VAT Registration Number Validator

Assembly Version 4.0.0.0

Reference Manual

Table of Contents

License Agreement – The MIT Open Source License 2
A note to the user 2
Installing the VAT Validator DLL 3
Classes 3
Enumerators 3
Exceptions 4
Methods 5
Constructors 5
Language Related Methods 5
Member State Related Methods 6
Data Retrieval Methods 7
Data Presentation Methods 8
Miscellaneous Methods 8
Using the VAT Validator Class Example 9
Conclusion and Final Remarks 10

Miroslav Bonchev Bonchev

MBBSoftware

June, 2015

License Agreement – The MIT Open Source License

{Your website/product} uses the VAT Registration Number Validator by <link-to>mbbsoftware.com</link>.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A note to the user

Dear User of the VAT Registration Number Validator,

Thank you for obtaining a copy of this software and using it in your website(s) or other product(s).

Since the above notice is included in the asp.net DLL assemblies and the C# code, it will not be visible by your website visitors or software users. We would like to kindly ask you to help us by adding a link to our website on your website, perhaps somewhere on one of your disclaimers or legal pages. For example, the following or similar link would be much appreciated to be added to your site:

{Your website/product} uses the VAT Registration Number Validator by <link-to>mbbsoftware.com</link>.

This will greatly assist our visibility to search engines and will help us to develop more technology to benefit the community of people using the Internet and the Internet as a whole, as well as helping us to make updates to the VAT Registration Number Validator as needed.

We thank you again for using the VAT Registration Number Validator, and especially if you will be so kind as to place a link on your website to http://www.mbbsoftware.com. We wish you success with your website(s), product(s), and endeavors.

Kind regards,
MBBSoftware

Installing the VAT Validator DLL

Read the license agreement of the VAT Validator software above, also found in the "VAT-Validator.cs" and "Licence Agreement.txt" files. By using this documentation and software you hereby accept the VAT Validator license agreement. If you do not agree with it, stop using the VAT Validator software and documentation now.

For websites copy the appropriate VAT Number Validator DLL into the Bin folder of your web application, or alternatively copy the "VAT-Validator.cs" file into the App_Code folder of your web application.
For application software use the required VAT Number Validator DLL or "VAT-Validator.cs" in your project as desired.

Classes

Create an instance of the `VATValidator` class and use it to validate VAT numbers as described in this document.

Enumerators

The following public enumerators of the `VATValidator` are defined:

```
public enum Language
{
   Bulgarian,  // bg
   Czech,      // cs
   Dansk,      // da
   Deutch,     // de
   Greek,      // el
   English,    // en
   Español,    // es
   Estonian,   // et
   Suomea,     // fi
   Français,   // fr
   Hungarian,  // hu
   Italiano,   // it
   Lithuanian, // lt
   Latvian,    // lv
   Maltese,    // mt
   Nederlands, // nl
   Polish,     // pl
   Português,  // pt
   Romanian,   // ro
   Slovak,     // sk
   Slovenian,  // sl
   Svenska     // sv
};
```

```
public enum MemberState
{
   Austria,          // AT
   Belgium,          // BE
   Bulgaria,         // BG
   Croatia,          // HR
   Cyprus,           // CY
   CzechRepublic,    // CZ
   Germany,          // DE
   Denmark,          // DK
   Estonia,          // EE
   Greece,           // EL
   Spain,            // ES
   Finland,          // FI
   France,           // FR
   UnitedKingdom,    // GB
   Hungary,          // HU
   Ireland,          // IE
   Italy,            // IT
   Lithuania,        // LT
   Luxembourg,       // LU
   Latvia,           // LV
   Malta,            // MT
   TheNetherlands,   // NL
   Poland,           // PL
   Portugal,         // PT
   Romania,          // RO
   Sweden,           // SE
   Slovenia,         // SI
   Slovakia          // SK
};
```

```
public enum ItemId
{
   Status            = 0,
   MemberState       = 1,
   VatNumber         = 2,
   TimeRequest       = 3,
   Name              = 4,
   Address           = 5,
   ConsultationNumber = 6
}
```

Exceptions

The **VATValidator** class throws the exceptions listed in this chapter. It is important that functions throwing exceptions are enclosed in appropriate try-catch block(s) in order to handle any possible errors, and maintain a high level of user experience.

InvalidParameter - This exception is thrown when invalid data is supplied to the methods of the VAT-Validator class.

ServerException - This exception is thrown only by the **bool IsValidVAT(parameters)** methods when they are unable to connect to the European VAT Number Information Exchange System (VIES) database.

ParsingException – This exception is thrown only by the **bool IsValidVAT(parameters)** methods when they are unable to parse the VIES database response data.

ResponseException – This exception is thrown only by the **bool IsValidVAT(parameters)** methods when they are unable to interpret the licensing information of the software. This exception would typically indicate attempts to bypass the licensing system or a need to upgrade the software. Note that after the issues becomes known to the software it will continue to work for up to three weeks reporting the issue through the **WebMasterMessage** output variable, before the exception is allowed to propagate.

Exception – This exception may be thrown by the .NET runtime and some internal for the **VATValidator** class functions. It is important that any function which is designated as throwing exception is eventually enclosed by **try-catch(Exception)** block to ensure that no exception leeks out and remains improperly handled.

Besides the appropriate handling of the exception errors in the code, it is good practice to also send an exception/error notification to the webmaster/administrator so they can take any additional measures if necessary.

Methods

Constructors

```
public VATValidator();
```

Synopsis:

Default constructor – use this constructor to create a VAT-Validator object which will not retrieve a consultation reference number.

Parameters:

None.

Exceptions:

Does not throw exceptions.

```
public VATValidator( MemberState stateRequester, string vatRequester );
```

Synopsis:

This constructor creates a VAT-Validator object which will request a consultation reference number when inquiring about a VAT number. The VAT number that is supplied through the parameters must be your own VAT number.

Parameters:

`MemberState` **`stateRequester`** – identifier of the member state which issued your VAT number.
`string` **`vatRequester`** – your VAT number, without member state prefix.

Exceptions:

Does not throw exceptions.

Language Related Methods

```
public static string GetLanguage( Language language );
```

Synopsis:

Static function which returns the two character European language code as a string, which are always in **lower case**.

Parameters:

`Language` **`language`** – identifier of the European language whose code is required.

Exceptions:

Throws `InvalidParameter` exception if the supplied language is invalid/unrecognized.

```
public static Language GetLanguage( string languageCode );
```

Synopsis:

Static function which returns the European language identifier from its two letter code.

Parameters:

string languageCode – European language code, e.g. “en” for English. The two character language codes are given in the **Language** enumerator in the Enumerators section.

Exceptions:

Throws **InvalidParameter** exception if the supplied language code is invalid/unrecognized.

```
public static string[] GetLanguages();
```

Synopsis:

Static function which returns an array of European language codes in alphabetical order. These language codes can be used immediately in the methods of the **VATValidator** class which require language code as a two character string. This function can be useful to populate the Value properties of Language selection combo-boxes.

Parameters:

None.

Exceptions:

None.

Member State Related Methods

```
public static string GetMemberState( MemberState state );
```

Synopsis:

Static function which returns the two character member state code as string. Member state codes are always in upper case.

Parameters:

MemberState state – identifier of the European Union member state whose code is required.

Exceptions:

Throws **InvalidParameter** exception if the supplied state is invalid/unrecognized.

```
public static MemberState GetMemberState( string stateCode );
```

Synopsis:

Static function which returns the European Union member state identifier from its two letter code.

Parameters:

string stateCode – European Union member state code, e.g. "AT" for Austria. The two character member state codes are given in the **MemberState** enumerator in the Enumerators section.

Exceptions:

Throws **InvalidParameter** exception if the supplied state code is invalid/unrecognized.

```
public static string[] GetMemberStates();
```

Synopsis:

Static function which returns an array of the European Union member state codes in alphabetical order. These state codes can be used immediately in the methods of the **VATValidator** class which requires state codes as two character strings. This function can be useful to populate the Value properties of member state selection combo-boxes.

Parameters:

None.

Exceptions:

None.

Data Retrieval Methods

```
public bool IsValidVAT( string strStateCode, string strVATNumber, string strLanguageCode,
                    ref Dictionary< DataItem, string > dictResults, ref string WebMasterMessage );

public bool IsValidVAT( MemberState eState, string strVATNumber, Language eLanguage,
                    ref Dictionary< DataItem, string > dictResults, ref string WebMasterMessage );
```

Synopsis:

These two methods return **TRUE** if the supplied VAT registration number is valid, and a **FALSE** if it is invalid or if the inquiry has failed, e.g. due to an error on the VIES database server.

To distinguish between an Invalid VAT Number and a failure to verify the supplied VAT number, use the **DataItem.Status** field from the **dictResults**. Invalid VAT numbers always begins with NO (in the requested language) followed by a coma, which is never the case for a failure to verify the VAT number. Thus the presence of a coma in the first 5 characters is a sufficient condition to distinguish between these two cases. This is shown in the examples section.

Parameters:

string strStateCode – Two letter member state code which issued the inquired about VAT number.
MemberState eState – member state which issued the inquired about VAT number.

string strVATNumber – inquired about VAT number, without member state prefix.

string strLanguageCode – two letter language code of the language in which the result should be acquired.
Language eLanguage – language in which the result should be acquired.

Dictionary< DataItem, string > dictResults – this dictionary contains all data returned by the VIES database, separated in respective **DataItem** fields. Use the data in this dictionary for your records, produce invoices and other needs.

string WebMasterMessage – This output data is used by the **VATValidator** class to communicate with the webmaster without throwing exceptions, thus allowing the software to work yet notify about conditions which might require administrator attention. The website using the script should always check that this string is empty, and if not it should pass the message to the webmaster/administrator for their consideration. Note that **WebMasterMessage** may be used to carry a message for the webmaster regardless if an exception is thrown or not.

Exceptions:

These methods may throw any of the **VATValidator** exceptions, including **Exception**.

Data Presentation Methods

```
public static string GetItemName( DataItem eItem, Language eLanguage );
```

Synopsis:

This method returns the name of the requested data item as text in the requested language.

Parameters:

DataItemDataItem eItem – identifier of the data item whose name as text is required.
Language eLanguage – language in which the returned item name is required.

Exceptions:

Throws **InvalidParameter** exception if any of the supplied parameters is invalid or unrecognized.

Miscellaneous Methods

```
public static List< DataItem > GetAllItems();
```

Synopsis:

This is an axillary helper method which returns a list containing the VAT registrant data items defined in the DataItem enumerator. This function is helpful to organize loops for utilizing the results from calls to the **bool IsValidVAT(parameters)** methods.

Parameters:

None.

Exceptions:

Does not throw exceptions.

Using the VAT Validator Class Example

Let the **ddlECCountry** and **ddlResponseLanguage** be two drop-down lists, populated with items to represent respective European Union Member Countries and Languages. Since in the example below, we will use the values of the selected list-items directly in the function calls, it is important that they are populated with valid European Union Member Country and Language Codes, which are shown in the `MemberState` and `Language` enumerators. Alternatively these drop-lists could be populated dynamically using:

```
public static string[] VATValidator.GetMemberStates();
public static string[] VATValidator.GetLanguages();
```

We will also assume that **textVATNumber** is a text box where the visitor will enter their VAT number, and that **checkObtainReference** is a check box which is used to indicate whether a Consultation Reference Number from the VIES database server is required or not. We will place the overall validation result in the **LabelVATValidationStatus** label field, and the obtained registrant data in two strings **AllItemsData** and **AllKnownItemsData**.

Then an eCommerce website using the VAT Registration Number Validator can use similar code to obtain the details of a VAT number.

--- Code Start ---

```
using MBBSoftware;


[some code to ensure that there is a valid selection in ddlECCountry]
[some code to ensure that there is a text (VAT number) entered in the textVATNumber]
[some code to ensure that there is a valid selection in ddlResponseLanguage]


// Dictionary which will be used to contain the VIES data for the queried VAT number.
Dictionary< VATValidator.DataItem, string > dictVATNumber = new Dictionary< VATValidator.DataItem, string >();


// Get the selected state and language in separate variables for example clarity. Note that both
// GetMemberState( param ) and GetLanguage( param ) throw exceptions when invalid arguments are supplied.
VATValidator.MemberState state    = VATValidator.GetMemberState( ddlECCountry.SelectedValue );
VATValidator.Language    language = VATValidator.GetLanguage( ddlResponseLanguage.SelectedValue );


// Variables for the results and any possible warnings that need to be send to the webmaster.
string AllItemsData = "";
string AllKnownItemsData = "";
string WebMasterMessage = "";


try
{
   // Now create a VAT validator instance which will be used to acquire the data. Use the default
   // constructor if you do not require a Consultation Reference Number. Use the parameters overload
   // constructor to supply your own VAT number as parameterized if you require a Consultation Reference Number.
   VATValidator validatorVAT = checkObtainReference.Checked ?
                               new VATValidator( VATValidator.MemberState.UnitedKingdom, "YourVATNnumber" ) :
                               new VATValidator();


   if( validatorVAT.IsValidVAT( state, textVATNumber.Text, language, ref dictVATNumber, ref WebMasterMessage ) )
   {
      // Valid VAT Number.
      LabelVATValidationStatus.Text = dictVATNumber[VATValidator.DataItem.Status];
```

```
      // Example: build a html string for all VAT validation data items.
      foreach( VATValidator.DataItem dataFileld in VATValidator.GetAllItems() )
      {
         AllItemsData += string.Format( "{0}: {1}<br />",
                                        VATValidator.GetItemName( dataFileld, VATValidator.Language.English ),
                                        dictVATNumber.ContainsKey( dataFileld ) ?
                                                     dictVATNumber[dataFileld] :
                                                     "[VAT-Validator: No Available Data]" );
      }


      // Example: build a html string only for the data items delivered through the request.
      foreach( KeyValuePair< VATValidator.DataItem, string > dataFileld in dictVATNumber )
      {
         // Ignore any empty or containing only "-" data items.
         if( !string.IsNullOrEmpty( dataFileld.Value.Trim( '-' ).Trim() ) )
         {
            AllKnownItemsData += string.Format( "{0}: {1}<br />",
                                        VATValidator.GetItemName( dataFileld.Key, VATValidator.Language.English ),
                                        dataFileld.Value );
         }
      }
   }
   else
   {
      // Invalid/Unconfirmed VAT Number. This message can be either "No, ..." for invalid VAT number or
      // ["Explanation what went wrong & why the VAT number could not be confirmed."]; use the coma as an indicator.
      LabelVATValidationStatus.Text = -1 != dictVATNumber[VATValidator.DataItem.Status].Substring(0,5).IndexOf(',') ?
                                          dictVATNumber[VATValidator.DataItem.Status] :
                                          ("The entered VAT number could not be validated. Response: " +
                                           dictVATNumber[VATValidator.DataItem.Status]);
   }
}
catch( Exception ex )
{
   LabelVATValidationStatus.Text = "The following exception occurred while trying to obtain the
requested VAT registration data: " + ex.Message;
}


if( !string.IsNullOrEmpty( WebMasterMessage ) )
{
   // There is a warning message from the VAT Registration Number Validator. Send an email to the webmaster.
   SendWarningNotification( "Warning from the VAT Registration Number Validator - [filename]",
                            string.Format( "Warning from the VAT-Validation assembly. Message: {0}",
                                           WebMasterMessage ) );
}
```

--- Code End ---

Conclusion and Final Remarks

Using the VAT Registration Number Validator is easy and straightforward. Be sure to use the proper exception handling to ensure smooth operation and best user experience. The different exceptions specialization allows for precise handling of any errors that might occur. Be sure to make use of any warnings that the software may issue via the **WebMasterMessage** variable and notify the webmaster about the possible problem. Naturally, the software will not be able to supply VAT data when the European Union database is not available or malfunctions. Be sure to always use the latest software, especially if the VIES database interface has been changed, and there are parsing exceptions thrown by the software. Should you need help and assistance please raise a support ticket at the http://www.mbbsoftware.com or contact support@mbbsoftwrae.com.

We have also developed a powerful all-in-one software suite called Act On File, which is designed for authentication, encryption, data compression, data comparison and five more data manipulation and utilization applications. You can use Act On File to add authentication and encryption keys to your website which will help to maintain safe online communication for your, and your customers benefit, as well as allow you to give and receive authenticated testimonials building the customer's trust in your website and business. For more information please visit http://www.mbbsoftware.com/Learning/Default.aspx.