## [7.3] Using language localization

This section was written by Lars Fredericksen.

There are basically four problems connected to writing programs/functions which have to run under language localisation:

1. AMS versions previous to 2.0X do not support language localisation.
2. Programs and functions can only be compiled (tokenised) in the language they were written in.
3. The names of the variable types (getType(var)="?") are unknown at the time of programming, because they are language dependent.
4. The names of the modes are unknown (getMode("?")="?":SetMode("?","?") ). There are other mode related function where the same problem occur, but the handling is principal the same so I will concentrate on the mentioned functions.

1) In AMS version 2.0X the mode functions support a new syntax: a mode number can replace the mode name. It is a very important feature, because it provides a way of setting the mode without knowing the language dependent name. But it does also give problems with calculators running AMS previous to 2.0X, because they are not compatible with the new syntax. If programs have to run on all AMS version and under different languages it is necessary to take both models into account.

To test if the AMS supports language localisation can easily be done in a program with this code:

```
Try
 GetMode("1")
 True»V2XX
Else
 False»V2XX
 ClrErr
EndTry
```

The V2XX variable will contain True if language localisation is supported, or False otherwise. It is convenient if the V2XX variable is a global variable, so that it can be used by functions to test for the version.

2) Almost all function names are language dependent so programs/functions can only be tokenised in the language they are written. In other words, if you have written a program in English it can only be run if the language setting is set to English. But that is only the first time it is run. If the program is executed one time with the correct language setting, the language can be changed to any other language and the program will still work correctly. The program can even be edited in another language, just remember to run the program one time before changing language. To save a language independent program, just make sure that the program is executed one time before transferring it with GraphLink.

3) Testing for a specific variable type in an unknown language using *getType()*

The following method can be used in programs/functions in all AMS versions.

```
Local TypeList,TypeMat
@ Acquiring the system for the type-name of a list.
{}»TypeList
getType(TypeList)»TypeList
@ Acquiring the system for the type-name of a matrix.
[0,0]»TypeMat
getType(TypeMat)»TypeMat
@ Testing if a variable is the type List.
If  getType(var)=TypeList Then
...
```

4) Setting/Testing a mode without knowing the language depending name.

When handling modes under language localisation only mode numbers should be used, because of some problem in the system with handling names containing foreign characters. This means is it necessary to know the mode numbering from Appendix D of the AMS 2.03 manual.

To set a mode and restore it:

```
Local Mode,OldMode
@ Set Graph=Function
If V2XX Then
 "1"»Mode
 SetMode(Mode,"1")»OldMode
ELSE
 "Graph"»Mode
 SetMode(Mode,"Function")»OldMode
Endif
...
SetMode(Mode,OldMode)
```

To test for a mode in a function is a little complicated. It is necessary for an installation program to get the language dependent mode names and store them in global variable, which can be used by the functions.

An example of translating Radian to the active language:

```
If V2XX Then
 "3"»MAngle
 SetMode(MAngle,"1")»OldMode
 GetMode(MAngle)»MRadian
 SetMode(MAngle,OldMode)
Else
 "Angle"»MAngle
 "Radian"»MRadian
Endif
```

When the modes have been translated and stored in global functions, they can be used in functions like this:

```
If GetMode(MAngle)=MRadian Then
...
```

*(Credit to Lars Fredericksen)*