`" file and uncomment and set the following property specifications to "`true`" (if property specification below does not already exist in the file, it can be added):

```
    startupInit.EventData.CANBUSFieldInfo=true
    startupInit.EventData.J1708FieldInfo=true
```

Then updating the EventData table columns:

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

This will add the various CANBUS fields to the EventData table.

3) Running the Server

To begin listening for incoming events the server must be started. This section describes the process for manually starting the **ATrack** server, and how to set up the system to have is automatically start the **ATrack** server on system reboot.

3.1) Manually Starting the Server

The command for manually starting the **ATRack** server is as follows:

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

To start the **ATrack** server with debug logging (useful when testing or debugging), the option "`-debug`" may be added to the command line.

The server will start, and logging information will be sent to the file "`$GTS_HOME/logs/atrack.log`".

For debug purposes, it is sometimes useful to have the logging output sent directly to the console, instead of the log file. In this case the option "`-i`" can also be added to the command-line. When logging to the console, hit control-C to stop the server.

To view the running server, you can use the "`bin/psjava`" command:

```
>  $GTS_HOME/bin/psjava

  PID  Parent  L User     Java class/jar
------ ------  - -------- ---------------------------------------------------
 54639(     1) 1 opengts  org.apache.catalina.startup.Bootstrap
 68936(     1) 1 opengts  /usr/local/GTS_2.3.0-B23/build/lib/atrack.jar
```

To stop the running **ATrack** server, enter the following command:

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

This will stop the running **ATrack** server.

3.2) Automatically Starting the Server on System Reboot

The auto-start script for Fedora is located at "`$GTS_HOME/bin/onboot/fedora/opengts`", and should have been installed into the system directory "`/etc/init.d`" when the GTS was installed.

This startup script uses the configuration specified in the file "`$GTS_HOME/bin/serverList`" to determine which device communication servers to start up when the system is rebooted. The line in this file that refers to the **ATrack** DCS should appear similar to the following:

```
execServer  "Atrack Technologies"  "Atrack"  "${option}"  ""
```

The first quoted string contains the DCS description. The second quoted string contains the ID of the device communication server and must match a library jar file with the same name at "`$GTS_HOME/build/lib/atrack.jar`". The third quoted string must contain the exact value "`${option}`", which is used within the startup script to pass command-line arguments to the DCS startup code. The forth quoted string is used to pass other optional default or constant arguments to the DCS startup code.

3.3) Monitoring the Log Files

When started, the **ATrack** DCS will create the following output log files:

`$GTS_HOME/logs/atrack.pid`
This file contains the process-id (PID) of the **ATrack** DCS execution process. This PID is used by the "`-kill`" option to terminate the running **ATrack** DCS.

`$GTS_HOME/logs/atrack.log`
This log file is where all other logging information is placed regarding the receipt and parsing of data from the remote **ATrack** tracking devices. As this file grows, it will be "rotated" into other log files timestamped as follows:

```
   atrack.log.YYYYMMDDHHMMSS.log
```

Where "*`YYYYMMDDHHMMSS`*" represents the Year/Month/Day/Hour/Minutes/Seconds time that the file was trimmed and rotated.

The "`atrack.log`" file will reflect any current connection attempts from remote **ATrack** tracking devices. As devices send their data to the server, the receipt of the incoming data packets, along with parsing results, will be displayed in this log file.

4) Adding Devices to an Account

When data is received from a remote **ATrack** tracking device, the **ATrack** server looks up the IMEI number in the Device table to determine which Account/Device owns this device. This section describes how to create a Device record and associate an **ATrack** tracking device with the Device record.

4.1) Creating a New Device Record

Using the web-interface, log in to the appropriate Account which should own the **ATrack** tracking device, then traverse to the "Device Admin" page (or "Vehicle Admin", etc, if so named). Create a new Device as indicated in the GTS Enterprise Tutorial documentation, then "Edit" the newly created Device record.

On the Edit page, there will be a field described as follows:

Unique ID: [   ]

In this field enter the value "**`atrack_`***`<IMEI_Number>`*", replacing "*`<IMEI_Number>`*" with the device IMEI number. For instance, if the IMEI number is "`123456789012345`", then enter the value "`atrack_123456789012345`" in this "Unique ID:" field.

After making changes to the Device record, click the "Change" button.

4.2) The "Server ID" Field

The "Server ID" field displayed as a column title on the Device list page, and as a read-only field on the Device Edit page, is assigned a value when the **ATrack** device sends its first data packet to the server. Until then, this value will remain blank.

When viewing a list of created Device records with assigned **ATrack** devices, records which still have blank "Server ID" fields indicate that no incoming data packet has been received for this particular Device.

5) Testing a New Configured Device

This section describes the process for monitoring newly configured **ATrack** devices that have been assigned to an Account/Device record.

5.1) Monitoring for Incoming Connections

The Account report "Last Known Device Location" can be used to display the last know location of a given device, which can also be used to determine whether any events have been received from a specific **ATrack** device.

The "Server ID" field on the Device record will also indicate if a data packet has arrived from a specific **ATrack** device and successfully assigned to the Device record.

If no indication on the Device reports, or "Server ID" field is evident, then the log file itself can be consulted for indications of incoming data packets from the **ATrack** device. The information in the log file can indicate whether an IMEI number may not have been properly assigned, so the **ATrack** DCS is unable to determine which Account/Device the incoming data packet belongs to.

5.2) Viewing the Unassigned Device Report

In the case where an **ATrack** device is put into service without having been assigned to an Account/Device record, or where the IMEI number was incorrectly entered in to the Device record, the **ATrack** DCS may not know to which Account/Device the incoming data packet belongs.

When the **ATrack** DCS cannot determine the ownership of an incoming data packet, it will place the IMEI and currently GPS location into the "UnassignedDevices" table. The "Unassigned Devices" report can be selected from the System Administrator login panel ("System Admin" tab, "SysAdmin Reports" menu option, "Unassigned Devices" report). This report will show the "Server ID" (**ATrack**), "Unique ID" (IMEI number), and the last time data was received from this device. This information can be used to determine whether an IMEI number was ever assigned to an Account/Device record, or if an IMEI number was incorrection assigned to an Account/Device (ie. transposed digits, etc).

Appendix)

A) Troubleshooting Device Connection Issues

The following are fequently-asked-questions regarding commonly occurring connection issues.

Q: I've configured an **ATrack** device to send data to the server, but have not received any data.
A: Monitor the "`atrack.log`" file for possible incoming connections from the device. If there is no indication that the server is receiving any communication from the remote device, the most common reasons to check include:

- Make sure device has a valid/active SIM card.
- Make sure the device has been programmed with the proper APN ("Access Point Name") configuration as specified by your wireless service provider.
- Make sure the device has been programmed with the proper host and port of your server.
- Make sure the server firewall allows incoming UDP/TCP connections on the specified port. If the server itself provides its own firewall, then check the firewall settings. On Linux, this is usually controlled by "`iptables`". The command to display the current iptables configuration is "`iptables-save`" (must be run as "`root`"). See "http://www.faqs.org/docs/iptables/iptables-save.html" for more information.

Q: I see data arriving for my device in the "`atrack.log`" file, but it is always the same event that is being sent over and over.
A: If this occurs for all configured/connected **ATrack** devices, the problem is likely that returned UDP acknowledgments are not being returned properly to the device. The most likely reason for this is that your computer responds to more than one IP address, and the returned UDP packets are being sent from a different IP address than the one that the device is configured to send data to. This can be fixed by setting the "`bindAddress`" attribute in the "`dcservers.xml`" file in the GTS installation directory (then restart the **ATrack** DCS). In some cases, the SIM card wireless service provider does not allow returned UDP packets to be sent from the server back to a device. In this case, it may be necessary to program the **ATrack** devices to not require a return acknowledgment.
A: If this occurs for only one device (ie. other devices are reporting as expected), this this is likely due to the GPS receiver's inability to obtain a new GPS fix, and the previous GPS fix is being resent. This usually means that the device is simply in an area where the GPS satellites cannot be seen (ie. Indoors, etc). On rare occasions, this can mean that the GPS antenna has become unplugged, or has been damaged.

Q: The received events have a valid latitude/longitude, but do not have an odometer value.
A: The **ATrack** DCS property "`estimateOdometer`" allows enabling a calculated odometer value, based on the distance traveled between successive GPS points. To enable a calculated estimated odometer value, make sure this property is set to "`true`".