Receipe: Timetables in scripts

Inhaltsverzeichnis

1 General

In the following procedures and functions no consideration is taken in the sense of the simplicity on special efficiency, since primarily at the example is to be explained, how the functions work. Therefore, at the end of each example it is generally suggested how the procedure can also be made more efficient.

2 Preliminary work

For various functions, the name of the AVLC server is required to which the AVLC on-board unit is to log on. This information is stored in the basic PIS file. Since it does not change from then on, it makes sense to store it in a variable. This should not be a "PUBLIC_VARS" variable, but one that is simply declared outside the procedures.

The variable should not be written in the `Initialize` procedure, but once during the first run of `SimStep`:

Code

```
var
NotFirstRun: boolean;
RBL_Server: string;
...
procedure Initialize;
begin
NotFirstRun := false;
...
end;
procedure SimStep;
begin
if not NotFirstRun then
begin
RBL_Server := PIS_GetITCSServer(Self);
... // other things that should only be called once from SimStep
NotFirstRun := true;
```

```
19. end;
20. ...
21. end;
```

Alles anzeigen

3 Routes (not times)

3.1 Where is the vehicle located on a particular route?

Assumed: The user has entered the line and route in the AVLC on-board unit and the system is now to determine whether and where the vehicle is on the route. The so-called "section" is returned. Here, counting is started at the start stop with number 0 and then the section between the stations and the stations themselves are counted alternately:

0: at the departure station

1: between the first and second station

2: at the second station

3: between the second and third station

...

Code

```
1. function WoAufDerRoute(linie: integer; route: integer): integer;
2. var
3. linieStr: string;
4. routeStr: string;
5. wayIndex: integer; // internal index of routes/ways
6. begin
7. // The procedures work with strings for the sake of flexibility.
8. // Therefore, the entered data must first be converted to strings:
9. linieStr := IntToStr(linie);
10. routeStr := IntToStr(route);
11. wayIndex := TimetableGetWayIndex(Self, RBL_Server, linieStr, routeStr);
12. // If the index is negative, then the entered route does not exist.
13. // Then simply -1 is to be returned:
14. if wayIndex < 0 then
15. begin
16. result := -1;
17. end
18. else
19. begin
20. result := TimetableAtSectionOfWay(Self, wayIndex);
21. end;
22. end;
```

Alles anzeigen

It is more efficient if the wayIndex is only recalculated when linieStr and/or routeStr change or when an input is made that potentially leads to a different wayIndex.

3.2 What is the name of the station where vehicle is located?

The following function returns the PIS ID that was entered in the configuration of the stations in the MapEditor. If there is one for the specific track, then this is returned. If there is only one for the entire station, then this is returned, otherwise the internal name of the station.

Again, the line and route numbers entered into the device must be passed to the function, as in the previous example. If no station was found, an empty string is returned:

Code

```
1. function AnWelcherHaltestelle_FIS_ID(linie: integer; route: integer): string;
2. var
3. linieStr: string;
4. routeStr: string;
5. wayIndex: integer;
6. section: integer;
7. stnID: string;
8. found: boolean;
9. begin
10. // Beginning like in the previous example
11. linieStr := IntToStr(linie);
12. routeStr := IntToStr(route);
13. wayIndex := TimetableGetWayIndex(Self, RBL_Server, linieStr, routeStr);
14. if wayIndex < 0 then
15. begin
16. result := '';
17. end
18. else
19. begin
20. section := TimetableAtSectionOfWay(Self, wayIndex);
21. // The function returns true if a station is found, in which case
22. // the FID ID is written to the variable stnID.
23. found := TimetableAtBusstopOfWay(Self, wayIndex, section, stnID);
24. if found then
25. begin
26. result := stnID;
27. end
28. else
29. begin
30. result := '';
31. end;
32. end;
33. end;
```

Alles anzeigen

Here, too, it would be more efficient if the wayindex were determined only when something changes in the line/route combination. But also the section has to be updated at least once per run - if at all so often, a "loose" interval is also sufficient here.

3.3 Distance between two stations along a route

This function can be used to determine - completely independently of where the vehicle is currently located! -

determine how far any two stations are from each other (measured along the route).

Code

```
function Stationsabstand(linie: integer; route: integer; FIS_ID_stnA, FIS_ID_stnB: string): single;
var
linieStr: string;
routeStr: string;
wayIndex: integer;
begin
// The beginning is like in the previous example
linieStr := IntToStr(linie);
routeStr := IntToStr(route);
wayIndex := TimetableGetWayIndex(Self, RBL_Server, linieStr, routeStr);
if wayIndex < 0 then
begin
result := -1;
end
else
begin
result := TimetableDistBetweenStns(Self, wayIndex, FIS_ID_stnA, FIS_ID_stnB);
end;
end;
```

Alles anzeigen

It is important here that the stations are searched for using the PIS ID stored in the MapEditor. If the route runs over a track that has its own PIS ID that differs from the station, then that ID is used. If the route runs over several stations (tracks) with the same PIS ID, the first station is used.

4 Trips (with times)

4.1 Convert hours and minutes to string

Code

```
function TimeToHHMM(time: single): string;
var
h: integer;
min: integer;
begin
// time is initially in days. Convert to hours:
time := time * 24.0;
// this number rounded down corresponds to the hour indication of a digital clock:
h := trunc(time);
result := IntToStrEnh(h, 2, '0') + ':';
// hours are subtracted from time, the remainder are the remaining
// minutes (but for this you have to multiply by 60:)
time := (time - h) * 60.0;
// only whole minutes are to be displayed, therefore again
// rounded down and the result is appended to the string:
min := trunc(time);
result := result + IntToStrEnh(min, 2, '0');
end;
```

Alles anzeigen

4.2 Show trips of a tour

For the following function, imagine that you first enter the line and course and then use the arrow keys to switch through the available journeys, which are then displayed in a text field with the first departure time, start and end station. `index` is the number, which is increased or decreased by 1 with the arrow keys.

For the following function, the system variables `TimeOfDay` and `Date`, both of which must be declared in the "PUBLIC_VARS" section.

Code

```
function Fahrtstring(linie: integer; kurs: integer; index: integer): string;
var
linieStr: string;
kursStr: string;
iTimetable, iTrip, iTourplan, iTour, iTourtrip: integer;
anzFahrten, anzStationen: integer;
linieDerFahrt, kursDerFahrt, routeDerFahrt: string;
globalFahrtIndex: integer;
abfahrtszeit: single;
stnFISID: string;
stnAnkunft, stnAbfahrt: single;
begin
linieStr := IntToStr(linie);
kursStr := IntToStr(kurs);
// the following indices must be determined based on the entered line/rate information
// and date/time. They identify the circulation and the
// trip on the roundabout:
TimetableGetTripAndTourIndexByLineCourseDate(Self, Date, TimeOfDay, RBL_Server, linieStr, kursStr, iTimetable, iTrip, iTourplan, iTour, iTourtrip);
// Now, a temporary list of the trips that have to be run on the
// round trip that has just been determined. The temporary list
// exists in the background until the following command is executed again
// is executed. Then the list will be replaced.
TimetableGenerateTempTripListByTour(Self, iTimetable, iTourplan, iTour);
// then write the length to a temporary variable:
anzFahrten := TimetableGetTempTripListCount(Self);
// Now check if the passed index is valid. If not, then directly
// return an empty string:
if (index < 0) or (index >= anzFahrten) then
begin
result := '';
end
else
begin
// now the data of this trip can be determined:
TimetableGetTripInfoByTempListIndex(Self, index, linieDerFahrt, kursDerFahrt, routeDerFahrt, globalFahrtIndex, abfahrtszeit);
// With the line and route information given here, the route functions described further
// route functions described above (e.g. to check if you are still on the route).
// still on the route) can be fed.
// since the first and last station is to be written to it, the station list must also be
// the station list must be generated. This is also generated temporarily
```

```
46. // according to the same principle as the trip list:
47. TimetableGenerateTempStnListByTrip(Self, iTimetable, globalFahrtIndex);
48.
49. // Determine the number of stations here as well:
50. anzStationen := TimetableGetTempStnListCount(Self);
51.
52. // If the list is empty, then return an empty string:
53. if anzStationen <= 0 then
54. begin
55. result := '';
56. end
57. else
58. begin
59. // Now assemble the string. First get the data of the first
60. // station:
61. TimetableGetInfoByTempStnListIndex(Self, 0, stnFISID, stnAnkunft, stnAbfahrt);
62. result := TimeToHHMM(abfahrtszeit) + ' : ' + stnFISID + ' => ';
63. // get the data of the last station (anzStationen-1):
64. TimetableGetInfoByTempStnListIndex(Self, anzStationen-1, stnFISID, stnAnkunft, stnAbfahrt);
65. result := result + stnFISID;
66. end;
67. end;
68. end;
```

Alles anzeigen

This example is full of inefficiencies, since all queries are made in one go. So, of course, the tour only has to be called up when the line and course change. Accordingly, the trip list only has to be updated then and the number of entries of this list can be written once into a global variable instead of into a function locale.

5 to be continued...