Trama Client-Server Commands   (Ver. 1396/09/24)

Contents:

1- List of Command Codes and Error Codes

1-1: List of Error_Codes

Code	Name	Description
0	No Error	
-1	Authentication Error	
-2	Device not Exists Error	
-3	Device Inactive Error	
-4	Bad Input Format Error	
-5	No Relating Data Exists	
-6	FCM_Token_Not_Registered	
-7	Device_Not_Alive_In_The_Network	
-8	Device_Not_Registered	
-9	Duplicate_Data	
-10	Record_Has_Been_Saved_Before	
-11	Payment_Not_Accomplished	
-100	General Error	

1-2: List of Commands

1-3: List of Command_Codes for FCM Server_2_Client Command (that need not to be acknowledged by clients)

Command Code	Name
200	FCM_Driver_Tracking_On_Driver_Location_Change
201	FCM_Driver_Travel_Reach_Point
202	FCM_Check_Server_4_Commands

2- Calculation of the password by using the Auth_Key

The password needed as one of the input parameters of the web services is calculated based on the Auth_Key obtained through the Login web-service (In the web-services *ping* and *register*, the input parameter may be something other than *Auth_Key*. Refer to such commands). Password should be calculated as follows:

```
public string Calculate_Password(string Auth_Key, string Private_Auth_Key)
{
    string Password="";
    string Key1= Private_Auth_Key;
    var Key2 = Auth_Key;
    var Key2_Length = Key2.Length;

    for (int i = Key1.Length-1; i >=0 ; i--)
    {
        Password= Password + (((int)Key1[i] -20) * ((int)Key2[i%Key2_Length] -10)).ToString();
    }
    return Password.ToString();
}
```

3- List of Web Services

Web-services are in RESTFul json protocol format which can be called through http GET and POST requests. The response for each of the web-services is a json record in the following format:

```
Status,
Error {
    Code,
    Message,
    Error_Data
},
Result_Data
```

Description:

Status is either true or false
For each web-service, if the *Status* is false, the *Result_Data* will be null and the error code and error message is returned back. For some error codes like *Payment_Not_Accomplished* for the *Login* api, relating *Error_Data* json object may also exists. For the case where *Status* is true, the returned *Result_Data* will have a specific format for each of the web-services.

For each web-service input parameters are contained json format record inside the body or request.

---

1. base_url/api/Tracking/Ping

(Request_Method=**POST**)

Input Parameters: `Password, Device_Code, Simcard_No, Device_Type ,Show_Device_Info, Reset_Private_Auth_Key(boolean)`

Result_Data

```
{
        Device_Code,
        Device_Registered(boolean),
        Register_Date(yyyyMMdd),
        Register_Time(hhMMdd),
        App_Nonforce_Version,
        App_Force_Version,
        Register_Type
}
```

Description: This method may be called prior to *Register* command or any other time to check connection to server and/or prepare to register the device. If *Show_Device_Info* is set to *true*, the *Result_Data* is returned back, otherwise *Result_Data* will be null. The device may set *Show_Device_Info=true* before the first register to get related *Result_Data* and *Show_Device_Info=false* in other cases for simple ping.

Note: If both Device_Code=-1 and *Simcard_No* is empty string, the result status is always true and no query is executed in the server to process *Device_Code*, *Device_Type, Device_Registered*, *Register_Date*, *Register_Time*. In this case Password is also ignored and thus this may only be used to see if the network connection to server is alive.

Registeration: If *Reset_Private_Auth_Key* is set to *true*, then a new *Private_Auth_Key* is assigned to the device and the device should register in order to communication with the server.
Two types of registrations may be used by the device depending on the *Register_Type* value.

a) If *Register_Type*=0, this means that the server sends an SMS containing the *Private_Auth_Key* and the device should use this Key in the *Register_Device* web-service in order to register.
b) If *Register_Type*=1, this means that the device should call *Client_2_Server_Register_Process* web-service in order to register. (refer to *Client_2_Server_Register_Process*).

Password Calculation: If *Device_Code*=-1 and *Simcard_No* is provided (i.e., not empty string), then for Calculate_Password function *Auth_Key*=**23415** and *Private_Auth_Key*=*Simcard_No*.
i.e.: Calculate_Password(”23415”, *Simcard_No*)

If *Device_Code*>0 and *Simcard_No* is empty, same as all other web-services (except *Register_Device*), the general *Auth_Key* and *Private_Auth_Key* should be used for calculating password, where *Auth_Key* is obtained through *Login* web-service. If *Device_Code=-1* and *Simcard_No* is empty, then *Password* is ignored.

2. base_url/api/Tracking/Client_2_Server_Register_Process

(Request_Method=**POST**)

Input Parameters: Password, Simcard_No, Device_Type(0 for driver, 1 for passenger), Mode, UDID, Device_Model_No, Android_Version

Result_Data

```
{
        Device_Code,
        SMS_Center_No,
        Interval_2_Call_Back(in seconds),
        Initial_Key(ignored in current version),
        Private_Auth_Key
}
```

Description: If *Register_Type*=1 (obtained in the response of *Ping* command), this means that the device should call *Client_2_Server_Register_Process* web-service in order to register. First, This web-service must be called by setting *Mode=0* in order to start registration process (In the response, the *Device_Code* corresponding to the given *Simcard_No* and alse *SMS_Center_No* is returned back). After that, the device should send (or ask the device user to send) an SMS to the *SMS_Center_No*, and check for getting *Private_Auth_Key* by calling *Client_2_Server_Register_Process ({Mode=1})* web-service after *Interval_2_Call_Back* seconds again. As soon as the *Private_Auth_Key* is captured (a positive returned number) or the returned parameter *Interval_2_Call_Back* is a negative number, the device must stop calling *Client_2_Server_Register_Process.* As soon as the *Private_Auth_Key* is captured, the device must call *Register_Device* in order to register the device.

Note that the valid *Device_Code* is only returned back when *Mode*=0, otherwise (*Mode*=1), the returned *Device_Code* is equal to -1.

Password:
Calculate_Password(”23415” , *Simcard_No*)

3. base_url/api/Tracking/Register_Device

(Request_Method=**POST**)

Input Parameters: `Password, Device_Code,` Private_Auth_Key, UDID, Device_Model_No, Android_Version

Description: This command should be called before the first time *login* command calling, in order to register the device. Prior to calling this command, the device should call *ping* web-service with *Reset_Private_Auth_Key=true* in order to get the *Private_Auth_Key* through SMS. Note that for this command, in order to calculate the input parameter *Password* through *Calculate_Password* method, the *Private_Auth_Key* together with public *Auth_Key*=**23415** must be used in the *Calculate_Password* function, i.e.: Calculate_Password(*Private_Auth_Key* ,"12345")

---

4. base_url/api/Tracking/Login

(Request_Method=**POST**)

Input Parameters: `Password, Device_Code`

Result_Data

```
{
    Device_Code,
    Device_Type(0:Driver, 1:Passenger),
    Auth_Key,
    Server_Latin_Date (yyyyMMdd),
    Server_Time(hhmmss),
    App_Nonforce_Version,
    App_Force_Version,
}
```

***Error_Data* { (for *Error_Code= Payment_Not_Accomplished*)**

```
    Payment_Fee,
    Redirect_Url_4_Payment
}
```

Description: If login succeeds, the corresponding *result_data* is returned back. The *Auth_Key* is needed for the device application in order to calculate the Password for other web-services. Note that since the Auth_Key may be unknown to the application prior to calling *Login* web-service, in order to calculate the input parameter *Password through Calculate_Password* method, *Private_Auth_Key* together with *Auth_Key*=**23415** must be used. The *Auth_Key* is changed automatically every day. At calling each web-service, if *Authentication_Error* is returned back, the client should call the *login* web-service again. If *Error_Code= Payment_Not_Accomplished,* then the Application must redirect user to the web page *Redirect_Url_4_Payment* and close the app, so that the user accomplishes the electronic payment. If the payment is successfully accomplished, the next *Login* will return a success result.

The android application must also check the output parameter *Device_Application_Version* and compare it to its Version Number. The android application should download the new version if its version is below *Device_Application_Version*)

---

5. base_url/api/tracking/Insert_Tracking_Records_In_Db

(Request_Method=**POST**)

Input Parameters: `Password, Device_Code, Records_Packet, New_Version(must be set to true)`

Records_Packet:

Record 1:	Direction,Shamsi_Date(yyyyMMdd),Time(hhmmss),Lat,Lon,Speed,Driver_Code,Backup_Driver_Code,Position_Capture_Type,Service_Code,Tracking_TimeTable_Item# Direction,Shamsi_Date(yyyyMMdd),Time(hhmmss),Lat,Lon,Speed,Driver_Code,Backup_Driver_Code,Position_Capture_Type,Service_Code,Tracking_TimeTable_Item#
	…
	…
Record n:	Direction,Shamsi_Date(yyyyMMdd),Time(hhmmss),Lat,Lon,Speed,Driver_Code,Backup_Driver_Code,Position_Capture_Type,Service_Code,Tracking_TimeTable_Item

Note:

a) Maximum value for number of records (*n*) is 30.

b) For each record, if driver is operating in default mode (main driver), *Backup_Driver_Code* could be either a null string or same as the *Driver_Code*; However if the driver is operating as the backup of some other driver, *Driver_Code* and *Backup_Driver_Code* should both be specified.

c) *Position_Capture_Type* is **0** for *GPS* capturing or **1** for *Network* capturing.

d) Speed is Km/Hour

e) Direction is **-1** (if no direction is distinguished), **0** (from passenger home to organization), or **1** (from organization to home)

---

6. base_url/api/tracking/Upload_Image

(Request_Method=**Post**)

Input Parameters: `Password, Device_Code, Device_Type, Device_Owner_Code, Image_File`

Note: `Image_File is contained in the` *`body form-data`* `and other input parameters must be placed in the url. The face image of the driver or passenger must have a size lower than 80 KB. Device_Type=0 for the driver and Device_Type=1 for the passenger.`

---

7. base_url/api/tracking/Message_2_Server

(Request_Method=**GET**)

Input Parameters: Password, Device_Code, Device_Type, Device_Owner_Code, Relating_Driver_Code, Organization_Code, Date(yyyyMMdd), Time(hhmmss), Message_Code, Param1, Param2

Message	Message_ Code	Param1	Param2
Message_2_Admin_Panel	0	Message Text	
Device_Location_Service_Is_Off	1		

Description:

Device_Type=0 if web-service called by driver device and *Device_Type*=1 if called from passenger device.
Device_Owner_Code is *Driver_Code* if web-service called by driver device and, *Passenger_Code* if web-service called by passenger device.
if *Device_Type*=1, *Relating_Driver_Code* can be set as the driver_code to whom the message relates or *Relating_Driver_Code* can be set equal to -1 if the message is a general one not relating to the driver

For passenger device, *Organization_Code* is the passengers assigned organization. For driver device, *Organization_Code* must be one the drivers assigned organization codes.

---

8. base_url/api/tracking/Set_Passenger_Address_Location_Point

(Request_Method=**GET**)

Input Parameters: Password, Device_Code, Passenger_Code, Driver_Codes(string comma-delimited), Lat, Lon

---

9. base_url/api/tracking/Set_Passenger_Alarm_Location_Point

(Request_Method=**GET**)

Input Parameters: Password, Device_Code, Passenger_Code, Driver_Code(string comma-delimited), Point_Code (1,2,3,4 or 5), Point_Descript, Lat, Lon

Note: This method is used to specify one of the five possible alarm locations near to the passenger's address location (If Lat=0 and Lon=0 the point will be deleted in the server). If the driver reaches to one of these locations, it must call the server through ***** command, so that the server informs the passenger through FCM push command.

---

10. base_url/api/tracking/Get_Driver_Device_Info

(Request_Method=**GET**)

Input Parameters: Device_Code

Result_Data

```
{
    Code,
    Name,
    Family,
    Mobile_No,
    Device_Simcard_No,
    Image_Url,
    Interval_2_Record_Tracking,
    Interval_2_Check_Server_For_Failed_FCM_Commands,
    I_am_Serving_Someone_Driver_Info{
        Active_Date_From;
        Active_Date_To;
        Driver_Info{
            Result_Data structure for Get_Driver_Device_Info …
        }
    }
    Someone_Serving_Me{
        Active_Date_From
        Active_Date_To
        Driver_Code
        Driver_Name
        Driver_Family
    }
    Services{
         Service_Code,
         Organization{
              Code,
              Descript,
              Address,
              Radius_From_Center,
              Lat,
              Lon
         }
         Tracking_Program
         {
              Code,
              Descript,
              Tracking_TimeTable_Items
              {
                     Code (Unique Code in Server Database),
                     Day_Code,
                     Direction (0 or 1),
                     Start_Time(hhmm),
                     Stop_Time(hhmm)
              }
         }
         Passengers
         {
              Code,
              Order_Code,
              Class(a number between 1 and 12 or -1 if unknown),
              Name,
              Family,
              Mobile_No,
              Image_Url,
              Address,
              Address_Lat,
              Address_Lon,
              Alarm_Distance_Driver_Getting_Close_2_Passenger(per meter),
                Alarm_Location_Points
                {
                        Point_Code(equal to 0, 1, 2 , 3, or 4),
                        Point_Descript,
                        Lat,
                        Lon
                }
            }
        }
    }
```

Note: *Interval_2_Record_Tracking* is the interval per second that device should record location points. *Interval_2_Check_Server_For_Failed_FCM_Commands* is the interval per minute that device should call *Commands_In_Server_Queue_For_The_Device* command for lost FCM commands. *Distance_4_Driver_Close_2_Passenger_Command* is the distance (per meter) that if the driver is within this distance to one of its passengers, it should notify the server through *Message_2_Server* command with *Message_Code*=0.

Distance_4_Driver_Reached_2_Passenger_Command is the distance (per meter) that if the driver is within this distance to one of its passengers, it should notify the server through *Message_2_Server* command *Message_Code*=1. *Organizations* are the schools associated with the driver. *Tracking_TimeTable_Items* is an array that specifies in which days and what times of each day, the device should save and upload location points to the server.

---

11. base_url/api/tracking/Get_Passenger_Device_Info

(Request_Method=**GET**)

Input Parameters: `Password, Device_Code`

Result_Data

```
{

        Code,
        Class(a number between 1 and 12 or -1 if unknown),
        Name,
        Family,
        Mobile_No,
        Device_Simcard_No,
        Image_Url,
        Address,
        Address_Lat,
        Address_Lon,
        Driver_Code,
        Driver_Name,
        Driver_Family,
        Driver_Mobile_No,
        Driver_Image_Url,
        Organization{
            Code,
            Descript,
            Address,
            Radius_From_Center,
            Lat,
            Lon
        },
        Alarm_Distance_Driver_Getting_Close_2_Passenger(per meter),
        Alarm_Location_Points{
            Point_Code,
            Point_Descript,
            Lat,
            Lon
        }
        Services{
                Service_Code,
                Driver_Code,
                Driver_Name,
                Driver_Family,
                Driver_Mobile_No,
                Driver_Image_Url,
                Tracking_Program
                {
                        Code,
                        Descript,
                        Tracking_TimeTable_Items
                        {
                                Code (Unique Code in Server Database),
                                Day_Code,
                                Direction (0 or 1),
                                Start_Time(hhmm),
                                Stop_Time(hhmm)
                        }
                }
         }
    }
```

Note: *Organization* is the school associated with the passenger.

12. base_url/api/tracking/Get_Driver_Backup_Device_Info

(Request_Method=**GET**)

Input Parameters: `Password, Device_Code`

Result_Data: Same format as that for *Get_Driver_Device_Info*

```
{
    Active_Date_From,
    Active_Date_To,
    Device_Code,
    Driver_Info
```

}

Description: This method gets the info of the device that is to be supported by this device driver for a specific period of time. At this period of time, the main device should ignore its main info and act as the backup device.

Backup_Driver_Info is an object with the same structure as the *Result_Data* for *Get_Driver_Device_Info* command.

---

13. base_url/api/tracking/Get_Text_Message

(Request_Method=**GET**)

Input Parameters: `Password,` Device_Code

Result_Data

```
{
      Messages
      {
            Code,
            Issue_Date,
            Issue_Time,
            Message_Text
      }
}
```

---

14. base_url/api/tracking/Get_Driver_Tracking_Points

(Request_Method=**GET**)

Input Parameters: Device_Code, Driver_Code,Date (yyyyMMdd),Time_From(hhmm),Time_To(hhmm)

Result_Data

```
{
    Points{
        Point_Code,
        Date,
        Time,
        Lat,
        Lon
    }
}
```

Description: By using this method, the passenger could get its driver locations for a given period of time

---

15. base_url/api/tracking/Get_Driver_Last_Location

(Request_Method=**GET**)

Input Parameters: `Password, Device_Code, Driver_Code`

Result_Data

```
{
    Point{
        Point_Code,
        Date,
        Time,
        Lat,
        Lon
    }
}
```

---

16. base_url/api/tracking/Start_FCM_Driver_Tracking

(Request_Method=**GET**)

Input Parameters: `Password, Device_Code, Driver_Code, Duration_Of_Online_Tracking_Per_Minute` (this is an integer between 1 and 30, in this interval, the server automatically sends tracking locations of the driver to the passenger)

Description: This command is called from passenger device to start online tracking of the driver. After calling this web-service, instantaneous location points of the driver is reported to the passenger's device through FCM message *FCM_Driver_Tracking_On_Driver_Location_Change* (as stated in the next section) upon any *Insert_Tracking_Records_In_Db* call from the corresponding driver device.

---

17. base_url/api/tracking/Stop_FCM_Driver_Tracking

(Request_Method=**GET**)

Input Parameters: `Password, Device_Code, Driver_Code`

---

18. base_url/api/tracking/Driver_Travel_Reach_Point

(Request_Method=**GET**)

Input Parameters: `Password, Device_Code, Driver_Code, Service_Code, Date(yyyyMMdd), Time(hhmmss), Location_Type_Code, Param1, Param2`

Location Type Descript	Location_Type_Code	Param1	Param2
Organization Address Coordinates	0	Organization_Code	
Passenger Address	1	Passenger_Code	

Coordinates			
Passenger Alarm Point Coordinates	2	Passenger_Code	Alarm_Point_Code(1,2,3,4,or 5)

---

19. base_url/api/tracking/Driver_Travel_Start

(Request_Method=**GET**)

Input Parameters: Password, Device_Code, Driver_Code, Organization_Code, Passenger_Code, Service_Code, Date(yyyyMMdd), Time(hhmmss)

Result_Data

```
{
        Travels
        {
                Passenger_Code,
                Travel_Code
        }
}
```

Description:

In two cases, the driver should call this method.

1- After the driver picked up some passenger, and when he starts leaving the passenger location zone (which means that the travel time for the corresponding passenger practically starts at this time). In this case, one member object *Travels* is returned through *Result_Data*. The *Organization_Code* is the Organization code corresponding to the passenger.
2- After picking up all passengers from the organization (school), when starting to leave the organization (school) zone to take passengers to their homes (registered location points). The driver device should call this method by setting *Passenger_Code* =-1 in this case. In this case, for each of the drivers passengers, a *Travel_Code* is assigned and thus an array of *Travels* object is returned through *Result_Data* for all of the driver's passengers

Note: *Travel_Code* is necessary to finalize each travel in *Driver_Travel_End* web-service.

---

20. base_url/api/tracking/Driver_Travel_End

(Request_Method=**POST**)

Input Parameters: Password, Device_Code, Travel_Info, Date(yyyyMMdd), Time(hhmmss)

```
Travel_Info{
        Travel_Code,
        Passenger_Code,
        Travel_Duration(hhmmss),
        Travel_Length(per Meters),
        Max_Speed,
        Average_Speed
}
```

Description: In two cases, the driver device calls this method.

1- Departing from the organization (school), after the driver reaches the address location zone of each of his passengers (in order to drop the passenger off), the driver device calls this method by specifying the related *Travel_Info* (Array with one member json object for the relating passenger)
2- Picking up all the passengers, when the driver reaches location zone of the organization (school), the driver device calls this method by specifying the *Travel_Info* (array of M json objects each for one of the passengers) in order to declare the end of the travel.

---

21. base_url/api/tracking/Get_Android_Application_Download_Url

(Request_Method=**GET**)

Input Parameters: Password, Device_Code

Result_Data

```
{
    App_Download_Url
}
```

Description: If success, the HttpResponse is automatically redirected to the url.

---

22. base_url/api/tracking/Get_Travels_Info

(Request_Method=**GET**)

Input Parameters: Password, Device_Code, Driver_Code, Organization_Code, Passenger_Code, Travel_Date(yyyyMMdd)

Result_Data:

```
{

    Travels
    {
        Travel_Code,
        Passenger_Code,
        Passenger_Name,
        Organization(نام مدرسه یا سازمان),
        Direction(رفت یا برگشت),
        Start_Date(yyyy/MM/dd),
        Start_Time(hh:mm:ss),
        End_Date(yyyy/MM/dd),
        End_Time(hh:mm:ss),
        Travel_Duration(hh:mm:ss),
        Travel_Length(بر حسب متر),
        Max_Speed(km/h),
        Average_Speed(km/h)
    }

}
```

Description: If Organization_Code=-1, all organizations all loaded. If Passenger_Code=-1 all passengers are loaded

---

23. base_url/api/tracking/Subscribe_FCM_Token_In_Server

(Request_Method=**GET**)

Input Parameters: `Password`, Device_Code, FCM_Token

---

24. base_url/api/tracking/Commands_In_Server_Queue_For_The_Device

(Request_Method=**GET**)

Input Parameters: `Password`, Device_Code

Result_Data

```
{
        FCM_Commands_4_Device
        {
                Command_Unique_Code(Unique Code In the server database needed for the ack response),
                Command_Code,
                Command_Data
        }
}
```

Note*:* The device should call the command *Commands_In_Server_Queue_For_The_Device* every *Interval_2_Check_Server_For_Failed_FCM_Per_Minute* minutes to catch a list (array) of server commands not caught by FCM.

Command Name	Command Code	Command_Data	Extra Description
Driver Device Info Must Be Updated	1	Result_Data for *Get_Driver_Device_Info* Api	
Passenger Device Info Must Be Updated	2	Result_Data for *Get_Passenger_Device_Info* Api	
Passenger will be absent	3	{Passenger_Code, Service_Code, Date, Direction}	Direction: 0: From passenger to organization (رفت) 1: From organization to passenger (برگشت) -1: Both directions (رفت و برگشت)
Personal Message From Panel 2 Driver	4	{Message}	
Personal Message From Panel 2 Passenger	5	{Message}	
Group Message From Panel 2 Driver	6	{Message}	
Group Message From Panel 2 Passenger	7	{Message}	

---

25. base_url/api/tracking/Set_Passenger_Absence

(Request_Method=**GET**)

Input Parameters: `Password`, Device_Code, Passenger_Code, Service_Code, `Driver_Code`, Date(yyyyMMdd shamsi), Direction

Note: Direction:
0: From passenger to organization (رفت)
1: From organization to passenger (برگشت)
-1: Both directions (رفت و برگشت)

26. base_url/api/tracking/Acknowledge_Command_Handled_By_Device

(Request_Method=**GET**)

Input Parameters: `Password`, Device_Code, `Command_Unique_Code`, `Command_Reference_Code`, `Command_Code`, `Ack_Only_This_Command`

Note: By calling this web-service, if *Ack_Only_This_Command*=true, then only the single command corresponding to the Command_Unique_Code is acknowledged, but for the case where *Ack_Only_This_Command*=false, all commands relating to the *Device_Code* whose command unique codes in the server are less than or equal to *Command_Unique_Code* will be acknowledged. Instead of *Command_Unique_Code* (which is a unique code the queue table), you can provide *Command_Reference_Code* for finding the record to Ack. Note that *Command_Reference_Code* is not unique in the commands queue table, but the combination of (*Device_Code*, *Command_Reference_Code*, *Command_Code*) is always unique. *(Command_Reference_Code, Command_Code)* can specially be used in FCM group commands where in a command with unique (*Command_Reference_Code, Command_Code)* (but different *Command_Unique_Codes*) is pushed to a group of devices (such as *FCM_Message_From_Panel*). The list of *Command_Codes* are given in the table in *Commands_In_Server_Queue_For_The_Device* web-service. If (*Command_Reference_Code, Command_Code)* is specified in the input parameters *Command_Unique_Code* must not be provided or must be set equal to -1.

27. base_url/api/tracking/SMS_Received_4_Device_Register

Input Parameters: `Mobile_No`, `Message`, `Date`, `Time`

---

28. base_url/api/Tracking/Get_All_Device_Info_4_Debugging

(Request_Method=**POST**)

Input Parameters: `Password`, `Password2`, `Simcard_No`, `Device_Type`

Result_Data

{

```
    Device_Code,
    Auth_Key,
    Private_Auth_Key
}
```

Description: This function is only used to get some device info to run in debugging mode for remote testing of a device

```
Password = Calculate_Password(”5632573” , Simcard_No)
Password2 = Now.Year + Now.Month + Now.Day*40 + Now.Hour*10 + 75*(Now.Minute+20)^3
```

4- FCM Server_2_Client Commands

The FCM message from server has the following properties:

```
data.info.Command_Code

data.info.Command_Data
```

Each of the FCM enabled web-services listed in Section 1-2 may be sent directly from the server to the client. If so, the client (device) should send acknowledge to the server via the web-service *Acknowledge_FCM_Server_Command*, as stated in the previous section. Besides these commands, the following messages may be passed from server to clients through FCM (Following commands do not need to be acknowledged by clients).

4-1) FCM_Driver_Tracking_On_Driver_Location_Change

```
data.info.Command_Code=200
data.info.Command_Data={Lat,Lon,Date(yyyyMMdd),Time(hhmmss)}
```

4-2) FCM_Driver_Travel_Reach_Point

```
data.info.Command_Code=201
data.info.Command_Data={Alarm_Point_Code, Organization_Code, Passenger_Code}
```

This Command is pushed to passengers device to inform the parents that driver has reached alarm or organization point or their address location point.

Alarm_Point_Code is 1, 2, 3, 4 or 5

4-3) FCM_Check_Server_4_Commands

```
data.info.Command_Code=202
data.info.Command_Data={}
```

Description: This command shows that the device has some command 2 receive from server and the device should call *Commands_In_Server_Queue_For_The_Device* imediately

---

4-4) FCM_Tracking_Program_Update

data.info.Command_Code=203

data.info.Command_Data={*Tracking_Program* object same as that in *Get_Driver_Device_Info*}

Description: This command is pushed upon any change in *tracking_program* or *tracking_program_timetable* to any driver registered in the following FCM topic:

DRIVER_CITY_*CITYNAME*_TRACKINGPROGRAMCODE_*CODE* (e.g. DRIVER_CITY_TEHRAN_TRACKINGPROGRAMCODE_25)

4-5) FCM_Organization_Update

data.info.Command_Code=204

data.info.Command_Data={*Organization* object same as that in the *Result_Data*→*Services*→*Organization* of *Get_Driver_Device_Info* web-service}

Description: This command is pushed upon any change in *the* corresponding organization in the panel to any driver registered in the following FCM topic:

DRIVER_CITY_*CITYNAME*_ORGANIZATION_*CODE* (e.g. DRIVER_CITY_TEHRAN_ORGANIZATION_2)

4-6) FCM_Alarm_Point_Update

data.info.Command_Code=205

data.info.Command_Data={Passenger_Code, Point_Code, Point_Descript, Lat, Lon}

Description: Upon setting (or deleting) an alarm point by the passenger, this FCM command is pushed to the corresponding driver. If *Lat*=0 and *Lon*=0, the corresponding point must be deleted. No Ack is needed for this command.

4-7) FCM_Message_From_Panel

data.info.Command_Code=206

data.info.Command_Data={Command_Reference_Code, Command_Code, Message_Text, Is_Group_Message(bool), Ack_Is_Neccessary(bool) }

Description: Command_Reference_Code is the code needed for the device to acknowledge this command via *Acknowledge_Command_Handled_By_Device* webservice. For personal messages (*Is_Group_Message*=false) this command is only pushed to the corresponding passenger or driver selected by the panel. For group messages (*Is_Group_Message*=true) this command is pushed to a group of devices registered to the corresponding topics:

Passengers:

PASSENGER_CITY_*CITYNAME* (e.g. PASSENGER_CITY_TEHRAN) : This is not active yet in the current version of panel.

PASSENGER_CITY_*CITYNAME*_ORGANIZATION_*CODE* (e.g. PASSENGER_CITY_TEHRAN_ORGANIZATION_1)

PASSENGER_CITY_*CITYNAME*_ORGANIZATION_*CODE*_CLASS_*CODE* (e.g. PASSENGER_CITY_TEHRAN_ORGANIZATION_1_CLASS_4)

Drivers:

DRIVER_CITY_*CITYNAME* (e.g. DRIVER_CITY_TEHRAN): This is not active yet in the current version of panel.

DRIVER_CITY_*CITYNAME*_ORGANIZATION_*CODE* (e.g. DRIVER_CITY_TEHRAN_ORGANIZATION_25)

DRIVER_CITY_*CITYNAME*_TRACKINGPROGRAMCODE_*CODE* (e.g. DRIVER_CITY_TEHRAN_TRACKINGPROGRAMCODE_25)
