

TomTom®  WORK

Connected Navigation

WEBFLEET.connect - Job dispatch



Always connected to your people on the road.

WEBFLEET.connect - Job dispatch

TomTom Business Solutions

Doc.WEBFLEET.connect-guide-orders

1.0

Published 2010-09-24

Copyright © 2010 TomTom Business Solutions, TomTom International B.V.

No part may be reproduced except as authorized by written permission. The copyright and the foregoing restriction extend to reproduction in all media.

All rights reserved.

Revision History

Revision	Date	Description	Author
1.0	2010-09-24	Initial release	IM

Table of Contents

Job dispatch in WEBFLEET.connect	4
1 Introduction	4
2 Sending jobs	4
3 Retrieving job status	6
4 Adding a destination	7
5 Going further	7

Job dispatch in WEBFLEET.connect

1 Introduction

Job dispatch support in WEBFLEET.connect allows you to send jobs to TomTom navigation devices that are connected to WEBFLEET. It enables you to manage them once send, receive status updates and check current workflow status. This article introduces job dispatch to those who are already familiar with jobs dispatch and address management in WEBFLEET. The article shows you how to manage jobs through the WEBFLEET.connect API.

Basic concepts

Job	A task that needs to be carried out, usually at a certain location. In order to complete a job, a logical sequence of actions, sometimes called workflow is necessary. In WEBFLEET jobs are named orders. Jobs can only be sent to Objects. Information about actions carried out by a user to advance a job is sent back to WEBFLEET as status updates.
Object	An object is the logical representation of the hardware equipment that directly communicates with WEBFLEET. Depending on the actual deployment, there can either be a fixed relationship with a vehicle (fixed mounting) or a dynamic relationship if the equipment is swapped between vehicles, e.g. when it is assigned to a specific person. An individual object is identified by an object number that is unique within a WEBFLEET account.
Address	An address is an entity within WEBFLEET that represents a certain point of interest - customers, driving destinations, etc. Typically it is described by street-level attributes (country, city, street name, house number, etc.). Addresses need to be geocoded in order to be useful - this either happens automatically during input or by the software that uploads addresses to WEBFLEET. An individual address is identified by an address number that is unique within an account.
Location	A location is the place were a job needs to be carried out. Therefore it is used as the driving destination. It can either refer to an address that exists in WEBFLEET (through an address number) or by explicitly stating geocoordinates and street-level attributes.
Message	Messages are sent from the object to WEBFLEET whenever there is new information available. This includes job status updates, changes to the estimated time of arrival, (pre-defined) text messages, etc. If available, each message contains the object's current geolocation (at the time of message generation).

Setting up your enviroment

WEBFLEET.connect is provided as a web service and can be accessed via plain HTTP requests or via SOAP. As long as your programming environment supports one of these access types, you don't need any additional tools. You might use a web browser to try out requests when using plain HTTP requests or to fetch WSDL files when using SOAP. In order to quickly try out SOAP requests, you can use SoapUI.

2 Sending jobs

In most cases, jobs are taken in and managed in a dedicated application, often an ERP, a CRM or any other application that is used to track job execution. How jobs are stored within those application can be quite complex, but let us start with a simple job table just holding an unique job id, a description and a flag that indicates whether the job has already been sent to WEBFLEET. Each job needs to be assigned to a specific object:

```
CREATE TABLE jobs
(
  jobid serial NOT NULL,
  jobtext character varying,
  jobsent boolean DEFAULT false,
  ttobject character varying(10),
  CONSTRAINT pk_jobs PRIMARY KEY (jobid)
);
```

After populating this table we have some jobs scheduled and ready to be sent to WEBFLEET:

```
1;"Do something.";FALSE;"001"
2;"Do something else.";FALSE;"001"
```

The next step is to create a program that transmits all outstanding jobs to WEBFLEET and keeps track of jobs already sent. In Perl, this could look like:

```
#!/usr/bin/perl -Ilib -w

use strict;
use DBI;
use WEBFLEET::Connect;

my $dbh = DBI->connect(...);
my $wfc = new WEBFLEET::Connect(
  account=>'...',
  username=>'...',
  password=>'...'
);

# fetch all pending jobs from database
my $sth = $dbh->prepare('select * from jobs where jobsent = false');
$sth->execute;
my @jobs;
while (my $h = $sth->fetchrow_hashref) {
  push @jobs, {%$h};
}

# send pending jobs and update state in database
$sth = $dbh->prepare('update jobs set jobsent=true where jobid = ?');
foreach my $h (@jobs) {
  my $r = $wfc->sendOrder(
    objectno=>$h->{ttobject},
    orderno=>$h->{jobid},
    ordertext=>$h->{jobtext}
  );
  $sth->execute($h->{jobid}) if ($r->is_success);
}
```

We are now able to send outstanding jobs. Optionally, this program could be scheduled to run periodically and send any new jobs that appear in the jobs database table. The easiest way on platforms that have **cron** is to add it to the crontab:

```
# send new jobs every 5 minutes on business days
*/5 * * * 1-5 ~/bin/sendjobs.pl
```

3 Retrieving job status

After a job has been sent out and received by the object and read by the driver the job will eventually be completed. It might be worth getting this information back as it helps staying informed about the completion status of jobs that have been sent out. There are two ways of retrieving the current status for a job:

- Retrieve the most recent status for each job as WEBFLEET keeps track of this.
- Create message queue and receive job status updates as they come in from the objects.

Tracking job completion

Information about job completion needs to be stored somewhere. For now and to keep it simple, this is stored directly in the jobs table. There are two new columns tracking the completion status and the time of completion.

```
ALTER TABLE jobs ADD COLUMN ttcompleted boolean;
ALTER TABLE jobs ALTER COLUMN ttcompleted SET DEFAULT false;
ALTER TABLE jobs ADD COLUMN ttcompletiontime timestamp with time zone;
```

It is now just a matter of extending the existing program with a call to `showOrderReport(Extern)`:

```
my $r = $wfc->showOrderReport(
    range_pattern=>'d0',
    useISO8601=>'true'
);
if ($r->is_success) {
    $sth = $dbh->prepare('update jobs set ttcompleted=true, ttcompletiontime=?
where jobid=?');
    foreach my $i (@{$r->content_arrayref}) {
        $sth->execute($i->{orderstate_time}, $i->{orderid}) if ($i->{orderstate}
eq '401');
    }
}
```

After one job has been completed, the jobs table might look like this:

```
1;"Do something.;"TRUE;"001";TRUE;"2010-09-24 10:06:25+02"
```

```
2;"Do something else. ";TRUE;"001";;" "
```

As an extension, you might not only want to track successful completion of a job, but jobs that get cancelled or rejected, too. This is just a matter of changing the column *ttcompleted* from boolean to an enumeration and checking for those states (301, 302).

4 Adding a destination

By now we have the basic machinery in place to send jobs and keep track of their completion status. But so far a driver could just accept the job, work his way through the workflow and finally complete it. What has been missing is a destination information that can be used to navigate to the location where the job needs to be carried out.

There are two ways of providing location information: refer to an existing address or attach all location details directly to the job. Let us assume that addresses are replicated to WEBFLEET in order to seem them on the map, then we just need to link those addresses to the jobs scheduled for each address. This means another database change:

```
ALTER TABLE jobs ADD COLUMN ttaddress character varying(10);
```

After adding some more jobs, the job table is now populated with jobs that have address attached to them.

```
1;"Do something. ";TRUE;"001";TRUE;"2010-09-24 10:06:25+02";" "  
2;"Do something else. ";TRUE;"001";;" ";"  
3;"Do something at a location. ";FALSE;"001";;" "; "a001"  
4;"Do something else at another location. ";FALSE;"001";;" "; "a002"
```

Of course, the program needs to be changed to transmit the location together with the job by using a different API method. The call to `sendOrder(Extern)` is replaced by a call to `sendDestinationOrder(Extern)`.

```
# ...  
my $r = $wfc->sendDestinationOrder(  
    objectno=>$h->{ttobject},  
    orderno=>$h->{jobid},  
    ordertext=>$h->{jobtext},  
    addrnr=>$h->{ttaddress}  
);  
# ...
```

5 Going further

There are many ways to move job dispatch further. Things that are missing to make that example a usable solution is address replication to WEBFLEET including geocoding. Error handling has not been considered and for big data volumes the access and update patterns for jobs certainly need to be optimised.

There are a lot of opportunities and use cases that haven't been covered:

- Keep a log of all incoming job status updates in a job history by using the message queues to retrieve all update messages.
- The API has methods to update or cancel orders once they have been sent out. Due to the complexity and the cooperation required from the job management application this has not been touched at all.
- Once an audit trail of all status updates is available, reports can be created that help identifying where time spent waiting can be reduced or planned arrival times have not been met and many more.