

TMS Spell Check for FireMonkey DEVELOPERS GUIDE

May 2016
Copyright © 2016 by tmssoftware.com bvba
Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability 3
Online references 3
Introduction 4
Getting started 6
Working with multiple languages 11
Using the predefined spell check user interface controls 12
Spell check events 18
Spell asynchronous handling and callback context 20
Spell check database files 23

Availability

TMS Spell Check for FireMonkey is available as FireMonkey component for Delphi and C++Builder.

TMS Spell Check for FireMonkey is available for Delphi XE6 & C++Builder XE6 or newer releases for cross-platform desktop applications (Windows / Mac OS-X)

TMS Spell Check for FireMonkey has been designed for and tested with: Windows 7, Windows 8, Mac OS-X.

TMS Spell Check is also available as VCL component:
http://www.tmssoftware.com/site/tmsspellcheck.asp

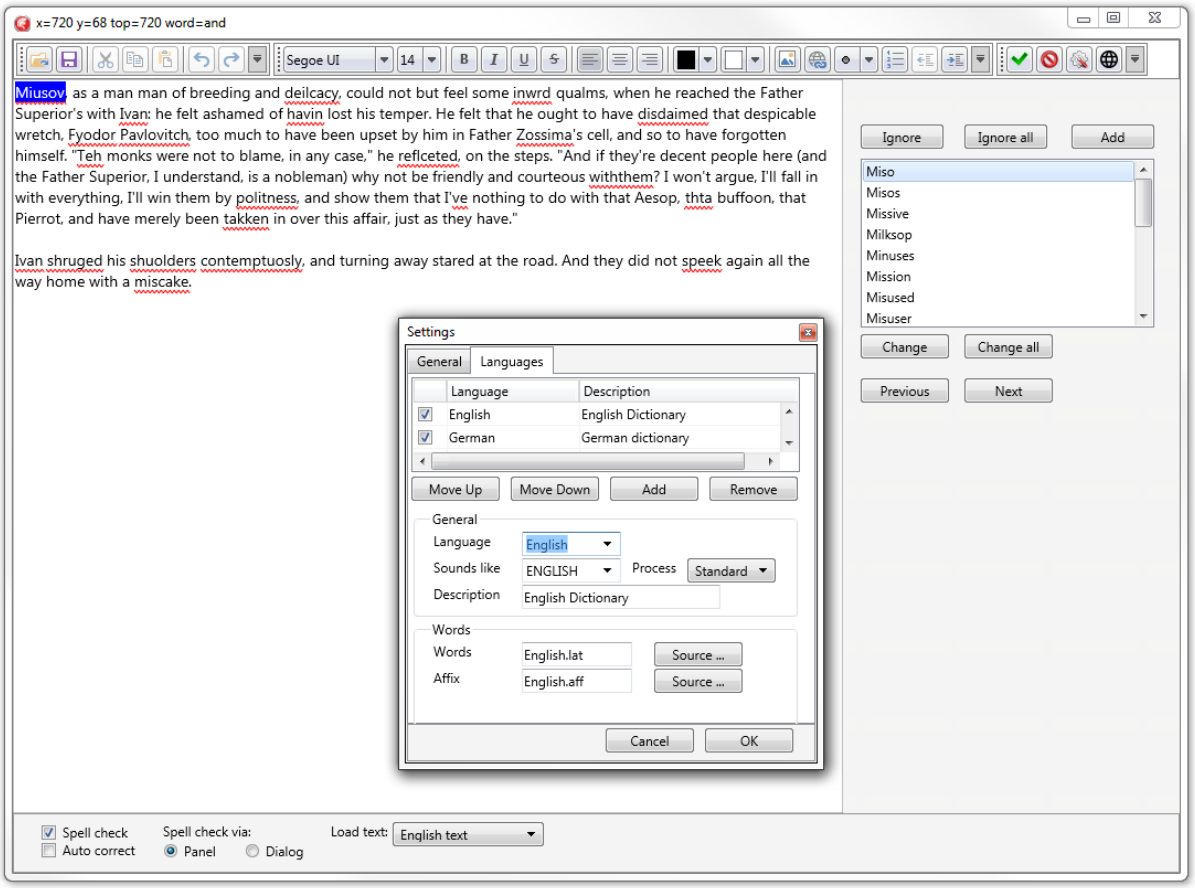
Online references

TMS software website:
http://www.tmssoftware.com

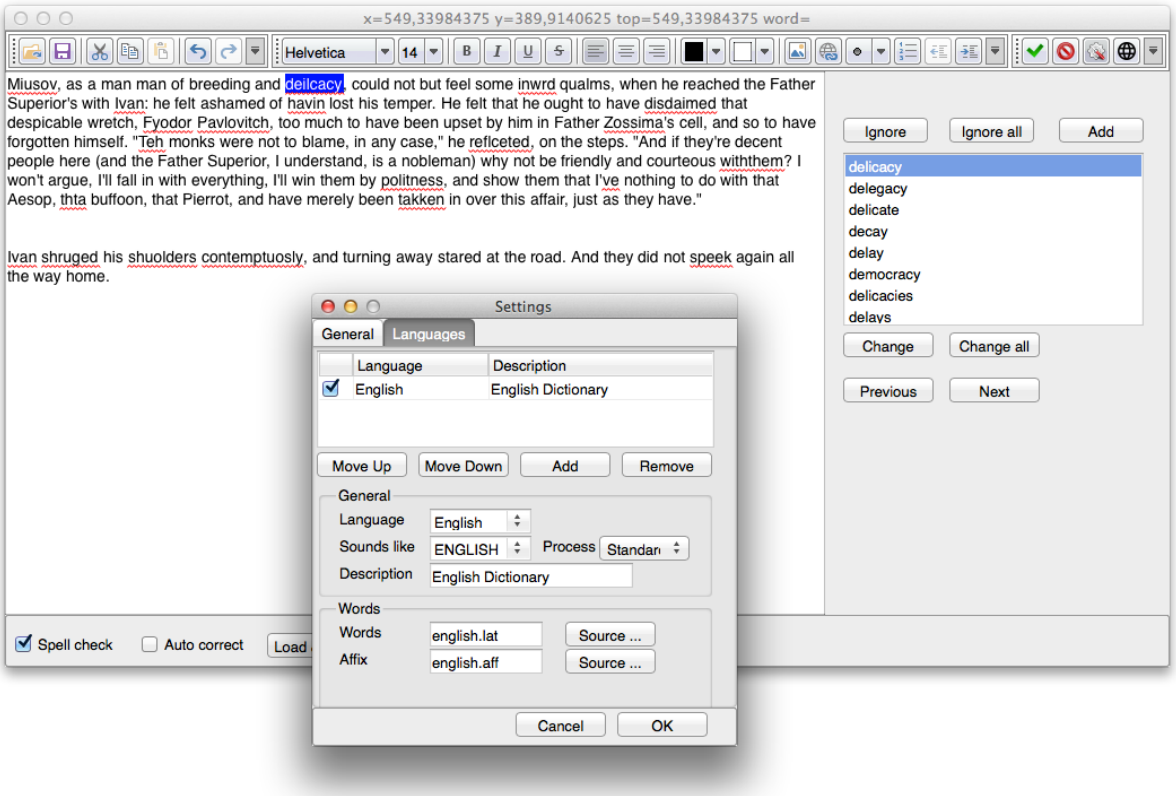
TMS Spell Check page:
http://www.tmssoftware.com/site/tmsfmxpack.asp

Introduction

TMS Spell Check for FireMonkey is a spell check engine with built-in support for single or multiple languages. TMS Spell Check for FireMonkey is based in ISpell. This means that ISpell dictionary files and ISpell affix files can be used with TMS Spell Check for FireMonkey. Via the technique of dictionary files and affix files, the TMS Spell Check for FireMonkey engine can generate very large word list from using the affixes. The soundex algorithm is used to find suggestions. TMS Spell Check for FireMonkey comes standard with dictionary and affix files for English (US & UK), German, French, Spanish, Italian and Dutch. Over time, more dictionaries & affix files will be included standard with TMS Spell Check for FireMonkey. TMS Spell Check for FireMonkey can directly interface to TTMSFMXRichEditor or it can be used standalone. TMS Spell Check for FireMonkey persists its dictionaries in a database. The dictionary database is typically initialized from a dictionary text file, affix file and optionally ignore list. The database can be restricted to in-memory use only (and will need to be loaded each time the application starts) or can use a database file (.SPLX file).



TMS Spell Check for FireMonkey with TMSFMXRichEditor on Windows

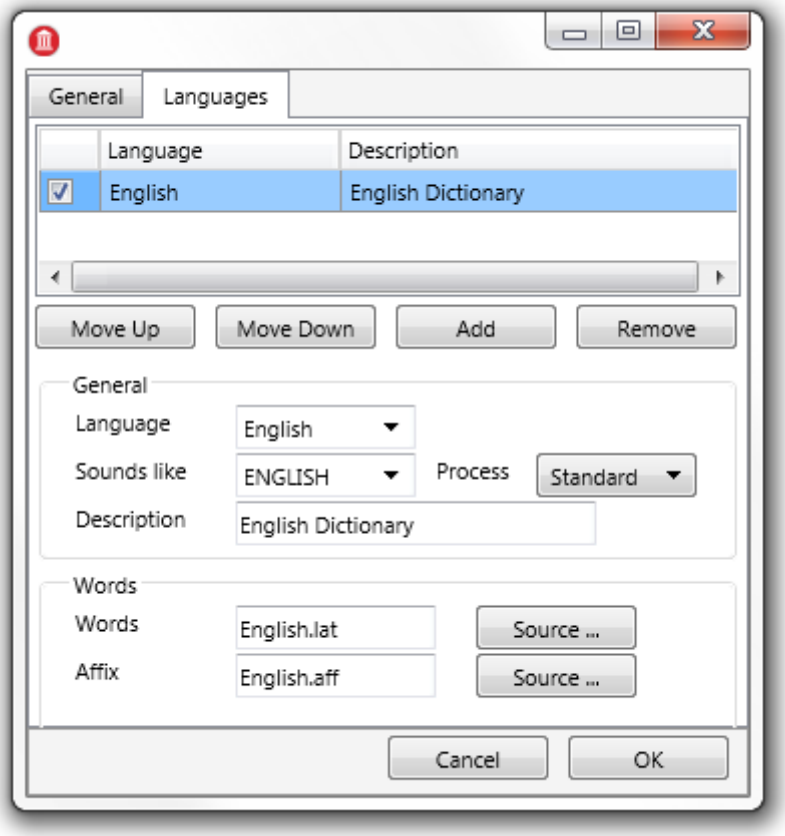


TMS Spell Check for FireMonkey with TMSFMXRichEditor on Mac OS-X

Getting started

Drop TTMSFMXSpellCheck on the form. Either Dbl-click on the TTMSFMXSpellCheck component to open the configuration dialog or directly edit the TTMSFMXSpellCheck.Languages collection property.

Configuration dialog:



Default, the English language is configured and this uses the word list: English.lat and the affix-file: English.aff. The files English.lat and English.aff are included with TMS Spell Check for FireMonkey and are used to initialize the database. Set TTMSFMXSpellCheck.Active = true as well as TTMSFMXSpellCheck.AutoUpdate = true and make sure the files English.lat and English.aff are present in the application executable folder. When the application starts, the first that will happen is the one-time generation of the spell check database TMSSPELLCHECK.SPLX (as defined with the TTMSFMXSpellCheck.DatabaseFileName property).

When the spell check engine is initialized, it is ready for use. It can be used either in a synchronous way or an asynchronous way.

Synchronous spell check:

Three functions operate synchronously:

TTMSFMXSpellCheck.Validate(AWord: string): TTMSFMXWordValidationResult;

The response for the Validate function is defined as:

wvrValidated, wvrNotValidated, wvrAttentionRequired

When the response is wvrValidate, this means the word is valid according to the active language dictionary. When the responsie is wvrAttentionRequired, it means the word might be correct but have for example an issue with the use of capital letters. When the response is wvrNotValidated, it means the word is incorrect.

Example code:

```
// This code performs a spell check on a word in a TEdit and sets
// its color to red when the word is incorrect.
procedure TForm1.Button1Click(Sender: TObject);
begin
  if TMSFMXSpellCheck1.Validate(Edit1.Text) = wvrNotValidated then
    Edit1.FontColor := claRed
  else
    Edit1.FontColor := claWhite;
end;
```

TTMSFMXSpellCheck.FirstSuggestion (AWord: string): string;

This function returns the first word from the suggestion list for a misspelled word. When the word is spelled correct, it returns the word as-is. This function can be used to implement an autocorrect function.

Example code:

```
// This code gets the first word from the suggestion list for an
// entered word. When found, it replaces the edit control value.
procedure TForm1.Edit1Exit(Sender: TObject);
var
  s: string;
begin
  s := TMSFMXSpellCheck1.FirstSuggestion(Edit1.Text);
  if (s <> '') and (Edit1.Text <> s) then
    Edit1.Text := s;
end;
```

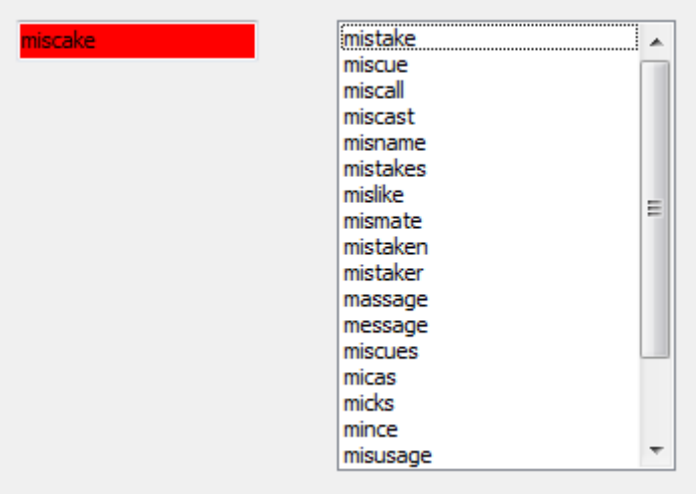
TTMSFMXSpellCheck.Suggestions(AWord: string): string;
The function TTMSFMXSpellCheck.Suggestions returns a list of possible suggestions for a misspelled word. The return is a list of words separated by a linebreak.

Example code:

```
// This code fills a listbox with suggestions for a misspelled word // in a
TEdit
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  if TMSFMXSpellCheck1.Validate(Edit1.Text) = wvrNotValidated then
  begin
    Edit1.FontColor := claRed;
    Listbox1.Items.Text := TMSFMXSpellCheck1.Suggestions(Edit1.Text);
  end;
```

```
end;
```

Result:



Asynchronous spell check:

For performing a spell check on large documents, an asynchronous method is provided. Via this asynchronous method, a large series of words can be sent to the engine that will asynchronously process the list of words and return the results via a callback.

First of all, in the unit FMX.TMSSpellParser, there is a helper function to retrieve a list of words from a text. When the list of words is available, the asynchronous request is started with a unique ID (to allow for multiple processed to use the same engine) via the method TTMSFMXSpellCheck.BeginRequest(ID). For each word, the method TTMSFMXSpellCheck.AddValidationRequest() is called. It is also possible to immediately asynchronously receive the suggestion list via TTMSFMXSpellCheck.AddSuggestionRequest. These method have 3 parameters: the word, a data object and the callback method. The data object can be used as identication of the word passed to the AdValidationRequest or AddSuggestionRequest. In this sample, the index of the word is used (the object is casted to an integer).

Example code:

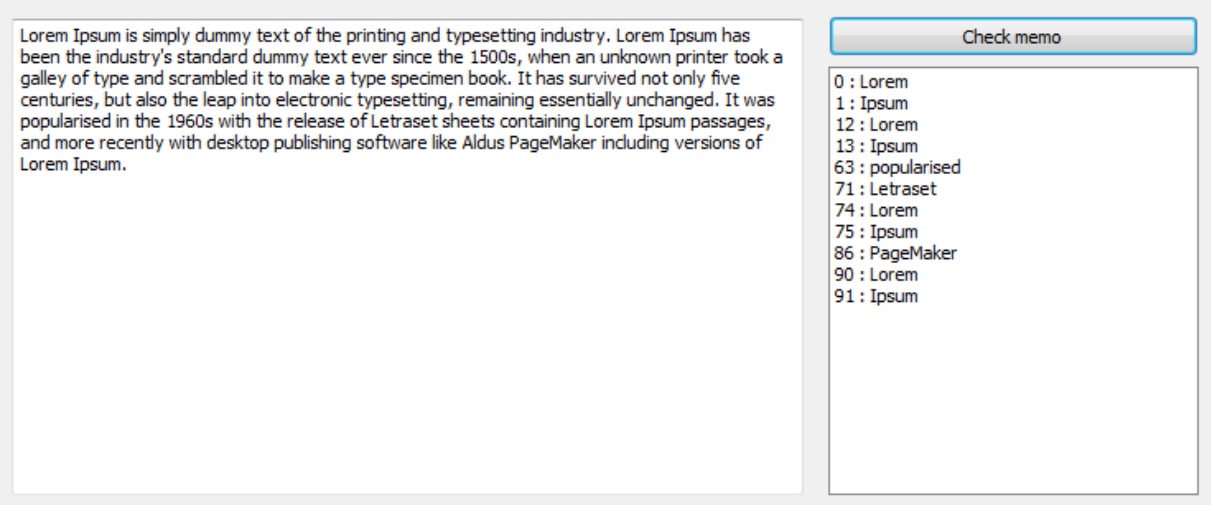
```
procedure TForm1.Button1Click(Sender: TObject);
var
  WordList: TStringList;
  i: integer;
  id: string;
  guid: TGUID;
begin
  WordList := TStringList.Create;
  try
    CreateGUID(guid);
    id := GUIDToString(guid);
    if ParseStringToWords(memo1.Lines.Text, WordList) > 0 then
    begin
      TMSFMXSpellCheck1.BeginRequest(id);
      for i := 0 to WordList.Count - 1 do
```

```
TMSFMXSpellCheck1.AddValidationRequest(WordList.Strings[i],TObject(i),
SpellCallback);
      TMSFMXSpellCheck1.EndRequest;
    end;
  finally
    WordList.Free;
  end;
end;
```

The callback is defined as:

```
procedure TForm1.SpellCallback(Sender: TObject;
  CallBackContext: TTMSFMXSpellCheckCallbackContext);
begin
  if callbackcontext.ValidationResult = wvrNotValidated then
  begin
    listbox1.items.Add(inttostr(integer(callbackcontext.Data))+' : '+
callbackcontext.OriginalRequest);
  end;
end;
```

The callback in this case does not much more than list the words found with incorrect spelling and the index of the word.



Ignore list

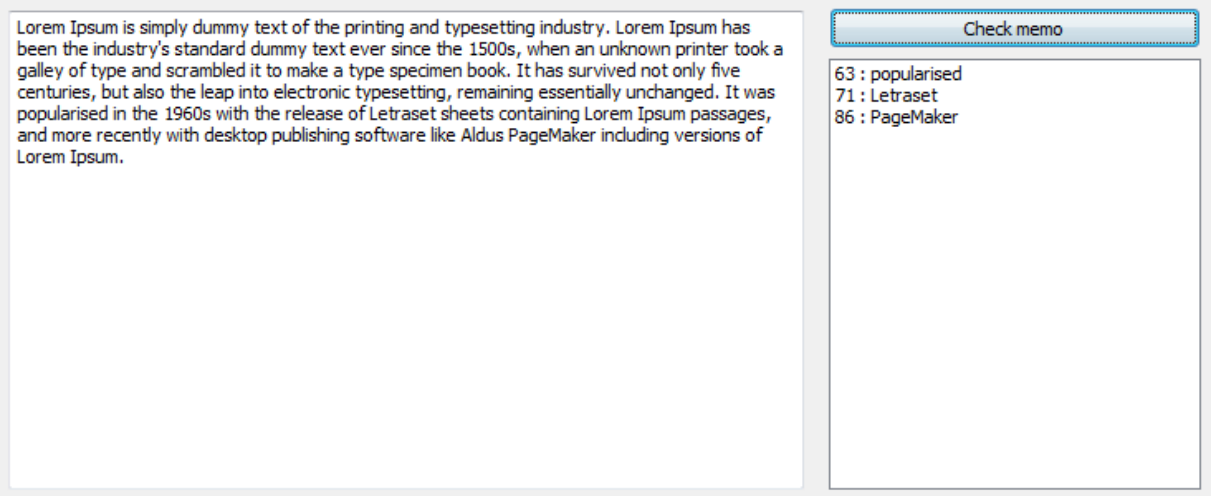
Adding words to the ignore list is easy. The method TTMSFMXSpellCheck.AddToIgnoreList can be used for this. With default setting TTMSFMXSpellCheck.StoreElements.sseIgnoreList = true, this also means that once a word is added to the ignore list, it is persisted.

Example code:

With this code executed before doing the spell check on the Lorem Ipsum document:

```
TMSFMXSpellCheck1.AddToIgnoreList('Lorem');
TMSFMXSpellCheck1.AddToIgnoreList('Ipsum');
```

the result is:



At any time, a word can be removed from the ignore list again with TTMSFMXSpellCheck.RemoveFromIgnoreList().

Adding words to the dictionary

Just like you can add words to the ignore list, it is possible to programmatically add words to the dictionary. This is done with the method TMSFMXSpellCheck.AddToDictionary(). The first parameter is the language specifier, the 2nd parameter is the list of words to add to the dictionary. This can contain multiple words when separated by a linebreak.

To add the word to the active language, use:

```
  TMSFMXSpellCheck1.AddToDictionary(TMSFMXSpellCheck1.ActiveLanguage,
'tmssoftware');
```

Similar to the ignore list, words can at any time also be programmatically removed from the dictionary with the method TTMSFMXSpellCheck.RemoveFromDictionary().

Working with multiple languages

TTMSFMXSpellCheck has built-in support to work with multiple languages. The dictionaries for multiple languages are configured via the TTMSFMXSpellCheck.Languages collection. Each language dictionary, affixes, ignorelist is an entry in this collection. The languages collection is edited via the TMSFMXSpellCheck configuration dialog (that is also accessible at runtime) or via the TMSFMXSpellCheck.Languages collection editor.

Each language entry is of the type TTMSFMXSpellCheckLanguagePack and has following properties:

AffixFileName	Sets the file name of the affix file for this language
Description	A description text for the language that is shown in the language picker component
Enabled	When true, this language is enabled for selection
Guid	Unique identifier of the language (this is an auto-generated unique value)
LanguageCode	Sets the international code name for the language
SoundExName	Name of the soundex to use
SoundExProcess	Selects the soundex algorithm variant for detection of similar words
SourceFileName	Sets the filename of the word list
Words	Collection of words in the dictionary defined outside the database. Can be used to programmatically add words to the dictionary that are not in the database.

When multiple languages are used, the language used to perform the spell check against is set with TTMSFMXSpellCheck.ActiveLanguageIndex: integer and this is simply the index of the language in the TTMSFMXSpellCheck.Languages collection.

Using the predefined spell check user interface controls

TMS Spell Check for FireMonkey comes with following predefined user interface controls:

TTMSFMXSpellCheckConfigDialog	dialog component for runtime configuration of the spell check engine
TTMSFMXSpellCheckLanguageSelectDialog	dialog component for runtime active language selection
TTMSFMXSpellCheckCorrectPanel	panel component offering the UI to do per word spell correction
TTMSFMXSpellCheckCorrectDialog	dialog with panel component offering the UI to do per word spell correction
TTMSFMXSpellCheckCorrectLinesPanel	panel component offering the UI to do per sentence spell correction
TTMSFMXSpellCheckCorrectLinesDialog	dialog with panel component offering the UI to do per sentence spell correction

TTMSFMXSpellCheckConfigDialog

Drop the component on the form, assign a TTMSFMXSpellCheck instance to TTMSFMXSpellCheckConfigDialog.SpellCheck
The dialog is shown at runtime by calling TTMSFMXSpellCheckConfigDialog.Execute.

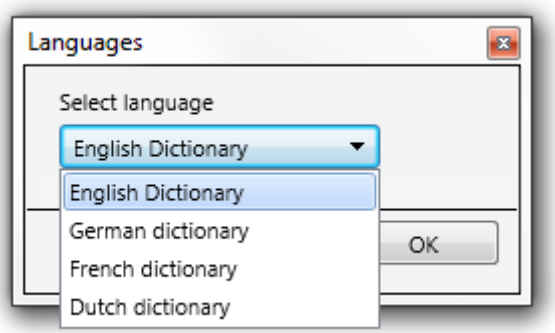


This brings up the configuration dialog for the language dictionaries in the spell check engine. From here the user can add & remove languages via setting or removing word dictionaries, affix files and

optionally ignore lists.

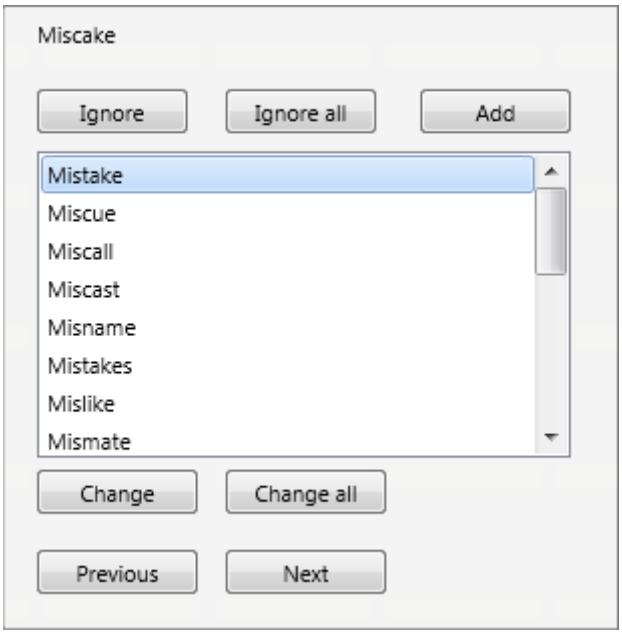
TTMSFMXSpellCheckLanguageSelectionDialog

Drop TTMSFMXSpellCheckLanguageSelectionDialog on the form, assign a TMSFMXSpellCheck instance to TTMSFMXSpellCheckLanguageSelectionDialog.SpellCheck and call TTMSFMXSpellCheckLanguageSelectionDialog.Execute. This brings up the dialog to allow to select from the active and enabled languages in the spell check engine:



TTMSFMXSpellCheckCorrectPanel

TTMSFMXSpellCheckCorrectPanel offers a panel with all common actions for performing spell check correction word by word. It contains the actions: Ignore, Ignore all, Add, Change, Change all and also Previous Error and Next Error.



The actual word to be corrected is retrieved via the event OnGetErrorWord. Return via this event the word to be corrected. For each button on the panel there is a corresponding event:

OnCorrectWord	event triggered with incorrect word and suggested word selected by the user.
OnCorrectWordAll	event triggered with incorrect word and suggested word selected by the user.

OnAddWord	event triggered with the word to be added to the dictionary
OnIgnoreWord	event triggered with the word that was added to the ignore list
OnIgnoreWordAll	event triggered with the word that was added to the ignore list
OnNextError	event triggered when button to go to the next error was triggered
OnPreviousError	event triggered when the button to go to the previous error was triggered

Example implementation:

A global word list of the type TStringList is created that will hold all found incorrect words and a global wordindex variable is used to track the word to be corrected. This incorrect word list is asynchronously retrieved via:

```
// routine that performs the spell check
procedure TForm1.SpellCheckClick(Sender: TObject);
var
  i: integer;
  id: string;
  guid: TGUID;
begin
  wordindex := 0;  // initialize index to first incorrect word
  CreateGUID(guid);
  id := GUIDToString(guid);
  if ParseStringToWords(memo1.Lines.Text, WordList) > 0 then
  begin
    TMSFMXSpellCheck1.BeginRequest(id);
    for i := 0 to WordList.Count - 1 do

TMSFMXSpellCheck1.AddValidattionRequest(WordList.Strings[i],TObject(i),
SpellCallback);
    TMSFMXSpellCheck1.EndRequest;
  end;
end;
// event triggered when the spell check is complete and initializes the
correction panel
procedure TForm5.TMSFMXSpellCheck1RequestsProcessed(Sender: TObject;
  Context: TProcessRequestContext);
begin
  TMSFMXSpellCheckCorrectPanel1.DoUpdate;
end;

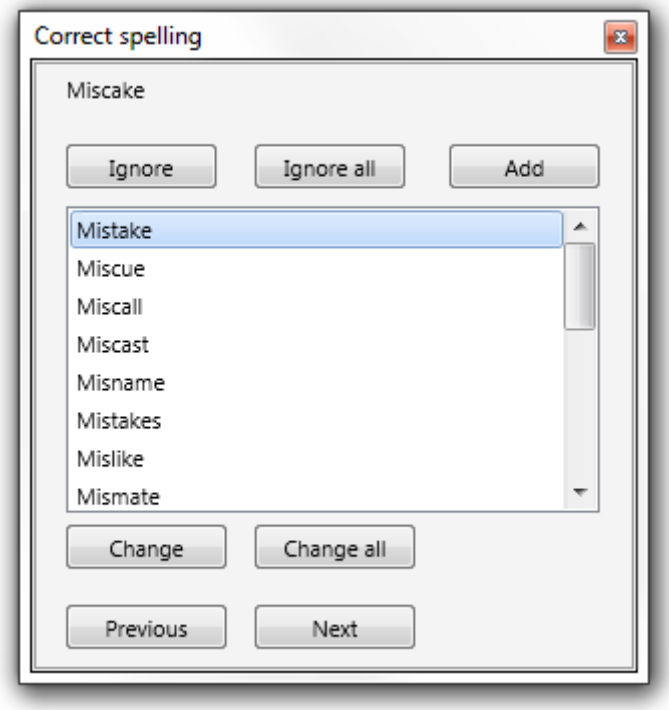
// event triggered from the panel to request the actual word to be
// corrected
procedure TForm1.TMSFMXSpellCheckCorrectPanel1GetErrorWord(Sender: TObject;
  var ErrorWord: string);
begin
  ErrorWord := WordList.Strings[wordindex];
end;

// event handler for the button to move to the next incorrect word
procedure TForm5.TMSFMXSpellCheckCorrectPanel1NextError(Sender: TObject);
begin
  if wordindex < WordList.Count - 1 then
    inc(wordindex);
end;
// event handler for the button to move to the previous incorrect word
```

```
procedure TForm1.TMSFMXSpellCheckCorrectPanel1PreviousError(Sender:
TObject);
begin
  if wordindex > 0 then
    dec(wordindex);
end;
```

TTMSFMXSpellCheckCorrectDialog

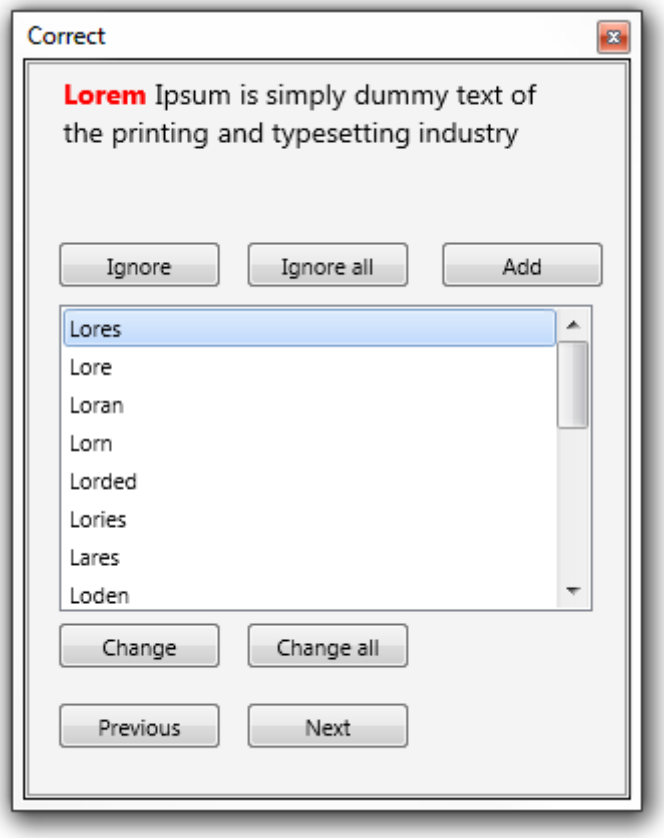
TTMSFMXSpellCheckCorrectDialog works exactly the same way as a TTMSFMXSpellCheckCorrectPanel except that it presents itself as a dialog. It exposes exactly the same events as TTMSFMXSpellCheckCorrectPanel to interact with the word list to be corrected.



Spell check correction dialog

TTMSFMXSpellCheckCorrectLinesPanel

TTMSFMXSpellCheckCorrectLinesPanel operates directly on a sentence, parses the sentence word by word, shows errors and allows to correct these errors one by one. The sentence is set via TTMSFMXSpellCheckCorrectLinesPanel.Init(ASentence: string);



When the user went through the corrections and no more errors are in the text, the OnSpellCheckComplete event is triggered:

```
procedure TForm1.TMSFMXSpellCheckCorrectLinesPanel1SpellCheckComplete(
  Sender: TObject; OriginalText, CorrectedText: string);
begin
  ShowMessage('Corrected:' + OriginalText + #13#10' to '#13#10 +
CorrectedText);
end;
```

TTMSFMXSpellCheckCorrectLinesPanel triggers events to inform the application of any action that happens on the panel during correction of the sentence, similar to TTMSFMXSpellCheckCorrectPanel.

TTMSFMXSpellCheckCorrectLinesDialog

TTMSFMXSpellCheckCorrectLinesDialog is the equivalent of TTMSFMXSpellCheckCorrectLinesPanel but presents itself via a dialog. The dialog is invoked via the TTMSFMXSpellCheckCorrectLinesPanel.Execute(var ASentence: string): TModalResult;

The corrected sentence is as such returned via the var parameter ASentence.

Example implementation:

```
var
 s:string;
begin
```

```
  s := 'Lorem Ipsum is simply dummy text of the printing and typesetting
industry';
  if TMSFMXSpellCheckCorrectLinesDialog1.Execute(s) = mrOK then
    ShowMessage('Corrected:'+s);
end;
```

Spell check events

OnBeforeAddToIgnoreWord, OnAfterAddToIgnoreWord

Events triggered before and after a word is added to the ignore list

OnBeforeRemoveFromIgnoreWord, OnAfterRemoveFromIgnoreWord

Events triggered before and after a word is removed from the ignore list.

OnBeforeAppendWordsToDictionary, OnAfterAppendWordsToDictionary

Events triggered before and after a word is added to the dictionary

OnBeforeBeginRequest, OnAfterBeginRequest

Events triggered before and after an asynchronous validation request is started

OnBeforeCleanupDictionary, OnAfterCleanupDictionary

Events triggers before and after a database is rebuilt

OnBeforeClose, OnAfterClose, OnBeforeOpen, OnAfterOpen

Events triggered before and after the spell check engine is activated (open) or desactivated (closed)

OnBeforeEndRequest , OnAfterEndRequest

Events triggered before and after the asynchronous validation request is closed

OnBeforeGetSuggestions, OnAfterGetSuggestions

Events triggered before and after a request for suggestions

OnBeforeGetValidation, OnAfterGetValidation

Events triggered before and after a request for validation

OnBeforeLoad, OnAfterLoad
OnBeforeLoadConfig, OnAfterLoadConfig
OnBeforeLoadDB, OnAfterLoadDB
OnBeforeLoadIgnoreList, OnAfterLoadIgnoreList
OnBeforeLoadIgnoreListText , OnAfterLoadIgnoreListText
OnBeforeSave, OnAfterSave
OnBeforeSaveConfig, OnAfterSaveConfig
OnBeforeSaveDB, OnAfterSaveDB
OnBeforeSaveIgnoreList, OnAfterSaveIgnoreList

Events triggered before and after loading or saving various databases.
Persisted spell check data consists of dictionaries, ignorelist, configuration settings of languages.
For each of these databases, events are triggered. OnBeforeLoad, OnAfterLoad, OnBeforeSave,

OnAfterSave apply to loading or saving the language configuration, ignorelists, language configuration.
OnBeforeLoadDB, OnAfterLoadDB, OnBeforeSaveDB, OnAfterSaveDB apply to saving and loading the dictionaries only.
OnBeforeLoadConfig, OnAfterLoadConfig, OnBeforeSaveConfig, OnAfterSaveConfig apply to saving and loading the language configuration only.
OnBeforeLoadIgnoreList, OnAfterLoadIgnoreList, OnBeforeSaveIgnoreList, OnAfterSaveIgnoreList apply to saving and loading the ignore list data only.
OnBeforeRefreshDictionary , OnAfterRefreshDictionary
Events triggered before and after a database is refreshed via the method RefreshDatabase.

OnRequestGroupResult

Event triggered when a result for a group of asynchronous validation requests is handled. Information about the validation request group is communicated via the TTMSFMXSpellcheckRequestGroup parameter.

OnRequestResult

Event triggered when an asynchronous validation request is handled. Information about the validation request is communicated via the TTMSFMXSpellCheckCallbackContext parameter.

OnRequestsProcessed

Event triggered when a series of asynchronous validation requests are completely handled.

Spell asynchronous handling and callback context

TTMSFMXSpellCheck can handle validation and suggestion requests asynchronously. Validation requests are asynchronously started with:

TTMSFMXSpellCheck.AddValidationRequest()

The full parameter list for AddValidationRequest is:

```
procedure TTMSFMXSpellCheck.AddValidationRequest(
  Word: string;
  Data: TSPObject;
  Callback: TTMSFMXSpellCheckRequestCallBack;
  WordLocation: TTMSFMXWordLocation;
  ValidationOptions: TTMSFMXWordCorrection);
```

Word: The word to be validated
Data: Custom extra data to pass along with the validation request that will be returned in the callback.
CallBack: Callback handler procedure reference
WordLocation: TTMSFMXWordLocation: defines whether the word is in the beginning, middle or end of a sentence. TTMSFMXWordLocation is defined as: TTMSFMXWordLocation = (wlStart, wlMiddle, wlEnd);
ValidationOptions: This is a set of extra options of the type TTMSFMXWordCorrections. TTMSFMXWordCorrections is defined as TTMSFMXWordCorrections = (wlcStartWithCapitalWords, wlcCaseInsensitive, wlcAllCapitalWords); This defines whether the validation should be handled with taking case in account or not, deal with start case of words in the beginning of a sentence or accept words in full capital letters too.

Requests for getting suggestions are asynchronously started with:

TTMSFMXSpellCheck.AddSuggestionRequest()

The full parameter list for AddSuggestionRequest is:

```
procedure TTMSFMXSpellCheck.AddSuggestionRequest(
  Word: string;
  Data: TSPObject;
  Callback: TTMSFMXSpellCheckRequestCallBack;
  SameCapsSuggestion: boolean);
```

Word: The word for which to get suggestions
Data: Custom extra data to pass along with the suggestions request that will be returned in the callback.
CallBack: Callback handler procedure reference
SameCapsSuggestion: When true, the suggestions will have the same caps as the original word.

The callback for both asynchronous requests has the following parameter list:

SpellCallback(Sender: TObject; CallBackContext: TTMSFMXSpellCheckCallbackContext);

The CallBackContext contains all the information about the original asynchronous started request. It is defined as:

```
 TTMSFMXSpellCheckCallbackContext = class
   property SameCaseSuggestions: boolean;
   property ValidationOptions: TTMSFMXWordCorrection;
   property ValidationResult: TTMSFMXWordValidationResult;
   property WordLocation: TTMSFMXWordLocation;
   property RequestType: TTMSFMXResultTypes;
   property OriginalRequest: String;
   property Data: TSPObject;
   property BooleanResult: boolean;
   property StringResult: String;
end;
```

The meaning of the properties is:

SameCaseSuggestions: setting as defined at the time a suggestion request was done with respect to handling of case for the suggestion.
ValidationOptions: setting as defined at the time a validation request was done
ValidationResult: contains the result of a validation request. The result types are:

- wvrValidated: word was validated
- wvrNotValidated: word was not validated
- wvrAttentionRequired: word was validated but has potential case issues

WordLocation: word location parameter as passed to the validation request
RequestType: indicates whether the callback was for a validation or suggestions request: rtValidation, rtSuggestions
OriginalRequest: Original word passed to the request
Data: Custom extra data passed to the request
BooleanResult: Simplified Boolean result
StringResult: linebreak separated list of suggested words in case of a suggestion request

Note that the callback is called for every request, irrespective of the result. That means that when memory was allocated or an object was created to pass along with the request, it can be freed when the callback is called.

Example:

In the case we want to pass the text selection offset and selection length of the word to be validated as information with the validation request and get that information in the callback, for example, to mark the word as error in a control, this can be done in following way:

The object that will be used to pass along this extra information could be:

```
TMemoHighlight = class
  ss: integer;
  sl: integer;
end;

procedure TMemoSpellCheck.CheckDocument;
var
  s: string;
  hl: TMemoHighlight;
begin
  CancelAllRequests('1');
  BeginRequest('1');

  s := Memo.CheckFirstWord;
  repeat
   hl := TMemoHighlight.Create;
   hl.ss := Memo.GetSelStart(Memo.CheckWord);
```

```
    hl.sl := Memo.GetSelLength(Memo.CheckWord);

    AddValidationRequest(s, hl, SpellCheckCallBack);

    s := Memo.CheckNextWord;
  until s = '';

  EndRequest;
end;


procedure TMemoSpellCheck.SpellCheckCallback(Sender: TObject;
  CallBackContext: TTMSFMXSpellCheckCallbackContext);
var
  hl: TMemoHighlight;
begin
  if CallBackContext.ValidationResult = wvrNotValidated then
  begin
    hl := TMemoHighlight(Callbackcontext.Data);
    Memo.SelStart := hl.ss;
    Memo.SelLength := hl.sl;
    Memo.MarkSelectionAsError;
    Memo.ClearSelection;
  end;
  TMemoHighlight(Callbackcontext.Data).Free;
end;
```

Spell check database files

The database filename where the dictionaries and settings are persisted is set with TTMSFMXSpellCheck.DatabaseFileName and defaults to TMSSPELLCHECK. This setting assumes the file TMSSPELLCHECK.SPLX is in the same folder where the application executable is but a full path can be specified for the DatabaseFileName as well.

By default, the word dictionary and ignore-list are both persisted in the database. This is controlled with the setting TTMSFMXSpellCheck.StoreElements where default sseSpellCheckDB and sseIgnoreList are set. This means the settings for each configured language as well as ignorelist will all be persisted in TMSSPELLCHECK.SPLX.

Language configuration can be saved separately. Default filename is TTMSFMXSpellCheck.DatabaFileName.SPLCFG. The language configuration can be saved or loaded separately programmatically with TTMSFMXSpellCheck.SaveConfig() / TTMSFMXSpellCheck.LoadConfig() or can be done via the configuration dialog.

The dictionary databases can also be saved separately. The default filename is TTMSFMXSpellCheck.DatabaFileName.SPL. The dictionary can be saved or loaded separately programmatically with TTMSFMXSpellCheck.SaveToFile() / TTMSFMXSpellCheck.LoadFromFile() or can be done via the configuration dialog.