MILES32.DLL Technical Documentation

DLL - Dynamic Link Library - is a software module that has a library of functions that can be called from an Executable File (like an *.EXE, which can be referred to as a Windows application). A DLL can be called by any Windows executable application, which will load that DLL into memory of the computer under Windows environment.

MILES32.DLL has all the Route Calculation functionality of non-graphical version of MileMaker. All Route Processing functions are stored inside the Library of Functions, so it can be used by a 3rd party executable application. Developers of that application will be responsible for screen display and proper calls to the MILES32.DLL Library Functions.

Under this design MILES32.DLL substitutes MMAKER2.EXE, and continue using all DLLs that are used by an INI or registry settings.

In order to test MILES32.DLL with TEST32.EXE or another C++ application, the application must reside in the same directory as the MILES32.DLL and the remaining MileMaker software or the PATH environment variable must be set to point to the MileMaker directory.

In order to test MILES32.DLL with TEST32VB.EXE or another Visual Basic application the PATH environment variable must be set to point to the \Program Files\Rand McNally\MMAKER32\BIN folder or its equivalent.

Library Functions that can be called by third-party software from MILES32.DLL:

GetHHGDistance
Parameters - (Starting Location, Ending Location)

Will calculate the HHG Mileage between Starting Location and Ending Location. Both Locations must exist inside the MileMaker data, otherwise an error will be returned.

NOTE: In this function and the rest of this documentation, the word “Location" means user input in the form of:

SPLC
City/County/State
5-digit U.S. Zip Code

All above locations are 22 byte ASCII character strings declared in C as follows:

```
char location[23];
```

The 23rd byte is for the NULL character which follows the actual 22 bytes of information.

City/County/State is 18 bytes starting with the 1st byte, optional County is 2 bytes, starting with byte 19, and State is 2 bytes, starting with byte 21.

SPLC or Zip must start from the 1st byte. The 9 digit SPLC must then be appended by 13 spaces, or the 5 digit Zip Code appended by 17 spaces, in order to create 22 bytes character string.

GetPracDistance
Parameters - (Starting Location, Ending Location)

Will calculate Practical Mileage between Starting Location and Ending Location. Both Locations must exist inside the MileMaker data, otherwise error will be returned.

GetHHGDistanceValid
Parameters - (Starting Location, Ending Location)

Will calculate Mileage between Starting Location and Ending Location. Locations do not have to be valid; they will be validated by using the Validation dialog, which will let the user choose the location name with the closest match to the location entered.

GetPracDistanceValid
Parameters - (Starting Location, Ending Location)

Will calculate Mileage between Starting Location and Ending Location. Locations do not have to be valid; they will be validated by using the Validation dialog, which will let the user choose the location name with the closest match to the location entered.

Route
Parameters - (String of Input Records, String of Output Records, Route Type, Validation Flag)

Route Type represents 8 route types:
1. HHG Mileage,
2. HHG Origin,
3. HHG Audit Route
4. HHG Route State Mileage Breakdown Only,
5. HHG Full Route with State Mileage Breakdown
6. Practical Route,
7. Practical Route State Mileage Breakdown Only,
8. Practical Route with State Mileage Breakdown,

String of Input Records will contain up to 28 input locations that will be processed according to Route Type.

Upon successful calculation, String of Output Records representing Mileage, Route, or State Mileage Breakdown (depending on Route Type) will be returned.

NOTE: Each Output Record is 71 bytes long (see Batch Processing documentation for complete information)

If Validation Flag is set to 1, then all locations will be validated by using our validation screen. If Validation Flag is set to 0, then it is assumed that all locations are valid and no validation dialog will be displayed.

Validate
Parameters - (Location,Val Flag)

Will take the Location and validate it with MileMaker data using Validate dialog. If the location exists, it will return it back inside Location. In some cases it will return it in the abbreviated form.

EXAMPLE: if Location is MORTON GROVE, IL then Returned Location will be MORTON GRV, IL (refer to the MileMaker Documentation for a complete list of abbreviations).

Val Flag must be 0 for HHG Validation, and 8 for Practical Validation.

ValidateList
Parameters - (Location, Val Flag, LocationsList)

Will take Location and validate it with the MileMaker data. If the location cannot be found, then a list of possible locations closely matching Location will be returned inside Locations List.

It is assumed that the whole list will then be shown to the user so s/he can choose the right location using 3rd party developer designed screen.

Val Flag must be 0 for HHG Validation, and 8 for Practical Validation.

TransactionUpdate
Parameters (NONE)

Calling this function, MILES32.DLL will display the transaction update screen to let the user view the number of transactions used and update the license.

NOTE: if MileMaker has been installed as LAN version, then only the user with the MMADMIN account will be able to update transactions, the rest of the users will only be able to view them.

SetProgressDisplay
Parameters (Display Flag)

If the Display Flag is set to 1 (default) then all Route and Mileage Calculations will have a small screen with percentage of job done displayed during the route calculation. If the Display Flag is set to 0 then the screen will not be displayed.

All numeric data that is passed to the functions needs to be passed as 4 byte binary data. For C++ this would be an int or long field. For Visual Basic this would be a long field.

The CPP subdirectory contains the source for TEST32.EXE which has been compiled using 32-bit Visual C++
The VB subdirectory contains the source for TEST32VB.EXE which has been compiled using 32-bit Visual Basic.