Email Template Framework

Document Version 1.6

Software Version 2.5

Document Revision History

Version	Date	Notes
1.0	1/11/2005	Initial release.
1.1	10/12/2006	Updated to reflect release 2.0 of the library.
1.2	10/2/2007	Updated to reflect changes in 2.1 of the library
1.3	8/6/2008	Updated to reflect changes in 2.2
1.4	14/10/2009	Updated to reflect changes in 2.3
1.5	30/03/2010	Updated to reflect bug fix in 2.4
1.6	07/02/2011	Updated for the feature in 2.5

Software Release History

Version	Date	Notes
1.0	1/6/2005	Initial release.
1.1	11/7/2005	Subject line can now contain user data items.
1.2	18/10/2005	Able to now send HTML format emails, introduction of namespaces with the template.
2.0	10/12/2006	Code base updated to .Net 2.0 Framework. Bug fixes.
2.1	10/2/2007	Add the ability to authenticate with the SMTP used to send the email templates.
2.2	8/6/2008	Allows relative attachments in the email template. Minor code clean up, improvement to unit tests.
2.3	14/10/2009	Fix to allow correct parsing of version number for non English cultures.
2.4	30/03/2010	Fix to dispose of the MailMessage object after sending the email.
2.5	07/02/2011	Added the ReplyTo property to email message.

Overview

ASP.NET provides an excellent way to separate content from code, however when trying to send an email via ASP.NET there is no clean separation of the email message to send and the code that sends it.
Email Templates attempts to address this issue, the email message is composed as an XML document which is then processed by the engine and send to the recipient(s).

The framework provides a mechanism where parts of the email can be merged with data at run time, in effect providing a mail merge function. The merging of data can be simple name value pair, IEnumerable or IDictionary objects.

The email message itself can be either plain text or HTML format.

The library is built using C# .Net Framework 2.0 and released under a BSD license. With support for log4net logging of emails sent via the framework.

Template Files

The email message is composed using XML, this file contains the email message as well as metadata need to send the email such as to address, from address, priority etc. Also in the template are optional user data elements which are used to merge data into the template at runtime.

A simple email template file is shown below using HTML formatting. This template doesn't contain any user data elements.

```
<?xml version="1.0" encoding="utf-8" ?>
<template:emailTemplate application="TestApplication" version="1.2"
xmlns:user="http://www.bitethebullet.co.uk/EmailProcessor/UserData"
 xmlns:template="http://bitethebullet.co.uk/EmailProcessor/Templ
ate">
<template:name>Simple HTML Template</template:name>
<template:description>Template used to test
HTML</template:description>
<template:mailFormat>HtmlText</template:mailFormat>
<template:message>
 <template:to>user@dummy.com</template:to>
 <template:from>admin@dummy.com</template:from>
 <template:subject>TestEmail</template:subject>
 <template:body><b>Sample email</b> from email template
processor<br/><br/>
 template:body>
 <template:attachments>
 </template:attachments>
</template:message>
</template:emailTemplate>
```

This template is used to send a very simple HTML email message, an important point to note at this stage is that the email to address can be override and set at runtime with any number of receipts. Each receipt will not be aware of any other receipts of the email.

XML Elements Described

<template:emailTemplate>
This is the root element in the XML, here we store an important attribute as well as defining the two namespaces we need.

Attribute Name	Notes
application	User defined value, here you can enter the name of the application that uses the framework. This value is optional and used only for your own reference.
value	This defines the version of the email template, we use this to maintain backwards compatibility with older templates. This value is mandatory and should be set to 2.0

<template:name>
Optional value used to name the template, not used by the framework just included to make managing the templates easier for the developer.

<template:description>
Optional value used to describe the template and its purpose used for, this value is not used in the framework.

<template:mailFormat>
Defines the email format, this value should be either HtmlText or PlainText

<template:message>
Element used to nest the details of the email itself.

<template:to>
The recipient of the email message. Just leave blank if you intend to set this value at run time, which in most cases is true.

<template:from>
Defines the from email address, this will appear of the email that the recipient receives.

<template:replyTo>
Optional email address that is to indicate an address other than the *From* address to use to reply to this message.

<template:subject>
Defines the subject of the email message, this element can contain user data items.

<template:body>
Defines the body of the email message, this element can contain user data items.

<template:attachments>
Defines attachments that the email message will contain. Each attachment will be defined in the nest element <template:attachment> as shown below.

```
<template:attachements>
   <template:attachment>c:\filename.jpg</template:attachment>
</template:attachements>
```

New in version 2.2 of the library is the ability to define the attachment relative to the location of the template.

For example if the template was stored in c:\templates\ and the attachment was stored in c:\templateAttachments the following xml snippet would allow you to attach the file.

```
<template:attachements>
        <template:attachment>
                ../templateAttachments/someimage.jpg
        </template:attachment>
</template:attachements>
```

User Data Items

As already stated it is possible to merge data into the subject and body parts of an email at run time using user data items. This is a powerful function since it allows you to create a template which defines the content and at runtime the relevant data is then merged into the email before then sending the email.

During the process of sending the email, a Hashtable is passed to the email template processor which is then merged into the email template. The Hashtable holds the values that the user data items map, the mapping is based on the keys in the Hashtable mapping on to name attribute of the user data item.

- There are two areas where user data item can be used; in the subject and body element of the template.

- The framework has three different types of user data items, <userData>, <userDataList> and <userDataDictionary>. Only <userData> elements can be used in the subject line. The body element however can use any of the three user data items.

- All three user data items have the namespace http://www.bitethebullet.co.uk/EmailProcessor/UserData

User Data

This is the simplest of the three user data element; this will replace the element with a simple string value. The element is defined as below.

```
<userData [mandatory="true/false"] name="" [default=""] />
```

Only the name attribute is a required field. This is the name of the data item that is stored in the Hashtable as a key the corresponding value is outputted in place on the userData element .
The default attribute value will be used when no data is given in the Hashtable.
The mandatory attribute is optional, the value can be either true or false. This indicates if the data item is mandatory. If mandatory is true then the Hashtable must contain a key with the userData name.

A userData element can be added to both a body and subject element

The item in the Hashtable that binds to this element should be a string or another object which has ToString() method.

User Data List

This user data element will output an IEnumerable object, so if you have an ArrayList of strings for example this will loop through the ArrayList and output each string.

```
<userDataList [mandatory="true/false"] name="">
<itemStart></itemStart>
 <itemEnd></itemEnd>
 <itemSeparator></itemSeparator>
</userDataList>
```

The name attribute is used to map the data item in the Hashtable to this element, this attribute is mandatory.
The nested elements are used to format the output of the each value outputted.

Element Name	Notes
itemStart	Takes a string value which is outputted before any of the enumerator values.
itemEnd	Takes a string value which is outputted after the last item in the Enumerator.
itemSeparator	Takes a string value which is outputted between each iteration, the last item in the enumerator doesn't have this value outputted after it.

The formatting of the outputted value will appear like the shown below, note however the last item in the list will not have the ItemSeparatorString.

ItemStartDataItemItemSeparatorDataItemItemSeparatorDataItemItemEnd

The item in the Hashtable for this element must support the IEnumerable interface, an exception will be thrown if it doesn't.

A userDataList element can only be used in a body element.

User Data Dictionary

This will output an object that supports the IDictionary interface as a key value pair.

```
<userDataDictionary [mandatory="true/false"] name="">
<itemStart></itemStart>
 <itemEnd></itemEnd>
 <itemSeparator></itemSeparator>
 <rowSeparator></rowSeparator>
</userDataDictionary>
```

The name attribute is mandatory and links the Hashtable data object to the element. The nested elements are used to format the output of the each value outputted.

Element Name	Notes
itemStart	Takes a string value which is outputted before the IDictionary values.
itemSeparator	Takes a string value which is outputted between each key value pair, so a value of ": " will mean that each item with be "key: value" when outputted.
rowSeparator	Takes a string value which will be outputted after each key value pair, except the final key value pair. This could be a linefeed, forcing each key value on to a new line for example.
itemEnd	Takes a string value which is outputted after the last item in the IDictionary object.

The formatting of the outputted values will appear like below.

ItemStartKeyItemSeparatorValueRowSeparatorKeyItemSeparatorValueRowSe
paratorItemEnd

The item in the Hashtable for this element must support the IDictionary, an exception will be thrown if the it doesn't.

A userDataDictionary element can only be used in a body element.

Simple Example

A simple example to demonstrate the email templates would be an email that is sent to a user when they first register with a website for instance.

Firstly we would define a template file that the email is based on, and then define what data values we are going to merge into it at runtime.

"[name]

Thank you for registering with www.dummysite.com, our account details are shown below
[account details]

Regards

Site Administrator"

The name value will be user's name and the account details will be all the account details that the user has registered with.
The name element will be a simple userDataElement where as the [account details] will be a userDataDictionary item.

So lets create the email template file that the application will need. I'm going to create this email as plain text email since it is easier. There is the source of the template file.

```
<?xml version="1.0" encoding="utf-8" ?>
<template:emailTemplate application="example application"
version="1.2"
xmlns:user="http://www.bitethebullet.co.uk/EmailProcessor/UserData"
 xmlns:template="http://bitethebullet.co.uk/EmailProcessor/Templ
ate">
<template:name>User Registered</template:name>
<template:description>Email template used as an example, sends a
welcome email to new registered users
along with there details</template:description>
<template:mailFormat>PlainText</template:mailFormat>
<template:message>
 <template:to></template:to>
 <template:from>admin@dummysite.com</template:from>
 <template:subject>Welcome to dummySite</template:subject>
 <template:body><user:userData name="user_name" mandatory="true"
/>

Thank you for registering with www.dummysite.com, our account details are shown
below
<user:userDataDictionary mandatory="true" name="userAccountDetails">
  <user:itemStart></user:itemStart>
  <user:itemEnd></user:itemEnd>
  <user:itemSeparator>:- </user:itemSeparator>
  <user:rowSeparator>/r/n</user:rowSeparator>
 </user:userDataDictionary>

Regards

Site Administrator
 </template:body>
 <template:attachments>
 </template:attachments>
</template:message>
</template:emailTemplate>
```

There is one important point to see from the above code, how we format a plain text email, we use \r, \n and \t as escape characters for white space values. These are the only white space escape characters that you can use to format the email message.

To start a new line you should use \r\n, to tab horizontal you should you \t. If you where creating this email in HTML format that you could use the standard HTML mark-up tags to layout the email as you require.

Normally the data that you would merge with this template would come from a database, however for this example we are simply going to load the data to into a Hashtable manually.

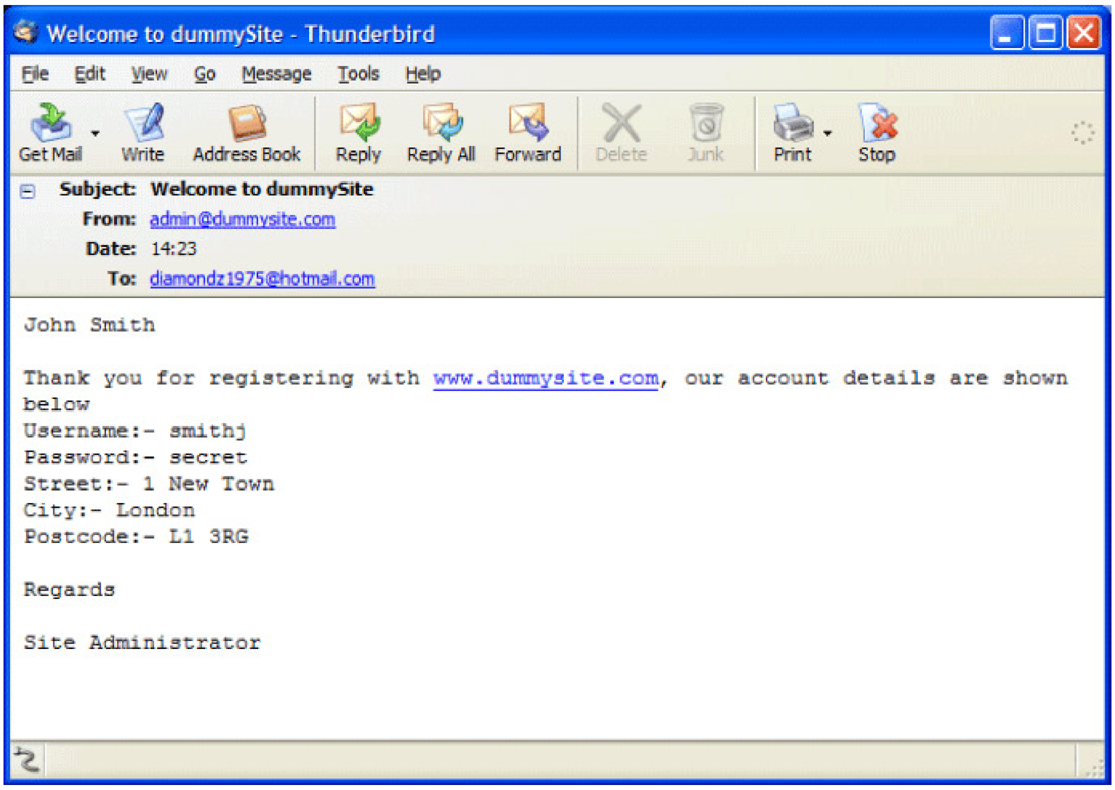
The complete C# code listing loading the template with data and the sending of the email is shown below.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Configuration;

using log4net;

using EmailTemplateProcessor;

namespace Email_Template_Example
{
        /// <summary>
        /// Example used to send a sample email using email template framework
        /// </summary>
        class Example
        {
         protected static readonly ILog Log = LogManager.GetLogger(typeof(Example));

                /// <summary>
                /// The main entry point for the application.
                /// </summary>
                [STAThread]
                static void Main(string[] args)
                {
             Log.Debug("Starting to send email message");

                        //normally the data would come from a database, here we will
                        //load the
                        //data into the hashtable ourselves
                        Hashtable data = new Hashtable();
                        data.Add("user_name", "John Smith");

                        //create some sample user account data in a hashtable
             OrderedDictionary userAccount = new OrderedDictionary();
                        userAccount.Add("Username", "smithj");
                        userAccount.Add("Password", "secret");
                        userAccount.Add("Street", "1 New Town");
                        userAccount.Add("City", "London");
```

```
                              userAccount.Add("Postcode", "L1 3RG");

                              //add the userdetails hashtable to data
                              data.Add("userAccountDetails", userAccount);


                              //merge the template with the data
                              EmailTemplate mailTemplate = new
EmailTemplate(System.Environment.CurrentDirectory,
@"\mailTemplate\simpleListTemplate.xml");
                              mailTemplate.LoadData(data);

               //since we are sending this email to one address we
               //can use the new property
               mailTemplate.ToSingleAddress =
ConfigurationManager.AppSettings["emailToAddress"];

               //other wise we could call this method with a string array
               //holding all the receipt email address
               //mailTemplate.To = new string[] {"some@emailaddresshere.info", "...",
"..."}; //this would come the database in real world

                              EmailProcessor processor = new EmailProcessor("localhost");
                              processor.SendEmail(mailTemplate);

               Log.Debug("Email message sent to user");
                   }
          }
}
```

You'll need to add a reference to the assembly EmailTemplateProcessor.dll to your project, then simply include the namespace EmailTemplateProcessor to have the classes available to project.
I've created this in a console C# application; the source is included in the zip file.

This is what the sample output will look like to the receipt.


Welcome to dummySite - Thunderbird
File Edit View Go Message Tools Help
Get Mail
Write
Address Book
Reply
Reply All
Forward
Delete
Junk
Print
Stop

Subject: Welcome to dummySite
From: admin@dummysite.com
Date: 14:23
To: diamondz1975@hotmail.com

```
John Smith

Thank you for registering with www.dummysite.com, our account details are shown
below
Username:- smithj
Password:- secret
Street:- 1 New Town
City:- London
Postcode:- L1 3RG

Regards

Site Administrator
```

Feel free to run the demo application the email is sent to www.mailinator.com so you can easily verify the message. The address used is *emailtemplate@mailinator.com*

Ordering Of Dictionary Items

There is one thing to notice and that is the order of the UserDataDictionary items are not the same order you added them unless you use an OrderedDictionary object.
This is just how the IDictionaryEnumerator works when outputting the items.

The UserDataList doesn't have this problem; items are outputted in a first in first out basis.

We have two different methods to set the recipients of the email. A simple property ToSingleAddress which is used to send the email to a single email account, or To which takes a string array of email addresses used to send to many recipients.

The two main classes are EmailTemplate which models an email allows for a given template to be loaded with data.
The second class is EmailProcessor which is used to send a EmailTemplate object via SMTP.
Further details can be found in EmailTemplate.chm the API documentation.

Getting Started

To use the email library in your own project simple reference the following two dlls

- */compiled/EmailTemplateProcessor.dll*
- */lib/log4net.dll*

Then include the namespace EmailTemplateProcess in your class.

If you're working with the library in a web project then when passing in the location of the template or the template folder be sure to use

Server.MapPath("~/pathToMailFoldersExample");

For more information about MapPath please follow this link
http://msdn.microsoft.com/en-us/library/system.web.httpserverutility.mappath.aspx

Limitations

There are some rough edges to this simple framework, however it does what I need it to do. Here are some bits that need to be looked at.

- HTML format emails needs to be well formed in order for the XML parser to read the template file, so for example line break markup should be entered then <br/>