Julius PETER

julius.peter1000@gmail.com

Abstract

Excel-Add-In to support multi-language enabled applications/spreadsheets

Excel Multi-Language Support (MLS) Add-In

Documentation

Table of Contents

[Synopsis 3](#_Toc17641724)

[MLS implementation (NLS table) 3](#_Toc17641725)

[MLS related functions 4](#_Toc17641726)

[Text in cells / GetNLSText function 4](#_Toc17641727)

[Shapes 4](#_Toc17641728)

[Text2 function 4](#_Toc17641729)

[SetLanguage 5](#_Toc17641730)

[VBA Code use of MLS related functions 5](#_Toc17641731)

[GetNlsText 5](#_Toc17641732)

[GetNlsAddition 6](#_Toc17641733)

[IsYes / IsNo 6](#_Toc17641734)

[ShowMessage/ShowConfirm 7](#_Toc17641735)

[SetNlsData 7](#_Toc17641736)

[NLS table editor 7](#_Toc17641737)

[All Languages 8](#_Toc17641738)

[Add Entry 8](#_Toc17641739)

[Edit/Clone 8](#_Toc17641740)

[Delete Entry 9](#_Toc17641741)

[Show NLS Calls 9](#_Toc17641742)

[Filter 9](#_Toc17641743)

[Clear 9](#_Toc17641744)

[Use selection 9](#_Toc17641745)

[Close Window 9](#_Toc17641746)

[Integration of the NLS table editor with the VBE and Excel Workbooks 9](#_Toc17641747)

[Reference & Info Functions 10](#_Toc17641748)

[Show NLS Calls (with orphan control) 10](#_Toc17641749)

[Show Widows 11](#_Toc17641750)

[Settings 11](#_Toc17641751)

[Stubs 11](#_Toc17641752)

[IsDeveloper 12](#_Toc17641753)

[IsPlatformDeveloper 12](#_Toc17641754)

[GetSheetProtectionPassword 12](#_Toc17641755)

[GetBookProtectionPassword 12](#_Toc17641756)

# Synopsis

This Excel Add-in (xlam file) offers advanced functionality for excel spreadsheets or applications with are meant to be used in a multi-language environment. It offers simple functions to display text elements as e.g. cell content, (shape) captions and messages in several languages by just selection of a language setting. This language setting can be independent/different than the selected Excel language setting, e.g. you can run an application in German on an English based Excel installation or vice versa.

This language setting can be based on the selected excel user interface language but is in general independent from such setting. It’s also possible to have some text elements bound to the excel system language, while others are dependent of the excel user language.

This excel add-in also offers advanced development support with advanced editing functionality that is integrated with both the excel and the excel VBE environment. These features are the most valuable elements of this add-in. Cross-referencing usage of text elements in cells, shapes and VBA (macro) code is also supported as well as widow (unused text elements) and orphan (undefined text elements) control.

It consists of a two-level database (table) of text elements, identified and selected by a key – module (larger application area) and identifier (identifying a specific element). The first level (optional) is a table usually placed in the specific excel workbook and a second table placed in the add-in itself. If a specific text element is requested and is available in the first level this will get returned. If not the second level will get searched and any find returned – else an error/default text. This allows to have a set of generic text messages that can be extended od superseded by application/document specific text elements.

The (sorted) tables are held in memory for fast access to all text elements.

# MLS implementation (NLS table)

Each text element is identified by a module (string) and identifier (string) value, which the system combines to a key (which is also provided as a pre-calculated value to avoid startup recalculation issues). In addition, the level of the text element (App for application = user workbook or Plt for platform = add-in). Two optional fields are available:

* Type (e.g. cell text, caption, message, error message) describing the intended use of the text element to support translation
* Additional is intended for any user defined purposes, e.g. specifying a face-id for the display of custom menu elements

Following those fields, the NLS table can have 1 – n columns, each one representing a supported language.

Text elements support optional placeholders, which will get replaced by provided text elements/strings, when a text element is requested. Currently up to 4 placeholders are supported (\_\_&1\_\_ to \_\_&4\_\_). Enlarging this number would be easy to implement. Additionally, you can use the “|” character as placeholder for a line break.

The selected language must be in list of supported languages (as provided in the add-in). Optionally you can add information about the to be used locale (LCID code; helpful for date to text conversion/formatting; e.g. Sunday vs. Sonntag) and the to be used decimal and thousands separators. Date separators are controlled by Windows and cannot be set by Excel.

The following example shows a two-language setup – supporting English and German (Deutsch). Additional languages would be represented by additional columns in the table and rows in the available language range.



# MLS related functions

## Text in cells / GetNLSText function

Cell text will be MLS enabled by use of the GetNLSText function.

A specific text element will be retrieved using the **GetNLSText** function. You would enter the following function call into a cell =GetNLSText(module,identifier,<P1>,<P2>,<P3>,<P4>), with up to 4 optional strings to replace the defined placeholders. If the specified module & identifier are not found in any of the two NLS tables (application level and platform level), a customized/language dependent default/error message will get displayed/returned.

Example: Unknown message <module/identifier>

As this function is non-volatile, all such cells need to get refreshed on a language change. This will be achieved using SetLanguage sub. You can run this sub either via a button/shape or automatically on any language change. The demo file (MultiLanguageDemo.xlsb) offers both options.

Of course, the GetNLSText function can also be used in your VBA code / macros.

## Shapes

Shapes will be MLS enabled, if the shape supports a textFrame property, which many shapes as rectangles etc. do, and the NLS table contains an entry with the module Button and an identifier that is equal to the shape name. Shapes with the same name on different sheets will be treated identically. Its good practice to use significant names for your shapes, so they are identifiable in the table.

## Text2 function

This function works like the regular XLS Text function, but also implements the use of dateLocale (if available). This allows to convert dates to strings in local language. Furthermore, it allows the use the international (English) codes for Year, Month, Day, Minute, Second etc. independently from the Excel System language. Thus, you can switch language without having to adjust the formatting code e.g. from YYYY (English) to JJJJ (German) for years. Usage: =Text2(DateValue, Format)

## SetLanguage

This VBA sub will execute a language change. It will refresh any cells that use the GetNLSText function. It will also pick up and use a DateLocale, as well as the settings for DecimalSeparator and ThousandsSeparator, if provided (not blank or named range not available) and have them available in the respective named ranges.

Furthermore, this sub will check all shapes and – if found in the NLS table – set the language specific text.

It can be called from a shape, event (e.g. change of the language selection field) or any other VBA function/sub.

## VBA Code use of MLS related functions

### GetNlsText

As already stated above this function will often be used as part of VBA code/macros. See above.

Definition:

***Function GetNlsText(ByVal module As String, Optional ByVal identifier As String, \_***

***Optional ByVal p1 As String, Optional ByVal p2 As String, Optional ByVal p3 As String, Optional ByVal p4 As String, Optional useSystemLanguage As Boolean, \_***

***Optional mandatory As Boolean, Optional quiet As Boolean = False) As String***

#### Parameter:

|  |  |  |
| --- | --- | --- |
| module | String | as explained above |
| (opt) identifier | String | As explained above  If not provided/blank, module will be interpreted either as fixed, language independent text or as a string of parameters for this function, separated by a parameter separator, defined in Sub SetNLSData (default = °°, should be a string that never will be part of a text).  This allows to construct the to-be-used parameter string in Excel or VBA. |
| (opt) P1 – to P4 | String | Strings to get inserted instead of the placeholders |
| (opt) useSystemLanguage | Boolean | If True, the excel system language will be used for retrieving the text and not the selected language; useful for text elements that change with the system language, e.g. how empty fields in pivots are named; (blank) or (Leer) |
| (opt) mandatory | Boolean | This option can used to change the behavior in case an entry is not provided in the NLS tables. If some text is critical for the function of e.g. VBA code, by setting this to True, a call that cannot get resolved will return a null string (and VBA code can react to this) and an error message will get displayed (optional) |
| (opt) quiet | Boolean | If true, an error message as above will be suppressed |

### GetNlsAddition

Used to retrieve the additional Information (e.g. FaceIds for menu elements) for a module/identifier combination.

Definition:

Function GetNlsAddition(module As String, identifier As String) As String

#### Use case:

Create an MLS enabled pop-up menu with FaceId (small graphic symbol) to call a macro (TestMacro). The FaceId number is stored as Additional Information in the NLS table

Sub testMenu()

Dim macroName As String, fidStr As String

Dim parentMenu As CommandBar

Set parentMenu = Application.CommandBars.Add(Name:="Test", position:=msoBarPopup, \_

MenuBar:=False, Temporary:=True)

With parentMenu

With .Controls.Add(Type:=msoControlButton)

macroName = "TestMacro"

.caption = GetNlsText("contextmenu", macroName)

fidStr = GetNlsAddition("ContextMenu", macroName)

If IsNumeric(fidStr) Then .faceId = CLng(fidStr)

.OnAction = "'" & ThisWorkbook.Name & "'!" & macroName

End With

End With

parentMenu.ShowPopup

parentMenu.Delete

End Sub

Sub TestMacro()

Call ShowMessage("Test", "Macro", smInfo)

End Sub

### IsYes / IsNo

While the range YesNo as shown in the demo workbook is a good technique to support multi-language data, any text that the user selected from such an MLS enabled validation list will (at least currently) not change with the language. If such a field/cell is used for programmatic purposes, it’s a good idea to recognize a positive or negative answer/setting independent from the selected language. IsYes/IsNo tries to achieve that by using a list of comma separated values that the function will identify as Yes or No across all supported languages (only feasible if there is no overlap of options). The following NLS table entries represent these options:



The above options would work correctly for both, English and German, without e.g. changing a user entered Ja/J to Yes/Y. If you add French as supported language you might want to add O and Oui as OptionsForYes and Non as OptionsForNo. Next to the data validated field in the demo file you can check out the use of these functions (will probably mostly be used in VBA code).

#### Definition:

Function IsYes(str, Optional acceptBlanks As Boolean, Optional opt As String) As Boolean

Function IsNo(str, Optional acceptBlanks As Boolean, Optional opt As String) As Boolean

Acceptblanks will accept a blank value as true, opt allows to use a different comma separated string for testing than the standard OptoinsForYes/OptionsForNo.

### ShowMessage/ShowConfirm

Those functions are not intrinsic to the package. They encapsulate message and confirmation pop-up windows by using the NLSText function and defining message and confirmation levels with some basic customization of the pop-up windows.

Public Enum msgLevel 'For showMessage

smInfo

smWarning

smError

smSystemError

End Enum

Public Enum confirmLevel 'For showConfirm

cfInfo

cfWarning

End Enum

For more details see the corresponding functions in the add-in, module NlsUsingFunctions.

### SetNlsData

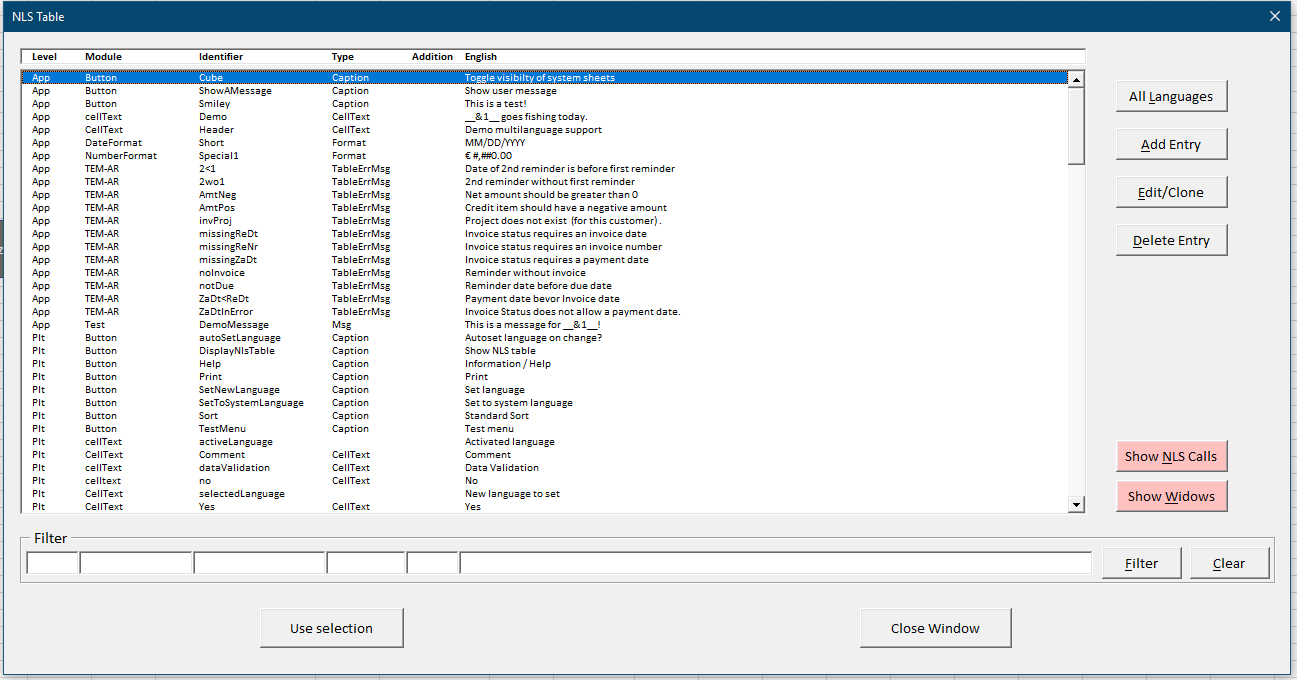
Is the internal function used to get all information to execute the MLS support into memory for quick access. It will get executed on book activation and whenever a related function is triggered. This function will check that a) the necessary data are already in memory and b) the data are valid for this specific workbook. If both holds true, no further action is taken, else the NLS data structures will get (re-) created. Any changes to the NLS tables will automatically invalidate the in-memory NLS data structures and they will get recreated on the next NLS call.

# NLS table editor

While the two NLS tables (App and Plt) can be edited manually like any other table, this add-in also provides an editor to support edit operation. It can be called via the developer menu (Ctrl-Shift-M), the shortcut key Ctrl-Shift-T or the cell context menu (right-click) in Excel, or by entering dm (developer menu) or nt (NLS table) in the immediate window of the Visual Basic Editor (VBE).

Development functions are only available if the user is authorized as [Developer](#_IsDeveloper) (= allowed to use development functions in the excel file) or [PlatformDeveloper](#_IsPlatformDeveloper) (allowed to make changes to the add-in). These checks are implemented as stub in the demo/add-in and probably needs to get fleshed out. Currently both checks will always return True. See the module [Stubs](#_Stubs).

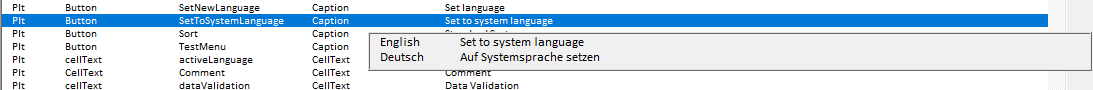
Note: To edit the Plt level NLS table in the add-in you need to remove the add-in property of the xlam file. The function Toggle add-in, available through the above developer menu or cell context menu helps you with that. Before saving you must set the add-in property again to True. Makes sure you save the add-in after making any changes. An Excel crash or closing the xlsb file without saving will let you lose all changes.



The editor shows all entries as above with usually just on language text (as defined in the pubic constant c\_primaryDevelopmentLanguage in the module Setting (either in the add-in or the excel file; any setting in the excel file will overwrite a setting in the add-in).

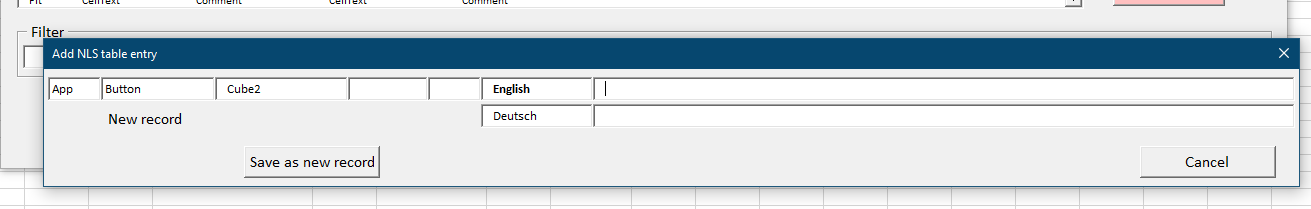
All Languages

(toggle on/off) allows you to see all languages per selected entry if wanted.



Add Entry

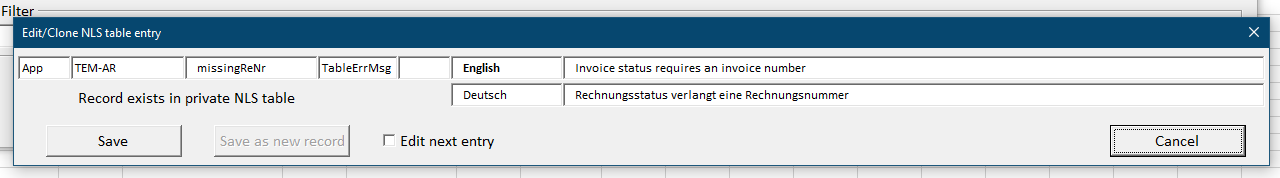
allows you to add an entry to the table. Only a PlatformDeveloper can add a Plt level entry.



Depending on the data you already entered into the Level, Module & Identifier fields a message will show the status of the new record and the Save as new record button will be activated. The fields for Module and Type allow you to select from the already existing entries in these columns. You can however also use new values.

Edit/Clone

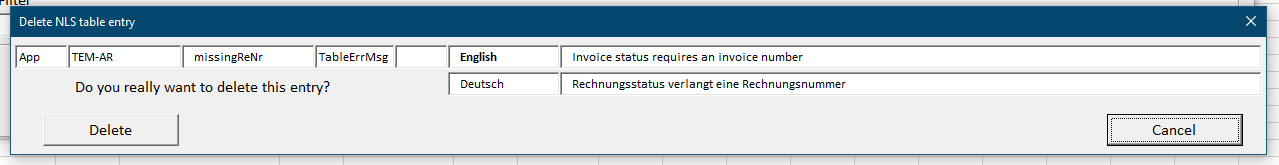
allows you to edit or clone (add an entry based on the selected entry) an entry. Only a PlatformDeveloper can edit a Plt level entry. A Developer can however clone a Plt level entry into an App level entry.



Depending on the data you already entered into the Level, Module & Identifier fields a message will show the status of the record and only the appropriate action buttons will get activated. As above, the fields for Module and Type allow you to select from the already existing entries in these columns. You can however also use new values.

## Delete Entry

allows you to delete an entry (after confirmation). Only a PlatformDeveloper can delete a Plt level entry.



## [Show NLS Calls](#_Show_NLS_Calls)

will close the editor and show a list of all uses/calls of NLS function (module & identifier as strings - not variables). This allows to check for orphans (module/identifier combinations in your file or code that cannot be found in the table. For more details see below.

[Show Widows](#_Show_Widows)

will close the editor and show a list of all entries that (presumably) are not used in your code or file. Bear in mind that calls, where module and/or identifier are passed as variables (not strings) will not get identified as used. For more details see below.

## Filter

allows you to filter your table by any (combination of) fields. Filter is not case sensitive and uses the [like operator](https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/operators/like-operator), wrapping the filter value in \*. Filter values are kept between calls to the NLS table.

## Clear

will reset all filter values to blanks.

Use selection

will close the editor and place module & identifier (as “module”, “identifier”) into the clipboard, so it can get pasted directly into a cell formula or VBA code line.

Close Window

(as well as the Esc key and the top-right x symbol) will close the NLS table editor.

## Integration of the NLS table editor with the VBE and Excel Workbooks

As already described above, you can access the NLS table editor via shortcut key in Excel (Ctrl-Shift-T) or the short command nt in the immediate window of the VBE. The other option is to use the development menu (shortcut Ctrl-Shift-M in Excel or the short command dm in the immediate window of the VBE. Furthermore, the development menu is accessible through many of the context menus (right-click), but not the Shape context menu (different internal structure).

When calling the NLS table editor, it checks if the active code line/cell/shape contains an MLS related call with module & identifier passed as strings (not as a string variable). If yes, the editor will open with the respective entry identified by module/identifier pre-selected. If both, an App and a Plt entry exist the App entry will get selected.

If no such entry is found, the editor will ask if you want to create such an entry and pre-set the module & identifier fields.

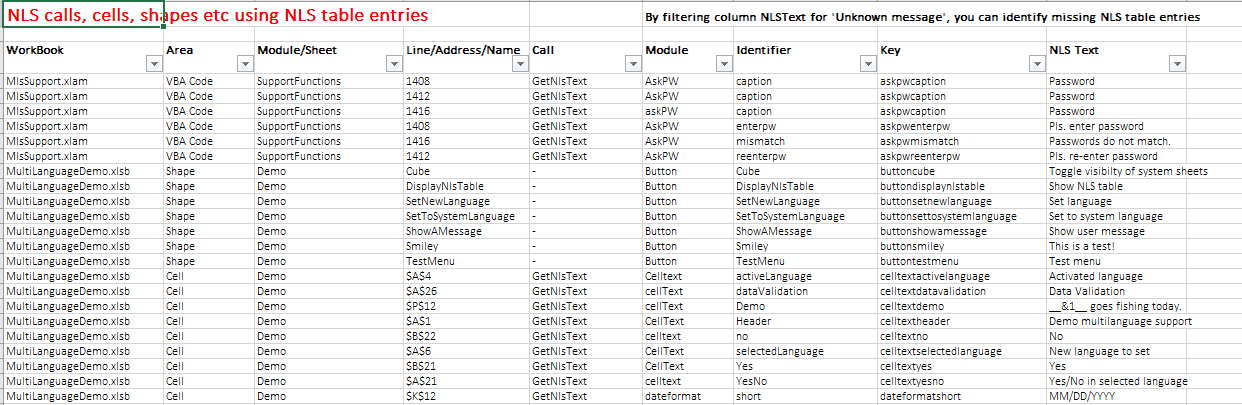
By pressing the Use Selection button, you can copy the module & identifier info of the selected entry into the clipboard and paste this info directly into a cell formula or VBE code. If a shape is selected, only the identifier is copied to the clipboard and can get pasted into the Excel name box.

As the getNLSText function is non-volatile, you need to execute a Set Language command to reflect changes in the NLS table in your workbooks. Depending on your set-up, it might be a good option to do this when a book is opened (of course also a performance question).

# Reference & Info Functions

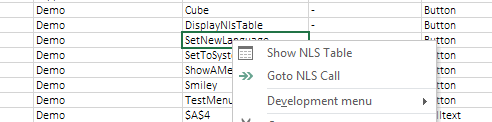
## Show NLS Calls (with orphan control)

This function, available via the NLS table editor and the development menu, shows a list of all uses of functions/constructs that access the NLS tables, as long as the call conforms to the module/identifier structure and both elements are passed/used as strings. Calls where these values are passed via variables will not get resolved.



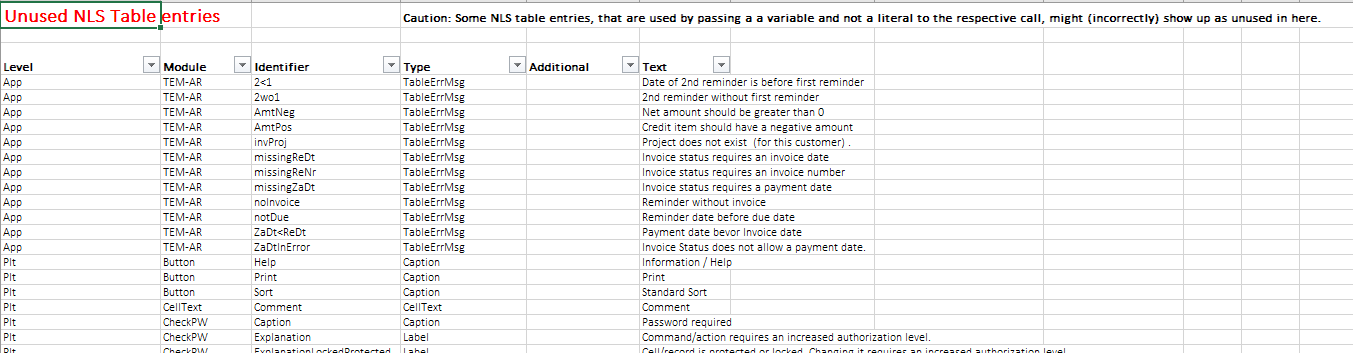
The NLS text will be displayed in the active language. By filtering the NLS text for the respective “missing entry” message you can identify orphans, that is calls where the respective NLS table entry is missing.

With a right-click, you can jump to the call in either the workbook or your VBE code or call the NLS table editor to change an entry or fix an orphan.



## Show Widows

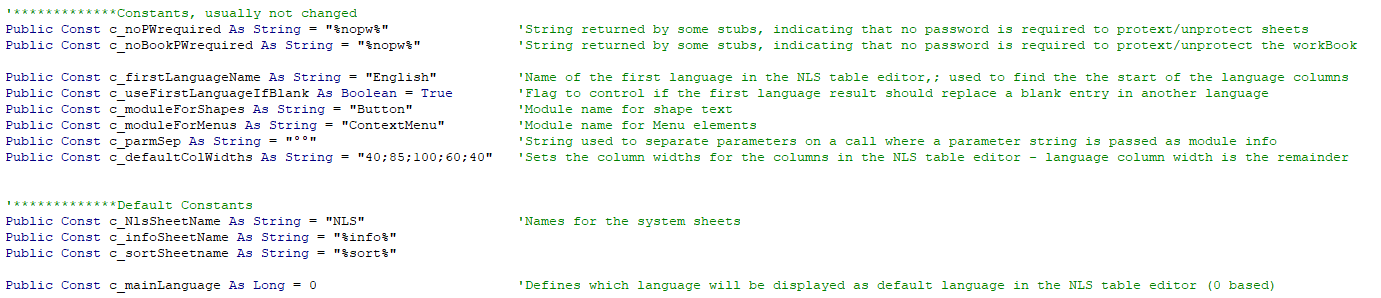
Widows are NLS table entries, where the respective module/identifier combination is not found in any NLS Call. They may be candidates for deletion, but pls. be cautious as many “irregular” uses (e.g. only calls where module & identifier are passed as strings, not variables will be shown in the NLS call list as above) will not get caught by the logic of the Show NLS call function.



# Settings

In the module Settings you will some customizable constants, that will govern the working and behavior of the MSL add-in.

E.g. you find in the add-in related Settings module



Two constants that you might want change:

Public Const c\_mainLanguage As Long = 0

Controls which language will get displayed as standard in the NLS table editor (0 …. First language; usually English, 1 …… 2nd language; e.g. Deutsch)

Public Const c\_useFirstLanguageIfBlank As Boolean = True

If set to true, any text item in a specific language that is left blank in the NLS table will default to the first language entry. This allows to have only selected entries of the table translated into a specific language.

If you need to have such settings changed by the normal, non-VBE user (unlikely), you need to change them from constants to public vars and control the settings of those vars with your own code, that e.g. reading in a value from a range at start-up/change.

All settings are valid for all client application that use the add-in. If you need different settings for client applications, you need multiple versions of the add-in.

## Add-In setup

I always found it quite comfortable to have the add-in in the same directory that the client file. Moving then add-in to another location might make it necessary to update the reference in the client file, unless the xlam file is in the same directory than the client-file. If you copy both files into another directory it will probably still reference the xlam in the original location. Only when the xlam cannot be found there (e.g. temporarily renamed), the client file will pick up the (new) local reference.

# Stubs

Stubs are provided in the module Stubs for functions that are (optionally) needed for the add-in to function but must be customized/fleshed out for a specific environment.

## IsDeveloper

Boolean. Returns true if a specific user is authorized to use development functions, e.g. call the NLS table editor.

## IsPlatformDeveloper

Boolean. Returns true if a specific user is authorized to alter the add-in, e.g. edit/add/delete platform level NLS table entries.

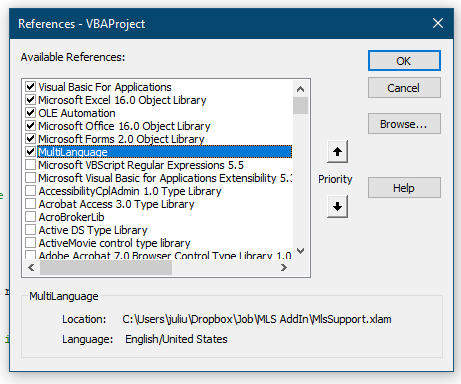
## GetSheetProtectionPassword

Returns a string with the password to protect/unprotect the sheet passed as parameter. Can also return a blank string, which tells other functions to ask for the password. If its set to return the constant c\_noPWrequired (default) other functions will get executed without password.

## GetBookProtectionPassword

Returns a string with the password to protect/unprotect the workbook. Can also return a blank string, which tells other functions to ask for the password. If its set to return the constant c\_noBookPWrequired (default) other functions will get executed without password.

# Installation

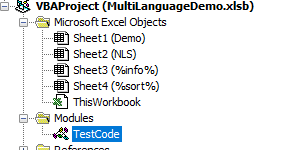
The easiest way of installing the MLS support is to copy both the add-in and the demo file into a directory of your choice and start building your app based on the demo app. You can delete the demo worksheet and add all your private add sheets. If you want to have the add-in in a central location of your choice, you need to manually set the reference to the correct location (in the VBE under Tools – References). 

Select Browse and look for the file MslSupport.xlam. If the reference for MultiLanguage points to the wrong location, uncheck Multilanguage, save the file and restart by re-adding the correct reference.

# Integrating MlsSupport.xlam into an existing Execl workbook

This is a little bit trickier. You need to copy the three system worksheets (NLS, %info% and %sort%) to you existing Excel app. Make sure the reference to the range langugeOptions in the add-in is added to your Excel file.

You also need the code in the object modules for ThisWorkoBook and the NLS worksheet (whatever programming name it has in your app – here it’s named sheet2). Usually a simple copy/paste or import is all you need to do. Testcode is not really needed.



If your application workbook is using the same event procedures below, you d to integrate the simple calls into your respective code.

Private Sub Workbook\_Activate()

Call BookActivate

End Sub

Private Sub Workbook\_BeforeClose(Cancel As Boolean)

Call bookBeforeClose

End Sub

Private Sub Workbook\_Deactivate()

Call BookDeactivate

End Sub

Private Sub Workbook\_Open()

Call BookOpen

End Sub

Private Sub Workbook\_SheetBeforeRightClick(ByVal sh As Object, ByVal Target As Range, Cancel As Boolean)

Cancel = SheetBeforeRightClick(sh, Target)

End Sub

Private Sub Workbook\_SheetChange(ByVal sh As Object, ByVal Target As Range)

Call SheetChange(sh, Target)

End Sub