Internationalization API for .NET and Delphi

Plurals, *genders*, *abbreviated numbers*, *ordinal numbers*, *runtime language change*, *overlap and truncation checking* and many more.

This project implements collection of I18N APIs for .NET and Delphi. Each API adds additional features to the standard I18N API of the platform. For example support for grammatical numbers, grammatical genders, abbreviated numbers and ordinal numbers. Library also includes APIs to perform runtime language switch of applications. The library also contains API that on runtime checks your user interface for issues such as truncated or overlapped controls. Finally the Delphi library contains a proper localization resource for FireMonkey.

Implementation

Each API is 100% native API. It means that it contains full source code and requires no additional files. .NET API contains only C# code and requires no other library or data files. In the same way Delphi API contains only Delphi code. The rules used by the code have been extracted from CLDR into source code files that are part of the API source code. You don't need ICU library or CLDR XML files but everything including logic and rules are compiled into your application file.

Grammatical numbers and genders

Most resource formats (e.g. .resx in .NET, .properties in Java, and resource strings in Delphi) only support plain strings. Grammatical numbers and genders require structural data where there are several language depend variants for a single string. For example in English "I have N cars" would require two variants:

Grammatical number	.NET	Delphi and Java
singular	I have {0} car	I have %d car
plural	I have {0} cars	I have %d cars

Some platforms such as Android and Angular have a built-in support for plurals and that is why they also have a special resource format to hold multiple patterns. However .NET, Delphi and Java do not have such a resource format. This library uses ICU message format to store all related patterns into a standard resource string. The syntax is:

`{parameter, kind, form1 {pattern1}[ form2 {pattern2}]...[ formN {patternN}]`

where

`kind` is either `plural`, `gender` or `select`

`form` is the code for grammatical number form or for grammatical gender form.

The following table contains possible forms.

Form	Used with	Description
zero	Grammatical numbers	Nullar
one	Grammatical numbers	Singular
two	Grammatical numbers	Dual
few	Grammatical numbers	Paucal, trial or similar
many	Grammatical numbers	Greater paucal or similar
other	Grammatical numbers Grammatical genders	Plural Neutral
male	Grammatical genders	Male
female	Grammatical genders	Female
neutral	Grammatical genders	Neutral. Same as other.

In addition of the above forms you can use operator with grammatical numbers. The operators are:

Form	Example	Description
=n	=1	Equal
~n	~12	Around
>n	>5	Greater than
<n	<10	Less than
>=n	>=5	Greater or equal to
<=n	<=10	Less or equal to
n..m	2..6	Range

The car sample for .NET would be:

```
{cars, plural, one {I have {0} car} other {I have {0} cars}}
```

Finnish version would be

```
{cars, plural, one {Minulla on {0} auto} other {Minulla on {0} autoa}}
```

Japanese version would be

```
{cars, plural, other {{0}車持っています}}
```

Japanese has only one form, universal, so the string contains only other pattern.

The pattern can contain text before and after the multi pattern syntax. The following example contains "I have " before the multi pattern and "." after the multi pattern.

```
I have {cars, plural, one {{0} car} other {{0} cars}}.
```

For Delphi the sample it would be

```
I have {cars, plural, one {%d car} other {%d cars}}.
```

If your pattern need to have braces ({ or }) they have to be escaped using \. \ character must also be escaped. For example *"I like skiing {alpine\s} however, I only have {0} pairs of skis"* would be

```
other {I like skiing \{alpine\\s\} however, I only have {0} pairs of skis}
```

See samples in `Samples\Delphi\VCL\Patterns`, `Samples\Delphi\FMX\Patterns`, `Samples\WindowsForms\Patterns` and `Samples\WPF\Patterns` directories.

The classes also support the legacy multi pattern format

```
one;I have {0} car;other;I have {0} cars
```

However support for this legacy format has been deprecated. The ICU message format is the recommended format.

Multiple placeholders

The API supports multiple plural/gender enabled placeholders. In that case you have split your string into parts each containing one placeholder and then chain the segments into a logical sentence. Use `next` form to start a new part. For example if I want to say "I have `c` cars and `s` skis" you will create the following string that contains two parts: one for car and another for ski. Each part contains two patterns: singular and plural.

.NET

```
I have {cars, plural, one {{0} car} other {{0} cars}} and {skis, plural, one {{0} ski} other {{0} skis}}.
```

The string contains two multi patterns: cars and skis.

Delphi

```
I have {cars, plural, one {%d car} other {%d cars}} and {skis, plural, one {%d ski} other {%d skis}}.
```

See `Samples\Delphi\VCL\Patterns\Multi`, `Samples\Delphi\FMX\Patterns\Multi`, `Samples\WindowsForms\Patterns\Multi` and `Samples\WPF\Patterns\Multi` samples.

Abbreviated numbers

If we need to show large numbers on screen it might be hard for user to easily understand magnitude of a number. For example if we have number 144563217 how should we show it. We could format it according to the rules of the locale of the user. For example in USA it would be 14,456,217. Although this is easier to understand than the unformatted number it still has few problems. The first one is that it requires some effort to understand the magnitude of the number. Secondly it requires quite a lot space. One solution is to round it like 14,000,000. It is easier to understand. To make is even easier to understand we can also abbreviate it like 14M. The abbreviated forms are getting more and more popular. Unfortunately each language as it own way to abbreviate. Most languages go by three digits. For example English: K, M, G, T, etc. However some Asian languages go by four digits. For example Japanese: 万 (10,000), 億 (100,000,000), etc.

CLDR contains rules to abbreviate numbers. The abbreviated number API uses those rules to format a number (interger or float) as a string. The result can either be long string such as 14 million, short as 14M, or a currency string such as $14M.

Ordinal numbers

CLDR does not contain information how to create ordinal numbers from a number. I have been collecting the rules from various sources. There are still many languages without a proper rule. A help from a native speaker would be appreciated.

Runtime language switch

Runtime language switch is a feature where the application can change the language of its user interface on runtime. This library contains code for that. The language change is implemented such way the there is not reload of forms or dialog and the current state of the application remains unchanged.

Runtime language switch APIs are for Delphi VCL, Delphi FireMonkey, .NET Windows Forms and .NET WPF.

User Interface Checker

When we translate strings to another language there is always possibility that the translation will be longer than the original string. This is especially true if the original language is English because English is more compact language as for example German or Finnish. When translated strings get longer there is a risk that part of the string gets truncated or two or more strings get overlapped. Situations like there are hard to find. User interface checker API helps. It is an API that you temporary link to your code and when you run the application is writes a report file with screenshots showing truncations and overlapped marked with clear colors. You can instantly see where a truncation or overlapping occurs.

User interface checker APIs are for Delphi VCL, Delphi FireMonkey, .NET Windows Forms and .NET WPF.

.NET

`Library\NET` contains the .NET API. `Library\NET\Standard` contains a .NET Standard library that contains API for grammatical numbers, grammatical genders, abbreviated numbers and ordinal numbers. Compile it and add that into your solution and finally add the library into the references of your project. Because the non UI part of the library is .NET Standard it works with Windows Forms, WPF, ASP.NET, .NET Core and Xamarin. You can also compile it using any .NET version starting with .NET 2.0.

Currently we have samples for Windows Forms and WPF. I will add ASP.NET and ASP.NET Core samples soon.

Delphi

`Library\Delphi` contains the Delphi API. Easiest way to include them into your application is to add the path to the search path of your project. Library supports Delphi 7 or later. However in order to get all features you need Delphi XE2 or later.

FireMonkey

FireMonkey does not have a proper localization resource. If your target mobile platforms such as iOS or Android you cannot use resource DLLs. Everything including the localized resource must be in the main application file. This library contains a solution for FireMonkey localization. It uses a special .ntres file format to store form (.fmx), string, image and audio resources into a single .ntres file. The file contains the resources in all the languages you want to support. Finally you add the .ntres file as a custom resource into your application and use library's API calls to access the resources. You don't have to modify your existing application that much in order to make is multilingual. Only exception are the resource strings. You cannot use the. If you have hard coded string you want to localize wrap it in _T function.

```
procedure TForm1.UpdateStrings;
begin
  Label1.Text := _T('Hello world');
end;
```

If you already have an existing resource string remove it and use the _T function. In addition you have to call _T for each form.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  _T(Self);
end;
```

See `Samples\Delphi\FMX\Patterns\Simple` sample first.

You can use .ntres resources also in VCL applications but it is recommended to use the standard VCL localization method with resource strings and resource DLLs or localized EXE files.

C++Builder

C++Builder is not officially supported. All these code should work with C++Builder but never been tested.

Other programming languages

I chose .NET and Delphi as the first platform for few reasons. First is that I love both Delphi and C#. Secondly both Delphi and .NET lack support for several I18N APIs such as plurals and genders. Third is that both Delphi and .NET would also benefit for runtime language switch. Fourth is that Delphi is used to create mobile application where the size matters. Fifth is that I believe that C# + .NET Core will have a brilliant future.

I am considering to implement similar classes for TypeScript (JavaScript comes in the process). However TypeScript is mostly used with Angular that already has a great support plurals and genders. I will probably implement only abbreviated numbers API for TypeScript.

I can implement similar libraries for Java. There is already ICU implementation for Java so if you use Java on server side you can use it. If you use Java on the client side it will most likely be Android and it supports plurals but not genders. I am still considering about Java support. Let me know if it would be useful.

I also would like to see other programming languages such as PHP, Python and Ruby. Because I am not that experienced PHP or Python programmer and I have never used Ruby it would be nice if somebody would implement PHP, Python and Ruby libraries and samples.

CldrToCode.exe

`Bin\CldrToCode.exe` is a tool that extracts rules from the CLDR XML files to C# and Delphi files. Normally you don't have to use it because the library already contains extracted files that contain rules in all CLDR languages. However if you want to create rule files that contain only some languages you can use `CldrToCode.exe` to create your own rules files. For example if you want to create rule files that only contains English, German and French use:

`CldrToCode.exe -lang:en;de;fr D:\CLDR\common`

This will create `NtPluralData.pas`, `NtNumberData.pas`, `PluralData.cs` and `NumberData.cs`.

With little effort I can make CldrToCode.exe to create Java, TypeScript and Python files too.

Localization

If you plan to localize your application using multiple patterns strings you better use a localization tool that has support for grammatical numbers and grammatical genders. In the second half of 2018 a new localization tool will be released. It has an excellent support for multiple patterns and it supports ASP.NET, .NET, Delphi, Java, Angular and many more.

In addition it supports continuous localization, machine translation, interactive fuzzy matching enabled translation memory, interactive terminology, import/export, build tools and cloud translation.