```
<?xml version='1.0' encoding='UTF-8'?>

<Root>

<process name="Email - POP3/SMTP" version="6.7.0.0" narrative="Retrieve email via POP3 protocol and

<appdef>

<element name="Application Root">

<id>011d2f54-1ac9-4681-a6e7-993aa460851d</id>

<type>Application</type>

<basetype>Application</basetype>

<datatype>unknown</datatype>

<diagnose>False</diagnose>

</element>

</appdef>

<view>

<camerax>0</camerax>

<cameray>0</cameray>

<zoom>1</zoom>

</view>

<preconditions/>

<endpoint narrative=""/>

<subsheet subsheetid="ad30cfbe-8a8c-44e4-8a26-6ca67f058f7d" type="CleanUp" published="True">

<name>Clean Up</name>

<view>

<camerax>0</camerax>

<cameray>0</cameray>

<zoom>1</zoom>

</view>

</subsheet>
```

```
<subsheet subsheetid="67bff276-3cf2-4985-a957-e462b800ac3b" type="Normal" published="True">
<name>Get Message</name>
<view>
<camerax>0</camerax>
<cameray>79</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<subsheet subsheetid="cc1b4ba2-0510-4980-92bd-d6312072c5bf" type="Normal" published="True">
<name>Send Message</name>
<view>
<camerax>0</camerax>
<cameray>14</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<subsheet subsheetid="8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e" type="Normal" published="True">
<name>Delete Message</name>
<view>
<camerax>0</camerax>
<cameray>-6</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<subsheet subsheetid="849d1a37-179b-40ae-9683-4ebb0d951576" type="Normal" published="True">
<name>Delete Messages</name>
<view>
```

```
<camerax>0</camerax>
<cameray>160</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<subsheet subsheetid="4688b112-6575-4f8a-980c-713566729518" type="Normal" published="False">
<name>Number From ID</name>
<view>
<camerax>0</camerax>
<cameray>64</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<subsheet subsheetid="544abb24-e623-4adb-a24d-3a5dba6164ba" type="Normal" published="True">
<name>Configure</name>
<view>
<camerax>63</camerax>
<cameray>0</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<subsheet subsheetid="954d358e-45d9-44b3-b56a-87e1c21f9d0f" type="Normal" published="False">
<name>Connect POP3</name>
<view>
<camerax>0</camerax>
<cameray>0</cameray>
<zoom>1</zoom>
```

```
</view>
</subsheet>
<subsheet subsheetid="c8b9e601-7444-4385-b2aa-6709658ad472" type="Normal" published="False">
<name>Disconnect POP3</name>
<view>
<camerax>0</camerax>
<cameray>0</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<subsheet subsheetid="e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f" type="Normal" published="True">
<name>List Messages</name>
<view>
<camerax>0</camerax>
<cameray>64</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<subsheet subsheetid="803f1a2c-38f9-4614-97aa-5ca66d454d2d" type="Normal" published="True">
<name>Save Attachments</name>
<view>
<camerax>0</camerax>
<cameray>-27</cameray>
<zoom>1</zoom>
</view>
</subsheet>
<stage stageid="f87d36b8-5544-47f1-bd36-115a2493fdcd" name="Start" type="Start">
```

<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>a378f6c1-35f0-4954-9186-e304278d369e</onsuccess>
</stage>
<stage stageid="a378f6c1-35f0-4954-9186-e304278d369e" name="End" type="End">
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>90</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="65d9a6b9-33b9-4e6b-8211-7aa840ca5eee" name="Stage1" type="ProcessInfo">
<narrative>
</narrative>
<displayx>-195</displayx>
<displayy>-105</displayy>
<displaywidth>150</displaywidth>
<displayheight>90</displayheight>

```
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<references>
<reference>System.dll</reference>
<reference>System.Data.dll</reference>
<reference>System.Xml.dll</reference>
</references>
<imports>
<import>System</import>
<import>System.IO</import>
<import>System.Net</import>
<import>System.Net.Mail</import>
<import>System.Net.Mime</import>
<import>System.Net.Sockets</import>
<import>System.Net.Security</import>
<import>System.Collections.Generic</import>
<import>System.Collections.Specialized</import>
<import>System.Text</import>
<import>System.Security.Cryptography</import>
<import>System.Globalization</import>
<import>System.Text.RegularExpressions</import>
<import>System.Data</import>
</imports>
<language>csharp</language>
<globalcode></globalcode>
<code>Pop3Client _rclient;

/// &lt;summary&gt;
/// This interface describes a MessageTraverser which is able to traverse a Message structure
/// and deliver some answer given some question.
/// &lt;/summary&gt;
/// &lt;typeparam name="TAnswer"&gt;This is the type of the answer you want to have delivered.&lt;/typepa
/// &lt;typeparam name="TQuestion"&gt;This is the type of the question you want to have answered.&lt;/typ
public interface IQuestionAnswerMessageTraverser&lt;TQuestion, TAnswer&gt;
{
/// &lt;summary&gt;
/// Call this when you want to apply this traverser on a &lt;see cref="Message"/&gt;.
/// &lt;/summary&gt;
/// &lt;param name="message"&gt;The &lt;see cref="Message"/&gt; which you want to traverse. Must not b
/// &lt;param name="question"&gt;The question&lt;/param&gt;
/// &lt;returns&gt;An answer&lt;/returns&gt;
TAnswer VisitMessage(Message message, TQuestion question);

/// &lt;summary&gt;
/// Call this when you want to apply this traverser on a &lt;see cref="MessagePart"/&gt;.
/// &lt;/summary&gt;
/// &lt;param name="messagePart"&gt;The &lt;see cref="MessagePart"/&gt; which you want to traverse. Mu
/// &lt;param name="question"&gt;The question&lt;/param&gt;
/// &lt;returns&gt;An answer&lt;/returns&gt;
TAnswer VisitMessagePart(MessagePart messagePart, TQuestion question);
}

///&lt;summary&gt;
/// Finds all the &lt;see cref="MessagePart"/&gt;s which have a given MediaType using a depth first travers
///&lt;/summary&gt;
```

```
internal class FindAllMessagePartsWithMediaType : IQuestionAnswerMessageTraverser&lt;string, List&lt;M

{

/// &lt;summary&gt;

/// Finds all the &lt;see cref="MessagePart"/&gt;s with the given MediaType

/// &lt;/summary&gt;

/// &lt;param name="message"&gt;The &lt;see cref="Message"/&gt; to start looking in&lt;/param&gt;

/// &lt;param name="question"&gt;The MediaType to look for. Case is ignored.&lt;/param&gt;

/// &lt;returns&gt;

/// A List of &lt;see cref="MessagePart"/&gt;s with the given MediaType.&lt;br/&gt;

/// &lt;br/&gt;

/// The List might be empty if no such &lt;see cref="MessagePart"/&gt;s were found.&lt;br/&gt;

/// The order of the elements in the list is the order which they are found using

/// a depth first traversal of the &lt;see cref="Message"/&gt; hierarchy.

/// &lt;/returns&gt;

public List&lt;MessagePart&gt; VisitMessage(Message message, string question)

{

if (message == null)

throw new ArgumentNullException("message");


return VisitMessagePart(message.MessagePart, question);

}



/// &lt;summary&gt;

/// Finds all the &lt;see cref="MessagePart"/&gt;s with the given MediaType

/// &lt;/summary&gt;

/// &lt;param name="messagePart"&gt;The &lt;see cref="MessagePart"/&gt; to start looking in&lt;/param&g

/// &lt;param name="question"&gt;The MediaType to look for. Case is ignored.&lt;/param&gt;
/// &lt;returns&gt;
/// A List of &lt;see cref="MessagePart"/&gt;s with the given MediaType.&lt;br/&gt;
/// &lt;br/&gt;
/// The List might be empty if no such &lt;see cref="MessagePart"/&gt;s were found.&lt;br/&gt;
/// The order of the elements in the list is the order which they are found using
/// a depth first traversal of the &lt;see cref="Message"/&gt; hierarchy.
/// &lt;/returns&gt;
public List&lt;MessagePart&gt; VisitMessagePart(MessagePart messagePart, string question)
{
if (messagePart == null)
throw new ArgumentNullException("messagePart");

List&lt;MessagePart&gt; results = new List&lt;MessagePart&gt;();

if (messagePart.ContentType.MediaType.Equals(question, StringComparison.OrdinalIgnoreCase))
results.Add(messagePart);

if (messagePart.IsMultiPart)
{
foreach (MessagePart part in messagePart.MessageParts)
{
List&lt;MessagePart&gt; result = VisitMessagePart(part, question);
results.AddRange(result);
}
}

return results;
}
}

///&lt;summary&gt;
/// Finds the first &lt;see cref="MessagePart"/&gt; which have a given MediaType in a depth first traversal.
///&lt;/summary&gt;
internal class FindFirstMessagePartWithMediaType : IQuestionAnswerMessageTraverser&lt;string, Messa
{
/// &lt;summary&gt;
/// Finds the first &lt;see cref="MessagePart"/&gt; with the given MediaType
/// &lt;/summary&gt;
/// &lt;param name="message"&gt;The &lt;see cref="Message"/&gt; to start looking in&lt;/param&gt;
/// &lt;param name="question"&gt;The MediaType to look for. Case is ignored.&lt;/param&gt;
/// &lt;returns&gt;A &lt;see cref="MessagePart"/&gt; with the given MediaType or &lt;see langword="null"/&
public MessagePart VisitMessage(Message message, string question)
{
if (message == null)
throw new ArgumentNullException("message");

return VisitMessagePart(message.MessagePart, question);
}

/// &lt;summary&gt;
/// Finds the first &lt;see cref="MessagePart"/&gt; with the given MediaType
/// &lt;/summary&gt;
/// &lt;param name="messagePart"&gt;The &lt;see cref="MessagePart"/&gt; to start looking in&lt;/param&g
/// &lt;param name="question"&gt;The MediaType to look for. Case is ignored.&lt;/param&gt;
/// &lt;returns&gt;A &lt;see cref="MessagePart"/&gt; with the given MediaType or &lt;see langword="null"/&g

public MessagePart VisitMessagePart(MessagePart messagePart, string question)

{

if (messagePart == null)

throw new ArgumentNullException("messagePart");


if (messagePart.ContentType.MediaType.Equals(question, StringComparison.OrdinalIgnoreCase))

return messagePart;


if (messagePart.IsMultiPart)

{

foreach (MessagePart part in messagePart.MessageParts)

{

MessagePart result = VisitMessagePart(part, question);

if (result != null)

return result;

}

}


return null;

}

}


/// &lt;summary&gt;

/// Finds all &lt;see cref="MessagePart"/&gt;s which are considered to be attachments

/// &lt;/summary&gt;

internal class AttachmentFinder : MultipleMessagePartFinder
```

```
{
protected override List&lt;MessagePart&gt; CaseLeaf(MessagePart messagePart)
{
if (messagePart == null)
throw new ArgumentNullException("messagePart");

// Maximum space needed is one
List&lt;MessagePart&gt; leafAnswer = new List&lt;MessagePart&gt;(1);

if (messagePart.IsAttachment)
leafAnswer.Add(messagePart);

return leafAnswer;
}
}

/// &lt;summary&gt;
/// This interface describes a MessageTraverser which is able to traverse a Message hierarchy structure
/// and deliver some answer.
/// &lt;/summary&gt;
/// &lt;typeparam name="TAnswer"&gt;This is the type of the answer you want to have delivered.&lt;/typepa
public interface IAnswerMessageTraverser&lt;TAnswer&gt;
{
/// &lt;summary&gt;
/// Call this when you want to apply this traverser on a &lt;see cref="Message"/&gt;.
/// &lt;/summary&gt;
/// &lt;param name="message"&gt;The &lt;see cref="Message"/&gt; which you want to traverse. Must not b
/// &lt;returns&gt;An answer&lt;/returns&gt;
TAnswer VisitMessage(Message message);

/// &lt;summary&gt;
/// Call this when you want to apply this traverser on a &lt;see cref="MessagePart"/&gt;.
/// &lt;/summary&gt;
/// &lt;param name="messagePart"&gt;The &lt;see cref="MessagePart"/&gt; which you want to traverse. Mu
/// &lt;returns&gt;An answer&lt;/returns&gt;
TAnswer VisitMessagePart(MessagePart messagePart);
}

/// &lt;summary&gt;
/// This is an abstract class which handles traversing of a &lt;see cref="Message"/&gt; tree structure.&lt;br/&
/// It runs through the message structure using a depth-first traversal.
/// &lt;/summary&gt;
/// &lt;typeparam name="TAnswer"&gt;The answer you want from traversing the message tree structure&lt;
public abstract class AnswerMessageTraverser&lt;TAnswer&gt; : IAnswerMessageTraverser&lt;TAnswer&
{
/// &lt;summary&gt;
/// Call this when you want an answer for a full message.
/// &lt;/summary&gt;
/// &lt;param name="message"&gt;The message you want to traverse&lt;/param&gt;
/// &lt;returns&gt;An answer&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;if &lt;paramref name="message"/&gt; is &lt;see langword
public TAnswer VisitMessage(Message message)
{
if (message == null)
```

```
throw new ArgumentNullException("message");


return VisitMessagePart(message.MessagePart);

}


/// &lt;summary&gt;

/// Call this method when you want to find an answer for a &lt;see cref="MessagePart"/&gt;

/// &lt;/summary&gt;

/// &lt;param name="messagePart"&gt;The &lt;see cref="MessagePart"/&gt; part you want an answer from.

/// &lt;returns&gt;An answer&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;if &lt;paramref name="messagePart"/&gt; is &lt;see lang

public TAnswer VisitMessagePart(MessagePart messagePart)

{

if (messagePart == null)

throw new ArgumentNullException("messagePart");


if (messagePart.IsMultiPart)

{

List&lt;TAnswer&gt; leafAnswers = new List&lt;TAnswer&gt;(messagePart.MessageParts.Count);

foreach (MessagePart part in messagePart.MessageParts)

{

leafAnswers.Add(VisitMessagePart(part));

}

return MergeLeafAnswers(leafAnswers);

}


return CaseLeaf(messagePart);
```

```
}

/// &lt;summary&gt;
/// For a concrete implementation an answer must be returned for a leaf &lt;see cref="MessagePart"/&gt;, w
/// MessageParts that are not &lt;see cref="MessagePart.IsMultiPart"&gt;MultiParts.&lt;/see&gt;
/// &lt;/summary&gt;
/// &lt;param name="messagePart"&gt;The message part which is a leaf and thereby not a MultiPart&lt;/par
/// &lt;returns&gt;An answer&lt;/returns&gt;
protected abstract TAnswer CaseLeaf(MessagePart messagePart);

/// &lt;summary&gt;
/// For a concrete implementation, when a MultiPart &lt;see cref="MessagePart"/&gt; has fetched it's answe
/// answers needs to be merged. This is the responsibility of this method.
/// &lt;/summary&gt;
/// &lt;param name="leafAnswers"&gt;The answer that the leafs gave&lt;/param&gt;
/// &lt;returns&gt;A merged answer&lt;/returns&gt;
protected abstract TAnswer MergeLeafAnswers(List&lt;TAnswer&gt; leafAnswers);
}

///&lt;summary&gt;
/// An abstract class that implements the MergeLeafAnswers method.&lt;br/&gt;
/// The method simply returns the union of all answers from the leaves.
///&lt;/summary&gt;
public abstract class MultipleMessagePartFinder : AnswerMessageTraverser&lt;List&lt;MessagePart&gt;&g
{
/// &lt;summary&gt;
/// Adds all the &lt;paramref name="leafAnswers"/&gt; in one big answer
/// &lt;/summary&gt;

/// &lt;param name="leafAnswers"&gt;The answers to merge&lt;/param&gt;

/// &lt;returns&gt;A list with has all the elements in the &lt;paramref name="leafAnswers"/&gt; lists&lt;/return

/// &lt;exception cref="ArgumentNullException"&gt;if &lt;paramref name="leafAnswers"/&gt; is &lt;see langw

protected override List&lt;MessagePart&gt; MergeLeafAnswers(List&lt;List&lt;MessagePart&gt;&gt; leafAns

{

if (leafAnswers == null)

throw new ArgumentNullException("leafAnswers");


// We simply create a list with all the answer generated from the leaves

List&lt;MessagePart&gt; mergedResults = new List&lt;MessagePart&gt;();


foreach (List&lt;MessagePart&gt; leafAnswer in leafAnswers)

{

mergedResults.AddRange(leafAnswer);

}


return mergedResults;

}

}


/// &lt;summary&gt;

/// Finds all text/[something] versions in a Message hierarchy

/// &lt;/summary&gt;

internal class TextVersionFinder : MultipleMessagePartFinder

{

protected override List&lt;MessagePart&gt; CaseLeaf(MessagePart messagePart)
{

if (messagePart == null)

throw new ArgumentNullException("messagePart");


// Maximum space needed is one

List&lt;MessagePart&gt; leafAnswer = new List&lt;MessagePart&gt;(1);


if (messagePart.IsText)

leafAnswer.Add(messagePart);


return leafAnswer;

}

}


/// &lt;summary&gt;

/// This class is responsible for decoding parameters that has been encoded with:&lt;br/&gt;

/// &lt;list type="bullet"&gt;

/// &lt;item&gt;

///    &lt;b&gt;Continuation&lt;/b&gt;&lt;br/&gt;

///    This is where a single parameter has such a long value that it could

///    be wrapped while in transit. Instead multiple parameters is used on each line.&lt;br/&gt;

///    &lt;br/&gt;

///    &lt;b&gt;Example&lt;/b&gt;&lt;br/&gt;

///    From: &lt;c&gt;Content-Type: text/html; boundary="someVeryLongStringHereWhichCouldBeWrapped

///    To: &lt;c&gt;Content-Type: text/html; boundary*0="someVeryLongStringHere" boundary*1="WhichCou

/// &lt;/item&gt;

/// &lt;item&gt;
///    &lt;b&gt;Encoding&lt;/b&gt;&lt;br/&gt;
///    Sometimes other characters then ASCII characters are needed in parameters.&lt;br/&gt;
///    The parameter is then given a different name to specify that it is encoded.&lt;br/&gt;
///    &lt;br/&gt;
///    &lt;b&gt;Example&lt;/b&gt;&lt;br/&gt;
///    From: &lt;c&gt;Content-Disposition attachment; filename="specialCharsÆØÅ"&lt;/c&gt;&lt;br/&gt;
///    To: &lt;c&gt;Content-Disposition attachment; filename*="ISO-8859-1'en-us'specialCharsC6D8C0"&lt;/c
///    This encoding is almost the same as &lt;see cref="EncodedWord"/&gt; encoding, and is used to decod
/// &lt;/item&gt;
/// &lt;item&gt;
///    &lt;b&gt;Continuation and Encoding&lt;/b&gt;&lt;br/&gt;
///    Both Continuation and Encoding can be used on the same time.&lt;br/&gt;
///    &lt;br/&gt;
///    &lt;b&gt;Example&lt;/b&gt;&lt;br/&gt;
///    From: &lt;c&gt;Content-Disposition attachment; filename="specialCharsÆØÅWhichIsSoLong"&lt;/c&gt
///    To: &lt;c&gt;Content-Disposition attachment; filename*0*="ISO-8859-1'en-us'specialCharsC6D8C0"; fi
///    This could also be encoded as:&lt;br/&gt;
///    To: &lt;c&gt;Content-Disposition attachment; filename*0*="ISO-8859-1'en-us'specialCharsC6D8C0"; fi
///    Notice that &lt;c&gt;filename*1&lt;/c&gt; does not have an &lt;c&gt;*&lt;/c&gt; after it - denoting it IS NO
///    There are some rules about this:&lt;br/&gt;
///    &lt;list type="number"&gt;
///      &lt;item&gt;The encoding must be mentioned in the first part (filename*0*), which has to be encoded.
///      &lt;item&gt;No other part must specify an encoding, but if encoded it uses the encoding mentioned in
///      &lt;item&gt;Parts may be encoded or not in any order.&lt;/item&gt;
///    &lt;/list&gt;
///    &lt;br/&gt;
/// &lt;/item&gt;
/// &lt;/list&gt;
/// More information and the specification is available in &lt;see href="http://tools.ietf.org/html/rfc2231"&gt;R
/// &lt;/summary&gt;
internal static class Rfc2231Decoder
{
/// &lt;summary&gt;
/// Decodes a string of the form:&lt;br/&gt;
/// &lt;c&gt;value0; key1=value1; key2=value2; key3=value3&lt;/c&gt;&lt;br/&gt;
/// The returned List of key value pairs will have the key as key and the decoded value as value.&lt;br/&gt;
/// The first value0 will have a key of &lt;see cref="string.Empty"/&gt;.&lt;br/&gt;
/// &lt;br/&gt;
/// If continuation is used, then multiple keys will be merged into one key with the different values
/// decoded into on big value for that key.&lt;br/&gt;
/// Example:&lt;br/&gt;
/// &lt;code&gt;
/// title*0=part1
/// title*1=part2
/// &lt;/code&gt;
/// will have key and value of:&lt;br&gt;&lt;/br&gt;
/// &lt;c&gt;title=decode(part1)decode(part2)&lt;/c&gt;
/// &lt;/summary&gt;
/// &lt;param name="toDecode"&gt;The string to decode.&lt;/param&gt;
/// &lt;returns&gt;A list of decoded key value pairs.&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="toDecode"/&gt; is &lt;see langwor
public static List&lt;KeyValuePair&lt;string, string&gt;&gt; Decode(string toDecode)
{
if(toDecode == null)
```

```
throw new ArgumentNullException("toDecode");


// Normalize the input to take account for missing semicolons after parameters.
// Example
// text/plain; charset=\"iso-8859-1\" name=\"somefile.txt\" or
// text/plain;\tcharset=\"iso-8859-1\"\tname=\"somefile.txt\"
// is normalized to
// text/plain; charset=\"iso-8859-1\"; name=\"somefile.txt\"
// Only works for parameters inside quotes
// \s = matches whitespace
toDecode = Regex.Replace(toDecode, "=\\s*\"(?&lt;value&gt;[^\"]*)\"\\s", "=\"${value}\"; ");


// Normalize
// Since the above only works for parameters inside quotes, we need to normalize
// the special case with the first parameter.
// Example:
// attachment filename="foo"
// is normalized to
// attachment; filename="foo"
// ^ = matches start of line (when not inside square bracets [])
toDecode = Regex.Replace(toDecode, @"^(?&lt;first&gt;[^;\s]+)\s(?&lt;second&gt;[^;\s]+)", "${first}; ${secon


// Split by semicolon, but only if not inside quotes
List&lt;string&gt; splitted = Utility.SplitStringWithCharNotInsideQuotes(toDecode.Trim(), ';');


List&lt;KeyValuePair&lt;string, string&gt;&gt; collection = new List&lt;KeyValuePair&lt;string, string&gt;&gt;(
foreach (string part in splitted)
{
// Empty strings should not be processed
if (part.Trim().Length == 0)
continue;

string[] keyValue = part.Trim().Split(new char[] {'='}, 2);
if(keyValue.Length == 1)
{
collection.Add(new KeyValuePair&lt;string, string&gt;("", keyValue[0]));
} else if (keyValue.Length == 2)
{
collection.Add(new KeyValuePair&lt;string, string&gt;(keyValue[0], keyValue[1]));
} else
{
throw new ArgumentException("When splitting the part \"" + part + "\" by = there was " + keyValue.Length +
}
}

return DecodePairs(collection);
}

/// &lt;summary&gt;
/// Decodes the list of key value pairs into a decoded list of key value pairs.&lt;br/&gt;
/// There may be less keys in the decoded list, but then the values for the lost keys will have been appende
/// to the new key.
/// &lt;/summary&gt;
/// &lt;param name="pairs"&gt;The pairs to decode&lt;/param&gt;

/// &lt;returns&gt;A decoded list of pairs&lt;/returns&gt;

private static List&lt;KeyValuePair&lt;string, string&gt;&gt; DecodePairs(List&lt;KeyValuePair&lt;string, strin

{

if (pairs == null)

throw new ArgumentNullException("pairs");



List&lt;KeyValuePair&lt;string, string&gt;&gt; resultPairs = new List&lt;KeyValuePair&lt;string, string&gt;&gt



int pairsCount = pairs.Count;

for (int i = 0; i &lt; pairsCount; i++)

{

KeyValuePair&lt;string, string&gt; currentPair = pairs[i];

string key = currentPair.Key;

string value = Utility.RemoveQuotesIfAny(currentPair.Value);



// Is it a continuation parameter? (encoded or not)

if (key.EndsWith("*0", StringComparison.OrdinalIgnoreCase) || key.EndsWith("*0*", StringComparison.Ordi

{

// This encoding will not be used if we get into the if which tells us

// that the whole continuation is not encoded



string encoding = "notEncoded - Value here is never used";



// Now lets find out if it is encoded too.

if (key.EndsWith("*0*", StringComparison.OrdinalIgnoreCase))

{
```

```
// It is encoded.

// Fetch out the encoding for later use and decode the value
// If the value was not encoded as the email specified
// encoding will be set to null. This will be used later.
value = DecodeSingleValue(value, out encoding);

// Find the right key to use to store the full value
// Remove the start *0 which tells is it is a continuation, and the first one
// And remove the * afterwards which tells us it is encoded
key = key.Replace("*0*", "");
}
else
{
// It is not encoded, and no parts of the continuation is encoded either

// Find the right key to use to store the full value
// Remove the start *0 which tells is it is a continuation, and the first one
key = key.Replace("*0", "");
}

// The StringBuilder will hold the full decoded value from all continuation parts
StringBuilder builder = new StringBuilder();

// Append the decoded value
builder.Append(value);
```

```
// Now go trough the next keys to see if they are part of the continuation
for (int j = i + 1, continuationCount = 1; j &lt; pairsCount; j++, continuationCount++)
{
string jKey = pairs[j].Key;
string valueJKey = Utility.RemoveQuotesIfAny(pairs[j].Value);

if (jKey.Equals(key + "*" + continuationCount))
{
// This value part of the continuation is not encoded
// Therefore remove qoutes if any and add to our stringbuilder
builder.Append(valueJKey);

// Remember to increment i, as we have now treated one more KeyValuePair
i++;
}
else if (jKey.Equals(key + "*" + continuationCount + "*"))
{
// We will not get into this part if the first part was not encoded
// Therefore the encoding will only be used if and only if the
// first part was encoded, in which case we have remembered the encoding used

// Sometimes an email creator says that a string was encoded, but it really
// `was not. This is to catch that problem.
if (encoding != null)
{
// This value part of the continuation is encoded
// the encoding is not given in the current value,
// but was given in the first continuation, which we remembered for use here
valueJKey = DecodeSingleValue(valueJKey, encoding);
}
builder.Append(valueJKey);

// Remember to increment i, as we have now treated one more KeyValuePair
i++;
}
else
{
// No more keys for this continuation
break;
}
}

// Add the key and the full value as a pair
value = builder.ToString();
resultPairs.Add(new KeyValuePair&lt;string, string&gt;(key, value));
}
else if (key.EndsWith("*", StringComparison.OrdinalIgnoreCase))
{
// This parameter is only encoded - it is not part of a continuation
// We need to change the key from "&lt;key&gt;*" to "&lt;key&gt;" and decode the value

// To get the key we want, we remove the last * that denotes
// that the value hold by the key was encoded
key = key.Replace("*", "");
```

```
// Decode the value
string throwAway;
value = DecodeSingleValue(value, out throwAway);

// Now input the new value with the new key
resultPairs.Add(new KeyValuePair&lt;string, string&gt;(key, value));
}
else
{
// Fully normal key - the value is not encoded
// Therefore nothing to do, and we can simply pass the pair
// as being decoded now
resultPairs.Add(currentPair);
}
}

return resultPairs;
}

/// &lt;summary&gt;
/// This will decode a single value of the form: &lt;c&gt;ISO-8859-1'en-us'%3D%3DIamHere&lt;/c&gt;&lt;br/&
/// Which is basically a &lt;see cref="EncodedWord"/&gt; form just using % instead of =&lt;br/&gt;
/// Notice that 'en-us' part is not used for anything.&lt;br/&gt;
/// &lt;br/&gt;
/// If the single value given is not on the correct form, it will be returned without
/// being decoded and &lt;paramref name="encodingUsed"/&gt; will be set to &lt;see langword="null"/&gt;.
/// &lt;/summary&gt;

/// &lt;param name="encodingUsed"&gt;

/// The encoding used to decode with - it is given back for later use.&lt;br/&gt;

/// &lt;see langword="null"/&gt; if input was not in the correct form.

/// &lt;/param&gt;

/// &lt;param name="toDecode"&gt;The value to decode&lt;/param&gt;

/// &lt;returns&gt;

/// The decoded value that corresponds to &lt;paramref name="toDecode"/&gt; or if

/// &lt;paramref name="toDecode"/&gt; is not on the correct form, it will be non-decoded.

/// &lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="toDecode"/&gt; is &lt;see langwor

private static string DecodeSingleValue(string toDecode, out string encodingUsed)

{

if (toDecode == null)

throw new ArgumentNullException("toDecode");



// Check if input has a part describing the encoding

if (toDecode.IndexOf('\'') == -1)

{

// The input was not encoded (at least not valid) and it is returned as is

//DefaultLogger.Log.LogDebug("Rfc2231Decoder: Someone asked me to decode a string which was not e

encodingUsed = null;

return toDecode;

}

encodingUsed = toDecode.Substring(0, toDecode.IndexOf('\''));

toDecode = toDecode.Substring(toDecode.LastIndexOf('\'') + 1);

return DecodeSingleValue(toDecode, encodingUsed);
}

/// &lt;summary&gt;
/// This will decode a single value of the form: %3D%3DIamHere
/// Which is basically a &lt;see cref="EncodedWord"/&gt; form just using % instead of =
/// &lt;/summary&gt;
/// &lt;param name="valueToDecode"&gt;The value to decode&lt;/param&gt;
/// &lt;param name="encoding"&gt;The encoding used to decode with&lt;/param&gt;
/// &lt;returns&gt;The decoded value that corresponds to &lt;paramref name="valueToDecode"/&gt;&lt;/retu
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="valueToDecode"/&gt; is &lt;see la
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="encoding"/&gt; is &lt;see langword
private static string DecodeSingleValue(string valueToDecode, string encoding)
{
if (valueToDecode == null)
throw new ArgumentNullException("valueToDecode");

if (encoding == null)
throw new ArgumentNullException("encoding");

// The encoding used is the same as QuotedPrintable, we only
// need to change % to =
// And otherwise make it look like the correct EncodedWord encoding
valueToDecode = "=?" + encoding + "?Q?" + valueToDecode.Replace("%", "=") + "?=";
return EncodedWord.Decode(valueToDecode);
}
}
```

```
/// &lt;summary&gt;

/// Class that can parse different fields in the header sections of a MIME message.

/// &lt;/summary&gt;

internal static class HeaderFieldParser

{

/// &lt;summary&gt;

/// Parses the Content-Transfer-Encoding header.

/// &lt;/summary&gt;

/// &lt;param name="headerValue"&gt;The value for the header to be parsed&lt;/param&gt;

/// &lt;returns&gt;A &lt;see cref="ContentTransferEncoding"/&gt;&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="headerValue"/&gt; is &lt;see lang

/// &lt;exception cref="ArgumentException"&gt;If the &lt;paramref name="headerValue"/&gt; could not be pa

public static ContentTransferEncoding ParseContentTransferEncoding(string headerValue)

{

if (headerValue == null)

throw new ArgumentNullException("headerValue");



switch (headerValue.Trim().ToUpperInvariant())

{

case "7BIT":

return ContentTransferEncoding.SevenBit;



case "8BIT":

return ContentTransferEncoding.EightBit;



case "QUOTED-PRINTABLE":

return ContentTransferEncoding.QuotedPrintable;
case "BASE64":

return ContentTransferEncoding.Base64;


case "BINARY":

return ContentTransferEncoding.Binary;


// If a wrong argument is passed to this parser method, then we assume

// default encoding, which is SevenBit.

// This is to ensure that we do not throw exceptions, even if the email not MIME valid.

default:

//DefaultLogger.Log.LogDebug("Wrong ContentTransferEncoding was used. It was: " + headerValue);

return ContentTransferEncoding.SevenBit;

}

}


/// &lt;summary&gt;

/// Parses an ImportanceType from a given Importance header value.

/// &lt;/summary&gt;

/// &lt;param name="headerValue"&gt;The value to be parsed&lt;/param&gt;

/// &lt;returns&gt;A &lt;see cref="MailPriority"/&gt;. If the &lt;paramref name="headerValue"/&gt; is not reco

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="headerValue"/&gt; is &lt;see lang

public static MailPriority ParseImportance(string headerValue)

{

if (headerValue == null)

throw new ArgumentNullException("headerValue");
```

```
switch (headerValue.ToUpperInvariant())

{

case "5":

case "HIGH":

return MailPriority.High;


case "3":

case "NORMAL":

return MailPriority.Normal;


case "1":

case "LOW":

return MailPriority.Low;


default:

//DefaultLogger.Log.LogDebug("HeaderFieldParser: Unknown importance value: \"" + headerValue + "\". U

return MailPriority.Normal;

}

}


/// &lt;summary&gt;

/// Parses a the value for the header Content-Type to

/// a &lt;see cref="ContentType"/&gt; object.

/// &lt;/summary&gt;

/// &lt;param name="headerValue"&gt;The value to be parsed&lt;/param&gt;

/// &lt;returns&gt;A &lt;see cref="ContentType"/&gt; object&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="headerValue"/&gt; is &lt;see lang
```

```
public static ContentType ParseContentType(string headerValue)
{
if (headerValue == null)
throw new ArgumentNullException("headerValue");

// We create an empty Content-Type which we will fill in when we see the values
ContentType contentType = new ContentType();

// Now decode the parameters
List&lt;KeyValuePair&lt;string, string&gt;&gt; parameters = Rfc2231Decoder.Decode(headerValue);

foreach (KeyValuePair&lt;string, string&gt; keyValuePair in parameters)
{
string key = keyValuePair.Key.ToUpperInvariant().Trim();
string value = Utility.RemoveQuotesIfAny(keyValuePair.Value.Trim());
switch (key)
{
case "":
// This is the MediaType - it has no key since it is the first one mentioned in the
// headerValue and has no = in it.

// Check for illegal content-type
if (value.ToUpperInvariant().Equals("TEXT"))
value = "text/plain";

contentType.MediaType = value;
break;
case "BOUNDARY":

contentType.Boundary = value;

break;


case "CHARSET":

contentType.CharSet = value;

break;


case "NAME":

contentType.Name = EncodedWord.Decode(value);

break;


default:

// This is to shut up the code help that is saying that contentType.Parameters

// can be null - which it cant!

if (contentType.Parameters == null)

throw new Exception("The ContentType parameters property is null. This will never be thrown.");


// We add the unknown value to our parameters list

// "Known" unknown values are:

// - title

// - report-type

contentType.Parameters.Add(key, value);

break;

}

}
```

```
return contentType;

}


/// &lt;summary&gt;

/// Parses a the value for the header Content-Disposition to a &lt;see cref="ContentDisposition"/&gt; object.

/// &lt;/summary&gt;

/// &lt;param name="headerValue"&gt;The value to be parsed&lt;/param&gt;

/// &lt;returns&gt;A &lt;see cref="ContentDisposition"/&gt; object&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="headerValue"/&gt; is &lt;see lang

public static ContentDisposition ParseContentDisposition(string headerValue)

{

if (headerValue == null)

throw new ArgumentNullException("headerValue");


// See http://www.ietf.org/rfc/rfc2183.txt for RFC definition


// Create empty ContentDisposition - we will fill in details as we read them

ContentDisposition contentDisposition = new ContentDisposition();


// Now decode the parameters

List&lt;KeyValuePair&lt;string, string&gt;&gt; parameters = Rfc2231Decoder.Decode(headerValue);


foreach (KeyValuePair&lt;string, string&gt; keyValuePair in parameters)

{

string key = keyValuePair.Key.ToUpperInvariant().Trim();

string value = keyValuePair.Value;
```

```
switch (key)
{
case "":
// This is the DispisitionType - it has no key since it is the first one
// and has no = in it.
contentDisposition.DispositionType = value;
break;

// The correct name of the parameter is filename, but some emails also contains the parameter
// name, which also holds the name of the file. Therefore we use both names for the same field.
case "NAME":
case "FILENAME":
// The filename might be in qoutes, and it might be encoded-word encoded
contentDisposition.FileName = EncodedWord.Decode(Utility.RemoveQuotesIfAny(value));
break;

case "CREATION-DATE":
// Notice that we need to create a new DateTime because of a failure in .NET 2.0.
// The failure is: you cannot give contentDisposition a DateTime with a Kind of UTC
// It will set the CreationDate correctly, but when trying to read it out it will throw an exception.
// It is the same with ModificationDate and ReadDate.
// This is fixed in 4.0 - maybe in 3.0 too.
// Therefore we create a new DateTime which have a DateTimeKind set to unspecified
DateTime creationDate = new DateTime(Rfc2822DateTime.StringToDate(Utility.RemoveQuotesIfAny(valu
contentDisposition.CreationDate = creationDate;
break;
```

```
case "MODIFICATION-DATE":

DateTime midificationDate = new DateTime(Rfc2822DateTime.StringToDate(Utility.RemoveQuotesIfAny(v

contentDisposition.ModificationDate = midificationDate;

break;


case "READ-DATE":

DateTime readDate = new DateTime(Rfc2822DateTime.StringToDate(Utility.RemoveQuotesIfAny(value)).T

contentDisposition.ReadDate = readDate;

break;


case "SIZE":

contentDisposition.Size = int.Parse(Utility.RemoveQuotesIfAny(value), CultureInfo.InvariantCulture);

break;


default:

if (key.StartsWith("X-"))

{

contentDisposition.Parameters.Add(key, Utility.RemoveQuotesIfAny(value));

break;

}


throw new ArgumentException("Unknown parameter in Content-Disposition. Ask developer to fix! Paramete

}

}


return contentDisposition;

}
```

```
/// &lt;summary&gt;
/// Parses an ID like Message-Id and Content-Id.&lt;br/&gt;
/// Example:&lt;br/&gt;
/// &lt;c&gt;&amp;amp;lt;test@test.com&amp;amp;gt;&lt;/c&gt;&lt;br/&gt;
/// into&lt;br/&gt;
/// &lt;c&gt;test@test.com&lt;/c&gt;
/// &lt;/summary&gt;
/// &lt;param name="headerValue"&gt;The id to parse&lt;/param&gt;
/// &lt;returns&gt;A parsed ID&lt;/returns&gt;
public static string ParseId(string headerValue)
{
// Remove whitespace in front and behind since
// whitespace is allowed there
// Remove the last &gt; and the first &lt;
return headerValue.Trim().TrimEnd('&gt;').TrimStart('&lt;');
}

/// &lt;summary&gt;
/// Parses multiple IDs from a single string like In-Reply-To.
/// &lt;/summary&gt;
/// &lt;param name="headerValue"&gt;The value to parse&lt;/param&gt;
/// &lt;returns&gt;A list of IDs&lt;/returns&gt;
public static List&lt;string&gt; ParseMultipleIDs(string headerValue)
{
List&lt;string&gt; returner = new List&lt;string&gt;();
// Split the string by &gt;
// We cannot use ' ' (space) here since this is a possible value:
// &lt;test@test.com&gt;&lt;test2@test.com&gt;
string[] ids = headerValue.Trim().Split(new char[]{ '&gt;' }, StringSplitOptions.RemoveEmptyEntries);
foreach (string id in ids)
{
returner.Add(ParseId(id));
}

return returner;
}
}

/// &lt;summary&gt;
/// Utility class for dealing with encoded word strings&lt;br/&gt;
/// &lt;br/&gt;
/// EncodedWord encoded strings are only in ASCII, but can embed information
/// about characters in other character sets.&lt;br/&gt;
/// &lt;br/&gt;
/// It is done by specifying the character set, an encoding that maps from ASCII to
/// the correct bytes and the actual encoded string.&lt;br/&gt;
/// &lt;br/&gt;
/// It is specified in a format that is best summarized by a BNF:&lt;br/&gt;
/// &lt;c&gt;"=?" character_set "?" encoding "?" encoded-text "?="&lt;/c&gt;&lt;br/&gt;
/// &lt;/summary&gt;
/// &lt;example&gt;
/// &lt;c&gt;=?ISO-8859-1?Q?=2D?=&lt;/c&gt;
/// Here &lt;c&gt;ISO-8859-1&lt;/c&gt; is the character set.&lt;br/&gt;
/// &lt;c&gt;Q&lt;/c&gt; is the encoding method (quoted-printable). &lt;c&gt;B&lt;/c&gt; is also supported (Ba
/// The encoded text is the &lt;c&gt;=2D&lt;/c&gt; part which is decoded to a space.
/// &lt;/example&gt;
internal static class EncodedWord
{
/// &lt;summary&gt;
/// Decode text that is encoded with the &lt;see cref="EncodedWord"/&gt; encoding.&lt;br/&gt;
///&lt;br/&gt;
/// This method will decode any encoded-word found in the string.&lt;br/&gt;
/// All parts which is not encoded will not be touched.&lt;br/&gt;
/// &lt;br/&gt;
/// From &lt;a href="http://tools.ietf.org/html/rfc2047"&gt;RFC 2047&lt;/a&gt;:&lt;br/&gt;
/// &lt;code&gt;
/// Generally, an "encoded-word" is a sequence of printable ASCII
/// characters that begins with "=?", ends with "?=", and has two "?"s in
/// between.  It specifies a character set and an encoding method, and
/// also includes the original text encoded as graphic ASCII characters,
/// according to the rules for that encoding method.
/// &lt;/code&gt;
/// Example:&lt;br/&gt;
/// &lt;c&gt;=?ISO-8859-1?q?this=20is=20some=20text?= other text here&lt;/c&gt;
/// &lt;/summary&gt;
/// &lt;remarks&gt;See &lt;a href="http://tools.ietf.org/html/rfc2047#section-2"&gt;RFC 2047 section 2&lt;/a&
/// &lt;param name="encodedWords"&gt;Source text. May be content which is not encoded.&lt;/param&gt;
/// &lt;returns&gt;Decoded text&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="encodedWords"/&gt; is &lt;see la
```

```
public static string Decode(string encodedWords)
{
if (encodedWords == null)
throw new ArgumentNullException("encodedWords");

// Notice that RFC2231 redefines the BNF to
// encoded-word := "=?" charset ["*" language] "?" encoded-text "?="
// but no usage of this BNF have been spotted yet. It is here to
// ease debugging if such a case is discovered.

// This is the regex that should fit the BNF
// RFC Says that NO WHITESPACE is allowed in this encoding, but there are examples
// where whitespace is there, and therefore this regex allows for such.
const string encodedWordRegex = @"\=\?(?&lt;Charset&gt;\S+?)\?(?&lt;Encoding&gt;\w)\?(?&lt;Content&g
// \w Matches any word character including underscore. Equivalent to "[A-Za-z0-9_]".
// \S Matches any nonwhite space character. Equivalent to "[^ \f\n\r\t\v]".
// +?   non-gready equivalent to +
// (?&lt;NAME&gt;REGEX) is a named group with name NAME and regular expression REGEX

// Any amount of linear-space-white between 'encoded-word's,
// even if it includes a CRLF followed by one or more SPACEs,
// is ignored for the purposes of display.
// http://tools.ietf.org/html/rfc2047#page-12
// Define a regular expression that captures two encoded words with some whitespace between them
const string replaceRegex = @"(?&lt;first&gt;" + encodedWordRegex + @")\s+(?&lt;second&gt;" + encoded

// Then, find an occourance of such an expression, but remove the whitespace inbetween when found
```

```
encodedWords = Regex.Replace(encodedWords, replaceRegex, "${first}${second}");

string decodedWords = encodedWords;

MatchCollection matches = Regex.Matches(encodedWords, encodedWordRegex);
foreach (Match match in matches)
{
// If this match was not a success, we should not use it
if (!match.Success) continue;

string fullMatchValue = match.Value;

string encodedText = match.Groups["Content"].Value;
string encoding = match.Groups["Encoding"].Value;
string charset = match.Groups["Charset"].Value;

// Get the encoding which corrosponds to the character set
Encoding charsetEncoding = EncodingFinder.FindEncoding(charset);

// Store decoded text here when done
string decodedText;

// Encoding may also be written in lowercase
switch (encoding.ToUpperInvariant())
{
// RFC:
// The "B" encoding is identical to the "BASE64"
// encoding defined by RFC 2045.
// http://tools.ietf.org/html/rfc2045#section-6.8
case "B":
decodedText = Base64.Decode(encodedText, charsetEncoding);
break;

// RFC:
// The "Q" encoding is similar to the "Quoted-Printable" content-
// transfer-encoding defined in RFC 2045.
// There are more details to this. Please check
// http://tools.ietf.org/html/rfc2047#section-4.2
//
case "Q":
decodedText = QuotedPrintable.DecodeEncodedWord(encodedText, charsetEncoding);
break;

default:
throw new ArgumentException("The encoding " + encoding + " was not recognized");
}

// Repalce our encoded value with our decoded value
decodedWords = decodedWords.Replace(fullMatchValue, decodedText);
}

return decodedWords;
}
}
```

```
/// &lt;summary&gt;

/// Contains common operations needed while decoding.

/// &lt;/summary&gt;

internal static class Utility

{

/// &lt;summary&gt;

/// Remove quotes, if found, around the string.

/// &lt;/summary&gt;

/// &lt;param name="text"&gt;Text with quotes or without quotes&lt;/param&gt;

/// &lt;returns&gt;Text without quotes&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="text"/&gt; is &lt;see langword="nu

public static string RemoveQuotesIfAny(string text)

{

if (text == null)

throw new ArgumentNullException("text");


// Check if there are qoutes at both ends

if (text[0] == "" &amp;amp;&amp;amp; text[text.Length - 1] == "")

{

// Remove quotes at both ends

return text.Substring(1, text.Length - 2);

}


// If no quotes were found, the text is just returned

return text;

}
```

```
/// &lt;summary&gt;

/// Split a string into a list of strings using a specified character.&lt;br/&gt;

/// Everything inside quotes are ignored.

/// &lt;/summary&gt;

/// &lt;param name="input"&gt;A string to split&lt;/param&gt;

/// &lt;param name="toSplitAt"&gt;The character to use to split with&lt;/param&gt;

/// &lt;returns&gt;A List of strings that was delimited by the &lt;paramref name="toSplitAt"/&gt; character&lt;

public static List&lt;string&gt; SplitStringWithCharNotInsideQuotes(string input, char toSplitAt)

{

List&lt;string&gt; elements = new List&lt;string&gt;();


int lastSplitLocation = 0;

bool insideQuote = false;


char[] characters = input.ToCharArray();


for (int i = 0; i &lt; characters.Length; i++)

{

char character = characters[i];

if (character == '\"')

insideQuote = !insideQuote;


// Only split if we are not inside quotes

if (character == toSplitAt &amp;amp;&amp;amp; !insideQuote)

{

// We need to split
int length = i - lastSplitLocation;
elements.Add(input.Substring(lastSplitLocation, length));

// Update last split location
// + 1 so that we do not include the character used to split with next time
lastSplitLocation = i + 1;
}
}

// Add the last part
elements.Add(input.Substring(lastSplitLocation, input.Length - lastSplitLocation));

return elements;
}
}

/// &lt;summary&gt;
/// Class used to decode RFC 2822 Date header fields.
/// &lt;/summary&gt;
internal static class Rfc2822DateTime
{
/// &lt;summary&gt;
/// Converts a string in RFC 2822 format into a &lt;see cref="DateTime"/&gt; object
/// &lt;/summary&gt;
/// &lt;param name="inputDate"&gt;The date to convert&lt;/param&gt;
/// &lt;returns&gt;
/// A valid &lt;see cref="DateTime"/&gt; object, which represents the same time as the string that was conve
/// If &lt;paramref name="inputDate"/&gt; is not a valid date representation, then &lt;see cref="DateTime.Mi
/// &lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;&lt;exception cref="ArgumentNullException"&gt;If &lt;par
/// &lt;exception cref="ArgumentException"&gt;If the &lt;paramref name="inputDate"/&gt; could not be pars
public static DateTime StringToDate(string inputDate)
{
if (inputDate == null)
throw new ArgumentNullException("inputDate");

// Old date specification allows comments and a lot of whitespace
inputDate = StripCommentsAndExcessWhitespace(inputDate);

try
{
// Extract the DateTime
DateTime dateTime = ExtractDateTime(inputDate);

// If a day-name is specified in the inputDate string, check if it fits with the date
ValidateDayNameIfAny(dateTime, inputDate);

// Convert the date into UTC
dateTime = new DateTime(dateTime.Ticks, DateTimeKind.Utc);

// Adjust according to the time zone
dateTime = AdjustTimezone(dateTime, inputDate);

// Return the parsed date
```

```
return dateTime;
}
catch (FormatException e) // Convert.ToDateTime() Failure
{
throw new ArgumentException("Could not parse date: " + e.Message + ". Input was: \"" + inputDate + "\"", e
}
catch (ArgumentException e)
{
throw new ArgumentException("Could not parse date: " + e.Message + ". Input was: \"" + inputDate + "\"", e
}
}

/// &lt;summary&gt;
/// Adjust the &lt;paramref name="dateTime"/&gt; object given according to the timezone specified in the &l
/// &lt;/summary&gt;
/// &lt;param name="dateTime"&gt;The date to alter&lt;/param&gt;
/// &lt;param name="dateInput"&gt;The input date, in which the timezone can be found&lt;/param&gt;
/// &lt;returns&gt;An date altered according to the timezone&lt;/returns&gt;
/// &lt;exception cref="ArgumentException"&gt;If no timezone was found in &lt;paramref name="dateInput"/
private static DateTime AdjustTimezone(DateTime dateTime, string dateInput)
{
// We know that the timezones are always in the last part of the date input
string[] parts = dateInput.Split(' ');
string lastPart = parts[parts.Length - 1];

// Convert timezones in older formats to [+-]dddd format.
lastPart = Regex.Replace(lastPart, @"UT|GMT|EST|EDT|CST|CDT|MST|MDT|PST|PDT|[A-I]|[K-Y]|Z", Ma
// Find the timezone specification

// Example: Fri, 21 Nov 1997 09:55:06 -0600

// finds -0600

Match match = Regex.Match(lastPart, @"[\+-](?&lt;hours&gt;\d\d)(?&lt;minutes&gt;\d\d)");

if (match.Success)

{

// We have found that the timezone is in +dddd or -dddd format

// Add the number of hours and minutes to our found date

int hours = int.Parse(match.Groups["hours"].Value);

int minutes = int.Parse(match.Groups["minutes"].Value);


int factor = match.Value[0] == '+' ? -1 : 1;


dateTime = dateTime.AddHours(factor * hours);

dateTime = dateTime.AddMinutes(factor * minutes);


return dateTime;

}


//DefaultLogger.Log.LogDebug("No timezone found in date: " + dateInput + ". Using -0000 as default.");


// A timezone of -0000 is the same as doing nothing

return dateTime;

}


/// &lt;summary&gt;
/// Convert timezones in older formats to [+-]dddd format.
/// &lt;/summary&gt;
/// &lt;param name="match"&gt;The match that was found&lt;/param&gt;
/// &lt;returns&gt;The string to replace the matched string with&lt;/returns&gt;
private static string MatchEvaluator(Match match)
{
if (!match.Success)
{
throw new ArgumentException("Match success are always true");
}

switch (match.Value)
{
// "A" through "I"
// are equivalent to "+0100" through "+0900" respectively
case "A": return "+0100";
case "B": return "+0200";
case "C": return "+0300";
case "D": return "+0400";
case "E": return "+0500";
case "F": return "+0600";
case "G": return "+0700";
case "H": return "+0800";
case "I": return "+0900";

// "K", "L", and "M"
// are equivalent to "+1000", "+1100", and "+1200" respectively
```

```
case "K": return "+1000";
case "L": return "+1100";
case "M": return "+1200";

// "N" through "Y"
// are equivalent to "-0100" through "-1200" respectively
case "N": return "-0100";
case "O": return "-0200";
case "P": return "-0300";
case "Q": return "-0400";
case "R": return "-0500";
case "S": return "-0600";
case "T": return "-0700";
case "U": return "-0800";
case "V": return "-0900";
case "W": return "-1000";
case "X": return "-1100";
case "Y": return "-1200";

// "Z", "UT" and "GMT"
// is equivalent to "+0000"
case "Z":
case "UT":
case "GMT":
return "+0000";

// US time zones
```

```
case "EDT": return "-0400"; // EDT is semantically equivalent to -0400
case "EST": return "-0500"; // EST is semantically equivalent to -0500
case "CDT": return "-0500"; // CDT is semantically equivalent to -0500
case "CST": return "-0600"; // CST is semantically equivalent to -0600
case "MDT": return "-0600"; // MDT is semantically equivalent to -0600
case "MST": return "-0700"; // MST is semantically equivalent to -0700
case "PDT": return "-0700"; // PDT is semantically equivalent to -0700
case "PST": return "-0800"; // PST is semantically equivalent to -0800

default:
throw new ArgumentException("Unexpected input");
}
}

/// &lt;summary&gt;
/// Extracts the date and time parts from the &lt;paramref name="dateInput"/&gt;
/// &lt;/summary&gt;
/// &lt;param name="dateInput"&gt;The date input string, from which to extract the date and time parts&lt;/p
/// &lt;returns&gt;The extracted date part or &lt;see langword="DateTime.MinValue"/&gt; if &lt;paramref nar
private static DateTime ExtractDateTime(string dateInput)
{
// Matches the date and time part of a string
// Example: Fri, 21 Nov 1997 09:55:06 -0600
// Finds: 21 Nov 1997 09:55:06
// Seconds does not need to be specified
// Even though it is illigal, sometimes hours, minutes or seconds are only specified with one digit
Match match = Regex.Match(dateInput, @"\d\d? .+ (\d\d\d\d|\d\d) \d?\d:\d?\d(:\d?\d)?");
```

```
if (match.Success)
{
return Convert.ToDateTime(match.Value, CultureInfo.InvariantCulture);
}

//DefaultLogger.Log.LogError("The given date does not appear to be in a valid format: " + dateInput);
return DateTime.MinValue;
}

/// &lt;summary&gt;
/// Validates that the given &lt;paramref name="dateTime"/&gt; agrees with a day-name specified
/// in &lt;paramref name="dateInput"/&gt;.
/// &lt;/summary&gt;
/// &lt;param name="dateTime"&gt;The time to check&lt;/param&gt;
/// &lt;param name="dateInput"&gt;The date input to extract the day-name from&lt;/param&gt;
/// &lt;exception cref="ArgumentException"&gt;If &lt;paramref name="dateTime"/&gt; and &lt;paramref nam
private static void ValidateDayNameIfAny(DateTime dateTime, string dateInput)
{
// Check if there is a day name in front of the date
// Example: Fri, 21 Nov 1997 09:55:06 -0600
if (dateInput.Length &gt;= 4 &amp;amp;&amp;amp; dateInput[3] == ',')
{
string dayName = dateInput.Substring(0, 3);

// If a dayName was specified. Check that the dateTime and the dayName
// agrees on which day it is
// This is just a failure-check and could be left out
if ((dateTime.DayOfWeek == DayOfWeek.Monday &amp;amp;&amp;amp; !dayName.Equals("Mon")) ||

(dateTime.DayOfWeek == DayOfWeek.Tuesday &amp;amp;&amp;amp; !dayName.Equals("Tue")) ||

(dateTime.DayOfWeek == DayOfWeek.Wednesday &amp;amp;&amp;amp; !dayName.Equals("Wed")) ||

(dateTime.DayOfWeek == DayOfWeek.Thursday &amp;amp;&amp;amp; !dayName.Equals("Thu")) ||

(dateTime.DayOfWeek == DayOfWeek.Friday &amp;amp;&amp;amp; !dayName.Equals("Fri")) ||

(dateTime.DayOfWeek == DayOfWeek.Saturday &amp;amp;&amp;amp; !dayName.Equals("Sat")) ||

(dateTime.DayOfWeek == DayOfWeek.Sunday &amp;amp;&amp;amp; !dayName.Equals("Sun")))

{

//DefaultLogger.Log.LogDebug("Day-name does not correspond to the weekday of the date: " + dateInput)

}

}


// If no day name was found no checks can be made

}


/// &lt;summary&gt;

/// Strips and removes all comments and excessive whitespace from the string

/// &lt;/summary&gt;

/// &lt;param name="input"&gt;The input to strip from&lt;/param&gt;

/// &lt;returns&gt;The stripped string&lt;/returns&gt;

private static string StripCommentsAndExcessWhitespace(string input)

{

// Strip out comments

// Also strips out nested comments

input = Regex.Replace(input, @"(\((?&gt;\((?&lt;C&gt;)|\)(?&lt;-C&gt;)|.?)*(?(C)(?!))\))", "");


// Reduce any whitespace character to one space only
```

```
input = Regex.Replace(input, @"\s+", " ");

// Remove all initial whitespace
input = Regex.Replace(input, @"^\s+", "");

// Remove all ending whitespace
input = Regex.Replace(input, @"\s+$", "");

// Remove spaces at colons
// Example: 22: 33 : 44 =&gt; 22:33:44
input = Regex.Replace(input, @" ?: ?", ":");

return input;
}
}

/// &lt;summary&gt;
/// Utility class for dealing with Base64 encoded strings
/// &lt;/summary&gt;
internal static class Base64
{
/// &lt;summary&gt;
/// Decodes a base64 encoded string into the bytes it describes
/// &lt;/summary&gt;
/// &lt;param name="base64Encoded"&gt;The string to decode&lt;/param&gt;
/// &lt;returns&gt;A byte array that the base64 string described&lt;/returns&gt;
public static byte[] Decode(string base64Encoded)
```

```
{
try
{
return Convert.FromBase64String(base64Encoded);
}
catch //(FormatException e)
{
//DefaultLogger.Log.LogError("Base64: (FormatException) " + e.Message + "\r\nOn string: " + base64Enco
throw;
}
}

/// &lt;summary&gt;
/// Decodes a Base64 encoded string using a specified &lt;see cref="System.Text.Encoding"/&gt;
/// &lt;/summary&gt;
/// &lt;param name="base64Encoded"&gt;Source string to decode&lt;/param&gt;
/// &lt;param name="encoding"&gt;The encoding to use for the decoded byte array that &lt;paramref name=
/// &lt;returns&gt;A decoded string&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="base64Encoded"/&gt; or &lt;para
/// &lt;exception cref="FormatException"&gt;If &lt;paramref name="base64Encoded"/&gt; is not a valid base
public static string Decode(string base64Encoded, Encoding encoding)
{
if (base64Encoded == null)
throw new ArgumentNullException("base64Encoded");

if (encoding == null)
throw new ArgumentNullException("encoding");
```

```
return encoding.GetString(Decode(base64Encoded));

}

}


/// &lt;summary&gt;

/// Used for decoding Quoted-Printable text.&lt;br/&gt;

/// This is a robust implementation of a Quoted-Printable decoder defined in &lt;a href="http://tools.ietf.org/h

/// Every measurement has been taken to conform to the RFC.

/// &lt;/summary&gt;

internal static class QuotedPrintable

{

/// &lt;summary&gt;

/// Decodes a Quoted-Printable string according to &lt;a href="http://tools.ietf.org/html/rfc2047"&gt;RFC 204

/// RFC 2047 is used for decoding Encoded-Word encoded strings.

/// &lt;/summary&gt;

/// &lt;param name="toDecode"&gt;Quoted-Printable encoded string&lt;/param&gt;

/// &lt;param name="encoding"&gt;Specifies which encoding the returned string will be in&lt;/param&gt;

/// &lt;returns&gt;A decoded string in the correct encoding&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="toDecode"/&gt; or &lt;paramref na

public static string DecodeEncodedWord(string toDecode, Encoding encoding)

{

if (toDecode == null)

throw new ArgumentNullException("toDecode");


if (encoding == null)

throw new ArgumentNullException("encoding");
```

```
// Decode the QuotedPrintable string and return it

return encoding.GetString(Rfc2047QuotedPrintableDecode(toDecode, true));

}


/// &lt;summary&gt;

/// Decodes a Quoted-Printable string according to &lt;a href="http://tools.ietf.org/html/rfc2045"&gt;RFC 204

/// RFC 2045 specifies the decoding of a body encoded with Content-Transfer-Encoding of quoted-printabl

/// &lt;/summary&gt;

/// &lt;param name="toDecode"&gt;Quoted-Printable encoded string&lt;/param&gt;

/// &lt;returns&gt;A decoded byte array that the Quoted-Printable encoded string described&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="toDecode"/&gt; is &lt;see langwor

public static byte[] DecodeContentTransferEncoding(string toDecode)

{

if (toDecode == null)

throw new ArgumentNullException("toDecode");


// Decode the QuotedPrintable string and return it

return Rfc2047QuotedPrintableDecode(toDecode, false);

}


/// &lt;summary&gt;

/// This is the actual decoder.

/// &lt;/summary&gt;

/// &lt;param name="toDecode"&gt;The string to be decoded from Quoted-Printable&lt;/param&gt;

/// &lt;param name="encodedWordVariant"&gt;

/// If &lt;see langword="true"/&gt;, specifies that RFC 2047 quoted printable decoding is used.&lt;br/&gt;
/// This is for quoted-printable encoded words&lt;br/&gt;
/// &lt;br/&gt;
/// If &lt;see langword="false"/&gt;, specifies that RFC 2045 quoted printable decoding is used.&lt;br/&gt;
/// This is for quoted-printable Content-Transfer-Encoding
/// &lt;/param&gt;
/// &lt;returns&gt;A decoded byte array that was described by &lt;paramref name="toDecode"/&gt;&lt;/retur
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="toDecode"/&gt; is &lt;see langwor
/// &lt;remarks&gt;See &lt;a href="http://tools.ietf.org/html/rfc2047#section-4.2"&gt;RFC 2047 section 4.2&lt
private static byte[] Rfc2047QuotedPrintableDecode(string toDecode, bool encodedWordVariant)
{
if (toDecode == null)
throw new ArgumentNullException("toDecode");

// Create a byte array builder which is roughly equivalent to a StringBuilder
using (MemoryStream byteArrayBuilder = new MemoryStream())
{
// Remove illegal control characters
toDecode = RemoveIllegalControlCharacters(toDecode);

// Run through the whole string that needs to be decoded
for (int i = 0; i &lt; toDecode.Length; i++)
{
char currentChar = toDecode[i];
if (currentChar == '=')
{
// Check that there is at least two characters behind the equal sign
if (toDecode.Length - i &lt; 3)
```

```
{
// We are at the end of the toDecode string, but something is missing. Handle it the way RFC 2045 states
WriteAllBytesToStream(byteArrayBuilder, DecodeEqualSignNotLongEnough(toDecode.Substring(i)));

// Since it was the last part, we should stop parsing anymore
break;
}

// Decode the Quoted-Printable part
string quotedPrintablePart = toDecode.Substring(i, 3);
WriteAllBytesToStream(byteArrayBuilder, DecodeEqualSign(quotedPrintablePart));

// We now consumed two extra characters. Go forward two extra characters
i += 2;
}
else
{
// This character is not quoted printable hex encoded.

// Could it be the _ character, which represents space
// and are we using the encoded word variant of QuotedPrintable
if (currentChar == '_' &amp;amp;&amp;amp; encodedWordVariant)
{
// The RFC specifies that the "_" always represents hexadecimal 20 even if the
// SPACE character occupies a different code position in the character set in use.
byteArrayBuilder.WriteByte(0x20);
}
```

```
else

{

// This is not encoded at all. This is a literal which should just be included into the output.

byteArrayBuilder.WriteByte((byte)currentChar);

}

}

}


return byteArrayBuilder.ToArray();

}

}


/// &lt;summary&gt;

/// Writes all bytes in a byte array to a stream

/// &lt;/summary&gt;

/// &lt;param name="stream"&gt;The stream to write to&lt;/param&gt;

/// &lt;param name="toWrite"&gt;The bytes to write to the &lt;paramref name="stream"/&gt;&lt;/param&gt;

private static void WriteAllBytesToStream(Stream stream, byte[] toWrite)

{

stream.Write(toWrite, 0, toWrite.Length);

}


/// &lt;summary&gt;

/// RFC 2045 states about robustness:&lt;br/&gt;

/// &lt;code&gt;

/// Control characters other than TAB, or CR and LF as parts of CRLF pairs,

/// must not appear. The same is true for octets with decimal values greater
/// than 126.  If found in incoming quoted-printable data by a decoder, a
/// robust implementation might exclude them from the decoded data and warn
/// the user that illegal characters were discovered.
/// &lt;/code&gt;
/// Control characters are defined in RFC 2396 as&lt;br/&gt;
/// &lt;c&gt;control = US-ASCII coded characters 00-1F and 7F hexadecimal&lt;/c&gt;
/// &lt;/summary&gt;
/// &lt;param name="input"&gt;String to be stripped from illegal control characters&lt;/param&gt;
/// &lt;returns&gt;A string with no illegal control characters&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="input"/&gt; is &lt;see langword="n
private static string RemoveIllegalControlCharacters(string input)
{
if (input == null)
throw new ArgumentNullException("input");

// First we remove any \r or \n which is not part of a \r\n pair
input = RemoveCarriageReturnAndNewLinewIfNotInPair(input);

// Here only legal \r\n is left over
// We now simply keep them, and the \t which is also allowed
// \x0A = \n
// \x0D = \r
// \x09 = \t)
return Regex.Replace(input, "[\x00-\x08\x0B\x0C\x0E-\x1F\x7F]", "");
}

/// &lt;summary&gt;
/// This method will remove any \r and \n which is not paired as \r\n
/// &lt;/summary&gt;
/// &lt;param name="input"&gt;String to remove lonely \r and \n's from&lt;/param&gt;
/// &lt;returns&gt;A string without lonely \r and \n's&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="input"/&gt; is &lt;see langword="n
private static string RemoveCarriageReturnAndNewLinewIfNotInPair(string input)
{
if (input == null)
throw new ArgumentNullException("input");

// Use this for building up the new string. This is used for performance instead
// of altering the input string each time a illegal token is found
StringBuilder newString = new StringBuilder(input.Length);

for (int i = 0; i &lt; input.Length; i++)
{
// There is a character after it
// Check for lonely \r
// There is a lonely \r if it is the last character in the input or if there
// is no \n following it
if (input[i] == '\r' &amp;amp;&amp;amp; (i + 1 &gt;= input.Length || input[i + 1] != '\n'))
{
// Illegal token \r found. Do not add it to the new string

// Check for lonely \n
// There is a lonely \n if \n is the first character or if there
// is no \r in front of it
}
else if (input[i] == '\n' &amp;amp;&amp;amp; (i - 1 &lt; 0 || input[i - 1] != '\r'))
{
// Illegal token \n found. Do not add it to the new string
}
else
{
// No illegal tokens found. Simply insert the character we are at
// in our new string
newString.Append(input[i]);
}
}

return newString.ToString();
}

/// &lt;summary&gt;
/// RFC 2045 says that a robust implementation should handle:&lt;br/&gt;
/// &lt;code&gt;
/// An "=" cannot be the ultimate or penultimate character in an encoded
/// object. This could be handled as in case (2) above.
/// &lt;/code&gt;
/// Case (2) is:&lt;br/&gt;
/// &lt;code&gt;
/// An "=" followed by a character that is neither a
/// hexadecimal digit (including "abcdef") nor the CR character of a CRLF pair
/// is illegal.  This case can be the result of US-ASCII text having been
/// included in a quoted-printable part of a message without itself having

/// been subjected to quoted-printable encoding.  A reasonable approach by a

/// robust implementation might be to include the "=" character and the

/// following character in the decoded data without any transformation and, if

/// possible, indicate to the user that proper decoding was not possible at

/// this point in the data.

/// &lt;/code&gt;

/// &lt;/summary&gt;

/// &lt;param name="decode"&gt;

/// The string to decode which cannot have length above or equal to 3

/// and must start with an equal sign.

/// &lt;/param&gt;

/// &lt;returns&gt;A decoded byte array&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="decode"/&gt; is &lt;see langword=

/// &lt;exception cref="ArgumentException"&gt;Thrown if a the &lt;paramref name="decode"/&gt; parameter

private static byte[] DecodeEqualSignNotLongEnough(string decode)

{

if (decode == null)

throw new ArgumentNullException("decode");



// We can only decode wrong length equal signs

if (decode.Length &gt;= 3)

throw new ArgumentException("decode must have length lower than 3", "decode");



// First char must be =

if (decode[0] != '=')

throw new ArgumentException("First part of decode must be an equal sign", "decode");
```

```
// We will now believe that the string sent to us, was actually not encoded
// Therefore it must be in US-ASCII and we will return the bytes it corrosponds to
return Encoding.ASCII.GetBytes(decode);
}

/// &lt;summary&gt;
/// This helper method will decode a string of the form "=XX" where X is any character.&lt;br/&gt;
/// This method will never fail, unless an argument of length not equal to three is passed.
/// &lt;/summary&gt;
/// &lt;param name="decode"&gt;The length 3 character that needs to be decoded&lt;/param&gt;
/// &lt;returns&gt;A decoded byte array&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="decode"/&gt; is &lt;see langword=
/// &lt;exception cref="ArgumentException"&gt;Thrown if a the &lt;paramref name="decode"/&gt; parameter
private static byte[] DecodeEqualSign(string decode)
{
if (decode == null)
throw new ArgumentNullException("decode");

// We can only decode the string if it has length 3 - other calls to this function is invalid
if (decode.Length != 3)
throw new ArgumentException("decode must have length 3", "decode");

// First char must be =
if (decode[0] != '=')
throw new ArgumentException("decode must start with an equal sign", "decode");
```

```
// There are two cases where an equal sign might appear
// It might be a
//   - hex-string like =3D, denoting the character with hex value 3D
//   - it might be the last character on the line before a CRLF
//     pair, denoting a soft linebreak, which simply
//     splits the text up, because of the 76 chars per line restriction
if (decode.Contains("\r\n"))
{
// Soft break detected
// We want to return string.Empty which is equivalent to a zero-length byte array
return new byte[0];
}

// Hex string detected. Convertion needed.
// It might be that the string located after the equal sign is not hex characters
// An example: =JU
// In that case we would like to catch the FormatException and do something else
try
{
// The number part of the string is the last two digits. Here we simply remove the equal sign
string numberString = decode.Substring(1);

// Now we create a byte array with the converted number encoded in the string as a hex value (base 16)
// This will also handle illegal encodings like =3d where the hex digits are not uppercase,
// which is a robustness requirement from RFC 2045.
byte[] oneByte = { Convert.ToByte(numberString, 16) };
// Simply return our one byte byte array
return oneByte;
} catch (FormatException)
{
// RFC 2045 says about robust implementation:
// An "=" followed by a character that is neither a
// hexadecimal digit (including "abcdef") nor the CR
// character of a CRLF pair is illegal.  This case can be
// the result of US-ASCII text having been included in a
// quoted-printable part of a message without itself
// having been subjected to quoted-printable encoding.  A
// reasonable approach by a robust implementation might be
// to include the "=" character and the following
// character in the decoded data without any
// transformation and, if possible, indicate to the user
// that proper decoding was not possible at this point in
// the data.

// So we choose to believe this is actually an un-encoded string
// Therefore it must be in US-ASCII and we will return the bytes it corrosponds to
return Encoding.ASCII.GetBytes(decode);
}
}
}

///&lt;summary&gt;
/// Utility class that divides a message into a body and a header.&lt;br/&gt;
/// The header is then parsed to a strongly typed &lt;see cref="MessageHeader"/&gt; object.

///&lt;/summary&gt;

internal static class HeaderExtractor

{

/// &lt;summary&gt;

/// Find the end of the header section in a byte array.&lt;br/&gt;

/// The headers have ended when a blank line is found

/// &lt;/summary&gt;

/// &lt;param name="messageContent"&gt;The full message stored as a byte array&lt;/param&gt;

/// &lt;returns&gt;The position of the line just after the header end blank line&lt;/returns&gt;

private static int FindHeaderEndPosition(byte[] messageContent)

{

// Convert the byte array into a stream

using (Stream stream = new MemoryStream(messageContent))

{

while (true)

{

// Read a line from the stream. We know headers are in US-ASCII

// therefore it is not problem to read them as such

string line = StreamUtility.ReadLineAsAscii(stream);



// The end of headers is signaled when a blank line is found

// or if the line is null - in which case the email is actually an email with

// only headers but no body

if (string.IsNullOrEmpty(line))

return (int)stream.Position;

}
```

```
}
}

/// &lt;summary&gt;
/// Extract the header part and body part of a message.&lt;br/&gt;
/// The headers are then parsed to a strongly typed &lt;see cref="MessageHeader"/&gt; object.
/// &lt;/summary&gt;
/// &lt;param name="fullRawMessage"&gt;The full message in bytes where header and body needs to be e
/// &lt;param name="headers"&gt;The extracted header parts of the message&lt;/param&gt;
/// &lt;param name="body"&gt;The body part of the message&lt;/param&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="fullRawMessage"/&gt; is &lt;see l
public static void ExtractHeadersAndBody(byte[] fullRawMessage, out MessageHeader headers, out byte[]
{
if (fullRawMessage == null)
throw new ArgumentNullException("fullRawMessage");

// Find the end location of the headers
int endOfHeaderLocation = FindHeaderEndPosition(fullRawMessage);

// The headers are always in ASCII - therefore we can convert the header part into a string
// using US-ASCII encoding
string headersString = Encoding.ASCII.GetString(fullRawMessage, 0, endOfHeaderLocation);

// Now parse the headers to a NameValueCollection
NameValueCollection headersUnparsedCollection = ExtractHeaders(headersString);

// Use the NameValueCollection to parse it into a strongly-typed MessageHeader header
```

```
headers = new MessageHeader(headersUnparsedCollection);


// Since we know where the headers end, we also know where the body is

// Copy the body part into the body parameter

body = new byte[fullRawMessage.Length - endOfHeaderLocation];

Array.Copy(fullRawMessage, endOfHeaderLocation, body, 0, body.Length);

}


/// &lt;summary&gt;

/// Method that takes a full message and extract the headers from it.

/// &lt;/summary&gt;

/// &lt;param name="messageContent"&gt;The message to extract headers from. Does not need the body p

/// &lt;returns&gt;A collection of Name and Value pairs of headers&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="messageContent"/&gt; is &lt;see l

private static NameValueCollection ExtractHeaders(string messageContent)

{

if (messageContent == null)

throw new ArgumentNullException("messageContent");


NameValueCollection headers = new NameValueCollection();


using (StringReader messageReader = new StringReader(messageContent))

{

// Read until all headers have ended.

// The headers ends when an empty line is encountered

// An empty message might actually not have an empty line, in which

// case the headers end with null value.
```

```
string line;
while (!string.IsNullOrEmpty(line = messageReader.ReadLine()))
{
// Split into name and value
KeyValuePair&lt;string, string&gt; header = SeparateHeaderNameAndValue(line);

// First index is header name
string headerName = header.Key;

// Second index is the header value.
// Use a StringBuilder since the header value may be continued on the next line
StringBuilder headerValue = new StringBuilder(header.Value);

// Keep reading until we would hit next header
// This if for handling multi line headers
while (IsMoreLinesInHeaderValue(messageReader))
{
// Unfolding is accomplished by simply removing any CRLF
// that is immediately followed by WSP
// This was done using ReadLine (it discards CRLF)
// See http://tools.ietf.org/html/rfc822#section-3.1.1 for more information
string moreHeaderValue = messageReader.ReadLine();

// If this exception is ever raised, there is an serious algorithm failure
// IsMoreLinesInHeaderValue does not return true if the next line does not exist
// This check is only included to stop the nagging "possibly null" code analysis hint
if (moreHeaderValue == null)
throw new ArgumentException("This will never happen");


// Simply append the line just read to the header value

headerValue.Append(moreHeaderValue);

}


// Now we have the name and full value. Add it

headers.Add(headerName, headerValue.ToString());

}

}


return headers;

}


/// &lt;summary&gt;

/// Check if the next line is part of the current header value we are parsing by

/// peeking on the next character of the &lt;see cref="TextReader"/&gt;.&lt;br/&gt;

/// This should only be called while parsing headers.

/// &lt;/summary&gt;

/// &lt;param name="reader"&gt;The reader from which the header is read from&lt;/param&gt;

/// &lt;returns&gt;&lt;see langword="true"/&gt; if multi-line header. &lt;see langword="false"/&gt; otherwise&

private static bool IsMoreLinesInHeaderValue(TextReader reader)

{

int peek = reader.Peek();

if (peek == -1)

return false;
```

```
char peekChar = (char)peek;


// A multi line header must have a whitespace character

// on the next line if it is to be continued

return peekChar == ' ' || peekChar == '\t';

}


/// &lt;summary&gt;

/// Separate a full header line into a header name and a header value.

/// &lt;/summary&gt;

/// &lt;param name="rawHeader"&gt;The raw header line to be separated&lt;/param&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="rawHeader"/&gt; is &lt;see langwo

internal static KeyValuePair&lt;string, string&gt; SeparateHeaderNameAndValue(string rawHeader)

{

if (rawHeader == null)

throw new ArgumentNullException("rawHeader");


string key = string.Empty;

string value = string.Empty;


int indexOfColon = rawHeader.IndexOf(':');


// Check if it is allowed to make substring calls

if (indexOfColon &gt;= 0 &amp;amp;&amp;amp; rawHeader.Length &gt;= indexOfColon + 1)

{

key = rawHeader.Substring(0, indexOfColon).Trim();

value = rawHeader.Substring(indexOfColon + 1).Trim();
}


return new KeyValuePair&lt;string, string&gt;(key, value);

}

}


/// &lt;summary&gt;

/// Utility class used by OpenPop for mapping from a characterSet to an &lt;see cref="Encoding"/&gt;.&lt;br/

/// &lt;br/&gt;

/// The functionality of the class can be altered by adding mappings

/// using &lt;see cref="AddMapping"/&gt; and by adding a &lt;see cref="FallbackDecoder"/&gt;.&lt;br/&gt;

/// &lt;br/&gt;

/// Given a characterSet, it will try to find the Encoding as follows:

/// &lt;list type="number"&gt;

///     &lt;item&gt;

///         &lt;description&gt;If a mapping for the characterSet was added, use the specified Encoding from th

///     &lt;/item&gt;

///     &lt;item&gt;

///         &lt;description&gt;Try to parse the characterSet and look it up using &lt;see cref="Encoding.GetEnc

///     &lt;/item&gt;

///     &lt;item&gt;

///         &lt;description&gt;If an encoding is not found yet, use the &lt;see cref="FallbackDecoder"/&gt; if def

///     &lt;/item&gt;

/// &lt;/list&gt;

/// &lt;/summary&gt;

public static class EncodingFinder

{
```

```
/// &lt;summary&gt;
/// Delegate that is used when the EncodingFinder is unable to find an encoding by
/// using the &lt;see cref="EncodingFinder.EncodingMap"/&gt; or general code.&lt;br/&gt;
/// This is used as a last resort and can be used for setting a default encoding or
/// for finding an encoding on runtime for some &lt;paramref name="characterSet"/&gt;.
/// &lt;/summary&gt;
/// &lt;param name="characterSet"&gt;The character set to find an encoding for.&lt;/param&gt;
/// &lt;returns&gt;An encoding for the &lt;paramref name="characterSet"/&gt; or &lt;see langword="null"/&gt
public delegate Encoding FallbackDecoderDelegate(string characterSet);

/// &lt;summary&gt;
/// Last resort decoder. &lt;seealso cref="FallbackDecoderDelegate"/&gt;.
/// &lt;/summary&gt;
public static FallbackDecoderDelegate FallbackDecoder { private get { return _fallbackDecoder; } set { _fall
private static FallbackDecoderDelegate _fallbackDecoder;

/// &lt;summary&gt;
/// Mapping from charactersets to encodings.
/// &lt;/summary&gt;
private static Dictionary&lt;string, Encoding&gt; EncodingMap { get { return _encodingMap; } set { _encodin
private static Dictionary&lt;string, Encoding&gt; _encodingMap;

/// &lt;summary&gt;
/// Initialize the EncodingFinder
/// &lt;/summary&gt;
static EncodingFinder()
{
```

```
Reset();
}

/// &lt;summary&gt;
/// Used to reset this static class to facilite isolated unit testing.
/// &lt;/summary&gt;
internal static void Reset()
{
EncodingMap = new Dictionary&lt;string, Encoding&gt;();
FallbackDecoder = null;

// Some emails incorrectly specify the encoding as utf8, but it should have been utf-8.
AddMapping("utf8", Encoding.UTF8);
}

/// &lt;summary&gt;
/// Parses a character set into an encoding.
/// &lt;/summary&gt;
/// &lt;param name="characterSet"&gt;The character set to parse&lt;/param&gt;
/// &lt;returns&gt;An encoding which corresponds to the character set&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="characterSet"/&gt; is &lt;see lang
internal static Encoding FindEncoding(string characterSet)
{
if (characterSet == null)
throw new ArgumentNullException("characterSet");

string charSetUpper = characterSet.ToUpperInvariant();
```

```
// Check if the characterSet is explicitly mapped to an encoding
if (EncodingMap.ContainsKey(charSetUpper))
return EncodingMap[charSetUpper];

// Try to find the generally find the encoding
try
{
if (charSetUpper.Contains("WINDOWS") || charSetUpper.Contains("CP"))
{
// It seems the characterSet contains an codepage value, which we should use to parse the encoding
charSetUpper = charSetUpper.Replace("CP", ""); // Remove cp
charSetUpper = charSetUpper.Replace("WINDOWS", ""); // Remove windows
charSetUpper = charSetUpper.Replace("-", ""); // Remove - which could be used as cp-1554

// Now we hope the only thing left in the characterSet is numbers.
int codepageNumber = int.Parse(charSetUpper, CultureInfo.InvariantCulture);

return Encoding.GetEncoding(codepageNumber);
}

// It seems there is no codepage value in the characterSet. It must be a named encoding
return Encoding.GetEncoding(characterSet);
}
catch (ArgumentException)
{
// The encoding could not be found generally.
// Try to use the FallbackDecoder if it is defined.

// Check if it is defined
if (FallbackDecoder == null)
throw; // It was not defined - throw catched exception

// Use the FallbackDecoder
Encoding fallbackDecoderResult = FallbackDecoder(characterSet);

// Check if the FallbackDecoder had a solution
if (fallbackDecoderResult != null)
return fallbackDecoderResult;

// If no solution was found, throw catched exception
throw;
}
}

/// &lt;summary&gt;
/// Puts a mapping from &lt;paramref name="characterSet"/&gt; to &lt;paramref name="encoding"/&gt;
/// into the &lt;see cref="EncodingFinder"/&gt;'s internal mapping Dictionary.
/// &lt;/summary&gt;
/// &lt;param name="characterSet"&gt;The string that maps to the &lt;paramref name="encoding"/&gt;&lt;/p
/// &lt;param name="encoding"&gt;The &lt;see cref="Encoding"/&gt; that should be mapped from &lt;param
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="characterSet"/&gt; is &lt;see lang
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="encoding"/&gt; is &lt;see langwor
public static void AddMapping(string characterSet, Encoding encoding)
```

```
{

if (characterSet == null)

throw new ArgumentNullException("characterSet");


if (encoding == null)

throw new ArgumentNullException("encoding");


// Add the mapping using uppercase

EncodingMap.Add(characterSet.ToUpperInvariant(), encoding);

}

}


/// &lt;summary&gt;

/// A MessagePart is a part of an email message used to describe the whole email parse tree.&lt;br/&gt;

/// &lt;br/&gt;

/// &lt;b&gt;Email messages are tree structures&lt;/b&gt;:&lt;br/&gt;

/// Email messages may contain large tree structures, and the MessagePart are the nodes of the this struct

/// A MessagePart may either be a leaf in the structure or a internal node with links to other MessageParts.

/// The root of the message tree is the &lt;see cref="Message"/&gt; class.&lt;br/&gt;

/// &lt;br/&gt;

/// &lt;b&gt;Leafs&lt;/b&gt;:&lt;br/&gt;

/// If a MessagePart is a leaf, the part is not a &lt;see cref="IsMultiPart"&gt;MultiPart&lt;/see&gt; message.&

/// Leafs are where the contents of an email are placed.&lt;br/&gt;

/// This includes, but is not limited to: attachments, text or images referenced from HTML.&lt;br/&gt;

/// The content of an attachment can be fetched by using the &lt;see cref="Body"/&gt; property.&lt;br/&gt;

/// If you want to have the text version of a MessagePart, use the &lt;see cref="GetBodyAsText"/&gt; metho

/// convert the &lt;see cref="Body"/&gt; into a string using the encoding the message was sent with.&lt;br/&
/// &lt;br/&gt;
/// &lt;b&gt;Internal nodes&lt;/b&gt;:&lt;br/&gt;
/// If a MessagePart is an internal node in the email tree structure, then the part is a &lt;see cref="IsMultiPa
/// The &lt;see cref="MessageParts"/&gt; property will then contain links to the parts it contain.&lt;br/&gt;
/// The &lt;see cref="Body"/&gt; property of the MessagePart will not be set.&lt;br/&gt;
/// &lt;br/&gt;
/// See the example for a parsing example.&lt;br/&gt;
/// This class cannot be instantiated from outside the library.
/// &lt;/summary&gt;
/// &lt;example&gt;
/// This example illustrates how the message parse tree looks like given a specific message&lt;br/&gt;
/// &lt;br/&gt;
/// The message source in this example is:&lt;br/&gt;
/// &lt;code&gt;
/// MIME-Version: 1.0
/// Content-Type: multipart/mixed; boundary="frontier"
///
/// This is a message with multiple parts in MIME format.
/// --frontier
/// Content-Type: text/plain
///
/// This is the body of the message.
/// --frontier
/// Content-Type: application/octet-stream
/// Content-Transfer-Encoding: base64
///
/// PGh0bWw+CiAgPGHLYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB0aGUg
/// Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==

/// --frontier--

/// &lt;/code&gt;

/// The tree will look as follows, where the content-type media type of the message is listed&lt;br/&gt;

/// &lt;code&gt;

/// - Message root

///   - multipart/mixed MessagePart

///     - text/plain MessagePart

///     - application/octet-stream MessagePart

/// &lt;/code&gt;

/// It is possible to have more complex message trees like the following:&lt;br/&gt;

/// &lt;code&gt;

/// - Message root

///   - multipart/mixed MessagePart

///     - text/plain MessagePart

///     - text/plain MessagePart

///     - multipart/parallel

///       - audio/basic

///       - image/tiff

///     - text/enriched

///     - message/rfc822

/// &lt;/code&gt;

/// But it is also possible to have very simple message trees like:&lt;br/&gt;

/// &lt;code&gt;

/// - Message root

///   - text/plain

/// &lt;/code&gt;
/// &lt;/example&gt;
public class MessagePart
{
#region Public properties
/// &lt;summary&gt;
/// The Content-Type header field.&lt;br/&gt;
/// &lt;br/&gt;
/// If not set, the ContentType is created by the default "text/plain; charset=us-ascii" which is
/// defined in &lt;a href="http://tools.ietf.org/html/rfc2045#section-5.2"&gt;RFC 2045 section 5.2&lt;/a&gt;.&lt
/// &lt;br/&gt;
/// If set, the default is overridden.
/// &lt;/summary&gt;
public ContentType ContentType { get { return _contentType; } private set { _contentType = value; } }
private ContentType _contentType;

/// &lt;summary&gt;
/// A human readable description of the body&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Content-Description header was present in the message.&lt;br/&gt;
/// &lt;/summary&gt;
public string ContentDescription { get { return _contentDescription; } private set { _contentDescription = val
private string _contentDescription;

/// &lt;summary&gt;
/// This header describes the Content encoding during transfer.&lt;br/&gt;
/// &lt;br/&gt;
/// If no Content-Transfer-Encoding header was present in the message, it is set
/// to the default of &lt;see cref="Header.ContentTransferEncoding.SevenBit"&gt;SevenBit&lt;/see&gt; in ac
/// &lt;/summary&gt;
/// &lt;remarks&gt;See &lt;a href="http://tools.ietf.org/html/rfc2045#section-6"&gt;RFC 2045 section 6&lt;/a&
public ContentTransferEncoding ContentTransferEncoding { get { return _contentTransferEncoding; } priva
private ContentTransferEncoding _contentTransferEncoding;

/// &lt;summary&gt;
/// ID of the content part (like an attached image). Used with MultiPart messages.&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Content-ID header field was present in the message.
/// &lt;/summary&gt;
public string ContentId { get { return _contentId; } private set { _contentId = value; } }
private string _contentId;

/// &lt;summary&gt;
/// Used to describe if a &lt;see cref="MessagePart"/&gt; is to be displayed or to be though of as an attachn
/// Also contains information about filename if such was sent.&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Content-Disposition header field was present in the message
/// &lt;/summary&gt;
public ContentDisposition ContentDisposition { get { return _contentDisposition; } private set { _contentDisp
private ContentDisposition _contentDisposition;

/// &lt;summary&gt;
/// This is the encoding used to parse the message body if the &lt;see cref="MessagePart"/&gt;&lt;br/&gt;
/// is not a MultiPart message. It is derived from the &lt;see cref="ContentType"/&gt; character set property.
/// &lt;/summary&gt;
```

```
public Encoding BodyEncoding { get { return _bodyEncoding; } private set { _bodyEncoding = value; } }

private Encoding _bodyEncoding;


/// &lt;summary&gt;

/// This is the parsed body of this &lt;see cref="MessagePart"/&gt;.&lt;br/&gt;

/// It is parsed in that way, if the body was ContentTransferEncoded, it has been decoded to the

/// correct bytes.&lt;br/&gt;

/// &lt;br/&gt;

/// It will be &lt;see langword="null"/&gt; if this &lt;see cref="MessagePart"/&gt; is a MultiPart message.&lt;b

/// Use &lt;see cref="IsMultiPart"/&gt; to check if this &lt;see cref="MessagePart"/&gt; is a MultiPart messag

/// &lt;/summary&gt;

public byte[] Body { get { return _body; } private set { _body = value; } }

private byte[] _body;


/// &lt;summary&gt;

/// Describes if this &lt;see cref="MessagePart"/&gt; is a MultiPart message&lt;br/&gt;

/// &lt;br/&gt;

/// The &lt;see cref="MessagePart"/&gt; is a MultiPart message if the &lt;see cref="ContentType"/&gt; medi

/// &lt;/summary&gt;

public bool IsMultiPart

{

get

{

return ContentType.MediaType.StartsWith("multipart/", StringComparison.OrdinalIgnoreCase);

}

}
```

```
/// &lt;summary&gt;
/// A &lt;see cref="MessagePart"/&gt; is considered to be holding text in it's body if the MediaType
/// starts either "text/" or is equal to "message/rfc822"
/// &lt;/summary&gt;
public bool IsText
{
get
{
string mediaType = ContentType.MediaType;
return mediaType.StartsWith("text/", StringComparison.OrdinalIgnoreCase) || mediaType.Equals("messag
}
}

/// &lt;summary&gt;
/// A &lt;see cref="MessagePart"/&gt; is considered to be an attachment, if&lt;br/&gt;
/// - it is not holding &lt;see cref="IsText"&gt;text&lt;/see&gt; and is not a &lt;see cref="IsMultiPart"&gt;Multi
/// or&lt;br/&gt;
/// - it has a Content-Disposition header that says it is an attachment
/// &lt;/summary&gt;
public bool IsAttachment
{
get
{
// Inline is the opposite of attachment
return (!IsText &amp;amp;&amp;amp; !IsMultiPart) || (ContentDisposition != null &amp;amp;&amp;amp; !C
}
}
```

```
/// &lt;summary&gt;
/// This is a convenient-property for figuring out a FileName for this &lt;see cref="MessagePart"/&gt;.&lt;br/&
/// If the &lt;see cref="MessagePart"/&gt; is a MultiPart message, then it makes no sense to try to find a File
/// &lt;br/&gt;
/// The FileName can be specified in the &lt;see cref="ContentDisposition"/&gt; or in the &lt;see cref="Conte
/// If none of these places two places tells about the FileName, a default "(no name)" is returned.
/// &lt;/summary&gt;
public string FileName { get { return _fileName; } private set { _fileName = value; } }
private string _fileName;

/// &lt;summary&gt;
/// If this &lt;see cref="MessagePart"/&gt; is a MultiPart message, then this property
/// has a list of each of the Multiple parts that the message consists of.&lt;br/&gt;
/// &lt;br/&gt;
/// It is &lt;see langword="null"/&gt; if it is not a MultiPart message.&lt;br/&gt;
/// Use &lt;see cref="IsMultiPart"/&gt; to check if this &lt;see cref="MessagePart"/&gt; is a MultiPart messag
/// &lt;/summary&gt;
public List&lt;MessagePart&gt; MessageParts { get { return _messageParts; } private set { _messageParts
private List&lt;MessagePart&gt; _messageParts;
#endregion

#region Constructors
/// &lt;summary&gt;
/// Used to construct the topmost message part
/// &lt;/summary&gt;
/// &lt;param name="rawBody"&gt;The body that needs to be parsed&lt;/param&gt;
/// &lt;param name="headers"&gt;The headers that should be used from the message&lt;/param&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="rawBody"/&gt; or &lt;paramref nar
internal MessagePart(byte[] rawBody, MessageHeader headers)
{
if (rawBody == null)
throw new ArgumentNullException("rawBody");

if (headers == null)
throw new ArgumentNullException("headers");

ContentType = headers.ContentType;
ContentDescription = headers.ContentDescription;
ContentTransferEncoding = headers.ContentTransferEncoding;
ContentId = headers.ContentId;
ContentDisposition = headers.ContentDisposition;

FileName = FindFileName(ContentType, ContentDisposition, "(no name)");
BodyEncoding = ParseBodyEncoding(ContentType.CharSet);

ParseBody(rawBody);
}
#endregion

#region Parsing
/// &lt;summary&gt;
/// Parses a character set into an encoding
/// &lt;/summary&gt;
/// &lt;param name="characterSet"&gt;The character set that needs to be parsed. &lt;see langword="null"/&

/// &lt;returns&gt;The encoding specified by the &lt;paramref name="characterSet"/&gt; parameter, or ASCI

private static Encoding ParseBodyEncoding(string characterSet)

{

// Default encoding in Mime messages is US-ASCII

Encoding encoding = Encoding.ASCII;


// If the character set was specified, find the encoding that the character

// set describes, and use that one instead

if (!string.IsNullOrEmpty(characterSet))

encoding = EncodingFinder.FindEncoding(characterSet);


return encoding;

}


/// &lt;summary&gt;

/// Figures out the filename of this message part from some headers.

/// &lt;see cref="FileName"/&gt; property.

/// &lt;/summary&gt;

/// &lt;param name="contentType"&gt;The Content-Type header&lt;/param&gt;

/// &lt;param name="contentDisposition"&gt;The Content-Disposition header&lt;/param&gt;

/// &lt;param name="defaultName"&gt;The default filename to use, if no other could be found&lt;/param&gt;

/// &lt;returns&gt;The filename found, or the default one if not such filename could be found in the headers&

/// &lt;exception cref="ArgumentNullException"&gt;if &lt;paramref name="contentType"/&gt; is &lt;see langw

private static string FindFileName(ContentType contentType, ContentDisposition contentDisposition, string

{

if (contentType == null)
```

```
throw new ArgumentNullException("contentType");

if (contentDisposition != null &amp;amp;&amp;amp; contentDisposition.FileName != null)
return contentDisposition.FileName;

if (contentType.Name != null)
return contentType.Name;

return defaultName;
}

/// &lt;summary&gt;
/// Parses a byte array as a body of an email message.
/// &lt;/summary&gt;
/// &lt;param name="rawBody"&gt;The byte array to parse as body of an email message. This array may no
private void ParseBody(byte[] rawBody)
{
if (IsMultiPart)
{
// Parses a MultiPart message
ParseMultiPartBody(rawBody);
}
else
{
// Parses a non MultiPart message
// Decode the body accodingly and set the Body property
Body = DecodeBody(rawBody, ContentTransferEncoding);
}

}


/// &lt;summary&gt;

/// Parses the &lt;paramref name="rawBody"/&gt; byte array as a MultiPart message.&lt;br/&gt;

/// It is not valid to call this method if &lt;see cref="IsMultiPart"/&gt; returned &lt;see langword="false"/&gt;.&

/// Fills the &lt;see cref="MessageParts"/&gt; property of this &lt;see cref="MessagePart"/&gt;.

/// &lt;/summary&gt;

/// &lt;param name="rawBody"&gt;The byte array which is to be parsed as a MultiPart message&lt;/param&

private void ParseMultiPartBody(byte[] rawBody)

{

// Fetch out the boundary used to delimit the messages within the body

string multipartBoundary = ContentType.Boundary;


// Fetch the individual MultiPart message parts using the MultiPart boundary

List&lt;byte[]&gt; bodyParts = GetMultiPartParts(rawBody, multipartBoundary);


// Initialize the MessageParts property, with room to as many bodies as we have found

MessageParts = new List&lt;MessagePart&gt;(bodyParts.Count);


// Now parse each byte array as a message body and add it the the MessageParts property

foreach (byte[] bodyPart in bodyParts)

{

MessagePart messagePart = GetMessagePart(bodyPart);

MessageParts.Add(messagePart);

}

}
```

```
/// &lt;summary&gt;

/// Given a byte array describing a full message.&lt;br/&gt;

/// Parses the byte array into a &lt;see cref="MessagePart"/&gt;.

/// &lt;/summary&gt;

/// &lt;param name="rawMessageContent"&gt;The byte array containing both headers and body of a messa

/// &lt;returns&gt;A &lt;see cref="MessagePart"/&gt; which was described by the &lt;paramref name="rawM

private static MessagePart GetMessagePart(byte[] rawMessageContent)

{

// Find the headers and the body parts of the byte array

MessageHeader headers;

byte[] body;

HeaderExtractor.ExtractHeadersAndBody(rawMessageContent, out headers, out body);


// Create a new MessagePart from the headers and the body

return new MessagePart(body, headers);

}


/// &lt;summary&gt;

/// Gets a list of byte arrays where each entry in the list is a full message of a message part

/// &lt;/summary&gt;

/// &lt;param name="rawBody"&gt;The raw byte array describing the body of a message which is a MultiPa

/// &lt;param name="multipPartBoundary"&gt;The delimiter that splits the different MultiPart bodies from ea

/// &lt;returns&gt;A list of byte arrays, each a full message of a &lt;see cref="MessagePart"/&gt;&lt;/returns&

private static List&lt;byte[]&gt; GetMultiPartParts(byte[] rawBody, string multipPartBoundary)

{

// This is the list we want to return
```

```
List&lt;byte[]&gt; messageBodies = new List&lt;byte[]&gt;();

// Create a stream from which we can find MultiPart boundaries
using (MemoryStream stream = new MemoryStream(rawBody))
{
bool lastMultipartBoundaryEncountered;

// Find the start of the first message in this multipart
// Since the method returns the first character on a the line containing the MultiPart boundary, we
// need to add the MultiPart boundary with prepended "--" and appended CRLF pair to the position returne
int startLocation = FindPositionOfNextMultiPartBoundary(stream, multipPartBoundary, out lastMultipartBou
while (true)
{
// When we have just parsed the last multipart entry, stop parsing on
if (lastMultipartBoundaryEncountered)
break;

// Find the end location of the current multipart
// Since the method returns the first character on a the line containing the MultiPart boundary, we
// need to go a CRLF pair back, so that we do not get that into the body of the message part
int stopLocation = FindPositionOfNextMultiPartBoundary(stream, multipPartBoundary, out lastMultipartBou

// If we could not find the next multipart boundary, but we had not yet discovered the last boundary, then
// we will consider the rest of the bytes as contained in a last message part.
if (stopLocation &lt;= -1)
{
// Include everything except the last CRLF.
stopLocation = (int)stream.Length - "\r\n".Length;

// We consider this as the last part
lastMultipartBoundaryEncountered = true;

// Special case: when the last multipart delimiter is not ending with "--", but is indeed the last
// one, then the next multipart would contain nothing, and we should not include such one.
if (startLocation &gt;= stopLocation)
break;
}

// We have now found the start and end of a message part
// Now we create a byte array with the correct length and put the message part's bytes into
// it and add it to our list we want to return
int length = stopLocation - startLocation;
byte[] messageBody = new byte[length];
Array.Copy(rawBody, startLocation, messageBody, 0, length);
messageBodies.Add(messageBody);

// We want to advance to the next message parts start.
// We can find this by jumping forward the MultiPart boundary from the last
// message parts end position
startLocation = stopLocation + ("\r\n" + "--" + multipPartBoundary + "\r\n").Length;
}
}

// We are done
```

```
return messageBodies;
}

/// &lt;summary&gt;
/// Method that is able to find a specific MultiPart boundary in a Stream.&lt;br/&gt;
/// The Stream passed should not be used for anything else then for looking for MultiPart boundaries
/// &lt;param name="stream"&gt;The stream to find the next MultiPart boundary in. Do not use it for anythin
/// &lt;param name="multiPartBoundary"&gt;The MultiPart boundary to look for. This should be found in the
/// &lt;param name="lastMultipartBoundaryFound"&gt;Is set to &lt;see langword="true"/&gt; if the next Multi
/// &lt;/summary&gt;
/// &lt;returns&gt;The position of the first character of the line that contained MultiPartBoundary or -1 if no (
private static int FindPositionOfNextMultiPartBoundary(Stream stream, string multiPartBoundary, out bool
{
lastMultipartBoundaryFound = false;
while (true)
{
// Get the current position. This is the first position on the line - no characters of the line will
// have been read yet
int currentPos = (int)stream.Position;

// Read the line
string line = StreamUtility.ReadLineAsAscii(stream);

// If we kept reading until there was no more lines, we did not meet
// the MultiPart boundary. -1 is then returned to describe this.
if (line == null)
return -1;
```

```
// The MultiPart boundary is the MultiPartBoundary with "--" in front of it
// which is to be at the very start of a line
if (line.StartsWith("--" + multiPartBoundary, StringComparison.Ordinal))
{
// Check if the found boundary was also the last one
lastMultipartBoundaryFound = line.StartsWith("--" + multiPartBoundary + "--", StringComparison.OrdinalIgn
return currentPos;
}
}
}

/// &lt;summary&gt;
/// Decodes a byte array into another byte array based upon the Content Transfer encoding
/// &lt;/summary&gt;
/// &lt;param name="messageBody"&gt;The byte array to decode into another byte array&lt;/param&gt;
/// &lt;param name="contentTransferEncoding"&gt;The &lt;see cref="ContentTransferEncoding"/&gt; of the
/// &lt;returns&gt;A byte array which comes from the &lt;paramref name="contentTransferEncoding"/&gt; be
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="messageBody"/&gt; is &lt;see lan
/// &lt;exception cref="ArgumentOutOfRangeException"&gt;Thrown if the &lt;paramref name="contentTrans
private static byte[] DecodeBody(byte[] messageBody, ContentTransferEncoding contentTransferEncoding
{
if (messageBody == null)
throw new ArgumentNullException("messageBody");

switch (contentTransferEncoding)
{
```

```
case ContentTransferEncoding.QuotedPrintable:
// If encoded in QuotedPrintable, everything in the body is in US-ASCII
return QuotedPrintable.DecodeContentTransferEncoding(Encoding.ASCII.GetString(messageBody));

case ContentTransferEncoding.Base64:
// If encoded in Base64, everything in the body is in US-ASCII
return Base64.Decode(Encoding.ASCII.GetString(messageBody));

case ContentTransferEncoding.SevenBit:
case ContentTransferEncoding.Binary:
case ContentTransferEncoding.EightBit:
// We do not have to do anything
return messageBody;

default:
throw new ArgumentOutOfRangeException("contentTransferEncoding");
}
}
#endregion

#region Public methods
/// &lt;summary&gt;
/// Gets this MessagePart's &lt;see cref="Body"/&gt; as text.&lt;br/&gt;
/// This is simply the &lt;see cref="BodyEncoding"/&gt; being used on the raw bytes of the &lt;see cref="Bo
/// This method is only valid to call if it is not a MultiPart message and therefore contains a body.&lt;br/&gt;
/// &lt;/summary&gt;
/// &lt;returns&gt;The &lt;see cref="Body"/&gt; property as a string&lt;/returns&gt;
```

```
public string GetBodyAsText()
{
return BodyEncoding.GetString(Body);
}

/// &lt;summary&gt;
/// Save this &lt;see cref="MessagePart"/&gt;'s contents to a file.&lt;br/&gt;
/// There are no methods to reload the file.
/// &lt;/summary&gt;
/// &lt;param name="file"&gt;The File location to save the &lt;see cref="MessagePart"/&gt; to. Existent files
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="file"/&gt; is &lt;see langword="nul
/// &lt;exception&gt;Other exceptions relevant to using a &lt;see cref="FileStream"/&gt; might be thrown as
public void Save(FileInfo file)
{
if (file == null)
throw new ArgumentNullException("file");

using (FileStream stream = new FileStream(file.FullName, FileMode.OpenOrCreate))
{
Save(stream);
}
}

/// &lt;summary&gt;
/// Save this &lt;see cref="MessagePart"/&gt;'s contents to a stream.&lt;br/&gt;
/// &lt;/summary&gt;
/// &lt;param name="messageStream"&gt;The stream to write to&lt;/param&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="messageStream"/&gt; is &lt;see la
/// &lt;exception&gt;Other exceptions relevant to &lt;see cref="Stream.Write"/&gt; might be thrown as well&
public void Save(Stream messageStream)
{
if (messageStream == null)
throw new ArgumentNullException("messageStream");

messageStream.Write(Body, 0, Body.Length);
}
#endregion
}

/// &lt;summary&gt;
/// &lt;see cref="Enum"/&gt; that describes the ContentTransferEncoding header field
/// &lt;/summary&gt;
/// &lt;remarks&gt;See &lt;a href="http://tools.ietf.org/html/rfc2045#section-6"&gt;RFC 2045 section 6&lt;/a&
public enum ContentTransferEncoding
{
/// &lt;summary&gt;
/// 7 bit Encoding
/// &lt;/summary&gt;
SevenBit,

/// &lt;summary&gt;
/// 8 bit Encoding
/// &lt;/summary&gt;
EightBit,
/// &lt;summary&gt;
/// Quoted Printable Encoding
/// &lt;/summary&gt;
QuotedPrintable,

/// &lt;summary&gt;
/// Base64 Encoding
/// &lt;/summary&gt;
Base64,

/// &lt;summary&gt;
/// Binary Encoding
/// &lt;/summary&gt;
Binary
}

/// &lt;summary&gt;
/// Class that hold information about one "Received:" header line.
///
/// Visit these RFCs for more information:
/// &lt;see href="http://tools.ietf.org/html/rfc5321#section-4.4"&gt;RFC 5321 section 4.4&lt;/see&gt;
/// &lt;see href="http://tools.ietf.org/html/rfc4021#section-3.6.7"&gt;RFC 4021 section 3.6.7&lt;/see&gt;
/// &lt;see href="http://tools.ietf.org/html/rfc2822#section-3.6.7"&gt;RFC 2822 section 3.6.7&lt;/see&gt;
/// &lt;see href="http://tools.ietf.org/html/rfc2821#section-4.4"&gt;RFC 2821 section 4.4&lt;/see&gt;
/// &lt;/summary&gt;
public class Received
```

```
{

/// &lt;summary&gt;

/// The date of this received line.

/// Is &lt;see cref="DateTime.MinValue"/&gt; if not present in the received header line.

/// &lt;/summary&gt;

public DateTime Date { get { return _date; } private set { _date = value; } }

private DateTime _date;


/// &lt;summary&gt;

/// A dictionary that contains the names and values of the

/// received header line.

/// If the received header is invalid and contained one name

/// multiple times, the first one is used and the rest is ignored.

/// &lt;/summary&gt;

/// &lt;example&gt;

/// If the header lines looks like:

/// &lt;code&gt;

/// from sending.com (localMachine [127.0.0.1]) by test.net (Postfix)

/// &lt;/code&gt;

/// then the dictionary will contain two keys: "from" and "by" with the values

/// "sending.com (localMachine [127.0.0.1])" and "test.net (Postfix)".

/// &lt;/example&gt;

public Dictionary&lt;string, string&gt; Names { get { return _names; } private set { _names = value; } }

private Dictionary&lt;string, string&gt; _names;


/// &lt;summary&gt;

/// The raw input string that was parsed into this class.
/// &lt;/summary&gt;
public string Raw { get { return _raw; } private set { _raw = value; } }
private string _raw;

/// &lt;summary&gt;
/// Parses a Received header value.
/// &lt;/summary&gt;
/// &lt;param name="headerValue"&gt;The value for the header to be parsed&lt;/param&gt;
/// &lt;exception cref="ArgumentNullException"&gt;&lt;exception cref="ArgumentNullException"&gt;If &lt;par
public Received(string headerValue)
{
if (headerValue == null)
throw new ArgumentNullException("headerValue");

// Remember the raw input if someone whishes to use it
Raw = headerValue;

// Default Date value
Date = DateTime.MinValue;

// The date part is the last part of the string, and is preceeded by a semicolon
// Some emails forgets to specify the date, therefore we need to check if it is there
if (headerValue.Contains(";"))
{
string datePart = headerValue.Substring(headerValue.LastIndexOf(";") + 1);
Date = Rfc2822DateTime.StringToDate(datePart);
}
```

```
Names = ParseDictionary(headerValue);
}


/// &lt;summary&gt;
/// Parses the Received header name-value-list into a dictionary.
/// &lt;/summary&gt;
/// &lt;param name="headerValue"&gt;The full header value for the Received header&lt;/param&gt;
/// &lt;returns&gt;A dictionary where the name-value-list has been parsed into&lt;/returns&gt;
private static Dictionary&lt;string, string&gt; ParseDictionary(string headerValue)
{
Dictionary&lt;string, string&gt; dictionary = new Dictionary&lt;string, string&gt;();


// Remove the date part from the full headerValue if it is present
string headerValueWithoutDate = headerValue;
if (headerValue.Contains(";"))
{
headerValueWithoutDate = headerValue.Substring(0, headerValue.LastIndexOf(";"));
}


// Reduce any whitespace character to one space only
headerValueWithoutDate = Regex.Replace(headerValueWithoutDate, @"\s+", " ");


// The regex below should capture the following:
// The name consists of non-whitespace characters followed by a whitespace and then the value follows.
// There are multiple cases for the value part:
//   1: Value is just some characters not including any whitespace
//   2: Value is some characters, a whitespace followed by an unlimited number of
//      parenthesized values which can contain whitespaces, each delimited by whitespace
//
// Cheat sheet for regex:
// \s means every whitespace character
// [^\s] means every character except whitespace characters
// +? is a non-greedy equivalent of +
const string pattern = @"(?&lt;name&gt;[^\s]+)\s(?&lt;value&gt;[^\s]+(\s\(.+?\))*)";

// Find each match in the string
MatchCollection matches = Regex.Matches(headerValueWithoutDate, pattern);
foreach (Match match in matches)
{
// Add the name and value part found in the matched result to the dictionary
string name = match.Groups["name"].Value;
string value = match.Groups["value"].Value;

// Check if the name is really a comment.
// In this case, the first entry in the header value
// is a comment
if (name.StartsWith("("))
{
continue;
}

// Only add the first name pair
// All subsequent pairs are ignored, as they are invalid anyway
if (!dictionary.ContainsKey(name))
dictionary.Add(name, value);
}

return dictionary;
}
}

/// &lt;summary&gt;
/// This class is used for RFC compliant email addresses.&lt;br/&gt;
/// &lt;br/&gt;
/// The class cannot be instantiated from outside the library.
/// &lt;/summary&gt;
/// &lt;remarks&gt;
/// The &lt;seealso cref="MailAddress"/&gt; does not cover all the possible formats
/// for &lt;a href="http://tools.ietf.org/html/rfc5322#section-3.4"&gt;RFC 5322 section 3.4&lt;/a&gt; complian
/// This class is used as an address wrapper to account for that deficiency.
/// &lt;/remarks&gt;
public class RfcMailAddress
{
#region Properties
///&lt;summary&gt;
/// The email address of this &lt;see cref="RfcMailAddress"/&gt;&lt;br/&gt;
/// It is possibly string.Empty since RFC mail addresses does not require an email address specified.
///&lt;/summary&gt;
///&lt;example&gt;
/// Example header with email address:&lt;br/&gt;
/// To: &lt;c&gt;Test test@mail.com&lt;/c&gt;&lt;br/&gt;
/// Address will be &lt;c&gt;test@mail.com&lt;/c&gt;&lt;br/&gt;
///&lt;/example&gt;
///&lt;example&gt;
/// Example header without email address:&lt;br/&gt;
/// To: &lt;c&gt;Test&lt;/c&gt;&lt;br/&gt;
/// Address will be &lt;see cref="string.Empty"/&gt;.
///&lt;/example&gt;
public string Address { get { return _address; } private set { _address = value; } }
private string _address;

///&lt;summary&gt;
/// The display name of this &lt;see cref="RfcMailAddress"/&gt;&lt;br/&gt;
/// It is possibly &lt;see cref="string.Empty"/&gt; since RFC mail addresses does not require a display name
///&lt;/summary&gt;
///&lt;example&gt;
/// Example header with display name:&lt;br/&gt;
/// To: &lt;c&gt;Test test@mail.com&lt;/c&gt;&lt;br/&gt;
/// DisplayName will be &lt;c&gt;Test&lt;/c&gt;
///&lt;/example&gt;
///&lt;example&gt;
/// Example header without display name:&lt;br/&gt;
/// To: &lt;c&gt;test@test.com&lt;/c&gt;&lt;br/&gt;
/// DisplayName will be &lt;see cref="string.Empty"/&gt;
///&lt;/example&gt;
public string DisplayName { get { return _displayName; } private set { _displayName = value; } }
private string _displayName;
```

```
/// &lt;summary&gt;

/// This is the Raw string used to describe the &lt;see cref="RfcMailAddress"/&gt;.

/// &lt;/summary&gt;

public string Raw { get { return _raw; } private set { _raw = value; } }

private string _raw;


/// &lt;summary&gt;

/// The &lt;see cref="MailAddress"/&gt; associated with the &lt;see cref="RfcMailAddress"/&gt;.

/// &lt;/summary&gt;

/// &lt;remarks&gt;

/// The value of this property can be &lt;see lanword="null"/&gt; in instances where the &lt;see cref="MailAd

/// Use &lt;see cref="HasValidMailAddress"/&gt; property to see if this property is valid.

/// &lt;/remarks&gt;

public MailAddress MailAddress { get { return _mailAddress; } private set { _mailAddress = value; } }

private MailAddress _mailAddress;


/// &lt;summary&gt;

/// Specifies if the object contains a valid &lt;see cref="MailAddress"/&gt; reference.

/// &lt;/summary&gt;

public bool HasValidMailAddress

{

get { return MailAddress != null; }

}

#endregion


#region Constructors
```

```
/// &lt;summary&gt;
/// Constructs an &lt;see cref="RfcMailAddress"/&gt; object from a &lt;see cref="MailAddress"/&gt; object.&
/// This constructor is used when we were able to construct a &lt;see cref="MailAddress"/&gt; from a string.
/// &lt;/summary&gt;
/// &lt;param name="mailAddress"&gt;The address that &lt;paramref name="raw"/&gt; was parsed into&lt;/p
/// &lt;param name="raw"&gt;The raw unparsed input which was parsed into the &lt;paramref name="mailA
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="mailAddress"/&gt; or &lt;paramref
private RfcMailAddress(MailAddress mailAddress, string raw)
{
if (mailAddress == null)
throw new ArgumentNullException("mailAddress");

if (raw == null)
throw new ArgumentNullException("raw");

MailAddress = mailAddress;
Address = mailAddress.Address;
DisplayName = mailAddress.DisplayName;
Raw = raw;
}

/// &lt;summary&gt;
/// When we were unable to parse a string into a &lt;see cref="MailAddress"/&gt;, this constructor can be
/// used. The Raw string is then used as the &lt;see cref="DisplayName"/&gt;.
/// &lt;/summary&gt;
/// &lt;param name="raw"&gt;The raw unparsed input which could not be parsed&lt;/param&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="raw"/&gt; is &lt;see langword="nu
```

```
private RfcMailAddress(string raw)
{
if (raw == null)
throw new ArgumentNullException("raw");

MailAddress = null;
Address = string.Empty;
DisplayName = raw;
Raw = raw;
}
#endregion

/// &lt;summary&gt;
/// A string representation of the &lt;see cref="RfcMailAddress"/&gt; object
/// &lt;/summary&gt;
/// &lt;returns&gt;Returns the string representation for the object&lt;/returns&gt;
public override string ToString()
{
if (HasValidMailAddress)
return MailAddress.ToString();

return Raw;
}

#region Parsing
/// &lt;summary&gt;
/// Parses an email address from a MIME header&lt;br/&gt;
/// &lt;br/&gt;
/// Examples of input:
/// &lt;c&gt;Eksperten mailrobot &amp;amp;lt;noreply@mail.eksperten.dk&amp;amp;gt;&lt;/c&gt;&lt;br/&gt;
/// &lt;c&gt;"Eksperten mailrobot" &amp;amp;lt;noreply@mail.eksperten.dk&amp;amp;gt;&lt;/c&gt;&lt;br/&gt
/// &lt;c&gt;&amp;amp;lt;noreply@mail.eksperten.dk&amp;amp;gt;&lt;/c&gt;&lt;br/&gt;
/// &lt;c&gt;noreply@mail.eksperten.dk&lt;/c&gt;&lt;br/&gt;
/// &lt;br/&gt;
/// It might also contain encoded text, which will then be decoded.
/// &lt;/summary&gt;
/// &lt;param name="input"&gt;The value to parse out and email and/or a username&lt;/param&gt;
/// &lt;returns&gt;A &lt;see cref="RfcMailAddress"/&gt;&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="input"/&gt; is &lt;see langword="n
/// &lt;remarks&gt;
/// &lt;see href="http://tools.ietf.org/html/rfc5322#section-3.4"&gt;RFC 5322 section 3.4&lt;/see&gt; for more
/// &lt;see cref="EncodedWord.Decode"&gt;For more information about encoded text&lt;/see&gt;.
/// &lt;/remarks&gt;
internal static RfcMailAddress ParseMailAddress(string input)
{
if (input == null)
throw new ArgumentNullException("input");

// Decode the value, if it was encoded
input = EncodedWord.Decode(input.Trim());

// Find the location of the email address
int indexStartEmail = input.LastIndexOf('&lt;');
int indexEndEmail = input.LastIndexOf('&gt;');
```

```
try
{
if (indexStartEmail &gt;= 0 &amp;amp;&amp;amp; indexEndEmail &gt;= 0)
{
string username;
// Check if there is a username in front of the email address
if (indexStartEmail &gt; 0)
{
// Parse out the user
username = input.Substring(0, indexStartEmail).Trim();
}
else
{
// There was no user
username = string.Empty;
}

// Parse out the email address without the "&lt;"  and "&gt;"
indexStartEmail = indexStartEmail + 1;
int emailLength = indexEndEmail - indexStartEmail;
string emailAddress = input.Substring(indexStartEmail, emailLength).Trim();

// There has been cases where there was no emailaddress between the &lt; and &gt;
if (!string.IsNullOrEmpty(emailAddress))
{
// If the username is quoted, MailAddress' constructor will remove them for us
return new RfcMailAddress(new MailAddress(emailAddress, username), input);
}
}

// This might be on the form noreply@mail.eksperten.dk
// Check if there is an email, if notm there is no need to try
if (input.Contains("@"))
return new RfcMailAddress(new MailAddress(input), input);
}
catch (FormatException)
{
// Sometimes invalid emails are sent, like sqlmap-user@sourceforge.net. (last period is illigal)
//DefaultLogger.Log.LogError("RfcMailAddress: Improper mail address: \"" + input + "\"");
}

// It could be that the format used was simply a name
// which is indeed valid according to the RFC
// Example:
// Eksperten mailrobot
return new RfcMailAddress(input);
}

/// &lt;summary&gt;
/// Parses input of the form&lt;br/&gt;
/// &lt;c&gt;Eksperten mailrobot &amp;amp;lt;noreply@mail.eksperten.dk&amp;amp;gt;, ...&lt;/c&gt;&lt;br/&
/// to a list of RFCMailAddresses
/// &lt;/summary&gt;
/// &lt;param name="input"&gt;The input that is a comma-separated list of EmailAddresses to parse&lt;/para

/// &lt;returns&gt;A List of &lt;seealso cref="RfcMailAddress"/&gt; objects extracted from the &lt;paramref na

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="input"/&gt; is &lt;see langword="n

internal static List&lt;RfcMailAddress&gt; ParseMailAddresses(string input)

{

if (input == null)

throw new ArgumentNullException("input");


List&lt;RfcMailAddress&gt; returner = new List&lt;RfcMailAddress&gt;();


// MailAddresses are split by commas

IEnumerable&lt;string&gt; mailAddresses = Utility.SplitStringWithCharNotInsideQuotes(input, ',');


// Parse each of these

foreach (string mailAddress in mailAddresses)

{

returner.Add(ParseMailAddress(mailAddress));

}


return returner;

}

#endregion

}


/// &lt;summary&gt;

/// Class that holds all headers for a message&lt;br/&gt;

/// Headers which are unknown the the parser will be held in the &lt;see cref="UnknownHeaders"/&gt; colle
/// &lt;br/&gt;
/// This class cannot be instantiated from outside the library.
/// &lt;/summary&gt;
/// &lt;remarks&gt;
/// See &lt;a href="http://tools.ietf.org/html/rfc4021"&gt;RFC 4021&lt;/a&gt; for a large list of headers.&lt;br/&
/// &lt;/remarks&gt;
public sealed class MessageHeader
{
#region Properties
/// &lt;summary&gt;
/// All headers which were not recognized and explicitly dealt with.&lt;br/&gt;
/// This should mostly be custom headers, which are marked as X-[name].&lt;br/&gt;
/// &lt;br/&gt;
/// This list will be empty if all headers were recognized and parsed.
/// &lt;/summary&gt;
/// &lt;remarks&gt;
/// If you as a user, feels that a header in this collection should
/// be parsed, feel free to notify the developers.
/// &lt;/remarks&gt;
public NameValueCollection UnknownHeaders { get { return _unknownHeaders; } private set { _unknownH
private NameValueCollection _unknownHeaders;

/// &lt;summary&gt;
/// A human readable description of the body&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Content-Description header was present in the message.
/// &lt;/summary&gt;
```

```
public string ContentDescription { get { return _contentDescription; } private set { _contentDescription = val

private string _contentDescription;


/// &lt;summary&gt;

/// ID of the content part (like an attached image). Used with MultiPart messages.&lt;br/&gt;

/// &lt;br/&gt;

/// &lt;see langword="null"/&gt; if no Content-ID header field was present in the message.

/// &lt;/summary&gt;

/// &lt;see cref="MessageId"&gt;For an ID of the message&lt;/see&gt;

public string ContentId { get { return _contentId; } private set { _contentId = value; } }

private string _contentId;


/// &lt;summary&gt;

/// Message keywords&lt;br/&gt;

/// &lt;br/&gt;

/// The list will be empty if no Keywords header was present in the message

/// &lt;/summary&gt;

public List&lt;string&gt; Keywords { get { return _keywords; } private set { _keywords = value; } }

private List&lt;string&gt; _keywords;


/// &lt;summary&gt;

/// A List of emails to people who wishes to be notified when some event happens.&lt;br/&gt;

/// These events could be email:

/// &lt;list type="bullet"&gt;

///   &lt;item&gt;deletion&lt;/item&gt;

///   &lt;item&gt;printing&lt;/item&gt;

///   &lt;item&gt;received&lt;/item&gt;
///   &lt;item&gt;...&lt;/item&gt;
/// &lt;/list&gt;
/// The list will be empty if no Disposition-Notification-To header was present in the message
/// &lt;/summary&gt;
/// &lt;remarks&gt;See &lt;a href="http://tools.ietf.org/html/rfc3798"&gt;RFC 3798&lt;/a&gt; for details&lt;/rer
public List&lt;RfcMailAddress&gt; DispositionNotificationTo { get { return _dispositionNotificationTo; } privat
private List&lt;RfcMailAddress&gt; _dispositionNotificationTo;

/// &lt;summary&gt;
/// This is the Received headers. This tells the path that the email went.&lt;br/&gt;
/// &lt;br/&gt;
/// The list will be empty if no Received header was present in the message
/// &lt;/summary&gt;
public List&lt;Received&gt; Received { get { return _recieved; } private set { _recieved = value; } }
private List&lt;Received&gt; _recieved;

/// &lt;summary&gt;
/// Importance of this email.&lt;br/&gt;
/// &lt;br/&gt;
/// The importance level is set to normal, if no Importance header field was mentioned or it contained
/// unknown information. This is the expected behavior according to the RFC.
/// &lt;/summary&gt;
public MailPriority Importance { get { return _importance; } private set { _importance = value; } }
private MailPriority _importance;

/// &lt;summary&gt;
/// This header describes the Content encoding during transfer.&lt;br/&gt;
/// &lt;br/&gt;
/// If no Content-Transfer-Encoding header was present in the message, it is set
/// to the default of &lt;see cref="Header.ContentTransferEncoding.SevenBit"&gt;SevenBit&lt;/see&gt; in ac
/// &lt;/summary&gt;
/// &lt;remarks&gt;See &lt;a href="http://tools.ietf.org/html/rfc2045#section-6"&gt;RFC 2045 section 6&lt;/a&
public ContentTransferEncoding ContentTransferEncoding { get { return _contentTransferEncoding; } priva
private ContentTransferEncoding _contentTransferEncoding;


/// &lt;summary&gt;
/// Carbon Copy. This specifies who got a copy of the message.&lt;br/&gt;
/// &lt;br/&gt;
/// The list will be empty if no Cc header was present in the message
/// &lt;/summary&gt;
public List&lt;RfcMailAddress&gt; Cc { get { return _cc; } private set { _cc = value; } }
private List&lt;RfcMailAddress&gt; _cc;


/// &lt;summary&gt;
/// Blind Carbon Copy. This specifies who got a copy of the message, but others
/// cannot see who these persons are.&lt;br/&gt;
/// &lt;br/&gt;
/// The list will be empty if no Received Bcc was present in the message
/// &lt;/summary&gt;
public List&lt;RfcMailAddress&gt; Bcc { get { return _bcc; } private set { _bcc = value; } }
private List&lt;RfcMailAddress&gt; _bcc;


/// &lt;summary&gt;
/// Specifies who this mail was for&lt;br/&gt;
/// &lt;br/&gt;
/// The list will be empty if no To header was present in the message
/// &lt;/summary&gt;
public List&lt;RfcMailAddress&gt; To { get { return _to; } private set { _to = value; } }
private List&lt;RfcMailAddress&gt; _to;

/// &lt;summary&gt;
/// Specifies who sent the email&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no From header field was present in the message
/// &lt;/summary&gt;
public RfcMailAddress From { get { return _from; } private set { _from = value; } }
private RfcMailAddress _from;

/// &lt;summary&gt;
/// Specifies who a reply to the message should be sent to&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Reply-To header field was present in the message
/// &lt;/summary&gt;
public RfcMailAddress ReplyTo { get { return _replyTo; } private set { _replyTo = value; } }
private RfcMailAddress _replyTo;

/// &lt;summary&gt;
/// The message identifier(s) of the original message(s) to which the
/// current message is a reply.&lt;br/&gt;
/// &lt;br/&gt;
/// The list will be empty if no In-Reply-To header was present in the message
/// &lt;/summary&gt;
public List&lt;string&gt; InReplyTo { get { return _inReplyTo; } private set { _inReplyTo = value; } }
private List&lt;string&gt; _inReplyTo;

/// &lt;summary&gt;
/// The message identifier(s) of other message(s) to which the current
/// message is related to.&lt;br/&gt;
/// &lt;br/&gt;
/// The list will be empty if no References header was present in the message
/// &lt;/summary&gt;
public List&lt;string&gt; References { get { return _references; } private set { _references = value; } }
private List&lt;string&gt; _references;

/// &lt;summary&gt;
/// This is the sender of the email address.&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Sender header field was present in the message
/// &lt;/summary&gt;
/// &lt;remarks&gt;
/// The RFC states that this field can be used if a secretary
/// is sending an email for someone she is working for.
/// The email here will then be the secretary's email, and
/// the Reply-To field would hold the address of the person she works for.&lt;br/&gt;
/// RFC states that if the Sender is the same as the From field,
/// sender should not be included in the message.
/// &lt;/remarks&gt;
public RfcMailAddress Sender { get { return _sender; } private set { _sender = value; } }
```

```
private RfcMailAddress _sender;


/// &lt;summary&gt;

/// The Content-Type header field.&lt;br/&gt;

/// &lt;br/&gt;

/// If not set, the ContentType is created by the default "text/plain; charset=us-ascii" which is

/// defined in &lt;a href="http://tools.ietf.org/html/rfc2045#section-5.2"&gt;RFC 2045 section 5.2&lt;/a&gt;.&lt

/// If set, the default is overridden.

/// &lt;/summary&gt;

public ContentType ContentType { get { return _contentType; } private set { _contentType = value; } }

private ContentType _contentType;


/// &lt;summary&gt;

/// Used to describe if a &lt;see cref="MessagePart"/&gt; is to be displayed or to be though of as an attachm

/// Also contains information about filename if such was sent.&lt;br/&gt;

/// &lt;br/&gt;

/// &lt;see langword="null"/&gt; if no Content-Disposition header field was present in the message

/// &lt;/summary&gt;

public ContentDisposition ContentDisposition { get { return _contentDispisition; } private set { _contentDispi

private ContentDisposition _contentDispisition;


/// &lt;summary&gt;

/// The Date when the email was sent.&lt;br/&gt;

/// This is the raw value. &lt;see cref="DateSent"/&gt; for a parsed up &lt;see cref="DateTime"/&gt; value of

/// &lt;br/&gt;

/// &lt;see langword="DateTime.MinValue"/&gt; if no Date header field was present in the message or if the

/// &lt;/summary&gt;
/// &lt;remarks&gt;See &lt;a href="http://tools.ietf.org/html/rfc5322#section-3.6.1"&gt;RFC 5322 section 3.6.
public string Date { get { return _date; } private set { _date = value; } }
private string _date;


/// &lt;summary&gt;
/// The Date when the email was sent.&lt;br/&gt;
/// This is the parsed equivalent of &lt;see cref="Date"/&gt;.&lt;br/&gt;
/// Notice that the &lt;see cref="TimeZone"/&gt; of the &lt;see cref="DateTime"/&gt; object is in UTC and ha
/// to local &lt;see cref="TimeZone"/&gt;.
/// &lt;/summary&gt;
/// &lt;remarks&gt;See &lt;a href="http://tools.ietf.org/html/rfc5322#section-3.6.1"&gt;RFC 5322 section 3.6.
public DateTime DateSent { get { return _dateSent; } private set { _dateSent = value; } }
private DateTime _dateSent;


/// &lt;summary&gt;
/// An ID of the message that is SUPPOSED to be in every message according to the RFC.&lt;br/&gt;
/// The ID is unique.&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Message-ID header field was present in the message
/// &lt;/summary&gt;
public string MessageId { get { return _messageId; } private set { _messageId = value; } }
private string _messageId;


/// &lt;summary&gt;
/// The Mime Version.&lt;br/&gt;
/// This field will almost always show 1.0&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Mime-Version header field was present in the message
/// &lt;/summary&gt;
public string MimeVersion { get { return _mimeVersion; } private set { _mimeVersion = value; } }
private string _mimeVersion;

/// &lt;summary&gt;
/// A single &lt;see cref="RfcMailAddress"/&gt; with no username inside.&lt;br/&gt;
/// This is a trace header field, that should be in all messages.&lt;br/&gt;
/// Replies should be sent to this address.&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Return-Path header field was present in the message
/// &lt;/summary&gt;
public RfcMailAddress ReturnPath { get { return _returnPath; } private set { _returnPath = value; } }
public RfcMailAddress _returnPath;

/// &lt;summary&gt;
/// The subject line of the message in decoded, one line state.&lt;br/&gt;
/// This should be in all messages.&lt;br/&gt;
/// &lt;br/&gt;
/// &lt;see langword="null"/&gt; if no Subject header field was present in the message
/// &lt;/summary&gt;
public string Subject { get { return _subject; } private set { _subject = value; } }
private string _subject;
#endregion

/// &lt;summary&gt;
/// Parses a &lt;see cref="NameValueCollection"/&gt; to a MessageHeader
/// &lt;/summary&gt;
/// &lt;param name="headers"&gt;The collection that should be traversed and parsed&lt;/param&gt;
/// &lt;returns&gt;A valid MessageHeader object&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="headers"/&gt; is &lt;see langword
internal MessageHeader(NameValueCollection headers)
{
if (headers == null)
throw new ArgumentNullException("headers");

// Create empty lists as defaults. We do not like null values
// List with an initial capacity set to zero will be replaced
// when a corrosponding header is found
To = new List&lt;RfcMailAddress&gt;(0);
Cc = new List&lt;RfcMailAddress&gt;(0);
Bcc = new List&lt;RfcMailAddress&gt;(0);
Received = new List&lt;Received&gt;();
Keywords = new List&lt;string&gt;();
InReplyTo = new List&lt;string&gt;(0);
References = new List&lt;string&gt;(0);
DispositionNotificationTo = new List&lt;RfcMailAddress&gt;();
UnknownHeaders = new NameValueCollection();

// Default importancetype is Normal (assumed if not set)
Importance = MailPriority.Normal;

// 7BIT is the default ContentTransferEncoding (assumed if not set)
ContentTransferEncoding = ContentTransferEncoding.SevenBit;
```

```
// text/plain; charset=us-ascii is the default ContentType
ContentType = new ContentType("text/plain; charset=us-ascii");

// Now parse the actual headers
ParseHeaders(headers);
}

/// &lt;summary&gt;
/// Parses a &lt;see cref="NameValueCollection"/&gt; to a &lt;see cref="MessageHeader"/&gt;
/// &lt;/summary&gt;
/// &lt;param name="headers"&gt;The collection that should be traversed and parsed&lt;/param&gt;
/// &lt;returns&gt;A valid &lt;see cref="MessageHeader"/&gt; object&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="headers"/&gt; is &lt;see langword
private void ParseHeaders(NameValueCollection headers)
{
if (headers == null)
throw new ArgumentNullException("headers");

// Now begin to parse the header values
foreach (string headerName in headers.Keys)
{
string[] headerValues = headers.GetValues(headerName);
if (headerValues != null)
{
foreach (string headerValue in headerValues)
{
```

```
ParseHeader(headerName, headerValue);
}
}
}
}

#region Header fields parsing
/// &lt;summary&gt;
/// Parses a single header and sets member variables according to it.
/// &lt;/summary&gt;
/// &lt;param name="headerName"&gt;The name of the header&lt;/param&gt;
/// &lt;param name="headerValue"&gt;The value of the header in unfolded state (only one line)&lt;/param&g
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="headerName"/&gt; or &lt;paramre
private void ParseHeader(string headerName, string headerValue)
{
if (headerName == null)
throw new ArgumentNullException("headerName");

if (headerValue == null)
throw new ArgumentNullException("headerValue");

switch (headerName.ToUpperInvariant())
{
// See http://tools.ietf.org/html/rfc5322#section-3.6.3
case "TO":
To = RfcMailAddress.ParseMailAddresses(headerValue);
break;
```

```
// See http://tools.ietf.org/html/rfc5322#section-3.6.3
case "CC":
Cc = RfcMailAddress.ParseMailAddresses(headerValue);
break;

// See http://tools.ietf.org/html/rfc5322#section-3.6.3
case "BCC":
Bcc = RfcMailAddress.ParseMailAddresses(headerValue);
break;

// See http://tools.ietf.org/html/rfc5322#section-3.6.2
case "FROM":
// There is only one MailAddress in the from field
From = RfcMailAddress.ParseMailAddress(headerValue);
break;

// http://tools.ietf.org/html/rfc5322#section-3.6.2
// The implementation here might be wrong
case "REPLY-TO":
// This field may actually be a list of addresses, but no
// such case has been encountered
ReplyTo = RfcMailAddress.ParseMailAddress(headerValue);
break;

// http://tools.ietf.org/html/rfc5322#section-3.6.2
case "SENDER":
```

```
Sender = RfcMailAddress.ParseMailAddress(headerValue);

break;


// See http://tools.ietf.org/html/rfc5322#section-3.6.5

// RFC 5322:

// The "Keywords:" field contains a comma-separated list of one or more

// words or quoted-strings.

// The field are intended to have only human-readable content

// with information about the message

case "KEYWORDS":

string[] keywordsTemp = headerValue.Split(',');

foreach (string keyword in keywordsTemp)

{

// Remove the quotes if there is any

Keywords.Add(Utility.RemoveQuotesIfAny(keyword.Trim()));

}

break;


// See http://tools.ietf.org/html/rfc5322#section-3.6.7

case "RECEIVED":

// Simply add the value to the list

Received.Add(new Received(headerValue.Trim()));

break;


case "IMPORTANCE":

Importance = HeaderFieldParser.ParseImportance(headerValue.Trim());

break;
```

```
// See http://tools.ietf.org/html/rfc3798#section-2.1
case "DISPOSITION-NOTIFICATION-TO":
DispositionNotificationTo = RfcMailAddress.ParseMailAddresses(headerValue);
break;

case "MIME-VERSION":
MimeVersion = headerValue.Trim();
break;

// See http://tools.ietf.org/html/rfc5322#section-3.6.5
case "SUBJECT":
Subject = EncodedWord.Decode(headerValue);
break;

// See http://tools.ietf.org/html/rfc5322#section-3.6.7
case "RETURN-PATH":
// Return-paths does not include a username, but we
// may still use the address parser
ReturnPath = RfcMailAddress.ParseMailAddress(headerValue);
break;

// See http://tools.ietf.org/html/rfc5322#section-3.6.4
// Example Message-ID
// &lt;33cdd74d6b89ab2250ecd75b40a41405@nfs.eksperten.dk&gt;
case "MESSAGE-ID":
MessageId = HeaderFieldParser.ParseId(headerValue);
```

```
break;


// See http://tools.ietf.org/html/rfc5322#section-3.6.4

case "IN-REPLY-TO":

InReplyTo = HeaderFieldParser.ParseMultipleIDs(headerValue);

break;


// See http://tools.ietf.org/html/rfc5322#section-3.6.4

case "REFERENCES":

References = HeaderFieldParser.ParseMultipleIDs(headerValue);

break;


// See http://tools.ietf.org/html/rfc5322#section-3.6.1))

case "DATE":

Date = headerValue.Trim();

DateSent = Rfc2822DateTime.StringToDate(headerValue);

break;


// See http://tools.ietf.org/html/rfc2045#section-6

// See ContentTransferEncoding class for more details

case "CONTENT-TRANSFER-ENCODING":

ContentTransferEncoding = HeaderFieldParser.ParseContentTransferEncoding(headerValue.Trim());

break;


// See http://tools.ietf.org/html/rfc2045#section-8

case "CONTENT-DESCRIPTION":

// Human description of for example a file. Can be encoded
```

```
ContentDescription = EncodedWord.Decode(headerValue.Trim());
break;

// See http://tools.ietf.org/html/rfc2045#section-5.1
// Example: Content-type: text/plain; charset="us-ascii"
case "CONTENT-TYPE":
ContentType = HeaderFieldParser.ParseContentType(headerValue);
break;

// See http://tools.ietf.org/html/rfc2183
case "CONTENT-DISPOSITION":
ContentDisposition = HeaderFieldParser.ParseContentDisposition(headerValue);
break;

// See http://tools.ietf.org/html/rfc2045#section-7
// Example: &lt;foo4*foo1@bar.net&gt;
case "CONTENT-ID":
ContentId = HeaderFieldParser.ParseId(headerValue);
break;

default:
// This is an unknown header

// Custom headers are allowed. That means headers
// that are not mentionen in the RFC.
// Such headers start with the letter "X"
// We do not have any special parsing of such
// Add it to unknown headers
UnknownHeaders.Add(headerName, headerValue);
break;
}
}
#endregion
}

/// &lt;summary&gt;
/// This is the root of the email tree structure.&lt;br/&gt;
/// &lt;see cref="Mime.MessagePart"/&gt; for a description about the structure.&lt;br/&gt;
/// &lt;br/&gt;
/// A Message (this class) contains the headers of an email message such as:
/// &lt;code&gt;
///  - To
///  - From
///  - Subject
///  - Content-Type
///  - Message-ID
/// &lt;/code&gt;
/// which are located in the &lt;see cref="Headers"/&gt; property.&lt;br/&gt;
/// &lt;br/&gt;
/// Use the &lt;see cref="Message.MessagePart"/&gt; property to find the actual content of the email messa
/// &lt;/summary&gt;
/// &lt;example&gt;
/// Examples are available on the &lt;a href="http://hpop.sourceforge.net/"&gt;project homepage&lt;/a&gt;.
/// &lt;/example&gt;

public class Message

{

#region Public properties

/// &lt;summary&gt;

/// Headers of the Message.

/// &lt;/summary&gt;

public MessageHeader Headers { get { return _headers; } private set { _headers = value; } }

private MessageHeader _headers;


/// &lt;summary&gt;

/// This is the body of the email Message.&lt;br/&gt;

/// &lt;br/&gt;

/// If the body was parsed for this Message, this property will never be &lt;see langword="null"/&gt;.

/// &lt;/summary&gt;

public MessagePart MessagePart { get { return _messagePart; } private set { _messagePart = value; } }

private MessagePart _messagePart;


/// &lt;summary&gt;

/// The raw content from which this message has been constructed.&lt;br/&gt;

/// These bytes can be persisted and later used to recreate the Message.

/// &lt;/summary&gt;

public byte[] RawMessage { get { return _rawMessage; } private set { _rawMessage = value; } }

private byte[] _rawMessage;

#endregion


#region Constructors
```

```
/// &lt;summary&gt;
/// Convenience constructor for &lt;see cref="Mime.Message(byte[], bool)"/&gt;.&lt;br/&gt;
/// &lt;br/&gt;
/// Creates a message from a byte array. The full message including its body is parsed.
/// &lt;/summary&gt;
/// &lt;param name="rawMessageContent"&gt;The byte array which is the message contents to parse&lt;/pa
public Message(byte[] rawMessageContent)
: this(rawMessageContent, true)
{
}

/// &lt;summary&gt;
/// Constructs a message from a byte array.&lt;br/&gt;
/// &lt;br/&gt;
/// The headers are always parsed, but if &lt;paramref name="parseBody"/&gt; is &lt;see langword="false"/&
/// &lt;/summary&gt;
/// &lt;param name="rawMessageContent"&gt;The byte array which is the message contents to parse&lt;/pa
/// &lt;param name="parseBody"&gt;
/// &lt;see langword="true"/&gt; if the body should be parsed,
/// &lt;see langword="false"/&gt; if only headers should be parsed out of the &lt;paramref name="rawMessa
/// &lt;/param&gt;
public Message(byte[] rawMessageContent, bool parseBody)
{
RawMessage = rawMessageContent;

// Find the headers and the body parts of the byte array
MessageHeader headersTemp;
byte[] body;
HeaderExtractor.ExtractHeadersAndBody(rawMessageContent, out headersTemp, out body);

// Set the Headers property
Headers = headersTemp;

// Should we also parse the body?
if (parseBody)
{
// Parse the body into a MessagePart
MessagePart = new MessagePart(body, Headers);
}
}
#endregion

/// &lt;summary&gt;
/// This method will convert this &lt;see cref="Message"/&gt; into a &lt;see cref="MailMessage"/&gt; equival
/// The returned &lt;see cref="MailMessage"/&gt; can be used with &lt;see cref="System.Net.Mail.SmtpClie
/// &lt;br/&gt;
/// You should be aware of the following about this method:
/// &lt;list type="bullet"&gt;
/// &lt;item&gt;
///    All sender and receiver mail addresses are set.
///    If you send this email using a &lt;see cref="System.Net.Mail.SmtpClient"/&gt; then all
///    receivers in To, From, Cc and Bcc will receive the email once again.
/// &lt;/item&gt;
/// &lt;item&gt;
///    If you view the source code of this Message and looks at the source code of the forwarded
///    &lt;see cref="MailMessage"/&gt; returned by this method, you will notice that the source codes are not
///    The content that is presented by a mail client reading the forwarded &lt;see cref="MailMessage"/&gt; s
///    same as the original, though.
/// &lt;/item&gt;
/// &lt;item&gt;
///    Content-Disposition headers will not be copied to the &lt;see cref="MailMessage"/&gt;.
///    It is simply not possible to set these on Attachments.
/// &lt;/item&gt;
/// &lt;item&gt;
///    HTML content will be treated as the preferred view for the &lt;see cref="MailMessage.Body"/&gt;. Plain
///    &lt;see cref="MailMessage.Body"/&gt; when HTML is not available.
/// &lt;/item&gt;
/// &lt;/list&gt;
/// &lt;/summary&gt;
/// &lt;returns&gt;A &lt;see cref="MailMessage"/&gt; object that contains the same information that this Mes
public MailMessage ToMailMessage()
{
// Construct an empty MailMessage to which we will gradually build up to look like the current Message obj
MailMessage message = new MailMessage();

message.Subject = Headers.Subject;

// We here set the encoding to be UTF-8
// We cannot determine what the encoding of the subject was at this point.
// But since we know that strings in .NET is stored in UTF, we can
// use UTF-8 to decode the subject into bytes
```

```
message.SubjectEncoding = Encoding.UTF8;

// The HTML version should take precedent over the plain text if it is available
MessagePart preferredVersion = FindFirstHtmlVersion();
if (preferredVersion != null)
{
// Make sure that the IsBodyHtml property is being set correctly for our content
message.IsBodyHtml = true;
}
else
{
// otherwise use the first plain text version as the body, if it exists
preferredVersion = FindFirstPlainTextVersion();
}

if (preferredVersion != null)
{
message.Body = preferredVersion.GetBodyAsText();
message.BodyEncoding = preferredVersion.BodyEncoding;
}

// Add body and alternative views (html and such) to the message
IEnumerable&lt;MessagePart&gt; textVersions = FindAllTextVersions();
foreach (MessagePart textVersion in textVersions)
{
// The textVersions also contain the preferred version, therefore
// we should skip that one
if (textVersion == preferredVersion)
continue;

MemoryStream stream = new MemoryStream(textVersion.Body);
AlternateView alternative = new AlternateView(stream);
alternative.ContentId = textVersion.ContentId;
alternative.ContentType = textVersion.ContentType;
message.AlternateViews.Add(alternative);
}

// Add attachments to the message
IEnumerable&lt;MessagePart&gt; attachments = FindAllAttachments();
foreach (MessagePart attachmentMessagePart in attachments)
{
MemoryStream stream = new MemoryStream(attachmentMessagePart.Body);
Attachment attachment = new Attachment(stream, attachmentMessagePart.ContentType);
attachment.ContentId = attachmentMessagePart.ContentId;
message.Attachments.Add(attachment);
}

if (Headers.From != null &amp;amp;&amp;amp; Headers.From.HasValidMailAddress)
message.From = Headers.From.MailAddress;

if (Headers.ReplyTo != null &amp;amp;&amp;amp; Headers.ReplyTo.HasValidMailAddress)
message.ReplyTo = Headers.ReplyTo.MailAddress;

if (Headers.Sender != null &amp;amp;&amp;amp; Headers.Sender.HasValidMailAddress)
```

```
message.Sender = Headers.Sender.MailAddress;


foreach (RfcMailAddress to in Headers.To)

{

if (to.HasValidMailAddress)

message.To.Add(to.MailAddress);

}


foreach (RfcMailAddress cc in Headers.Cc)

{

if (cc.HasValidMailAddress)

message.CC.Add(cc.MailAddress);

}


foreach (RfcMailAddress bcc in Headers.Bcc)

{

if (bcc.HasValidMailAddress)

message.Bcc.Add(bcc.MailAddress);

}


return message;

}


#region MessagePart Searching Methods


/// &lt;summary&gt;

/// Finds the first text/plain &lt;see cref="MessagePart"/&gt; in this message.&lt;br/&gt;
/// This is a convenience method - it simply propagates the call to &lt;see cref="FindFirstMessagePartWithM
/// &lt;br/&gt;
/// If no text/plain version is found, &lt;see langword="null"/&gt; is returned.
/// &lt;/summary&gt;
/// &lt;returns&gt;
/// &lt;see cref="MessagePart"/&gt; which has a MediaType of text/plain or &lt;see langword="null"/&gt;
/// if such &lt;see cref="MessagePart"/&gt; could not be found.
/// &lt;/returns&gt;
public MessagePart FindFirstPlainTextVersion()
{
return FindFirstMessagePartWithMediaType("text/plain");
}

/// &lt;summary&gt;
/// Finds the first text/html &lt;see cref="MessagePart"/&gt; in this message.&lt;br/&gt;
/// This is a convenience method - it simply propagates the call to &lt;see cref="FindFirstMessagePartWithM
/// &lt;br/&gt;
/// If no text/html version is found, &lt;see langword="null"/&gt; is returned.
/// &lt;/summary&gt;
/// &lt;returns&gt;
/// &lt;see cref="MessagePart"/&gt; which has a MediaType of text/html or &lt;see langword="null"/&gt;
/// if such &lt;see cref="MessagePart"/&gt; could not be found.
/// &lt;/returns&gt;
public MessagePart FindFirstHtmlVersion()
{
return FindFirstMessagePartWithMediaType("text/html");
}
```

```
/// &lt;summary&gt;

/// Finds all the &lt;see cref="MessagePart"/&gt;'s which contains a text version.&lt;br/&gt;

/// &lt;br/&gt;

/// &lt;see cref="Mime.MessagePart.IsText"/&gt; for MessageParts which are considered to be text versions

/// &lt;br/&gt;

/// Examples of MessageParts media types are:

/// &lt;list type="bullet"&gt;

///    &lt;item&gt;text/plain&lt;/item&gt;

///    &lt;item&gt;text/html&lt;/item&gt;

///    &lt;item&gt;text/xml&lt;/item&gt;

/// &lt;/list&gt;

/// &lt;/summary&gt;

/// &lt;returns&gt;A List of MessageParts where each part is a text version&lt;/returns&gt;

public List&lt;MessagePart&gt; FindAllTextVersions()

{

return new TextVersionFinder().VisitMessage(this);

}


/// &lt;summary&gt;

/// Finds all the &lt;see cref="MessagePart"/&gt;'s which are attachments to this message.&lt;br/&gt;

/// &lt;br/&gt;

/// &lt;see cref="Mime.MessagePart.IsAttachment"/&gt; for MessageParts which are considered to be attac

/// &lt;/summary&gt;

/// &lt;returns&gt;A List of MessageParts where each is considered an attachment&lt;/returns&gt;

public List&lt;MessagePart&gt; FindAllAttachments()

{
```

```
return new AttachmentFinder().VisitMessage(this);
}


/// &lt;summary&gt;
/// Finds the first &lt;see cref="MessagePart"/&gt; in the &lt;see cref="Message"/&gt; hierarchy with the give
/// &lt;br/&gt;
/// The search in the hierarchy is a depth-first traversal.
/// &lt;/summary&gt;
/// &lt;param name="mediaType"&gt;The MediaType to search for. Case is ignored.&lt;/param&gt;
/// &lt;returns&gt;
/// A &lt;see cref="MessagePart"/&gt; with the given MediaType or &lt;see langword="null"/&gt; if no such &
/// &lt;/returns&gt;
public MessagePart FindFirstMessagePartWithMediaType(string mediaType)
{
return new FindFirstMessagePartWithMediaType().VisitMessage(this, mediaType);
}


/// &lt;summary&gt;
/// Finds all the &lt;see cref="MessagePart"/&gt;s in the &lt;see cref="Message"/&gt; hierarchy with the give
/// &lt;/summary&gt;
/// &lt;param name="mediaType"&gt;The MediaType to search for. Case is ignored.&lt;/param&gt;
/// &lt;returns&gt;
/// A List of &lt;see cref="MessagePart"/&gt;s with the given MediaType.&lt;br/&gt;
/// The List might be empty if no such &lt;see cref="MessagePart"/&gt;s were found.&lt;br/&gt;
/// The order of the elements in the list is the order which they are found using
/// a depth first traversal of the &lt;see cref="Message"/&gt; hierarchy.
/// &lt;/returns&gt;
```

```
public List&lt;MessagePart&gt; FindAllMessagePartsWithMediaType(string mediaType)

{

return new FindAllMessagePartsWithMediaType().VisitMessage(this, mediaType);

}


#endregion


#region Message Persistence


/// &lt;summary&gt;

/// Save this &lt;see cref="Message"/&gt; to a file.&lt;br/&gt;

/// &lt;br/&gt;

/// Can be loaded at a later time using the &lt;see cref="Load(FileInfo)"/&gt; method.

/// &lt;/summary&gt;

/// &lt;param name="file"&gt;The File location to save the &lt;see cref="Message"/&gt; to. Existent files will

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="file"/&gt; is &lt;see langword="nul

/// &lt;exception&gt;Other exceptions relevant to using a &lt;see cref="FileStream"/&gt; might be thrown as

public void Save(FileInfo file)

{

if (file == null)

throw new ArgumentNullException("file");


using (FileStream stream = new FileStream(file.FullName, FileMode.OpenOrCreate))

{

Save(stream);

}

}
```

```
/// &lt;summary&gt;

/// Save this &lt;see cref="Message"/&gt; to a stream.&lt;br/&gt;

/// &lt;/summary&gt;

/// &lt;param name="messageStream"&gt;The stream to write to&lt;/param&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="messageStream"/&gt; is &lt;see la

/// &lt;exception&gt;Other exceptions relevant to &lt;see cref="Stream.Write"/&gt; might be thrown as well&

public void Save(Stream messageStream)

{

if (messageStream == null)

throw new ArgumentNullException("messageStream");


messageStream.Write(RawMessage, 0, RawMessage.Length);

}


/// &lt;summary&gt;

/// Loads a &lt;see cref="Message"/&gt; from a file containing a raw email.

/// &lt;/summary&gt;

/// &lt;param name="file"&gt;The File location to load the &lt;see cref="Message"/&gt; from. The file must ex

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="file"/&gt; is &lt;see langword="nul

/// &lt;exception cref="FileNotFoundException"&gt;If &lt;paramref name="file"/&gt; does not exist&lt;/except

/// &lt;exception&gt;Other exceptions relevant to a &lt;see cref="FileStream"/&gt; might be thrown as well&l

/// &lt;returns&gt;A &lt;see cref="Message"/&gt; with the content loaded from the &lt;paramref name="file"/&

public static Message Load(FileInfo file)

{

if (file == null)

throw new ArgumentNullException("file");
```

```
if (!file.Exists)

throw new FileNotFoundException("Cannot load message from non-existent file", file.FullName);


using (FileStream stream = new FileStream(file.FullName, FileMode.Open))

{

return Load(stream);

}

}




/// &lt;summary&gt;

/// Loads a &lt;see cref="Message"/&gt; from a &lt;see cref="Stream"/&gt; containing a raw email.

/// &lt;/summary&gt;

/// &lt;param name="messageStream"&gt;The &lt;see cref="Stream"/&gt; from which to load the raw &lt;se

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="messageStream"/&gt; is &lt;see l

/// &lt;exception&gt;Other exceptions relevant to &lt;see cref="Stream.Read"/&gt; might be thrown as well&

/// &lt;returns&gt;A &lt;see cref="Message"/&gt; with the content loaded from the &lt;paramref name="mess

public static Message Load(Stream messageStream)

{

if (messageStream == null)

throw new ArgumentNullException("messageStream");


using (MemoryStream outStream = new MemoryStream())

{

#if DOTNET4

// TODO: Enable using native v4 framework methods when support is formally added.
messageStream.CopyTo(outStream);
#else
int bytesRead;
byte[] buffer = new byte[4096];

while ((bytesRead = messageStream.Read(buffer, 0, 4096)) &gt; 0)
{
outStream.Write(buffer, 0, bytesRead);
}
#endif
byte[] content = outStream.ToArray();

return new Message(content);
}
}
#endregion
}

/// &lt;summary&gt;
/// Implements the CRAM-MD5 algorithm as specified in &lt;a href="http://tools.ietf.org/html/rfc2195"&gt;RF
/// &lt;/summary&gt;
internal static class CramMd5
{
/// &lt;summary&gt;
/// Defined by &lt;a href="http://tools.ietf.org/html/rfc2104#section-2"&gt;RFC 2104&lt;/a&gt;
/// Is a 64 byte array with all entries set to 0x36.
/// &lt;/summary&gt;
```

```
private static readonly byte[] ipad;

/// &lt;summary&gt;
/// Defined by &lt;a href="http://tools.ietf.org/html/rfc2104#section-2"&gt;RFC 2104&lt;/a&gt;
/// Is a 64 byte array with all entries set to 0x5C.
/// &lt;/summary&gt;
private static readonly byte[] opad;

/// &lt;summary&gt;
/// Initializes the static fields
/// &lt;/summary&gt;
static CramMd5()
{
ipad = new byte[64];
opad = new byte[64];
for (int i = 0; i &lt; ipad.Length; i++)
{
ipad[i] = 0x36;
opad[i] = 0x5C;
}
}

/// &lt;summary&gt;
/// Computes the digest needed to login to a server using CRAM-MD5.&lt;br/&gt;
/// &lt;br/&gt;
/// This computes:&lt;br/&gt;
/// MD5((password XOR opad), MD5((password XOR ipad), challenge))
/// &lt;/summary&gt;
/// &lt;param name="username"&gt;The username of the user who wants to log in&lt;/param&gt;
/// &lt;param name="password"&gt;The password for the &lt;paramref name="username"/&gt;&lt;/param&g
/// &lt;param name="challenge"&gt;
/// The challenge received from the server when telling it CRAM-MD5 authenticated is wanted.
/// Is a base64 encoded string.
/// &lt;/param&gt;
/// &lt;returns&gt;The response to the challenge, which the server can validate and log in the user if correct
/// &lt;exception cref="ArgumentNullException"&gt;
/// If &lt;paramref name="username"/&gt;,
/// &lt;paramref name="password"/&gt; or
/// &lt;paramref name="challenge"/&gt; is &lt;see langword="null"/&gt;
/// &lt;/exception&gt;
internal static string ComputeDigest(string username, string password, string challenge)
{
if (username == null)
throw new ArgumentNullException("username");

if (password == null)
throw new ArgumentNullException("password");

if (challenge == null)
throw new ArgumentNullException("challenge");

// Get the password bytes
byte[] passwordBytes = GetSharedSecretInBytes(password);
```

```
// The challenge is encoded in base64
byte[] challengeBytes = Convert.FromBase64String(challenge);

// Now XOR the password with the opad and ipad magic bytes
byte[] passwordOpad = Xor(passwordBytes, opad);
byte[] passwordIpad = Xor(passwordBytes, ipad);

// Now do the computation: MD5((password XOR opad), MD5((password XOR ipad), challenge))
byte[] digestValue = Hash(Concatenate(passwordOpad, Hash(Concatenate(passwordIpad, challengeBytes

// Convert the bytes to a hex string
// BitConverter writes the output as AF-B3-...
// We need lower-case output without "-"
string hex = BitConverter.ToString(digestValue).Replace("-", "").ToLowerInvariant();

// Include the username in the resulting base64 encoded response
return Convert.ToBase64String(Encoding.ASCII.GetBytes(username + " " + hex));
}

/// &lt;summary&gt;
/// Hashes a byte array using the MD5 algorithm.
/// &lt;/summary&gt;
/// &lt;param name="toHash"&gt;The byte array to hash&lt;/param&gt;
/// &lt;returns&gt;The result of hashing the &lt;paramref name="toHash"/&gt; bytes with the MD5 algorithm&
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="toHash"/&gt; is &lt;see langword=
private static byte[] Hash(byte[] toHash)
{
```

```
if (toHash == null)

throw new ArgumentNullException("toHash");


using (MD5 md5 = new MD5CryptoServiceProvider())

{

return md5.ComputeHash(toHash);

}

}


/// &lt;summary&gt;

/// Concatenates two byte arrays into one

/// &lt;/summary&gt;

/// &lt;param name="one"&gt;The first byte array&lt;/param&gt;

/// &lt;param name="two"&gt;The second byte array&lt;/param&gt;

/// &lt;returns&gt;A concatenated byte array&lt;/returns&gt;

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="one"/&gt; or &lt;paramref name="

private static byte[] Concatenate(byte[] one, byte[] two)

{

if (one == null)

throw new ArgumentNullException("one");


if (two == null)

throw new ArgumentNullException("two");


// Create space for both byte arrays in one

byte[] concatenated = new byte[one.Length + two.Length];
```

```
// Copy the first one over
Buffer.BlockCopy(one, 0, concatenated, 0, one.Length);

// Copy the second one over
Buffer.BlockCopy(two, 0, concatenated, one.Length, two.Length);

// Return result
return concatenated;
}

/// &lt;summary&gt;
/// XORs a byte array with another.&lt;br/&gt;
/// Each byte in &lt;paramref name="toXor"/&gt; is XORed with the corresponding byte
/// in &lt;paramref name="toXorWith"/&gt; until the end of &lt;paramref name="toXor"/&gt; is encountered.
/// &lt;/summary&gt;
/// &lt;param name="toXor"&gt;The byte array to XOR&lt;/param&gt;
/// &lt;param name="toXorWith"&gt;The byte array to XOR with&lt;/param&gt;
/// &lt;returns&gt;A new byte array with the XORed results&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="toXor"/&gt; or &lt;paramref name=
/// &lt;exception cref="ArgumentException"&gt;If the lengths of the arrays are not equal&lt;/exception&gt;
private static byte[] Xor(byte[] toXor, byte[] toXorWith)
{
if (toXor == null)
throw new ArgumentNullException("toXor");

if (toXorWith == null)
throw new ArgumentNullException("toXorWith");
```

```
if (toXor.Length != toXorWith.Length)
throw new ArgumentException("The lengths of the arrays must be equal");

// Create a new array to store results in
byte[] xored = new byte[toXor.Length];

// XOR each individual byte.
for (int i = 0; i &lt; toXor.Length; i++)
{
xored[i] = toXor[i];
xored[i] ^= toXorWith[i];
}

// Return result
return xored;
}
/// &lt;summary&gt;
/// This method is responsible to generate the byte array needed
/// from the shared secret - the password.&lt;br/&gt;
///
/// RFC 2195 says:&lt;br/&gt;
/// The shared secret is null-padded to a length of 64 bytes. If the
/// shared secret is longer than 64 bytes, the MD5 digest of the
/// shared secret is used as a 16 byte input to the keyed MD5
/// calculation.
/// &lt;/summary&gt;
/// &lt;param name="password"&gt;This is the shared secret&lt;/param&gt;
/// &lt;returns&gt;The 64 bytes that is to be used from the shared secret&lt;/returns&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="password"/&gt; is &lt;see langwor
private static byte[] GetSharedSecretInBytes(string password)
{
if (password == null)
throw new ArgumentNullException("password");

// Get the password in bytes
byte[] passwordBytes = Encoding.ASCII.GetBytes(password);

// If the length is larger than 64, we need to
if (passwordBytes.Length &gt; 64)
{
passwordBytes = new MD5CryptoServiceProvider().ComputeHash(passwordBytes);
}

if (passwordBytes.Length != 64)
{
byte[] returner = new byte[64];
for (int i = 0; i &lt; passwordBytes.Length; i++)
{
returner[i] = passwordBytes[i];
}
return returner;
}
```

```
return passwordBytes;
}
}


/// &lt;summary&gt;
/// Class for computing the digest needed when issuing the APOP command to a POP3 server.
/// &lt;/summary&gt;
internal static class Apop
{
/// &lt;summary&gt;
/// Create the digest for the APOP command so that the server can validate
/// we know the password for some user.
/// &lt;/summary&gt;
/// &lt;param name="password"&gt;The password for the user&lt;/param&gt;
/// &lt;param name="serverTimestamp"&gt;The timestamp advertised in the server greeting to the POP3 cli
/// &lt;returns&gt;The password and timestamp hashed with the MD5 algorithm outputted as a HEX string&
public static string ComputeDigest(string password, string serverTimestamp)
{
if (password == null)
throw new ArgumentNullException("password");

if (serverTimestamp == null)
throw new ArgumentNullException("serverTimestamp");

// The APOP command authorizes itself by using the password together
// with the server timestamp. This way the password is not transmitted
// in clear text, and the server can still verify we have the password.
```

```
byte[] digestToHash = Encoding.ASCII.GetBytes(serverTimestamp + password);


using (MD5 md5 = new MD5CryptoServiceProvider())

{

// MD5 hash the digest

byte[] result = md5.ComputeHash(digestToHash);


// Convert the bytes to a hex string

// BitConverter writes the output as AF-B3-...

// We need lower-case output without "-"

return BitConverter.ToString(result).Replace("-", "").ToLowerInvariant();

}

}

}


internal static class StreamUtility

{

/// &lt;summary&gt;

/// Read a line from the stream.

/// A line is interpreted as all the bytes read until a CRLF or LF is encountered.&lt;br/&gt;

/// CRLF pair or LF is not included in the string.

/// &lt;/summary&gt;

/// &lt;param name="stream"&gt;The stream from which the line is to be read&lt;/param&gt;

/// &lt;returns&gt;A line read from the stream returned as a byte array or &lt;see langword="null"/&gt; if no b

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="stream"/&gt; is &lt;see langword=

public static byte[] ReadLineAsBytes(Stream stream)

{
```

```
if (stream == null)

throw new ArgumentNullException("stream");


using (MemoryStream memoryStream = new MemoryStream())

{

while (true)

{

int justRead = stream.ReadByte();

if (justRead == -1 &amp;amp;&amp;amp; memoryStream.Length &gt; 0)

break;


// Check if we started at the end of the stream we read from

// and we have not read anything from it yet

if (justRead == -1 &amp;amp;&amp;amp; memoryStream.Length == 0)

return null;


char readChar = (char)justRead;


// Do not write \r or \n

if (readChar != '\r' &amp;amp;&amp;amp; readChar != '\n')

memoryStream.WriteByte((byte)justRead);


// Last point in CRLF pair

if (readChar == '\n')

break;

}
return memoryStream.ToArray();
}
}

/// &lt;summary&gt;
/// Read a line from the stream. &lt;see cref="ReadLineAsBytes"/&gt; for more documentation.
/// &lt;/summary&gt;
/// &lt;param name="stream"&gt;The stream to read from&lt;/param&gt;
/// &lt;returns&gt;A line read from the stream or &lt;see langword="null"/&gt; if nothing could be read from th
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="stream"/&gt; is &lt;see langword=
public static string ReadLineAsAscii(Stream stream)
{
byte[] readFromStream = ReadLineAsBytes(stream);
return readFromStream != null ? Encoding.ASCII.GetString(readFromStream) : null;
}
}

/// &lt;summary&gt;
/// Some of these states are defined by &lt;a href="http://tools.ietf.org/html/rfc1939"&gt;RFC 1939&lt;/a&gt;.
/// Which commands that are allowed in which state can be seen in the same RFC.&lt;br/&gt;
/// &lt;br/&gt;
/// Used to keep track of which state the &lt;see cref="Pop3Client"/&gt; is in.
/// &lt;/summary&gt;
internal enum ConnectionState
{
/// &lt;summary&gt;
/// This is when the Pop3Client is not even connected to the server
/// &lt;/summary&gt;
Disconnected,

/// &lt;summary&gt;
/// This is when the server is awaiting user credentials
/// &lt;/summary&gt;
Authorization,

/// &lt;summary&gt;
/// This is when the server has been given the user credentials, and we are allowed
/// to use commands specific to this users mail drop
/// &lt;/summary&gt;
Transaction
}

/// &lt;summary&gt;
/// Describes the authentication method to use when authenticating towards a POP3 server.
/// &lt;/summary&gt;
public enum AuthenticationMethod
{
/// &lt;summary&gt;
/// Authenticate using the UsernameAndPassword method.&lt;br/&gt;
/// This will pass the username and password to the server in cleartext.&lt;br/&gt;
/// &lt;see cref="Apop"/&gt; is more secure but might not be supported on a server.&lt;br/&gt;
/// This method is not recommended. Use &lt;see cref="Auto"/&gt; instead.
/// &lt;br/&gt;
/// If SSL is used, there is no loss of security by using this authentication method.
/// &lt;/summary&gt;
UsernameAndPassword,

/// &lt;summary&gt;
/// Authenticate using the Authenticated Post Office Protocol method, which is more secure then
/// &lt;see cref="UsernameAndPassword"/&gt; since it is a request-response protocol where server checks
///  client knows a shared secret, which is the password, without the password itself being transmitted.&lt;br
/// This authentication method uses MD5 under its hood.&lt;br/&gt;
/// &lt;br/&gt;
/// This authentication method is not supported by many servers.&lt;br/&gt;
/// Choose this option if you want maximum security.
/// &lt;/summary&gt;
Apop,

/// &lt;summary&gt;
/// This is the recomended method to authenticate with.&lt;br/&gt;
/// If &lt;see cref="Apop"/&gt; is supported by the server, &lt;see cref="Apop"/&gt; is used for authentication
/// If &lt;see cref="Apop"/&gt; is not supported, Auto will fall back to &lt;see cref="UsernameAndPassword"/
/// &lt;/summary&gt;
Auto,

/// &lt;summary&gt;
/// Logs in the the POP3 server using CRAM-MD5 authentication scheme.&lt;br/&gt;
/// This in essence uses the MD5 hashing algorithm on the user password and a server challenge.
/// &lt;/summary&gt;
CramMd5
}
```

```
/// &lt;summary&gt;
/// POP3 compliant POP Client&lt;br/&gt;
/// &lt;br/&gt;
/// If you want to override where logging is sent, look at &lt;see cref="DefaultLogger"/&gt;
/// &lt;/summary&gt;
/// &lt;example&gt;
/// Examples are available on the &lt;a href="http://hpop.sourceforge.net/"&gt;project homepage&lt;/a&gt;.
/// &lt;/example&gt;
public class Pop3Client : IDisposable
{
#region Private member properties
/// &lt;summary&gt;
/// The stream used to communicate with the server
/// &lt;/summary&gt;
private Stream Stream { get { return _stream; } set { _stream = value; } }
private Stream _stream;

/// &lt;summary&gt;
/// This is the last response the server sent back when a command was issued to it
/// &lt;/summary&gt;
private string LastServerResponse { get { return _lastServerResponse; } set { _lastServerResponse = valu
private string _lastServerResponse;

/// &lt;summary&gt;
/// The APOP time stamp sent by the server in it's welcome message if APOP is supported.
/// &lt;/summary&gt;
```

```
private string ApopTimeStamp { get { return _apopTimeStamp; } set { _apopTimeStamp = value; } }
private string _apopTimeStamp;

/// &lt;summary&gt;
/// Describes what state the &lt;see cref="Pop3Client"/&gt; is in
/// &lt;/summary&gt;
private ConnectionState State { get { return _state; } set { _state = value; } }
private ConnectionState _state;
#endregion

#region Public member properties
/// &lt;summary&gt;
/// Tells whether the &lt;see cref="Pop3Client"/&gt; is connected to a POP server or not
/// &lt;/summary&gt;
public bool Connected { get { return _connected; } private set { _connected = value; } }
private bool _connected;

/// &lt;summary&gt;
/// Allows you to check if the server supports
/// the &lt;see cref="AuthenticationMethod.Apop"/&gt; authentication method.&lt;br/&gt;
/// &lt;br/&gt;
/// This value is filled when the connect method has returned,
/// as the server tells in its welcome message if APOP is supported.
/// &lt;/summary&gt;
public bool ApopSupported { get { return _apopSupported; } private set { _apopSupported = value; } }
private bool _apopSupported;
#endregion
```

```
#region Constructors
/// &lt;summary&gt;
/// Constructs a new Pop3Client for you to use.
/// &lt;/summary&gt;
public Pop3Client()
{
SetInitialValues();
}
#endregion

#region IDisposable implementation

public void Dispose()
{
if (!_isDisposed)
{
try
{
Dispose(true);
}
finally
{
_isDisposed = true;
GC.SuppressFinalize(this);
}
}
```

```
}

private bool _isDisposed=false;


/// &lt;summary&gt;

/// Disposes the &lt;see cref="Pop3Client"/&gt;.&lt;br/&gt;

/// This is the implementation of the &lt;see cref="IDisposable"/&gt; interface.&lt;br/&gt;

/// Sends the QUIT command to the server before closing the streams.

/// &lt;/summary&gt;

/// &lt;param name="disposing"&gt;&lt;see langword="true"/&gt; if managed and unmanaged code should b

protected  void Dispose(bool disposing)

{

if (disposing &amp;amp;&amp;amp; !_isDisposed)

{

if (Connected)

{

Disconnect();

}

}

}


protected void AssertDisposed()

{

if (_isDisposed)

{

string typeName = GetType().FullName;

throw new ObjectDisposedException(typeName, String.Format(System.Globalization.CultureInfo.Invariant

}
```

```
}

#endregion

#region Connection managing methods
/// &lt;summary&gt;
/// Connect to the server using user supplied stream
/// &lt;/summary&gt;
/// &lt;param name="stream"&gt;The stream used to communicate with the server&lt;/param&gt;
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="stream"/&gt; is &lt;see langword=
public void Connect(Stream stream)
{
AssertDisposed();

if (State != ConnectionState.Disconnected)
throw new Exception("You cannot ask to connect to a POP3 server, when we are already connected to one

if (stream == null)
throw new ArgumentNullException("stream");

Stream = stream;

// Fetch the server one-line welcome greeting
string response = StreamUtility.ReadLineAsAscii(Stream);

// Check if the response was an OK response
try
{
// Assume we now need the user to supply credentials
// If we do not connect correctly, Disconnect will set the
// state to Disconnected
// If this is not set, Disconnect will throw an exception
State = ConnectionState.Authorization;

IsOkResponse(response);
ExtractApopTimestamp(response);
Connected = true;
}
catch (Exception e)
{
// If not close down the connection and abort
DisconnectStreams();

//DefaultLogger.Log.LogError("Connect(): " + "Error with connection, maybe POP3 server not exist");
//DefaultLogger.Log.LogDebug("Last response from server was: " + LastServerResponse);
throw new Exception("Server is not available", e);
}
}

/// &lt;summary&gt;
/// Connects to a remote POP3 server using default timeouts of 60.000 milliseconds
/// &lt;/summary&gt;
/// &lt;param name="hostname"&gt;The &lt;paramref name="hostname"/&gt; of the POP3 server&lt;/param
/// &lt;param name="port"&gt;The port of the POP3 server&lt;/param&gt;
/// &lt;param name="useSsl"&gt;&lt;see langword="true"/&gt; if SSL should be used. &lt;see langword="fals
/// &lt;exception cref="PopServerNotAvailableException"&gt;If the server did not send an OK message whe
/// &lt;exception cref="PopServerNotFoundException"&gt;If it was not possible to connect to the server&lt;/e
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="hostname"/&gt; is &lt;see langwor
/// &lt;exception cref="ArgumentOutOfRangeException"&gt;If port is not in the range [&lt;see cref="IPEndPo
public void Connect(string hostname, int port, bool useSsl)
{
const int defaultTimeOut = 60000;
Connect(hostname, port, useSsl, defaultTimeOut, defaultTimeOut, null);
}

/// &lt;summary&gt;
/// Connects to a remote POP3 server
/// &lt;/summary&gt;
/// &lt;param name="hostname"&gt;The &lt;paramref name="hostname"/&gt; of the POP3 server&lt;/param
/// &lt;param name="port"&gt;The port of the POP3 server&lt;/param&gt;
/// &lt;param name="useSsl"&gt;&lt;see langword="true"/&gt; if SSL should be used. &lt;see langword="fals
/// &lt;param name="receiveTimeout"&gt;Timeout in milliseconds before a socket should time out from read
/// &lt;param name="sendTimeout"&gt;Timeout in milliseconds before a socket should time out from sendin
/// &lt;param name="certificateValidator"&gt;If you want to validate the certificate in a SSL connection, pass
/// &lt;exception cref="PopServerNotAvailableException"&gt;If the server did not send an OK message whe
/// &lt;exception cref="PopServerNotFoundException"&gt;If it was not possible to connect to the server&lt;/e
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="hostname"/&gt; is &lt;see langwor
/// &lt;exception cref="ArgumentOutOfRangeException"&gt;If port is not in the range [&lt;see cref="IPEndPo
public void Connect(string hostname, int port, bool useSsl, int receiveTimeout, int sendTimeout, RemoteCe
{
AssertDisposed();
```

```
if (hostname == null)
throw new ArgumentNullException("hostname");

if (hostname.Length == 0)
throw new ArgumentException("hostname cannot be empty", "hostname");

if (port &gt; IPEndPoint.MaxPort || port &lt; IPEndPoint.MinPort)
throw new ArgumentOutOfRangeException("port");

if (receiveTimeout &lt; -1)
throw new ArgumentOutOfRangeException("receiveTimeout");

if (sendTimeout &lt; -1)
throw new ArgumentOutOfRangeException("sendTimeout");

if (State != ConnectionState.Disconnected)
throw new Exception("You cannot ask to connect to a POP3 server, when we are already connected to one

TcpClient clientSocket = new TcpClient();
clientSocket.ReceiveTimeout = receiveTimeout;
clientSocket.SendTimeout = sendTimeout;

try
{
clientSocket.Connect(hostname, port);
}
```

```
catch (SocketException e)

{

// Close the socket - we are not connected, so no need to close stream underneath

clientSocket.Close();


//DefaultLogger.Log.LogError("Connect(): " + e.Message);

throw new Exception("Server not found", e);

}


Stream stream;

if (useSsl)

{

// If we want to use SSL, open a new SSLStream on top of the open TCP stream.

// We also want to close the TCP stream when the SSL stream is closed

// If a validator was passed to us, use it.

SslStream sslStream;

if (certificateValidator == null)

{

sslStream = new SslStream(clientSocket.GetStream(), false);

}

else

{

sslStream = new SslStream(clientSocket.GetStream(), false, certificateValidator);

}

sslStream.ReadTimeout = receiveTimeout;

sslStream.WriteTimeout = sendTimeout;
// Authenticate the server
sslStream.AuthenticateAsClient(hostname);

stream = sslStream;
}
else
{
// If we do not want to use SSL, use plain TCP
stream = clientSocket.GetStream();
}

// Now do the connect with the same stream being used to read and write to
Connect(stream);
}

/// &lt;summary&gt;
/// Disconnects from POP3 server.
/// Sends the QUIT command before closing the connection, which deletes all the messages that was mark
/// &lt;/summary&gt;
public void Disconnect()
{
AssertDisposed();

if (State == ConnectionState.Disconnected)
throw new Exception("You cannot disconnect a connection which is already disconnected");

try
```

```
{
SendCommand("QUIT");
}
finally
{
DisconnectStreams();
}
}
#endregion

#region Authentication methods
/// &lt;summary&gt;
/// Authenticates a user towards the POP server using &lt;see cref="AuthenticationMethod.Auto"/&gt;.&lt;br
/// If this authentication fails but you are sure that the username and password is correct, it might
/// be that that the POP3 server is wrongly telling the client it supports &lt;see cref="AuthenticationMethod.A
/// You should try using &lt;see cref="Authenticate(string, string, AuthenticationMethod)"/&gt; while passing
/// &lt;/summary&gt;
/// &lt;param name="username"&gt;The username&lt;/param&gt;
/// &lt;param name="password"&gt;The user password&lt;/param&gt;
/// &lt;exception cref="InvalidLoginException"&gt;If the user credentials was not accepted&lt;/exception&gt;
/// &lt;exception cref="PopServerLockedException"&gt;If the server said the the mailbox was locked&lt;/exc
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="username"/&gt; or &lt;paramref na
/// &lt;exception cref="LoginDelayException"&gt;If the server rejects the login because of too recent logins&
public void Authenticate(string username, string password)
{
AssertDisposed();
Authenticate(username, password, AuthenticationMethod.Auto);
```

```
}


/// &lt;summary&gt;

/// Authenticates a user towards the POP server using some &lt;see cref="AuthenticationMethod"/&gt;.

/// &lt;/summary&gt;

/// &lt;param name="username"&gt;The username&lt;/param&gt;

/// &lt;param name="password"&gt;The user password&lt;/param&gt;

/// &lt;param name="authenticationMethod"&gt;The way that the client should authenticate towards the serv

/// &lt;exception cref="NotSupportedException"&gt;If &lt;see cref="AuthenticationMethod.Apop"/&gt; is used

/// &lt;exception cref="InvalidLoginException"&gt;If the user credentials was not accepted&lt;/exception&gt;

/// &lt;exception cref="PopServerLockedException"&gt;If the server said the the mailbox was locked&lt;/exc

/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="username"/&gt; or &lt;paramref na

/// &lt;exception cref="LoginDelayException"&gt;If the server rejects the login because of too recent logins&

public void Authenticate(string username, string password, AuthenticationMethod authenticationMethod)

{

AssertDisposed();


if (username == null)

throw new ArgumentNullException("username");


if (password == null)

throw new ArgumentNullException("password");


if (State != ConnectionState.Authorization)

throw new Exception("You have to be connected and not authorized when trying to authorize yourself");


try
```

```
{
switch (authenticationMethod)
{
case AuthenticationMethod.UsernameAndPassword:
AuthenticateUsingUserAndPassword(username, password);
break;

case AuthenticationMethod.Apop:
AuthenticateUsingApop(username, password);
break;

case AuthenticationMethod.Auto:
if (ApopSupported)
AuthenticateUsingApop(username, password);
else
AuthenticateUsingUserAndPassword(username, password);
break;

case AuthenticationMethod.CramMd5:
AuthenticateUsingCramMd5(username, password);
break;
}
}
catch (Exception e)
{
//DefaultLogger.Log.LogError("Problem logging in using method " + authenticationMethod + ". Server respo
// Throw a more specific exception if special cases of failure is detected
// using the response the server generated when the last command was sent
CheckFailedLoginServerResponse(LastServerResponse, e);

// If no special failure is detected, tell that the login credentials were wrong
throw e;
}

// We are now authenticated and therefore we enter the transaction state
State = ConnectionState.Transaction;
}

/// &lt;summary&gt;
/// Authenticates a user towards the POP server using the USER and PASSWORD commands
/// &lt;/summary&gt;
/// &lt;param name="username"&gt;The username&lt;/param&gt;
/// &lt;param name="password"&gt;The user password&lt;/param&gt;
/// &lt;exception cref="PopServerException"&gt;If the server responded with -ERR&lt;/exception&gt;
private void AuthenticateUsingUserAndPassword(string username, string password)
{
SendCommand("USER " + username);
SendCommand("PASS " + password);

// Authentication was successful if no exceptions thrown before getting here
}

/// &lt;summary&gt;
/// Authenticates a user towards the POP server using APOP
/// &lt;/summary&gt;
/// &lt;param name="username"&gt;The username&lt;/param&gt;
/// &lt;param name="password"&gt;The user password&lt;/param&gt;
/// &lt;exception cref="NotSupportedException"&gt;Thrown when the server does not support APOP&lt;/exc
/// &lt;exception cref="PopServerException"&gt;If the server responded with -ERR&lt;/exception&gt;
private void AuthenticateUsingApop(string username, string password)
{
if (!ApopSupported)
throw new NotSupportedException("APOP is not supported on this server");

SendCommand("APOP " + username + " " + Apop.ComputeDigest(password, ApopTimeStamp));

// Authentication was successful if no exceptions thrown before getting here
}


/// &lt;summary&gt;
/// Authenticates using the CRAM-MD5 authentication method
/// &lt;/summary&gt;
/// &lt;param name="username"&gt;The username&lt;/param&gt;
/// &lt;param name="password"&gt;The user password&lt;/param&gt;
/// &lt;exception cref="NotSupportedException"&gt;Thrown when the server does not support AUTH CRAM
/// &lt;exception cref="InvalidLoginException"&gt;If the user credentials was not accepted&lt;/exception&gt;
/// &lt;exception cref="PopServerLockedException"&gt;If the server said the the mailbox was locked&lt;/exc
/// &lt;exception cref="LoginDelayException"&gt;If the server rejects the login because of too recent logins&
private void AuthenticateUsingCramMd5(string username, string password)
{
```

```
// Example of communication:

// C: AUTH CRAM-MD5

// S: + PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+

// C: dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw

// S: +OK CRAM authentication successful


// Other example, where AUTH CRAM-MD5 is not supported

// C: AUTH CRAM-MD5

// S: -ERR Authentication method CRAM-MD5 not supported


try

{

SendCommand("AUTH CRAM-MD5");

}

catch (Exception e)

{

// A PopServerException will be thrown if the server responds with a -ERR not supported

throw new NotSupportedException("CRAM-MD5 authentication not supported", e);

}


// Fetch out the challenge from the server response

string challenge = LastServerResponse.Substring(2);


// Compute the challenge response

string response = CramMd5.ComputeDigest(username, password, challenge);


// Send the response to the server
```

```
SendCommand(response);

// Authentication was successful if no exceptions thrown before getting here
}
#endregion

#region Public POP3 commands
/// &lt;summary&gt;
/// Get the number of messages on the server using a STAT command
/// &lt;/summary&gt;
/// &lt;returns&gt;The message count on the server&lt;/returns&gt;
/// &lt;exception cref="PopServerException"&gt;If the server did not accept the STAT command&lt;/exceptio
public int GetMessageCount()
{
AssertDisposed();

if (State != ConnectionState.Transaction)
throw new Exception("You cannot get the message count without authenticating yourself towards the serve

return SendCommandIntResponse("STAT", 1);
}

/// &lt;summary&gt;
/// Marks the message with the given message number as deleted.&lt;br/&gt;
/// &lt;br/&gt;
/// The message will not be deleted until a QUIT command is sent to the server.&lt;br/&gt;
/// This is done when you call &lt;see cref="Disconnect()"/&gt; or when the Pop3Client is &lt;see cref="Disp
/// &lt;/summary&gt;
/// &lt;param name="messageNumber"&gt;
/// The number of the message to be deleted. This message may not already have been deleted.&lt;br/&gt;
/// The &lt;paramref name="messageNumber"/&gt; must be inside the range [1, messageCount]
/// &lt;/param&gt;
/// &lt;exception cref="PopServerException"&gt;If the server did not accept the delete command&lt;/excepti
public void DeleteMessage(int messageNumber)
{
AssertDisposed();

ValidateMessageNumber(messageNumber);

if (State != ConnectionState.Transaction)
throw new Exception("You cannot delete any messages without authenticating yourself towards the server

SendCommand("DELE " + messageNumber);
}

/// &lt;summary&gt;
/// Marks all messages as deleted.&lt;br/&gt;
/// &lt;br/&gt;
/// The messages will not be deleted until a QUIT command is sent to the server.&lt;br/&gt;
/// This is done when you call &lt;see cref="Disconnect()"/&gt; or when the Pop3Client is &lt;see cref="Disp
/// The method assumes that no prior message has been marked as deleted, and is not valid to call if this i
/// &lt;/summary&gt;
/// &lt;exception cref="PopServerException"&gt;If the server did not accept one of the delete commands. A
public void DeleteAllMessages()
```

```
{
AssertDisposed();

int messageCount = GetMessageCount();

for (int messageItem = messageCount; messageItem &gt; 0; messageItem--)
{
DeleteMessage(messageItem);
}
}

/// &lt;summary&gt;
/// Keep server active by sending a NOOP command.&lt;br/&gt;
/// This might keep the server from closing the connection due to inactivity.&lt;br/&gt;
/// &lt;br/&gt;
/// RFC:&lt;br/&gt;
/// The POP3 server does nothing, it merely replies with a positive response.
/// &lt;/summary&gt;
/// &lt;exception cref="PopServerException"&gt;If the server did not accept the NOOP command&lt;/except
public void NoOperation()
{
AssertDisposed();

if (State != ConnectionState.Transaction)
throw new Exception("You cannot use the NOOP command unless you are authenticated to the server");

SendCommand("NOOP");
}


/// &lt;summary&gt;
/// Send a reset command to the server.&lt;br/&gt;
/// &lt;br/&gt;
/// RFC:&lt;br/&gt;
/// If any messages have been marked as deleted by the POP3
/// server, they are unmarked. The POP3 server then replies
/// with a positive response.
/// &lt;/summary&gt;
/// &lt;exception cref="PopServerException"&gt;If the server did not accept the RSET command&lt;/excepti
public void Reset()
{
AssertDisposed();

if (State != ConnectionState.Transaction)
throw new Exception("You cannot use the RSET command unless you are authenticated to the server");

SendCommand("RSET");
}


/// &lt;summary&gt;
/// Get a unique ID for a single message.&lt;br/&gt;
/// &lt;/summary&gt;
/// &lt;param name="messageNumber"&gt;
/// Message number, which may not be marked as deleted.&lt;br/&gt;
/// The &lt;paramref name="messageNumber"/&gt; must be inside the range [1, messageCount]
/// &lt;/param&gt;

/// &lt;returns&gt;The unique ID for the message&lt;/returns&gt;

/// &lt;exception cref="PopServerException"&gt;If the server did not accept the UIDL command. This could

public string GetMessageUid(int messageNumber)

{

AssertDisposed();


ValidateMessageNumber(messageNumber);


if (State != ConnectionState.Transaction)

throw new Exception("Cannot get message ID, when the user has not been authenticated yet");


// Example from RFC:

//C: UIDL 2

//S: +OK 2 QhdPYR:00WBw1Ph7x7


SendCommand("UIDL " + messageNumber);


// Parse out the unique ID

return LastServerResponse.Split(' ')[2];

}


/// &lt;summary&gt;

/// Gets a list of unique IDs for all messages.&lt;br/&gt;

/// Messages marked as deleted are not listed.

/// &lt;/summary&gt;

/// &lt;returns&gt;
/// A list containing the unique IDs in sorted order from message number 1 and upwards.
/// &lt;/returns&gt;
/// &lt;exception cref="PopServerException"&gt;If the server did not accept the UIDL command&lt;/exceptio
public List&lt;string&gt; GetMessageUids()
{
AssertDisposed();

if (State != ConnectionState.Transaction)
throw new Exception("Cannot get message IDs, when the user has not been authenticated yet");

// RFC Example:
// C: UIDL
// S: +OK
// S: 1 whqtswO00WBw418f9t5JxYwZ
// S: 2 QhdPYR:00WBw1Ph7x7
// S: .      // this is the end

SendCommand("UIDL");

List&lt;string&gt; uids = new List&lt;string&gt;();

string response;
// Keep reading until multi-line ends with a "."
while (!IsLastLineInMultiLineResponse(response = StreamUtility.ReadLineAsAscii(Stream)))
{
// Add the unique ID to the list
uids.Add(response.Split(' ')[1]);
}


return uids;

}


/// &lt;summary&gt;

/// Gets the size in bytes of a single message

/// &lt;/summary&gt;

/// &lt;param name="messageNumber"&gt;

/// The number of a message which may not be a message marked as deleted.&lt;br/&gt;

/// The &lt;paramref name="messageNumber"/&gt; must be inside the range [1, messageCount]

/// &lt;/param&gt;

/// &lt;returns&gt;Size of the message&lt;/returns&gt;

/// &lt;exception cref="PopServerException"&gt;If the server did not accept the LIST command&lt;/exceptio

public int GetMessageSize(int messageNumber)

{

AssertDisposed();


ValidateMessageNumber(messageNumber);


if (State != ConnectionState.Transaction)

throw new Exception("Cannot get message size, when the user has not been authenticated yet");


// RFC Example:

// C: LIST 2

// S: +OK 2 200

return SendCommandIntResponse("LIST " + messageNumber, 2);
```

```
}

/// &lt;summary&gt;
/// Get the sizes in bytes of all the messages.&lt;br/&gt;
/// Messages marked as deleted are not listed.
/// &lt;/summary&gt;
/// &lt;returns&gt;Size of each message excluding deleted ones&lt;/returns&gt;
/// &lt;exception cref="PopServerException"&gt;If the server did not accept the LIST command&lt;/exceptio
public List&lt;int&gt; GetMessageSizes()
{
AssertDisposed();

if (State != ConnectionState.Transaction)
throw new Exception("Cannot get message sizes, when the user has not been authenticated yet");

// RFC Example:
// C: LIST
// S: +OK 2 messages (320 octets)
// S: 1 120
// S: 2 200
// S: .       // End of multi-line

SendCommand("LIST");

List&lt;int&gt; sizes = new List&lt;int&gt;();

string response;
```

```
// Read until end of multi-line

while (!".".Equals(response = StreamUtility.ReadLineAsAscii(Stream)))

{

sizes.Add(int.Parse(response.Split(' ')[1], CultureInfo.InvariantCulture));

}


return sizes;

}


/// &lt;summary&gt;

/// Fetches a message from the server and parses it

/// &lt;/summary&gt;

/// &lt;param name="messageNumber"&gt;

/// Message number on server, which may not be marked as deleted.&lt;br/&gt;

/// Must be inside the range [1, messageCount]

/// &lt;/param&gt;

/// &lt;returns&gt;The message, containing the email message&lt;/returns&gt;

/// &lt;exception cref="PopServerException"&gt;If the server did not accept the command sent to fetch the r

public Message GetMessage(int messageNumber)

{

AssertDisposed();


ValidateMessageNumber(messageNumber);


if (State != ConnectionState.Transaction)

throw new Exception("Cannot fetch a message, when the user has not been authenticated yet");
```

```
byte[] messageContent = GetMessageAsBytes(messageNumber);


return new Message(messageContent);

}


/// &lt;summary&gt;

/// Fetches a message in raw form from the server

/// &lt;/summary&gt;

/// &lt;param name="messageNumber"&gt;

/// Message number on server, which may not be marked as deleted.&lt;br/&gt;

/// Must be inside the range [1, messageCount]

/// &lt;/param&gt;

/// &lt;returns&gt;The raw bytes of the message&lt;/returns&gt;

/// &lt;exception cref="PopServerException"&gt;If the server did not accept the command sent to fetch the

public byte[] GetMessageAsBytes(int messageNumber)

{

AssertDisposed();


ValidateMessageNumber(messageNumber);


if (State != ConnectionState.Transaction)

throw new Exception("Cannot fetch a message, when the user has not been authenticated yet");


// Get the full message

return GetMessageAsBytes(messageNumber, false);

}
```

```
/// &lt;summary&gt;

/// Get all the headers for a message.&lt;br/&gt;

/// The server will not need to send the body of the message.

/// &lt;/summary&gt;

/// &lt;param name="messageNumber"&gt;

/// Message number, which may not be marked as deleted.&lt;br/&gt;

/// Must be inside the range [1, messageCount]

/// &lt;/param&gt;

/// &lt;returns&gt;MessageHeaders object&lt;/returns&gt;

/// &lt;exception cref="PopServerException"&gt;If the server did not accept the command sent to fetch the r

public MessageHeader GetMessageHeaders(int messageNumber)

{

AssertDisposed();


ValidateMessageNumber(messageNumber);


if (State != ConnectionState.Transaction)

throw new Exception("Cannot fetch a message, when the user has not been authenticated yet");


// Only fetch the header part of the message

byte[] messageContent = GetMessageAsBytes(messageNumber, true);


// Do not parse the body - as it is not in the byte array

return new Message(messageContent, false).Headers;

}


/// &lt;summary&gt;
/// Asks the server to return it's capability listing.&lt;br/&gt;

/// This is an optional command, which a server is not enforced to accept.

/// &lt;/summary&gt;

/// &lt;returns&gt;

/// The returned Dictionary keys are the capability names.&lt;br/&gt;

/// The Lists pointed to are the capability parameters fitting that certain capability name.

/// See &lt;a href="http://tools.ietf.org/html/rfc2449#section-6"&gt;RFC section 6&lt;/a&gt; for explanation fo

/// &lt;/returns&gt;

/// &lt;remarks&gt;

/// Capabilities are case-insensitive.&lt;br/&gt;

/// The dictionary uses case-insensitive searching, but the Lists inside

/// does not. Therefore you will have to use something like the code below

/// to search for a capability parameter.&lt;br/&gt;

/// foo is the capability name and bar is the capability parameter.

/// &lt;code&gt;

/// List&amp;amp;lt;string&amp;amp;gt; arguments = capabilities["foo"];

/// bool contains = null != arguments.Find(delegate(string str)

///    {

///      return String.Compare(str, "bar", true) == 0;

///    });

/// &lt;/code&gt;

/// If we were running on .NET framework &gt;= 3.5, a HashSet could have been used.

/// &lt;/remarks&gt;

/// &lt;exception cref="PopServerException"&gt;If the server did not accept the capability command&lt;/exce

public Dictionary&lt;string, List&lt;string&gt;&gt; Capabilities()

{

AssertDisposed();
```

```
if (State != ConnectionState.Authorization &amp;amp;&amp;amp; State != ConnectionState.Transaction)

throw new Exception("Capability command only available while connected or authenticated");


// RFC Example

// Examples:

// C: CAPA

// S: +OK Capability list follows

// S: TOP

// S: USER

// S: SASL CRAM-MD5 KERBEROS_V4

// S: RESP-CODES

// S: LOGIN-DELAY 900

// S: PIPELINING

// S: EXPIRE 60

// S: UIDL

// S: IMPLEMENTATION Shlemazle-Plotz-v302

// S: .

SendCommand("CAPA");


// Capablities are case-insensitive

Dictionary&lt;string, List&lt;string&gt;&gt; capabilities = new Dictionary&lt;string, List&lt;string&gt;&gt;(String


string lineRead;

// Keep reading until we are at the end of the multi line response

while (!IsLastLineInMultiLineResponse(lineRead = StreamUtility.ReadLineAsAscii(Stream)))

{
```

```
// Example of read line
// SASL CRAM-MD5 KERBEROS_V4
// SASL is the name of the capability while
// CRAM-MD5 and KERBEROS_V4 are arguments to SASL
string[] splitted = lineRead.Split(' ');

// There should always be a capability name
string capabilityName = splitted[0];

// Find all the arguments
List&lt;string&gt; capabilityArguments = new List&lt;string&gt;();
for (int i = 1; i &lt; splitted.Length; i++)
{
capabilityArguments.Add(splitted[i]);
}

// Add the capability found to the dictionary
capabilities.Add(capabilityName, capabilityArguments);
}

return capabilities;
}
#endregion

#region Private helper methods
/// &lt;summary&gt;
/// Examines string to see if it contains a time stamp to use with the APOP command.&lt;br/&gt;
/// If it does, sets the &lt;see cref="ApopTimeStamp"/&gt; property to this value.
/// &lt;/summary&gt;
/// &lt;param name="response"&gt;The string to examine&lt;/param&gt;
private void ExtractApopTimestamp(string response)
{
// RFC Example:
// +OK POP3 server ready &lt;1896.697170952@dbc.mtview.ca.us&gt;
Match match = Regex.Match(response, "&lt;.+&gt;");
if (match.Success)
{
ApopTimeStamp = match.Value;
ApopSupported = true;
}
}

/// &lt;summary&gt;
/// Tests a string to see if it is a "+" string.&lt;br/&gt;
/// An "+" string should be returned by a compliant POP3
/// server if the request could be served.&lt;br/&gt;
/// &lt;br/&gt;
/// The method does only check if it starts with "+".
/// &lt;/summary&gt;
/// &lt;param name="response"&gt;The string to examine&lt;/param&gt;
/// &lt;exception cref="PopServerException"&gt;Thrown if server did not respond with "+" message&lt;/exce
private static void IsOkResponse(string response)
{
if (response == null)
```

```
throw new Exception("The stream used to retrieve responses from was closed");


if (response.StartsWith("+", StringComparison.OrdinalIgnoreCase))

return;


throw new Exception("The server did not respond with a + response. The response was: \"" + response + "

}


/// &lt;summary&gt;

/// Sends a command to the POP server.&lt;br/&gt;

/// If this fails, an exception is thrown.

/// &lt;/summary&gt;

/// &lt;param name="command"&gt;The command to send to server&lt;/param&gt;

/// &lt;exception cref="PopServerException"&gt;If the server did not send an OK message to the command

private void SendCommand(string command)

{

// Convert the command with CRLF afterwards as per RFC to a byte array which we can write

byte[] commandBytes = Encoding.ASCII.GetBytes(command + "\r\n");


// Write the command to the server

Stream.Write(commandBytes, 0, commandBytes.Length);

Stream.Flush(); // Flush the content as we now wait for a response


// Read the response from the server. The response should be in ASCII

LastServerResponse = StreamUtility.ReadLineAsAscii(Stream);


IsOkResponse(LastServerResponse);
}


/// &lt;summary&gt;

/// Sends a command to the POP server, expects an integer reply in the response

/// &lt;/summary&gt;

/// &lt;param name="command"&gt;command to send to server&lt;/param&gt;

/// &lt;param name="location"&gt;

/// The location of the int to return.&lt;br/&gt;

/// Example:&lt;br/&gt;

/// &lt;c&gt;S: +OK 2 200&lt;/c&gt;&lt;br/&gt;

/// Set &lt;paramref name="location"/&gt;=1 to get 2&lt;br/&gt;

/// Set &lt;paramref name="location"/&gt;=2 to get 200&lt;br/&gt;

/// &lt;/param&gt;

/// &lt;returns&gt;Integer value in the reply&lt;/returns&gt;

/// &lt;exception cref="PopServerException"&gt;If the server did not accept the command&lt;/exception&gt;

private int SendCommandIntResponse(string command, int location)

{

SendCommand(command);


return int.Parse(LastServerResponse.Split(' ')[location], CultureInfo.InvariantCulture);

}


/// &lt;summary&gt;

/// Asks the server for a message and returns the message response as a byte array.

/// &lt;/summary&gt;

/// &lt;param name="messageNumber"&gt;

/// Message number on server, which may not be marked as deleted.&lt;br/&gt;
/// Must be inside the range [1, messageCount]
/// &lt;/param&gt;
/// &lt;param name="askOnlyForHeaders"&gt;If &lt;see langword="true"/&gt; only the header part of the me
/// &lt;returns&gt;A byte array that the message requested consists of&lt;/returns&gt;
/// &lt;exception cref="PopServerException"&gt;If the server did not accept the command sent to fetch the r
private byte[] GetMessageAsBytes(int messageNumber, bool askOnlyForHeaders)
{
AssertDisposed();

ValidateMessageNumber(messageNumber);

if (State != ConnectionState.Transaction)
throw new Exception("Cannot fetch a message, when the user has not been authenticated yet");

if (askOnlyForHeaders)
{
// 0 is the number of lines of the message body to fetch, therefore it is set to zero to fetch only headers
SendCommand("TOP " + messageNumber + " 0");
}
else
{
// Ask for the full message
SendCommand("RETR " + messageNumber);
}

// RFC 1939 Example
// C: RETR 1
// S: +OK 120 octets

// S: &lt;the POP3 server sends the entire message here&gt;

// S: .


// Create a byte array builder which we use to write the bytes too

// When done, we can get the byte array out

using (MemoryStream byteArrayBuilder = new MemoryStream())

{

bool first = true;

byte[] lineRead;


// Keep reading until we are at the end of the multi line response

while (!IsLastLineInMultiLineResponse(lineRead = StreamUtility.ReadLineAsBytes(Stream)))

{

// We should not write CRLF on the very last line, therefore we do this

if (!first)

{

// Write CRLF which was not included in the lineRead bytes of last line

byte[] crlfPair = Encoding.ASCII.GetBytes("\r\n");

byteArrayBuilder.Write(crlfPair, 0, crlfPair.Length);

}

else

{

// We are now not the first anymore

first = false;

}
// This is a multi-line. See http://tools.ietf.org/html/rfc1939#section-3

// It says that a line starting with "." and not having CRLF after it

// is a multi line, and the "." should be stripped

if (lineRead.Length &gt; 0 &amp;amp;&amp;amp; lineRead[0] == '.')

{

// Do not write the first period

byteArrayBuilder.Write(lineRead, 1, lineRead.Length - 1);

}

else

{

// Write everything

byteArrayBuilder.Write(lineRead, 0, lineRead.Length);

}

}


// If we are fetching a header - add an extra line to denote the headers ended

if (askOnlyForHeaders)

{

byte[] crlfPair = Encoding.ASCII.GetBytes("\r\n");

byteArrayBuilder.Write(crlfPair, 0, crlfPair.Length);

}


// Get out the bytes we have written to byteArrayBuilder

byte[] receivedBytes = byteArrayBuilder.ToArray();


return receivedBytes;

}
```

```
}

/// &lt;summary&gt;
/// Check if the bytes received is the last line in a multi line response
/// from the pop3 server. It is the last line if the line contains only a "."
/// &lt;/summary&gt;
/// &lt;param name="bytesReceived"&gt;The last line received from the server, which could be the last resp
/// &lt;returns&gt;&lt;see langword="true"/&gt; if last line in a multi line response, &lt;see langword="false"/&
/// &lt;exception cref="ArgumentNullException"&gt;If &lt;paramref name="bytesReceived"/&gt; is &lt;see lar
private static bool IsLastLineInMultiLineResponse(byte[] bytesReceived)
{
if (bytesReceived == null)
throw new ArgumentNullException("bytesReceived");

return bytesReceived.Length == 1 &amp;amp;&amp;amp; bytesReceived[0] == '.';
}

/// &lt;see cref="IsLastLineInMultiLineResponse(byte[])"&gt; for documentation&lt;/see&gt;
private static bool IsLastLineInMultiLineResponse(string lineReceived)
{
if (lineReceived == null)
throw new ArgumentNullException("lineReceived");

// If the string is indeed the last line, then it is okay to do ASCII encoding
// on it. For performance reasons we check if the length is equal to 1
// so that we do not need to decode a long message string just to see if
// it is the last line
```

```
return lineReceived.Length == 1 &amp;amp;&amp;amp; IsLastLineInMultiLineResponse(Encoding.ASCII.G

}


/// &lt;summary&gt;

/// Method for checking that a &lt;paramref name="messageNumber"/&gt; argument given to some method

/// is indeed valid. If not, &lt;see cref="InvalidUseException"/&gt; will be thrown.

/// &lt;/summary&gt;

/// &lt;param name="messageNumber"&gt;The message number to validate&lt;/param&gt;

private static void ValidateMessageNumber(int messageNumber)

{

if (messageNumber &lt;= 0)

throw new Exception("The messageNumber argument cannot have a value of zero or less. Valid message

}


/// &lt;summary&gt;

/// Closes down the streams and sets the Pop3Client into the initial configuration

/// &lt;/summary&gt;

private void DisconnectStreams()

{

try

{

Stream.Close();

}

finally

{

// Reset values to initial state

SetInitialValues();
}
}


/// &lt;summary&gt;
/// Sets the initial values on the public properties of this Pop3Client.
/// &lt;/summary&gt;
private void SetInitialValues()
{
// We have not seen the APOPTimestamp yet
ApopTimeStamp = null;


// We are not connected
Connected = false;
State = ConnectionState.Disconnected;


// APOP is not supported before we check on login
ApopSupported = false;
}


/// &lt;summary&gt;
/// Checks for extra response codes when an authentication has failed and throws
/// the correct exception.
/// If no such response codes is found, nothing happens.
/// &lt;/summary&gt;
/// &lt;param name="serverErrorResponse"&gt;The server response string&lt;/param&gt;
/// &lt;param name="e"&gt;The exception thrown because the server responded with -ERR&lt;/param&gt;
/// &lt;exception cref="PopServerLockedException"&gt;If the account is locked or in use&lt;/exception&gt;
/// &lt;exception cref="LoginDelayException"&gt;If the server rejects the login because of too recent logins&

private static void CheckFailedLoginServerResponse(string serverErrorResponse, Exception e)

{

string upper = serverErrorResponse.ToUpperInvariant();


// Bracketed strings are extra response codes addded

// in RFC http://tools.ietf.org/html/rfc2449

// together with the CAPA command.


// Specifies the account is in use

if (upper.Contains("[IN-USE]") || upper.Contains("LOCK"))

{

//DefaultLogger.Log.LogError("Authentication: maildrop is locked or in-use");

throw e;

}


// Specifies that there must go some time between logins

if (upper.Contains("[LOGIN-DELAY]"))

{

throw e;

}

}

#endregion

}

</code>

</stage>

<stage stageid="e74c516a-5c0f-4539-88d1-4af07d49b311" name="Clean Up" type="SubSheetInfo">
```

```
<subsheetid>ad30cfbe-8a8c-44e4-8a26-6ca67f058f7d</subsheetid>
<narrative>
</narrative>
<displayx>-195</displayx>
<displayy>-105</displayy>
<displaywidth>150</displaywidth>
<displayheight>90</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="453e6ca1-71dc-4ce2-8a1c-d6512bd6b502" name="Start" type="Start">
<subsheetid>ad30cfbe-8a8c-44e4-8a26-6ca67f058f7d</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>98566d14-9c68-423f-a446-0c63a9d56369</onsuccess>
</stage>
<stage stageid="98566d14-9c68-423f-a446-0c63a9d56369" name="End" type="End">
<subsheetid>ad30cfbe-8a8c-44e4-8a26-6ca67f058f7d</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
```

```
<displayy>90</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="dfd23e65-5e00-48c2-9222-b028de2d5425" name="new" type="Note">

<subsheetid>ad30cfbe-8a8c-44e4-8a26-6ca67f058f7d</subsheetid>

<narrative>Clean Up Page


This is an optional page where you might choose to perform some finalisation (or "cleanup") tasks as your


The cleanup action will be called automatically immediately after closing your business object at the end of


You will not be able to call this action from a business process, nor will it be called at any other time than b

<displayx>-180</displayx>

<displayy>60</displayy>

<displaywidth>180</displaywidth>

<displayheight>230</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="9ffa7f6a-6f1a-4b3a-800b-c5b3d02f1053" name="new" type="Note">

<narrative>Initialise Page


This is an optional page where you might choose to perform some initialisation tasks after your business ol


The initialise action will be called automatically immediately after loading your business object.
```

```
You will not be able to call this action from a business process, nor will it be called at any other time than a

<displayx>-180</displayx>

<displayy>60</displayy>

<displaywidth>180</displaywidth>

<displayheight>230</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="3d27a452-2516-48ee-a57b-363989e0a677" name="Delete Message" type="SubSheetInf

<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>

<narrative>Deletes a message from the POP3 server.</narrative>

<displayx>-195</displayx>

<displayy>-105</displayy>

<displaywidth>150</displaywidth>

<displayheight>90</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="7120f051-794a-428f-b0ca-ada23ab0da34" name="Start" type="Start">

<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-135</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>
```

```
<input type="text" name="MessageID" narrative="The ID of the message to delete." stage="MessageID"/>
</inputs>
<onsuccess>fdb5293f-358f-4a67-b88b-4caa37a0f16f</onsuccess>
</stage>
<stage stageid="10c06eb3-10d6-4759-9f38-22fdb99f5635" name="MessageID" type="Data">
<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-135</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="f7d6db43-bdda-4cf0-bee9-37c1bfaadd6c" name="Configure" type="SubSheetInfo">
<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>
<narrative>Configures POP3 and SMTP settings for the Business Object</narrative>
<displayx>-195</displayx>
<displayy>-105</displayy>
<displaywidth>150</displaywidth>
<displayheight>90</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
```

```
<stage stageid="26f68cd5-5a95-4e00-863e-0e3393c2e05c" name="End" type="End">

<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-15</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="0c9b2a5e-a669-43e8-94ce-0a93ac299716" name="Start" type="Start">

<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-105</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="text" name="Username" narrative="The POP3 and SMTP Username" stage="Username"/>

<input type="password" name="Password" narrative="The POP3 and SMTP Password" stage="Password"

<input type="text" name="POP3 Server" narrative="The POP3 Server address e.g. pop.domain.com" stage

<input type="text" name="SMTP Server" narrative="The SMTP Server address e.g. smtp.domain.com" stag

<input type="number" name="POP3 Port" narrative="The POP3 Port, e.g. 110 or 995" stage="POP3 Port"/
```

```
<input type="number" name="SMTP Port" narrative="The SMTP Port, e.g 25 or 465" stage="SMTP Port"/>
<input type="flag" name="POP3 UseSSL" narrative="Whether to use SSL for POP3" stage="POP3 UseSS
<input type="flag" name="SMTP UseSSL" narrative="Whether to use SSL for SMTP" stage="SMTP UseSS
</inputs>
<onsuccess>5fc137ae-22ff-4a9a-8e93-7c3923a7775f</onsuccess>
</stage>
<stage stageid="6200a401-764c-40bb-9c01-710765a7198a" name="Username" type="Data">
<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>
<narrative>
</narrative>
<displayx>165</displayx>
<displayy>-150</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<alwaysinit/>
</stage>
<stage stageid="949b007a-73ec-49c8-bf67-97989d93b9d2" name="Password" type="Data">
<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>
<narrative>
</narrative>
<displayx>240</displayx>
<displayy>-150</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
```

```
<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>password</datatype>

<initialvalueenc>

</initialvalueenc>

<alwaysinit/>

</stage>

<stage stageid="c92379ab-616d-4e9f-bbdf-76d8f4a464de" name="Disconnect POP3" type="SubSheetInfo

<subsheetid>c8b9e601-7444-4385-b2aa-6709658ad472</subsheetid>

<narrative>

</narrative>

<displayx>-195</displayx>

<displayy>-105</displayy>

<displaywidth>150</displaywidth>

<displayheight>90</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="68269076-5ba4-49b3-bb2b-e89927d49df3" name="Start" type="Start">

<subsheetid>c8b9e601-7444-4385-b2aa-6709658ad472</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-105</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<onsuccess>d4140c57-5f78-45ff-8e59-d720e57c6bb2</onsuccess>
```

```
</stage>
<stage stageid="f304774e-9251-4092-b2a7-32424accffdb" name="End" type="End">
<subsheetid>c8b9e601-7444-4385-b2aa-6709658ad472</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>30</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="d4140c57-5f78-45ff-8e59-d720e57c6bb2" name="Disconnect" type="Code">
<subsheetid>c8b9e601-7444-4385-b2aa-6709658ad472</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-60</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>367756ad-7a12-4c98-95ef-e1c9324c9ffa</onsuccess>
<code>_rclient.Disconnect();</code>
</stage>
<stage stageid="d85d6404-ed67-4802-b251-3c6e4f553bc3" name="Delete Message" type="Code">
<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>
```

```
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>0</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="number" name="MessageNumber" expr="[Message Number]"/>
</inputs>
<onsuccess>1027af62-c9e7-443b-a361-2b2fa991aed1</onsuccess>
<code>_rclient.DeleteMessage((int)MessageNumber);</code>
</stage>
<stage stageid="64b00b6b-bad6-4e81-bdc2-72fcaaf69961" name="End" type="End">
<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>90</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="5fc137ae-22ff-4a9a-8e93-7c3923a7775f" name="Set Configured" type="Calculation">
<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>
```

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-60</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<onsuccess>26f68cd5-5a95-4e00-863e-0e3393c2e05c</onsuccess>

<calculation expression="True" stage="Configured"/>

</stage>

<stage stageid="e691cdbf-68a7-46c5-9469-911182bc8475" name="Connect POP3" type="SubSheetInfo">

<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>

<narrative>

</narrative>

<displayx>-195</displayx>

<displayy>-105</displayy>

<displaywidth>150</displaywidth>

<displayheight>90</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="15b67dac-088d-4cf3-a6a7-2feca9d5c99d" name="Start" type="Start">

<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

```
<displayy>-180</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="Action" stage="Action"/>
</inputs>
<onsuccess>3ca489aa-ab27-465b-b189-8875b7dd615d</onsuccess>
</stage>
<stage stageid="710abe98-0572-4ea2-adab-5ac1226ed5ac" name="End" type="End">
<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>120</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="73bf49d5-81cc-4f1a-b89d-ed0c90a05a86" name="Configured" type="Data">
<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-60</displayy>
```

```
<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>flag</datatype>

<initialvalue>False</initialvalue>

<alwaysinit/>

</stage>

<stage stageid="bcb58d07-9f70-40cf-9df3-eeff2d3a75b6" name="Configured" type="Decision">

<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-60</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<decision expression="[Configured]"/>

<ontrue>6a90465e-93b4-4c91-95c2-cfc40b74d63c</ontrue>

<onfalse>2f2d374d-c22d-470e-99e5-d1ca2fce2c95</onfalse>

</stage>

<stage stageid="6a90465e-93b4-4c91-95c2-cfc40b74d63c" name="Connect and Authenticate" type="Cod

<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>
```

```
<displayy>0</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="Username" expr="[Username]"/>
<input type="text" name="Password" expr="[Password]"/>
<input type="text" name="Server" expr="[POP3 Server]"/>
<input type="number" name="Port" expr="[POP3 Port]"/>
<input type="flag" name="Use SSL" expr="[POP3 UseSSL]"/>
</inputs>
<onsuccess>5827b973-4390-423c-b457-4a225cd63717</onsuccess>
<code>_rclient = new Pop3Client();
_rclient.Connect(Server,(int)Port,Use_SSL);
_rclient.Authenticate(Username,Password);</code>
</stage>
<stage stageid="2f2d374d-c22d-470e-99e5-d1ca2fce2c95" name="Exception1" type="Exception">
<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-60</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<exception type="" detail="&amp;quot;Cannot connect to Server you must use Configure first&amp;quot; "/
</stage>
```

```
<stage stageid="7d982653-252b-40ea-aa32-027aa81ece7c" name="Action" type="Data">
<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-120</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="5827b973-4390-423c-b457-4a225cd63717" name="Set Connected" type="Calculation">
<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>60</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>710abe98-0572-4ea2-adab-5ac1226ed5ac</onsuccess>
<calculation expression="True" stage="POP3 Connected"/>
</stage>
```

```
<stage stageid="aee3539f-93d1-4661-8e48-e207e7fda381" name="POP3 Connected" type="Data">

<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>

<narrative>

</narrative>

<displayx>90</displayx>

<displayy>60</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>flag</datatype>

<initialvalue>False</initialvalue>

</stage>

<stage stageid="3ca489aa-ab27-465b-b189-8875b7dd615d" name="Connected" type="Decision">

<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-120</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<decision expression="[POP3 Connected]"/>

<ontrue>d99cd226-dd16-4417-af86-11e756edfb93</ontrue>

<onfalse>bcb58d07-9f70-40cf-9df3-eeff2d3a75b6</onfalse>

</stage>

<stage stageid="d99cd226-dd16-4417-af86-11e756edfb93" name="End" type="End">
```

<subsheetid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>-75</displayx>

<displayy>-120</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="1fa82664-ea42-4255-84ab-50b61958fe56" name="POP3 Server" type="Data">

<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>

<narrative>

</narrative>

<displayx>165</displayx>

<displayy>-105</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>text</datatype>

<initialvalue/>

<alwaysinit/>

</stage>

<stage stageid="23794a48-9b7c-4d9c-b4ea-93c47ab545fe" name="SMTP Server" type="Data">

<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>

<narrative>

</narrative>

```xml
<displayx>240</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<alwaysinit/>
</stage>
<stage stageid="bb607f8e-6326-4b20-9d1c-be2061af1a0a" name="SMTP Port" type="Data">
<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>
<narrative>
</narrative>
<displayx>240</displayx>
<displayy>-60</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>number</datatype>
<initialvalue/>
<alwaysinit/>
</stage>
<stage stageid="8d588294-9205-4d6f-abb1-bdc102beac7e" name="POP3 Port" type="Data">
<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>
<narrative>
</narrative>
<displayx>165</displayx>
```

```
<displayy>-60</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>number</datatype>

<initialvalue/>

<alwaysinit/>

</stage>

<stage stageid="4b819ad9-214e-4499-b11b-73d0fd122070" name="SMTP UseSSL" type="Data">

<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>

<narrative>

</narrative>

<displayx>240</displayx>

<displayy>-15</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>flag</datatype>

<initialvalue/>

<alwaysinit/>

</stage>

<stage stageid="47441526-8d37-4599-921c-9e4103ac87aa" name="POP3 UseSSL" type="Data">

<subsheetid>544abb24-e623-4adb-a24d-3a5dba6164ba</subsheetid>

<narrative>

</narrative>

<displayx>165</displayx>

<displayy>-15</displayy>
```

```
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>flag</datatype>
<initialvalue/>
<alwaysinit/>
</stage>
<stage stageid="367756ad-7a12-4c98-95ef-e1c9324c9ffa" name="Set Connected" type="Calculation">
<subsheetid>c8b9e601-7444-4385-b2aa-6709658ad472</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-15</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>f304774e-9251-4092-b2a7-32424accffdb</onsuccess>
<calculation expression="False" stage="POP3 Connected"/>
</stage>
<stage stageid="1027af62-c9e7-443b-a361-2b2fa991aed1" name="Disconnect POP3" type="SubSheet">
<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>45</displayy>
```

```
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>64b00b6b-bad6-4e81-bdc2-72fcaaf69961</onsuccess>
<processid>c8b9e601-7444-4385-b2aa-6709658ad472</processid>
</stage>
<stage stageid="943a7e4b-9dbd-427b-a3a1-4f2810050f82" name="Send Message" type="SubSheetInfo">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<narrative>Sends a message using the SMTP protocol.</narrative>
<displayx>-195</displayx>
<displayy>-105</displayy>
<displaywidth>150</displaywidth>
<displayheight>90</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="eb463c09-de32-4dde-94c3-cf0507ee79e3" name="Start" type="Start">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>-45</displayx>
<displayy>-135</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="From" narrative="The email address of the sender." stage="From"/>
```

```
<input type="text" name="To" narrative="The email address of the recipient. To address multiple recipients
<input type="text" name="Subject" narrative="The subject of the email." stage="Subject"/>
<input type="text" name="Body" narrative="The body of the email." stage="Body"/>
<input type="collection" name="Attachments" narrative="A Collection containing a list of files to add to the
<input type="flag" name="BodyIsHTML" narrative="Indicates if the body should be sent as HTML" stage="
</inputs>
<onsuccess>8e55392e-ae1e-443f-9d5b-37fbeb948c47</onsuccess>
</stage>
<stage stageid="7e86926f-9f8a-4e11-aff3-b0ea75a9e350" name="End" type="End">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>-45</displayx>
<displayy>0</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="18563c9d-55d4-4aea-9713-58595b565208" name="From" type="Data">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<narrative>
</narrative>
<displayx>-225</displayx>
<displayy>-30</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
```

```
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="7e3eba6a-938c-4747-91fb-f065fd8e1ecd" name="To" type="Data">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<narrative>
</narrative>
<displayx>-165</displayx>
<displayy>-30</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="6cbf338c-56d6-45b9-bfa6-40dcc60f55f4" name="Subject" type="Data">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<narrative>
</narrative>
<displayx>-225</displayx>
<displayy>15</displayy>
<displaywidth>60</displaywidth>
```

```
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="364ba15a-711a-471f-a468-2fe38c091107" name="Body" type="Data">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<narrative>
</narrative>
<displayx>-165</displayx>
<displayy>15</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="a212d081-53fb-4398-bda0-462ce18c07c2" name="Send Message" type="Code">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>-45</displayx>
```

```
<displayy>-45</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="Server" expr="[SMTP Server]"/>
<input type="number" name="Port" expr="[SMTP Port]"/>
<input type="text" name="Username" expr="[Username]"/>
<input type="password" name="Password" expr="[Password]"/>
<input type="flag" name="UseSSL" expr="[SMTP UseSSL]"/>
<input type="text" name="From" expr="[From]"/>
<input type="text" name="To" expr="[To]"/>
<input type="text" name="Subject" expr="[Subject]"/>
<input type="text" name="Body" expr="[Body]"/>
<input type="collection" name="Attachments" expr="[Attachments]"/>
<input type="flag" name="BodyIsHTML" expr="[BodyIsHTML]"/>
</inputs>
<onsuccess>7e86926f-9f8a-4e11-aff3-b0ea75a9e350</onsuccess>
<code>SmtpClient client = new SmtpClient();
try
{
client.Host = Server;
client.Port = (int)Port;
if (Username != "")
client.Credentials = new NetworkCredential(Username,Password);
client.EnableSsl = UseSSL;
```

```
using(MailMessage mail = new MailMessage())
{
mail.From = new MailAddress(From);
mail.To.Add(To);
mail.Subject = Subject;
mail.IsBodyHtml = BodyIsHTML;
mail.Body = Body;

foreach(DataRow dr in Attachments.Rows)
{
string file = dr["Path"].ToString();
Attachment data = new Attachment(file, MediaTypeNames.Application.Octet);
ContentDisposition dis = data.ContentDisposition;
dis.CreationDate = File.GetCreationTime(file);
dis.ModificationDate = File.GetLastWriteTime(file);
dis.ReadDate = File.GetLastAccessTime(file);
mail.Attachments.Add(data);
}

client.Send(mail);
}
}
catch(Exception ex)
{
string msg = ex.Message;
if(ex.InnerException != null) {
msg += " - " + ex.InnerException.Message;
}

throw new Exception(msg);

}

finally

{

IDisposable disposableClient = client as IDisposable;

if (disposableClient!=null)

disposableClient.Dispose();

}</code>

</stage>

<stage stageid="fdb5293f-358f-4a67-b88b-4caa37a0f16f" name="Connect POP3" type="SubSheet">

<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-90</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="text" name="Action" expr="&amp;quot;Get All Messages&amp;quot;"/>

</inputs>

<onsuccess>7bb623a1-88ca-4162-8eba-a0d384df4aec</onsuccess>

<processid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</processid>

</stage>

<stage stageid="5c4dcdf6-a371-4878-9c4c-3fd08aa568f4" name="List Messages" type="SubSheetInfo">
```

```
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>

<narrative>

</narrative>

<displayx>-195</displayx>

<displayy>-105</displayy>

<displaywidth>150</displaywidth>

<displayheight>90</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="f8e7202a-f413-4d68-b38e-4b6d789318f1" name="Start" type="Start">

<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-105</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<onsuccess>c47a1fa4-9fa6-4b4c-b35c-5d03d65fad46</onsuccess>

</stage>

<stage stageid="76845230-717a-4efe-954c-b10309c6cc6c" name="End" type="End">

<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>165</displayx>
```

```
<displayy>60</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<outputs>
<output type="collection" name="Emails" stage="Emails"/>
</outputs>
</stage>
<stage stageid="797b45e9-3197-454c-9068-7baefc351f75" name="MessageCount" type="Data">
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>0</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>number</datatype>
<initialvalue>0</initialvalue>
<private/>
<alwaysinit/>
</stage>
<stage stageid="b94061e0-c242-4e91-8bc6-ed1cc12cd5e9" name="Messages Left?" type="Decision">
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<loginhibit/>
<narrative>
</narrative>
```

<displayx>15</displayx>
<displayy>60</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<decision expression="[MessageCount] &amp;gt;0"/>
<ontrue>0f6d2f90-451c-4d3b-89cf-517bddb760d8</ontrue>
<onfalse>76845230-717a-4efe-954c-b10309c6cc6c</onfalse>
</stage>
<stage stageid="cecc8228-7a4a-4e96-b398-8757b6db702d" name="Decrement" type="Calculation">
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>255</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>aafa7d4a-dfb0-4446-9323-8afa0b721cf0</onsuccess>
<calculation expression="[MessageCount]-1" stage="MessageCount"/>
</stage>
<stage stageid="002f58e7-8123-4b0a-bcca-2319263c33d1" name="Get Message Headers1" type="Code"
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<loginhibit/>
<narrative>
</narrative>

```
<displayx>15</displayx>
<displayy>180</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="number" name="MessageNumber" expr="[MessageCount]"/>
</inputs>
<outputs>
<output type="text" name="Subject" stage="Emails.Subject"/>
<output type="text" name="From Name" stage="Emails.From Name"/>
<output type="text" name="From Email" stage="Emails.From Address"/>
<output type="datetime" name="Date Sent" stage="Emails.Date Sent"/>
<output type="text" name="MessageID" stage="Emails.MessageID"/>
</outputs>
<onsuccess>cecc8228-7a4a-4e96-b398-8757b6db702d</onsuccess>
<code>MessageHeader result =_rclient.GetMessageHeaders((int)MessageNumber);
Subject = result.Subject;
From_Name = result.From.DisplayName;
From_Email = result.From.Address;
Date_Sent = result.DateSent;
MessageID = result.MessageId;</code>
</stage>
<stage stageid="aafa7d4a-dfb0-4446-9323-8afa0b721cf0" name="anchor1" type="Anchor">
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<loginhibit/>
<narrative>
```

```
</narrative>
<displayx>-75</displayx>
<displayy>255</displayy>
<displaywidth>10</displaywidth>
<displayheight>10</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>6a475ba3-f7b4-4ead-8691-921b64e9e896</onsuccess>
</stage>
<stage stageid="6a475ba3-f7b4-4ead-8691-921b64e9e896" name="anchor2" type="Anchor">
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>-75</displayx>
<displayy>60</displayy>
<displaywidth>10</displaywidth>
<displayheight>10</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>b94061e0-c242-4e91-8bc6-ed1cc12cd5e9</onsuccess>
</stage>
<stage stageid="0f6d2f90-451c-4d3b-89cf-517bddb760d8" name="Add Row" type="Action">
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>120</displayy>
```

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="text" name="Collection Name" narrative="The name of the collection to be modified" expr="&a

</inputs>

<onsuccess>002f58e7-8123-4b0a-bcca-2319263c33d1</onsuccess>

<resource object="Blueprism.AutomateProcessCore.clsCollectionActions" action="Add Row"/>

</stage>

<stage stageid="38f14f0c-f87e-4c96-8836-8f5b8faced14" name="Emails" type="Collection">

<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>165</displayx>

<displayy>105</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>collection</datatype>

<private/>

<alwaysinit/>

<collectioninfo>

<field name="Subject" type="text" description="The subject of the email."/>

<field name="From Name" type="text" description="The name of the sender."/>

<field name="From Address" type="text" description="The email address of the sender."/>

<field name="Date Sent" type="datetime" description="The date and time at which the email was sent."/>

```
<field name="MessageID" type="text" description="The ID of the email message."/>
</collectioninfo>
</stage>
<stage stageid="0bf982d4-8f0e-4d69-aa12-0ac3996f3d84" name="Get Message Count2" type="Code">
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>0</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<outputs>
<output type="number" name="MessageCount" stage="MessageCount"/>
</outputs>
<onsuccess>b94061e0-c242-4e91-8bc6-ed1cc12cd5e9</onsuccess>
<code>MessageCount=_rclient.GetMessageCount();</code>
</stage>
<stage stageid="c47a1fa4-9fa6-4b4c-b35c-5d03d65fad46" name="Connect POP3" type="SubSheet">
<subsheetid>e0fb4432-bf1a-4bdf-856a-a52d9acb4d9f</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-60</displayy>
<displaywidth>60</displaywidth>
```

<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="Action" expr="&amp;quot;Get All Messages&amp;quot;"/>
</inputs>
<onsuccess>0bf982d4-8f0e-4d69-aa12-0ac3996f3d84</onsuccess>
<processid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</processid>
</stage>
<stage stageid="9879be0d-2188-4595-91b6-ac07d1e348dc" name="Get Message" type="SubSheetInfo">
<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>
<narrative>Gets a message using the POP3 protocol. By default, for multipart messages the plain text part
<displayx>-195</displayx>
<displayy>-60</displayy>
<displaywidth>150</displaywidth>
<displayheight>180</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="6a181147-fc29-471a-8902-d9fffbb71d86" name="Start" type="Start">
<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>

```
<inputs>
<input type="text" name="MessageID" narrative="The ID of the message to retrieve." stage="MessageID"/
<input type="flag" name="HTMLPreferred" narrative="Return HTML content in preference to plain text (mu
</inputs>
<onsuccess>b218b56f-ef0d-4199-91ee-f1ccb4eec209</onsuccess>
</stage>
<stage stageid="f44ba718-e5c7-48fd-8233-0e9bfbf2048d" name="MessageID" type="Data">
<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="39a05d5d-741a-4c0a-a59c-f5952b55bb44" name="End" type="End">
<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>75</displayy>
```

```
<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<outputs>

<output type="text" name="Body" narrative="The email body" stage="Body"/>

<output type="text" name="Content-Type" narrative="The content type of the returned body (e.g. text/plain

</outputs>

</stage>

<stage stageid="b218b56f-ef0d-4199-91ee-f1ccb4eec209" name="Connect POP3" type="SubSheet">

<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-60</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="text" name="Action" expr="&amp;quot;Get All Messages&amp;quot;"/>

</inputs>

<onsuccess>fb2ca3a4-1e9e-4b39-b820-40e7c035d7d8</onsuccess>

<processid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</processid>

</stage>

<stage stageid="e0ceae00-8c9e-4201-8a4b-2e3b1a6b5293" name="Get Message" type="Code">

<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>

<loginhibit/>
```

```
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>30</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="number" name="MessageNumber" expr="[Message Number]"/>
<input type="flag" name="HTMLPreferred" expr="[HTMLPreferred]"/>
</inputs>
<outputs>
<output type="text" name="Body" stage="Body"/>
<output type="text" name="Content-Type" stage="Content-Type"/>
</outputs>
<onsuccess>39a05d5d-741a-4c0a-a59c-f5952b55bb44</onsuccess>
<code>Message m =_rclient.GetMessage((int)MessageNumber);
MessagePart p;
if (HTMLPreferred) {
p = m.FindFirstHtmlVersion();
if (p == null) p = m.FindFirstPlainTextVersion();
} else {
p = m.FindFirstPlainTextVersion();
if (p == null) p = m.FindFirstHtmlVersion();
}
if (p != null) {
Body = p.GetBodyAsText();
Content_Type = p.ContentType.ToString();

} else {

Body = "";

Content_Type = "";

}</code>

</stage>

<stage stageid="aa902ce3-629f-4d73-87a8-5e04f070405f" name="Body" type="Data">

<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>

<narrative>

</narrative>

<displayx>90</displayx>

<displayy>75</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>text</datatype>

<initialvalue/>

<private/>

<alwaysinit/>

</stage>

<stage stageid="d64e2310-ca59-4f22-8594-a0cd08d3d78f" name="Number From ID" type="SubSheetInfo

<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>

<narrative>

</narrative>

<displayx>-195</displayx>

<displayy>-105</displayy>

<displaywidth>150</displaywidth>
```

```
<displayheight>90</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="63a1c937-63b0-4a26-aba7-716654c5da18" name="Start" type="Start">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="MessageID" stage="MessageID"/>
</inputs>
<onsuccess>f6de7ad3-6964-449c-bdc0-135401e01d62</onsuccess>
</stage>
<stage stageid="9e58393f-f316-40c2-bf5b-7aaa994e24ec" name="MessageID" type="Data">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
```

```
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="40f3248b-d376-4904-aef8-505a14c96731" name="End" type="End">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>165</displayx>
<displayy>15</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<outputs>
<output type="number" name="Message Number" stage="MessageCount"/>
</outputs>
</stage>
<stage stageid="f6de7ad3-6964-449c-bdc0-135401e01d62" name="Get Message Count" type="Code">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-45</displayy>
<displaywidth>60</displaywidth>
```

```
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<outputs>
<output type="number" name="MessageCount" stage="MessageCount"/>
</outputs>
<onsuccess>0623c4e8-9fd1-40c7-8d57-d9b00ad7f140</onsuccess>
<code>MessageCount = _rclient.GetMessageCount();</code>
</stage>
<stage stageid="f1f6e86c-668e-46ae-8fad-4da426311236" name="MessageCount" type="Data">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-45</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>number</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="0623c4e8-9fd1-40c7-8d57-d9b00ad7f140" name="Messages Left?" type="Decision">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<loginhibit/>
<narrative>
</narrative>
```

```
<displayx>15</displayx>
<displayy>15</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<decision expression="[MessageCount] &amp;gt;0"/>
<ontrue>c9e08180-d230-4857-a8c5-361f80b97a8c</ontrue>
<onfalse>40f3248b-d376-4904-aef8-505a14c96731</onfalse>
</stage>
<stage stageid="dbf5258e-83cb-432e-b51e-596ef2ccbb01" name="Decrement" type="Calculation">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>195</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>a21a95f1-5257-48c7-bf2b-db274edbc70f</onsuccess>
<calculation expression="[MessageCount]-1" stage="MessageCount"/>
</stage>
<stage stageid="c9e08180-d230-4857-a8c5-361f80b97a8c" name="Get Message Headers" type="Code">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<loginhibit/>
<narrative>
</narrative>
```

```
<displayx>15</displayx>

<displayy>75</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="number" name="MessageNumber" expr="[MessageCount]"/>

</inputs>

<outputs>

<output type="text" name="MessageID" stage="FoundMessageID"/>

</outputs>

<onsuccess>20b52376-80d0-49b8-8b7f-12e300c62410</onsuccess>

<code>MessageHeader result =_rclient.GetMessageHeaders((int)MessageNumber);

MessageID = result.MessageId;</code>

</stage>

<stage stageid="a21a95f1-5257-48c7-bf2b-db274edbc70f" name="anchor1" type="Anchor">

<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>-75</displayx>

<displayy>195</displayy>

<displaywidth>10</displaywidth>

<displayheight>10</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<onsuccess>37af2a50-a79d-4d9c-913e-cb970afaa33f</onsuccess>

</stage>
```

```
<stage stageid="37af2a50-a79d-4d9c-913e-cb970afaa33f" name="anchor2" type="Anchor">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>-75</displayx>
<displayy>15</displayy>
<displaywidth>10</displaywidth>
<displayheight>10</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>0623c4e8-9fd1-40c7-8d57-d9b00ad7f140</onsuccess>
</stage>
<stage stageid="ddcb6ae8-4ccf-4a15-a377-97a5535a287b" name="FoundMessageID" type="Data">
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>75</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="20b52376-80d0-49b8-8b7f-12e300c62410" name="Message Match" type="Decision">
```

```
<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>120</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<decision expression="[MessageID]=[FoundMessageID]"/>

<ontrue>73b0921c-7ac9-4ede-ac69-891397d2cd3b</ontrue>

<onfalse>dbf5258e-83cb-432e-b51e-596ef2ccbb01</onfalse>

</stage>

<stage stageid="73b0921c-7ac9-4ede-ac69-891397d2cd3b" name="End" type="End">

<subsheetid>4688b112-6575-4f8a-980c-713566729518</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>165</displayx>

<displayy>120</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<outputs>

<output type="number" name="Message Number" stage="MessageCount"/>

</outputs>

</stage>
```

```
<stage stageid="7bb623a1-88ca-4162-8eba-a0d384df4aec" name="Number From ID" type="SubSheet">
<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-45</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="MessageID" expr="[MessageID]"/>
</inputs>
<outputs>
<output type="number" name="Message Number" stage="Message Number"/>
</outputs>
<onsuccess>d85d6404-ed67-4802-b251-3c6e4f553bc3</onsuccess>
<processid>4688b112-6575-4f8a-980c-713566729518</processid>
</stage>
<stage stageid="356a7e1e-cf63-4cde-a441-7f6889494278" name="Message Number" type="Data">
<subsheetid>8279ef2e-ecb1-434b-a10d-cc7c6d1a5e8e</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-45</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
```

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>number</datatype>

<initialvalue/>

<private/>

<alwaysinit/>

</stage>

<stage stageid="fb2ca3a4-1e9e-4b39-b820-40e7c035d7d8" name="Number From ID" type="SubSheet">

<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-15</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="text" name="MessageID" expr="[MessageID]"/>

</inputs>

<outputs>

<output type="number" name="Message Number" stage="Message Number"/>

</outputs>

<onsuccess>e0ceae00-8c9e-4201-8a4b-2e3b1a6b5293</onsuccess>

<processid>4688b112-6575-4f8a-980c-713566729518</processid>

</stage>

<stage stageid="8fabc428-4d6b-46b9-90d6-27513ef82084" name="Message Number" type="Data">

<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>

<narrative>

</narrative>

<displayx>90</displayx>

<displayy>-15</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>number</datatype>

<initialvalue/>

<private/>

<alwaysinit/>

</stage>

<stage stageid="f06ff7dc-dee1-4319-8fb8-e7324bbeda46" name="Delete Messages" type="SubSheetInfo"

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<narrative>Deletes a collection identifying messages from the POP3 server.</narrative>

<displayx>-195</displayx>

<displayy>-105</displayy>

<displaywidth>150</displaywidth>

<displayheight>90</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="d878691d-ebbf-449b-b3e9-1a4dbc780391" name="Start" type="Start">

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-105</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="collection" name="MessageIDs" narrative="A collection containing the IDs of the messages to

</inputs>

<onsuccess>12f5806c-a210-4e38-8f15-5d8cbaa6fc9b</onsuccess>

</stage>

<stage stageid="b2f67ca9-9e76-4ae8-ab11-bbeb3991f9f1" name="Delete Message1" type="Code">

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>210</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="number" name="MessageNumber" expr="[MessageNumbers.MessageNumber]"/>

</inputs>

<onsuccess>5acbc759-910b-43a4-aad3-277348fe735e</onsuccess>

<code>_rclient.DeleteMessage((int)MessageNumber);</code>

</stage>

<stage stageid="592900a1-6fa2-4aae-8df6-4aceeb59cd3c" name="End" type="End">

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

```
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>345</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="b5f506e0-8569-45f6-bc34-f11ca13a180f" name="Disconnect POP3" type="SubSheet">
<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>300</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<onsuccess>592900a1-6fa2-4aae-8df6-4aceeb59cd3c</onsuccess>
<processid>c8b9e601-7444-4385-b2aa-6709658ad472</processid>
</stage>
<stage stageid="12f5806c-a210-4e38-8f15-5d8cbaa6fc9b" name="Connect POP3" type="SubSheet">
<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>
<loginhibit/>
<narrative>
</narrative>
```

```
<displayx>15</displayx>
<displayy>-60</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="Action" expr="&amp;quot;Get All Messages&amp;quot;"/>
</inputs>
<onsuccess>b542fa7b-76a0-4ac7-bef8-b7523eed14dc</onsuccess>
<processid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</processid>
</stage>
<stage stageid="d1c8749a-8674-4230-980f-d75287257eb7" name="Number From ID" type="SubSheet">
<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>75</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="MessageID" expr="[MessageIDs.MessageID]"/>
</inputs>
<outputs>
<output type="number" name="Message Number" stage="MessageNumbers.MessageNumber"/>
</outputs>
```

```
<onsuccess>37fa1082-1d8f-4a93-a6bf-ebfa986abde1</onsuccess>

<processid>4688b112-6575-4f8a-980c-713566729518</processid>

</stage>

<stage stageid="b542fa7b-76a0-4ac7-bef8-b7523eed14dc" name="For Each MessageID" type="LoopStar

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>-15</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<onsuccess>05f7d6d8-79ad-4aa7-823f-6d114dfdff84</onsuccess>

<groupid>d1ff5376-69c7-47a4-a5a5-71e609d81702</groupid>

<looptype>ForEach</looptype>

<loopdata>MessageIDs</loopdata>

</stage>

<stage stageid="37fa1082-1d8f-4a93-a6bf-ebfa986abde1" name="Next" type="LoopEnd">

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>120</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>
```

```
<font family="Tahoma" size="10" style="Regular" color="000000"/>

<onsuccess>227fbd4a-9ce9-42be-b920-b00ae83b5705</onsuccess>

<groupid>d1ff5376-69c7-47a4-a5a5-71e609d81702</groupid>

</stage>

<stage stageid="05f7d6d8-79ad-4aa7-823f-6d114dfdff84" name="Add Row" type="Action">

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>30</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="text" name="Collection Name" narrative="The name of the collection to be modified" expr="&a

</inputs>

<onsuccess>d1c8749a-8674-4230-980f-d75287257eb7</onsuccess>

<resource object="Blueprism.AutomateProcessCore.clsCollectionActions" action="Add Row"/>

</stage>

<stage stageid="26d19bda-3d6e-41ad-817e-cbec74832a0f" name="MessageNumbers" type="Collection">

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<narrative>

</narrative>

<displayx>90</displayx>

<displayy>75</displayy>

<displaywidth>60</displaywidth>
```

```
<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>collection</datatype>

<private/>

<alwaysinit/>

<collectioninfo>

<field name="MessageNumber" type="number"/>

</collectioninfo>

</stage>

<stage stageid="227fbd4a-9ce9-42be-b920-b00ae83b5705" name="For Each Message Number" type="Lo

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>165</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<onsuccess>b2f67ca9-9e76-4ae8-ab11-bbeb3991f9f1</onsuccess>

<groupid>0faefc94-574d-4561-8432-32eb13a0c75d</groupid>

<looptype>ForEach</looptype>

<loopdata>MessageNumbers</loopdata>

</stage>

<stage stageid="5acbc759-910b-43a4-aad3-277348fe735e" name="Next" type="LoopEnd">

<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<loginhibit/>
```

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>255</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<onsuccess>b5f506e0-8569-45f6-bc34-f11ca13a180f</onsuccess>

<groupid>0faefc94-574d-4561-8432-32eb13a0c75d</groupid>

</stage>

<stage stageid="4c85d0ab-d738-40a4-b0b2-d59986be4091" name="Save Attachments" type="SubSheetI

<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>

<narrative>Gets a message attachment and saves it to a given folder.</narrative>

<displayx>-195</displayx>

<displayy>-105</displayy>

<displaywidth>150</displaywidth>

<displayheight>90</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

</stage>

<stage stageid="795b56fc-9b21-4bfe-9766-bb07f4b1280a" name="End" type="End">

<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>15</displayx>

<displayy>90</displayy>

<displaywidth>60</displaywidth>

```
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
</stage>
<stage stageid="1bb24baa-c27d-4ab7-9081-fc6414148434" name="Start" type="Start">
<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="MessageID" narrative="The ID of the email message." stage="MessageID"/>
<input type="text" name="Folder" narrative="A Folder in which to save all the attachments." stage="Folder"
</inputs>
<onsuccess>61363752-629c-4ad8-b3fb-c6de4554c3f9</onsuccess>
</stage>
<stage stageid="c0b68cab-c88f-45cd-9b05-9fafa3790fd2" name="MessageID" type="Data">
<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>
<narrative>
</narrative>
<displayx>90</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
```

```
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="61363752-629c-4ad8-b3fb-c6de4554c3f9" name="Connect POP3" type="SubSheet">
<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>-60</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="text" name="Action" expr="&amp;quot;Get All Messages&amp;quot;"/>
</inputs>
<onsuccess>2f95ac8b-e89b-4ca2-9c6f-877551c2bd64</onsuccess>
<processid>954d358e-45d9-44b3-b56a-87e1c21f9d0f</processid>
</stage>
<stage stageid="2f95ac8b-e89b-4ca2-9c6f-877551c2bd64" name="Number From ID" type="SubSheet">
<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>
<loginhibit/>
<narrative>
</narrative>
```

```
<displayx>15</displayx>

<displayy>-15</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<inputs>

<input type="text" name="MessageID" expr="[MessageID]"/>

</inputs>

<outputs>

<output type="number" name="Message Number" stage="Message Number"/>

</outputs>

<onsuccess>03fcd315-a8c0-4853-ad26-2b4808074ab9</onsuccess>

<processid>4688b112-6575-4f8a-980c-713566729518</processid>

</stage>

<stage stageid="5e21321e-a465-46f7-83f0-8d7e2912c93f" name="Message Number" type="Data">

<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>

<narrative>

</narrative>

<displayx>90</displayx>

<displayy>-15</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>number</datatype>

<initialvalue/>

<private/>

<alwaysinit/>
```

```
</stage>
<stage stageid="9c9a7a0d-e77a-4368-b7f9-269d1feadbe2" name="Folder" type="Data">
<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>
<narrative>
</narrative>
<displayx>165</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>text</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="03fcd315-a8c0-4853-ad26-2b4808074ab9" name="Save Attachments" type="Code">
<subsheetid>803f1a2c-38f9-4614-97aa-5ca66d454d2d</subsheetid>
<loginhibit/>
<narrative>
</narrative>
<displayx>15</displayx>
<displayy>45</displayy>
<displaywidth>90</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<inputs>
<input type="number" name="MessageNumber" expr="[Message Number]"/>
```

```
<input type="text" name="Folder" expr="[Folder]"/>
</inputs>
<onsuccess>795b56fc-9b21-4bfe-9766-bb07f4b1280a</onsuccess>
<code>Message m = _rclient.GetMessage((int)MessageNumber);
foreach(MessagePart a in m.FindAllAttachments())
{
string path=System.IO.Path.Combine(Folder,a.FileName);
a.Save(new FileInfo(path));
}</code>
</stage>
<stage stageid="fb9ba538-acf2-42b4-879b-a26cc69aa7e9" name="Attachments" type="Collection">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<narrative>
</narrative>
<displayx>-195</displayx>
<displayy>60</displayy>
<displaywidth>120</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>collection</datatype>
<private/>
<alwaysinit/>
<collectioninfo>
<field name="Path" type="text" description="The path to a file to include as an attachment in the email."/>
</collectioninfo>
</stage>
<stage stageid="c57bad69-0560-4732-b2cd-ecdac5946b2b" name="MessageIDs" type="Collection">
```

```
<subsheetid>849d1a37-179b-40ae-9683-4ebb0d951576</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>105</displayx>

<displayy>-105</displayy>

<displaywidth>60</displaywidth>

<displayheight>30</displayheight>

<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>collection</datatype>

<private/>

<alwaysinit/>

<collectioninfo>

<field name="Subject" type="text" description="The subject of the email."/>

<field name="From Name" type="text" description="The name of the sender."/>

<field name="From Address" type="text" description="The email address of the sender."/>

<field name="Date Sent" type="datetime" description="The date and time at which the email was sent."/>

<field name="MessageID" type="text" description="The ID of the email message."/>

</collectioninfo>

</stage>

<stage stageid="8e55392e-ae1e-443f-9d5b-37fbeb948c47" name="Configured" type="Decision">

<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>

<loginhibit/>

<narrative>

</narrative>

<displayx>-45</displayx>

<displayy>-90</displayy>
```

```
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<decision expression="[Configured]"/>
<ontrue>a212d081-53fb-4398-bda0-462ce18c07c2</ontrue>
<onfalse>7b2c9cad-ad8c-4892-8a7e-a77fceda0298</onfalse>
</stage>
<stage stageid="7b2c9cad-ad8c-4892-8a7e-a77fceda0298" name="FAIL" type="Exception">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<narrative>
</narrative>
<displayx>30</displayx>
<displayy>-90</displayy>
<displaywidth>30</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<exception type="ConfigurationException" detail="&amp;quot;Cannot connect to Server you must use Con
</stage>
<stage stageid="8166f9b5-deea-4781-8584-e8777525856c" name="BodyIsHTML" type="Data">
<subsheetid>cc1b4ba2-0510-4980-92bd-d6312072c5bf</subsheetid>
<narrative>
</narrative>
<displayx>-210</displayx>
<displayy>105</displayy>
<displaywidth>90</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
```

```
<datatype>flag</datatype>
<initialvalue>False</initialvalue>
<private/>
<alwaysinit/>
</stage>
<stage stageid="106bc5d9-ced2-4091-a8b6-05f0774acd71" name="HTMLPreferred" type="Data">
<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>
<narrative>
</narrative>
<displayx>-60</displayx>
<displayy>-105</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
<font family="Tahoma" size="10" style="Regular" color="000000"/>
<datatype>flag</datatype>
<initialvalue/>
<private/>
<alwaysinit/>
</stage>
<stage stageid="f085c445-853e-460a-b673-4b230f5cc2bb" name="Content-Type" type="Data">
<subsheetid>67bff276-3cf2-4985-a957-e462b800ac3b</subsheetid>
<narrative>
</narrative>
<displayx>165</displayx>
<displayy>75</displayy>
<displaywidth>60</displaywidth>
<displayheight>30</displayheight>
```

```
<font family="Tahoma" size="10" style="Regular" color="000000"/>

<datatype>text</datatype>

<initialvalue/>

<private/>

<alwaysinit/>

</stage>

</process>

</Root>
```