Table of Contents

1 Components Overview

SendMail - used for sending e-mail messages via SMTP servers. The messages can be either in plain text or HTML format, contain attachments or embedded images for HTML contents. You can use the SendMail for dispatching of messages by mailing lists, specifying multiple recipients.

TextTemplateConverter - the utility which translates some specified %keywords% inside the text into some specified values. It can be used as plug-in for SendMail component to replace some keywords in the email templates (like %recipient_email% or %sender_name%) to their actual values.

---

2 Installation Instructions

to Delphi 5
1. Create "..\Lib\SendMail" directory.
2. Unzip files and copy them to "..\Lib\SendMail".
3. Start Delphi 5 IDE.
4. Open "SendMailD5.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 6
1. Create "..\Lib\SendMail" directory.
2. Unzip files and copy them to "..\Lib\SendMail".
3. Start Delphi 6 IDE.
4. Open "SendMailD6.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 7
1. Create "..\Lib\SendMail" directory.
2. Unzip files and copy them to "..\Lib\SendMail".
3. Start Delphi 7 IDE.
4. Open "SendMailD7.dpk" file.
5. Install package to the components palette ("Install" button).

to C++ Builder 5
1. Create "..\Lib\SendMail" directory.
2. Unzip files and copy them to "..\Lib\SendMail".
3. Start C++ Builder 5 IDE.
4. Open "SendMailCB5.bpk" file.
5. Install package to the components palette ("Install" button).

to C++ Builder 6
1. Create "..\Lib\SendMail" directory.
2. Unzip files and copy them to "..\Lib\SendMail".
3. Start C++ Builder 6 IDE.

4. Open "SendMailCB6.bpk" file.
5. Install package to the components palette ("Install" button).

Source Code

1. Uninstall / delete all previous (trial) instances of SendMail.
2. Create "..\Lib\SendMail" directory.
3. Unzip files from "Sources" directory and copy them to "..\Lib\SendMail".
4. Run Delphi IDE.
5. Select "Component \ Install..." menu item.
6. Press "Add" button and select "SendMail.pas" file.
7. Rebuild library.

Notes for C++ Builder users


When you will build the project with SendMail, you may get following compiler error:
[C++ Error] SHDocVw.hpp(893): E2293) expected, at the following line:

```
/* TWinControl.CreateParented */ inline __fastcall TWebBrowser(HWND ParentWindow)
: Olectrls::TOleControl(ParentWindow) { }
```

Please manually edit the SHDocVw.hpp and change this line to

```
/* TWinControl.CreateParented */ inline __fastcall TWebBrowser(HANDLE
ParentWindow) : Olectrls::TOleControl(ParentWindow) { }
```

Also you can get another linker error:
SendMail.hpp E2209 Unable to open include file 'MSHTML.hpp'.
Please just replace MSHTML.hpp to MSHTML.h. For some reason BCB comes with MSHTML.h file instead of *.hpp.

---

3 Registration Information

SendMail component is SHAREWARE. This means that you can try it out for free, but if you like it and want to use it you have to register it with the author. Before continue read and accept license agreement please.

The only difference between the unregistered and registered versions is that the registered one has not message box with remind to register and works without Delphi (C++ Builder) running. You can also purchase the source code, if you would like to have it, and be able to compile or modify the SendMail on any 32bit version of Delphi or C++ Builder (higher than Delphi 5 or BCB 5).

If you would like to use the SendMail and receive full, unrestricted version, priority support or even source code — you have to purchase proper license.

All prices in US dollars. Registering entitles you to unlimited support via E-Mail, minor version updates indefinitely and major version updates for 6 month from date of purchase. You can use registered components in any number of projects, there is no deployment and royaltee fees.

Registration types:

Full, unrestricted version without source code:

Single user license:

- **https://secure.element5.com/register.html?productid=197300 - $17,95**

Site license:

- https://secure.element5.com/register.html?productid=197301 - $59,95

Full version including 100% Source Code:

Single user license:

- **https://secure.element5.com/register.html?productid=197302 - $27,95**

Site license:

- https://secure.element5.com/register.html?productid=197303 - $89,95

Comments

1. ***Site license*** covers a single organisation in one location (building complex). If you buy a site license, you may use the software in unlimited number of your company's computers withing this area. Site license is very cost-effective if you have many computers (many software developers).

See license agreement for more details.

SendMail (http://www.appcontrols.com)

Documentation created with **Help&Manual**, best authoring tool.

4 License Agreement

Copyright

The SendMail component (software) is Copyright © 2002-2005, by Utilmind Solutions® (Utilmind). All rights reserved.
The authors - Utilmind Solutions® and Aleksey Kuznetsov (founder of Utilmind), exclusively own all copyrights to the Advanced Application Controls (AppControls) and all other products distributed by Utilmind Solutions®.

Liability disclaimer

THIS SOFTWARE IS DISTRIBUTED "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED. YOU USE IT AT YOUR OWN RISK. THE AUTHOR WILL NOT BE LIABLE FOR DATA LOSS, DAMAGES, LOSS OF PROFITS OR ANY OTHER KIND OF LOSS WHILE USING OR MISUSING THIS SOFTWARE.

Restrictions

You may not attempt to reverse compile, modify, translate or disassemble the software in whole or in part. You may not remove or modify any copyright notice or the method by which it may be invoked.

Operating license

Unregistered version

You may distribute the unregistered version of software freely, provided that all files are included and remain unmodified and that no extra files have been added to the package. You may not ask any money for the distribution. You may use the unregistered version of software free of charge for testing purposes, but if you want to use it for other purposes than testing - you have to register it

with the author.

Registered version (single user license)
Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use registered version of the software only by a single person, on a single computer at a time. You may physically transfer the software from one computer to another, provided that the software is used only by a single person, on a single computer at a time. In group projects where multiple persons will use the software, you must purchase an individual license for each member of the group or purchase site license. Use over a "local area network" (within the same locale) is permitted provided that the software is used only by a single person, on a single computer at a time. Use over a "wide area network" (outside the same locale) is strictly prohibited under any and all circumstances.

Registered version (site/team license)
Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your company or your team only in one location (building complex). If you purchase a site license, you may use the program in an unlimited number of your company's computers within this area.

Registered version (Educational site license)
Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your educational organisation (school/college/university etc) in one location (building complex). If you buy a educational site license, you may use the program in an unlimited number of your edicational organisation's computers within this area.

Registered version (World-wide license)
Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your company or your team world-wide. If your company has many branches even with thouthands of computers, world wide license covers them all.

Notes (clarification)
"Single-user license" means "single-developer license". "Site license" means that it can be used by any number of software developers within your company.
You can use purchased components in ANY number of your projects and deploy the "end-user" software to ANY number of your users/customers without any additional royalty fees. However you are not permitted to distribute the component itself (the source code or .dcu files of components).

Back-up and transfer

You may make one copy of the software solely for "back-up" purposes, as prescribed by international copyright laws. You must reproduce and include the copyright notice on the back-up copy.

Terms

This license is effective until terminated. You may terminate it by destroying the program, the documentation and copies thereof. This license will also terminate if you fail to comply with any terms or conditions of this agreement. You agree upon such termination to destroy all copies of the

program and of the documentation, or return them to author.

Other rights and restrictions
All other rights and restrictions not specifically granted in this license are reserved by authors.

---

5 SendMail component

5.1 TSendMail

Overview
The SendMail component used for sending e-mail messages via SMTP servers. The messages can be either in plain text or HTML format, contain attachments or *embedded images* for HTML contents. You can use the SendMail for mass dispatching of messages by mailing lists, specifying multiple recipients.

For more information about SMTP protocol see RFC 821.

How to use?
SMTP server
First you should specify SMTP server which will relay messages to recipients. Set the hostname and port number to SMTPHost and SMTPPort properties. In case if specified SMTP server requires secure authentication — set Authentication.Enabled property to True and provide the Username and Password.

Sender and recipients
The name and email address of sender should be specified in FromName and FromAddress properties (or leave them blank if you sending anonymous message). Recipients should be listed in ToAddr, ToCC or ToBCC properties. (Addresses listed in ToBCC (blind carbon-copy) will not be included to the header of message.)

Message
Set the body of the message to MsgBody and subject of message to MsgSubject property. If you wish to send HTML-formatted message — set MsgContentType to `ctHTML`. If you use foreign language (with non-latin character) — specify content charset in MsgContentCharset property. Also you can specify the priority (level of importance) of message in MsgPriority property.

Attachment
All files which should be attached to message should be listed in Attach property. There are two attachment types — normal and embedded (see AttachType property). Use `ctEmbedded` type if you wish to use some attached pictures in HTML-based message (see also MsgContentType property).

Sending or saving the message
Use Send method to dispatch the message to SMTP server or SaveToFile to save generated message to disk. Use Abort to terminate connection with SMTP server.

Event handlers
Write OnProgress and OnResponse event handlers to display the progress and status of

dispatching. Hook OnHostUnreachable, OnConnLost and OnSMTPError events to be notified errors (or use OnAnyError event to be notified about ANY fatal error). Write OnSuccess event to notify user about successful dispatching of e-mail message.

new! *Email templates*
The SendMail can automatically translate the email templates and replace all the keywords like %RECIPIENT_NAME% or %RECIPIENT_EMAIL% to their actual values. You just need to specify some keywords which are used in your template and values which should replace the keywords in the outgoing message. See TemplateConverter property for more details.

Usage example
In Delphi: http://www.appcontrols.com/demos/tsendmaildemo-delphi.zip
In C++ Builder: http://www.appcontrols.com/demos/tsendmaildemo-bcb.zip
Compiled executable: http://www.appcontrols.com/demos/exe/SendMailDemo.exe

See also
TextTemplateConverter component;
SMTP protocol reference (rfc821); MIME references (rfc2045, rfc2046, rfc2047, rfc2048, rfc2049).

5.2 Properties

5.2.1 AddHeaders

Applies to
SendMail component.

Declaration
```
property AddHeaders: TStringList;
```

Description
The AddHeaders property specifies additional headers of the e-mail message. You can specify ANY optional headers that may be required by mail client to process and display the message.

Specify one header per line. The format should looks like:
FieldName: Value

For example
X-ResentFrom: <email@address.com>
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2600.0000
X-Antivir-Status: Clean
and so forth…

See also *(fields automatically included to the header of message)*
Agent, FromAddress, FromName, FromOrganization, MsgContentCharset, MsgContentType, MsgPriority, MsgSubject, ReplyTo, ToAddr and ToCC properties.

5.2.2 Agent

Applies to
SendMail component.

Declaration
```
property Agent: String;
```

Description
The Agent property is the value that specifies the name and type of the e-mail client. You can

specify ANY value, this is just optional identifier of the program which sends the message.

The value of Agent property automatically includes to the header of message as "X-Mailer" field. Leave the Agent value blank if you don't want to include the program identifier to the message.

See also *(optional headers)*
AddHeaders and MsgPriority properties.

5.2.3 Attach

Applies to
SendMail component.

Declaration
```
property Attach: TStringList;
```

Description
The Attach property specifies the file names which should be included to the e-mail message on dispatching.

The Attach property is the list of strings (TStringList), where every line is the file name including full path to the file.

The files can be attached as "external" or as "internal" (embedded) data. For example, you might want to embed additional pictures to the body of HTML message. See AttachType property for more details.

If attached file could not be found (moved, renamed or deleted before sending), the OnCantAttach event occurs. However, this is not fatal error and SendMail usually continue sending the e-mail, skipping the lost file.

Example
```
procedure TForm1.AttachBtnClick(Sender: TObject);
var
  I: Integer;
begin
  with OpenDialog1 do // TOpenDialog on form
   if Execute and (I <> Files.Count) then
     for I := 0 to Files.Count - 1 do
       SendMail1.Attach.Add(Files[I]);
end;
```

See also
AttachType property;
OnCantAttach event.

5.2.4 AttachType

Applies to
SendMail component.

Declaration
```
type
  TacSMTPAttachType = (atNormal, atEmbedded);

property AttachType: TacSMTPAttachType;
```

Description

The AttachType property controls how the SendMail should envelop the attached files (listed in Attach property).

There are two possible types of attachment:

Value	Meaning
atNormal	normal attachment. User will be able to save attached files to disk or execute attached files directly from message;
atEmbedded	embed attached pictures to the body of message and "hide" them. Use this attachment type only for HTML-based message (when MsgContentType = ctHTML). *Note: pictures will be embedded only if you're showing them in HTML-based message (use <img src="cid:picture.gif"> tags).*

If you would like to show embedded pictures in the HTML message, use the filename of attached message as "content-identifer". To show embedded pictures use following tag:

```
<img src="cid:filename"> or...
<body background="cid:filename">
```

Example:

```
<body background="cid:background.jpg">
  <img src="cid:Banner.gif" width=468 height=60>
</body>
```

(*Note: "background.jpg" and ""Banner.gif" files should be attached to the message).*

See also

Attach and MsgContentType properties.

5.2.5 Authentication

Applies to

SendMail component.

Declaration

```
type
  TSendMailAuthentication = class
  published
    property Enabled: Boolean default False;
    property Username: String;
    property Password: String;
  end;
```

Description

The Authentication structure used to specify whether the SMTP server requires authentication (set Enabled to True, if it requires), and to provide the Username and Password for user authentication.

See also

SMTPHost and SMTPPort properties.

5.2.5.1 Enabled

Applies to

SendMail component as sub-property of Authentication structure.

Declaration

```
property Enabled: Boolean default False;
```

Description

The Enabled property specifies whether the component should send Username and Password for secure authentication with SMTP server.

Set Enabled to True if authentication required, or False otherwise.

See also
Enabled, Username and Password properties.

5.2.5.2 Password

Applies to
SendMail component as sub-property of Authentication structure.

Declaration
```
property Password: String;
```

Description
The Password property specifies the password required for secure authentication with SMTP server.

See also
Enabled, Username and Password properties.

5.2.5.3 Username

Applies to
SendMail component as sub-property of Authentication structure.

Declaration
```
property Username: String;
```

Description
The Username property specifies the username required for secure authentication with SMTP server.

See also
Enabled, Username and Password properties.

5.2.6 Busy

Applies to
SendMail component.

Declaration
```
property Busy: Boolean; // Read-only !!
```

Description
The Busy property determines whether the SendMail component (its thread) is busy on some operations. When Busy property is True, the SendMail currently sending e-mail messages to recipients.

Note

You can NOT send any new e-mail message when the component is busy, and must wait until component done all operations. The Send method will return False when the component is busy.

See also
Send method; OnSuccess and OnAnyError events.

5.2.7 FromAddress

Applies to
SendMail component.

Declaration
```
property FromAddress: String;
```

Description
The FormAddress property specifies the name of sender (i.e: 'john@doe.com'). Leave it blank if you don't want to include the e-mail address to the header of message.

Note
Some SMTP servers don't relay anonymous messages and requires e-mail address of sender to be specified.

See also
FromName, FromOrganization and ReplyTo properties.

5.2.8 FromName

Applies to
SendMail component.

Declaration
```
property FromName: String;
```

Description
The FormName property specifies the name of sender (i.e: 'John Doe'). Leave it blank if you don't want to include the sender's name to the header of e-mail message.

See also
FromAddress and FromOrganization properties.

5.2.9 FromOrganization

Applies to
SendMail component.

Declaration
```
property FromOrganization: String;
```

Description
The FormOrganization property is optional value that specifies the company name of sender. Leave it blank if you don't want to include this value to the header of e-mail message.

If the value of FormOrganization property specified, it automatically includes to the header of message as "Organization" field.

See also
FromAddress and FromName properties.

5.2.10 MsgBody

Applies to
SendMail component.

Declaration

```
property MsgBody: String;
```

Description

The MsgBody property specifies the text of e-mail message.

The text can be either in plain or HTML format (depending on MsgContentType value). To specify the language of message (content encoding) — use MsgContentCharset property.

See also

MsgSubject and TemplateConverter properties.

5.2.11 MsgContentCharset

Applies to

SendMail component.

Declaration

```
property MsgContentCharset: String; // 'iso-8859-1' by default
```

Description

The MsgContentCharset property determines the charaters set used in the text of e-mail message. This parameter always included to the message to allow the e-mail client to determinate which charset should be used to display the message.

Note


You should programmatically handle conversion of text to required code table if the content charset of e-mail is different than charset used on typing a message. Otherwise the text mey appears to be broken. (FYI: 'iso-8859-1' is the standard Western European code table, AFAIK, 'iso-8859-1' is the same as 'us-ascii'.)


The SendMail automatically converts the text from Windows to KOI8 table ('koi8-r' is the standard Russian charset used in e-mail messages), if the When MsgContentCharset = 'koi8-r'. (So Russian programmers may do not worry about this.)

See also

MsgContentType, MsgBody and MsgSubject properties.

5.2.12 MsgContentType

Applies to

SendMail component.

Declaration

```
type
  TacSMTPContentType = (ctPlainText, ctHTML);

property MsgContentType: TacSMTPContentType;
```

Description

The MsgContentType property specifies the type of message content. The MsgContentType allows to email reader to determinate which format should be used to display the message.

Set MsgContentType to `ctPlainText` if you wish the MsgBody contains only plain text, or set it to `ctHTML` to send HTML-formatted message.

See also
MsgContentCharset, MsgBody and MsgSubject properties.

5.2.13 MsgDateTime

Applies to
SendMail component.

Declaration
```
property MsgDateTime: TDateTime;
```

Description
The MsgDateTime property specifies the date and time of outgoing message.

Usually you don't need to specify this property, because if the MsgDateTime is 0, the component will use current date/time on sending the message. Also the component automatically specifies the shift from GMT, using the international settings specifies on sender's machine.

See also
MsgContentCharset, MsgBody and MsgSubject properties.

5.2.14 MsgPriority

Applies to
SendMail component.

Declaration
```
type
  TacSMTPPriority = (mpHighest, mpHigh, mpNormal, mpLow, mpLowest);

property MsgPriority: TacSMTPPriority; // mpNormal by default
```

Description
The MsgPriority property specifies an importance and priority of e-mail message. The priority of message will be shown by e-mail reader program. Also, most SMTP servers relays messages with the higher priorities earlier than messages with low priorities.

The value of MsgPriority property automatically includes to the header of message as "X-Priority" field.

See also *(optional headers)*
AddHeaders and Agent properties.

5.2.15 MsgSubject

Applies to
SendMail component.

Declaration
```
property MsgSubject: String;
```

Description
The MsgSubject property specifies the subject of e-mail message.

See also
MsgBody, MsgContentType and MsgContentCharset properties.

5.2.16 ReplyTo

Applies to
SendMail component.

Declaration
```
property ReplyTo: String;
```

Description
The ReplyTo is the optional property which specifies that you want replies to your outgoing messages sent to a different e-mail address, specified in this property. Specify the ReplyTo address if you wish to get replies to different address than specified in FromAddress property.

Usually when the message could not be delivered to recipient it will be returned to the sender, by address specified in FromAddress property. So you can specify "trashcan" address in FromAddress but your real address in ReplyTo.

If the value of ReplyTo property specified, it automatically includes to the header of message as "ReplyTo" field. Leave the ReplyTo value blank if you don't want to include this field to the header of message.

See also
FromAddress property.

5.2.17 SMTPHost

Applies to
SendMail component.

Declaration
```
property SMTPHost: String;
```

Description
The SMTPHost property specifies the hostname or IP address of the SMTP server which should be used to relay the e-mail messages to recipients.

See also
SMTPPort property.

5.2.18 SMTPPort

Applies to
SendMail component.

Declaration
```
property SMTPPort: Integer; // 25 by default (standard SMTP port)
```

Description
The SMTPPort specifies the port number of SMTP server (specifies the port which listen the server). The standard port number for SMTP protocol is 25.

See also
SMTPHost property.

5.2.19 Suspended

Applies to
SendMail component.

Declaration
```
property Suspended: Boolean;
```

Description
The Suspended property indicates whether a thread (used for downloading) is currenly suspended.

Set Suspended to True to suspend the process temporary; set it False to resume it.

5.2.20 TemplateConverter

Applies to
SendMail component.

Declaration
```
property TemplateConverter: TTextTemplateConverter;
```

Description
The TemplateConverter can be pointed to TextTemplateConverter component, if you are using the template keywords in your email messages, to translate all the %keywords% in message body and headers of outgoing email, to replace them with actual "Values", specified as the parameters of converter component.

For example, if your message contains keywords like %RECIPIENT_NAME% or %RECIPIENT_EMAIL%, the component will automatically replace them to the actual name and email address of recipient, and the message like
```
To: %RECIPIENT_NAME% <%RECIPIENT_EMAIL%>
Dear %RECIPIENT_NAME%,
etc, etc...
```

will be automatically translated to
```
To: Bill Gates <billgates@microsoft.com>
Dear Bill Gates,
etc, etc...
```

See also
TTextTemplateConverter component.

5.2.21 Thread

Applies to
SendMail component.

Declaration
```
property Thread: TCustomThread; // Read-only !!
```

Description
The Thread property is the pointer to the process thread, which used for sending the data to SMTP server, avoiding suspending of main application thread. This is read-only public property.


This property can be used only in when Threaded property is True, since the component does not create any separate thread in non-threaded model. Otherwise this property will return **nil**

(NULL), because the thread not yet created.

See also
Suspended, ThreadPriority and Threaded properties.

5.2.22 Threaded

Applies to
SendMail component.

Declaration
```
property Threaded: Boolean; // True by default
```

Description
The Threaded property used to choose between two working models of acSendMail component.

This property specifies whether the message should be prepared and sent via Internet to SMTP server in *current, caller's* thread, or should create *separate* thread which will process the message delivery. Set Threaded to True to create separate thread, or to False if the part of code which calls the Send method are already in separate, non-interface thread.

Basically there is two possible working models:

1. **Threaded**: when Threaded is True, the component creates new separate thread to produce and send email message to specified SMTP host. In this case the SendMail does not locks the application interface when it connects to the SMTP server and sends the message via Internet, and only produce events synchronized with main application thread (interface).

   ⚠ However, in this model the component can send ONLY ONE message simultaneously, since it can create only one thread to send message. If you will try to send additional message when the component is busy, next email will not be sent, and Send method will just return False. You should implement some message queue to operate with SendMail in this threading model and don't call Send method while previous message not yet sent. Otherwise — set Threaded to False and use non-threaded model to send any number of email messages simultaneously from any number of different threads.

2. **Non-threaded**: when Threaded is False, the component does not creates any separate thread to produce and send email message. This model should be used only if your application uses many threads which could send email message using this component. This model should NOT be used in single-threaded application, since when you call the Send method, all your application interface will be locked while the component producing and sending the email message.

See also
Thread property;
Send method.

5.2.23 ThreadPriority

Applies to
SendMail component.

Declaration
```
property ThreadPriority: TThreadPriority;
```

Description
ThreadPriority indicates the priority used when scheduling the thread. Adjust the priority higher or

lower as needed.

TThreadPriority type defines the possible values for the priority of SendMail thread, as defined in the following table. The system schedules CPU cycles to each thread based on a priority scale; the Priority property adjusts a thread's priority higher or lower on the scale.

Values	Meaning
tpIdle	The thread executes only when the system is idle. The system will not interrupt other threads to execute a thread with tpIdle priority.
tpLowest	The thread's priority is two points below normal.
tpLower	The thread's priority is one point below normal.
tpNormal	The thread has normal priority.
tpHigher	The thread's priority is one point above normal.
tpHighest	The thread's priority is two points above normal.
tpTimeCritical	The thread gets highest priority.

Warning

Boosting the thread priority of a CPU intensive operation may "starve" the other threads in the application. Only apply priority boosts to threads that spend most of their time waiting for external events.

See also

Thread and WaitThread properties.

5.2.24 ToAddr

Applies to

SendMail component.

Declaration

```
property ToAddr: String;
```

Description

The ToAddr property specifies the recipients of the message. You can specify any number of e-mail addresses and recipient names.

The value of ToAddr property must have following format:
"Name1" <address1>; "Name2" <address2> , "NameN" <addressN>

The name of the recipient marked blue (must be between quotes ("), i.e: "John Smith"). You can leave the name blank and use just email address;
E-mail address of recipient marked green (must be between brackets "<>", i.e: <john@smith.com>).
Multiple recipients should be separated by semicolon (;) or comma (,).

Examples

<john@smith.com>
"John Smith" <john@smith.com>, <info@appcontrols.com>, <webmaster@utilmind.com>
"Aleksey Kuznetsov" <aleksey@utilmind.com>; "Webmaster" <webmaster@utilmind.com>

Alternatively, if you wish to specify multiple recipients, you can use ToCC property (carbon-copy). If you wish to hide some email addresses from message header — use ToBCC property (blind carbon-copy).

Warning

Please don't use brackets (< and >) in the "Name" fields (between quotes). Otherwise the address will be parsed incorrectly. For example, following address will be parsed incorrectly!!:
"John<g> Smith" <john@smith.com>

However, the parser is smart enough to parse definitions like these:
"John Smith, Inc;" <john@smith.com>
"Smith, "A" John" <john@smith.com>

See also
ToCC and ToBCC properties.

5.2.25 ToBCC

Applies to
SendMail component.

Declaration
```
property ToBCC: String;
```

Description
The ToBCC property used to specify recipients of the message which addresses should NOT be visible in the message header. This is field known as blind carbon-copy and only the message sender knows to whom the message was sent. All e-mails used in ToBCC will be hidden and recipients will NOT see these e-mail addresses.

The value of ToBCC property must have following format:
"Name1" <address1>; "Name2" <address2> , "NameN" <addressN>

The name of the recipient marked blue (must be between quotes ("), i.e: "John Smith"). You can leave the name blank and use just email address;
E-mail address of recipient marked green (must be between brackets "<>", i.e: <john@smith.com>).
Multiple recipients should be separated by semicolon (;) or comma (,).

Examples
<john@smith.com>
"John Smith" <john@smith.com>, <info@appcontrols.com>, <webmaster@utilmind.com>
"Aleksey Kuznetsov" <aleksey@utilmind.com>; "Webmaster" <webmaster@utilmind.com>

Alternatively, if you wish to specify multiple recipients, you can use ToCC property (visible carbon-copy).

Warning
Please don't use brackets (< and >) in the "Name" fields (between quotes). Otherwise the address will be parsed incorrectly. For example, following address will be parsed incorrectly!!:
"John<g> Smith" <john@smith.com>

However, the parser is smart enough to parse definitions like these:
"John Smith, Inc;" <john@smith.com>
"Smith, "A" John" <john@smith.com>

See also
ToAddr and ToCC properties.

5.2.26 ToCC

Applies to
SendMail component.

Declaration
```
property ToCC: String;
```

Description
The ToCC property used to specify multiple recipients of the message. This is standard field known as carbon-copy and recipient will be able to reply to all others recipients.

The value of ToCC property must have following format:
"Name1" <address1>; "Name2" <address2> , "NameN" <addressN>

The name of the recipient marked blue (must be between quotes ("), i.e: "John Smith"). You can leave the name blank and use just email address;
E-mail address of recipient marked green (must be between brackets "<>", i.e: <john@smith.com>).
Multiple recipients should be separated by semicolon (;) or comma (,).

Examples
<john@smith.com>
"John Smith" <john@smith.com>, <info@appcontrols.com>, <webmaster@utilmind.com>
"Aleksey Kuznetsov" <aleksey@utilmind.com>; "Webmaster" <webmaster@utilmind.com>

If you wish to hide recipients from the message header — use ToBCC property (blind carbon-copy).

Warning
Please don't use brackets (< and >) in the "Name" fields (between quotes). Otherwise the address will be parsed incorrectly. For example, following address will be parsed incorrectly!!:
"John<g> Smith" <john@smith.com>

However, the parser is smart enough to parse definitions like these:
"John Smith, Inc;" <john@smith.com>
"Smith, "A" John" <john@smith.com>

See also
ToAddr and ToBCC properties.

5.2.27 WaitThread

Applies to
SendMail component.

Declaration
```
property WaitThread: Boolean;
```

Description
The WaitThread property controls whether the procedure that calls the Send method (which downloads the data from the Web) should be suspended and wait until the scanning process will be done.

Set the WaitThread to True, if you would like to read the data from the Web so that the application does not continue with next lines of code after calling the Send method. Your application will done download (or inform about error) before continuing to next step.

If your application can wait, until the the email will be sent, only for some limited time interval — specify WaitTimeout property.

See also
WaitTimeout, Thread, Threaded and Suspended properties;
Send method.

5.2.28 WaitTimeout

Applies to
SendMail component.

Declaration
```
property WaitTimeout: Integer;
```

Description
The WaitTimeout property specifies the time interval (limit), in milliseconds unit, which application able to wait while the mail will be sent.

For example, if the maximum time which you can allow to send an email is 5 seconds, set this value to 5000 (milliseconds). If application can wait infinitely, set WaitTimeout to 0.

When the timeout is expired, the component automatically terminates the process of sending an email. To be notified when the WaitTimeout is expired — write OnWaitTimeoutExpired event handler.

Notes
The WaitTimeout only works together with WaitThread property, only when it set to True.

See also
WaitThread, ThreadPriority and Suspended properties;
Send and Abort methods;
OnWaitTimeoutExpired event.

5.3 Methods

5.3.1 SaveToFile

Applies to
SendMail component.

Declaration
```
function SaveToFile(const FileName: String): Boolean;
```

Description
The SaveToFile property generates the e-mail message (including encoded attachment) and stores it to the file (specified in *FileName* parameter). Set *Unsent* parameter to True to add "X-Unsent: 1" field (which means that message not yet sent) to the stored message.

Function returns True if message has successfully saved, or False if it could not be stored by specified *FileName*.

The created message has standard format (.eml) which can be easily viewed and sent by MS-Outlook e-mail clients.

See also
Send property.

5.3.2 Send

Applies to
SendMail component.

Declaration
```
function Send: Boolean; // returns False if busy or WaitTimeout is expired
```

Description
The Send method generates and sends the e-mail message to all recipients specified in ToAddr, ToCC and ToBCC properties.

If you are using threaded model (Threaded property is True), this method may returns False if component currently busy (already sending the message), OR WaitTimeout is expired (if you waiting for completion of request in the function that calls this medod, using WaitThread property).

See also
Abort method;
Threaded, ToAddr, ToCC and ToBCC properties;
SendMail function.

5.3.3 Abort

Applies to
SendMail component.

Declaration
```
procedure Abort(HardTerminate: Boolean = True);
```

Description
The Abort method imediately terminates the thread which sends the data via SMTP protocol. After calling the Abort method, the OnAborted event occurs.

The Abort method contains 1 optional parameter, *HardTerminate* (True by default), which controls whether the application should imediately terminate thread which sends data to SMTP server, or wait when server terminate connection itself.

See also
Send method; OnAborted event.

5.4 Events

5.4.1 OnAborted

Applies to
SendMail component.

Declaration
```
property OnAborted: TNotifyEvent;
```

Description
The OnAborted event occurs when user interrupts the process of sending the e-mail to SMTP

server, after calling the Abort method.

See also
Abort method;
OnWaitTimeoutExpired event.

5.4.2 OnAnyError

Applies to
SendMail component.

Declaration
```
property OnAnyError: TNotifyEvent;
```

Description
The OnAnyError event occurs when ANY fatal error has occured on delivering the message to server *(except OnCantAttach)*.

The fatal error means that message could not be delivered. Such errors can be also handled by following events: OnConnLost, OnHostUnreachable and OnSMTPError.

See also
OnConnLost, OnHostUnreachable and OnSMTPError events.

5.4.3 OnCantAttach

Applies to
SendMail component.

Declaration
```
type
  TSMTPCantAttachEvent = procedure(Sender: TObject; FileName: String) of
object;

property OnCantAttach: TSMTPCantAttachEvent;
```

Description
The OnCantAttach event occurs if the component could not read the file which should be attached (listed in Attach property).

This is not fatal error, so SendMail will try to continue sending the message without missed file. If you don't want to continue process — just call Abort method in the event handler.

See also
Attach property; Abort method.

5.4.4 OnConnLost

Applies to
SendMail component.

Declaration
```
property OnConnLost: TNotifyEvent;
```

Description
The OnConnLost event occurs when the connection with remote server lost for some reason, at the

moment of sending or receiving the data by SMTP protocol.

This is fatal error and OnAnyError event will occured after this event.

See also
OnAnyError event.

5.4.5 OnHostUnreachable

Applies to
SendMail component.

Declaration
```
property OnHostUnrachable: TNotifyEvent;
```

Description
The OnHostUnreachable event occurs if the SendMail can not establish connection to SMTP server, specified in SMTPHost property. Possible reasons of this problem is:

1. User currently not connected to the Internet;
2. Hostname is unknown (check spelling of domain name in SMTPHost property);
3. Invalid SMTP port specified (check the number of SMTP port in SMTPPort property);
4. Remote server is down (disconnected from Internet).

Example
Delphi:
```
procedure TForm1.SendMail1HostUnreachable(Sender: TObject);
begin
  Application.MessageBox(PChar('Specified SMTP server is
unreachable.'#13#10#10'Please check your Internet connection'#13#10'or
specify another SMTP host or/and port.'),
                         PChar(Application.Title),
                         MB_OK or MB_ICONSTOP);
end;
```

C++ Builder:
```
void __fastcall TForm1::SendMail1HostUnreachable(TObject *Sender)
{
  AnsiString Msg =
      "Specified SMTP server is unreachable.\n\n"
      "Please check your Internet connection\n"
      "or specify another SMTP host or/and port.";
  Application->MessageBox(Msg.c_str(),
                          Application->Title.c_str(),
                          MB_OK | MB_ICONSTOP);
}
```

See also
SMTPHost and SMTPPort properties;
OnAnyError event.

5.4.6 OnProgress

Applies to
SendMail component.

Declaration

```
type
  TSMTPProgressEvent = procedure(Sender: TObject; MessageSize,
BytesSent: Integer; PercentsDone: Byte) of object;

property OnProgress: TSMTPProgressEvent;
```

Description

The OnProgress event occurs every time when the component has successfully sent the part of e-mail message to SMTP server.

Write OnProgress event handler to show the download progress. The SendMail passes to the event handler following parameters:

Parameter	Meaning
MessageSize	total size of e-mail message in bytes;
BytesSent	number of bytes that already sent to SMTP server;
PercentsDone	progress in percents (0%..100%).

Example

```
procedure TForm1.SendMail1Progress(Sender: TObject; MessageSize,
  BytesSent: Integer; PercentsDone: Byte);
begin
  ProgressBar1.Position := PercentsDone;
end;
```

See also

OnSuccess event.

5.4.7 OnResponse

Applies to

SendMail component.

Declaration

```
type
  TSMTPResponseEvent = procedure(Sender: TObject; Code: Integer; const
Response: String) of object;

property OnResponse: TSMTPResponseEvent;
```

Description

The OnResponse event occurs every time when component receives the response for SMTP request from server. The response contains the status information (code and optional text) of request.

For more information about status codes of SMTP protocol see RFC 821.

Example

```
procedure TForm1.SendMail1Response(Sender: TObject; Code: Integer;
  const Response: string);
begin
  with ListView1 do // status view
   begin
    Items.Add(Response);
    ItemIndex := Items.Count - 1;
   end;
end;
```

See also
OnSMTPError, OnSuccess and OnProgress events.

5.4.8 OnSMTPError

Applies to
SendMail component.

Declaration
```
type
  TSMTPResponseEvent = procedure(Sender: TObject; Code: Integer; const
Response: String) of object;

property OnSMTPError: TSMTPResponseEvent;
```

Description
The OnSMTPError event occurs when the SMTP server returns an error code with response (see OnResponse event). After receiving error code, the SendMail component automatically terminates connection with SMTP server, then triggers the OnSMTPError event.

For more information about status codes of SMTP protocol see RFC 821.

Example
```
procedure TForm1.SendMail1SMTPError(Sender: TObject;
  Code: Integer; const Response: string);
begin
  with StatusList do
   begin
    Items.Add('ERROR: ' + Response);
    ItemIndex := Items.Count - 1;
   end;
end;
```

See also
OnResponse and OnAnyError events.

5.4.9 OnSuccess

Applies to
SendMail component.

Declaration
```
property OnSuccess: TNotifyEvent;
```

Description
The OnSuccess event occurs when the e-mail message has been successfully sent to SMTP server. Use OnSuccess event to be notified when the SendMail successfuly dispatched the email message to SMTP server (message will be relayed to recipients shortly by server).

Note

Successful dispatching of message to SMTP server not gives guarantee of successful delivery to recipients. SMTP server may return the message if it could not be relayed to the mailbox.

See also
OnResponse and OnSMTPError events.

5.4.10 OnWaitTimeoutExpired

Applies to
SendMail component.

Declaration
```
property OnWaitTimeoutExpired: TNotifyEvent;
```

Description
The OnWaitTimeoutExpired event occurs when the component could not send email for the time interval specified in WaitTimeout property (when WaitTimeout is expired).

Notes
The WaitTimeout only works together with WaitThread property, only when it set to True.

See also
WaitThread and WaitTimeout properties;
Abort method.

5.5 QuickSendMail function

Unit
SendMail

Declaration
```
function SendMail(const FromAddr, FromName, ToAddr, ToName, Subject,
Body: String;
  const SMTPHost: String = 'localhost'; SMTPPort: Word = 25;
  ContentType: TSMTPContentType = ctHTML; Priority: TSMTPPriority =
mpNormal;
  const ContentCharset: String = 'iso-8859-1'; RunInSeparateThread:
Boolean = True): Boolean;
```

Description
The QuickSendMail function is alternative way to send email messages without having the SendMail component on the form, and without dynamic creation of SendMail component instances. When you use this function — everything created/released internally.

The QuickSendMail have following set of parameters (some of parameters are optional and may be not specified):

Parameter	Meaning
FromAddr	specifies the email address of sender;
FromName	specifies the name of sender (the name which displayed in email message, can be empty string);
ToAddr	email address of recipient;
ToName	name of recipient (can be empty string);
Subject	the subject of email message;
Body	the text of email message (can contain HTML tags if *ContentType* is ctHTML, or just plain text if *ContentType* is ctPlainText);
SMTPHost	the hostname or IP address of SMTP server;
SMTPPort	the port number of SMTP server (standard port for SMTP

	servers is 25);
ContentType	can be ctPlainText if your email message contains just plain text, or ctHTML if it should be HTML-formatted;
Priority	priority of email message (npNormal by default);
ContentCharset	the set of characters of text in email message. Use appropriate definition of charset for language used in email message;
RunInSeparateThread	specifies whether the message should be prepared and sent via Internet to SMTP server in *current, caller's* thread, or should create *separate* thread which will process the message delivery. Set this parameter to True to create separate thread, or to False if the part of code which calls the QuickSendMail function are already in separate, non-interface thread.

See also

SendMail component.

6 TextTemplateConverter compont

6.1 TTextTemplateConverter

Overview

The TextTemplateConverter is the utility which translates some specified %keywords% inside the text into some specified values. It can be used as plug-in for SendMail component to replace some keywords in email templates (like %recipient_email% or %sender_name%) to their actual values.

How to use?

Drop component on your form and specify some keywords which you are using in your text templates to the **Name** properties of Params collection. Also specify the text strings, which should be replace that keywords, to the **Value** propertes of Params. You can modify the collection of Names/Values both at design and run-time.

When your program calls the **Convert** method, it will find all the keywords (specified in *Name* properties) within the text template and replace them to their actual values (specified in *Value* properties). For example, our Params property contains 2 following items:

```
Item1:
  Name  = %RECIPIENT_NAME%
  Value = Bill Gates
Item2:
  Name  = %RECIPIENT_EMAIL%
  Value = billgates@microsoft.com
```

Plus we have the text template, which contains %RECIPIENT_NAME% and %RECIPIENT_EMAIL% keywords instead of real name and email address. For example, the text like:

```
To: %RECIPIENT_NAME% <%RECIPIENT_EMAIL%>
Dear %RECIPIENT_NAME%,
etc, etc...
```

After calling the Convert method, all the keywords specified in Name property of each item will be replaced by the words specified in Value property. For example, after convertion, the text displayed above will looks like follows:

```
  To: Bill Gates <billgates@microsoft.com>
  Dear Bill Gates,
etc, etc...
```

The keywords can looks like **%KEYNAME%**, or **<KEYNAME>**, or just **KEYNAME**, whatever you like. In most cases you should specify the keywords which are used in your text templates just once, and you can do it at design-time and don't modify them at run-time at all.

However, your program should specify required *Values* every time before processing of the template (before the program calls Convert method to translate the keywords into the values). To specify the *Values* for each required keyword (*Names*) you can use ParamByName method, or just enumerate each item of Params collection. Here is some examples:

```
  // how to add the the template parameters at run-time
  var
    Param: TacTemplateParameter;
  begin
    Param := TextTemplateConverter1.Params.Add;
    Param.Name := '%keyword%';
    Param.Value := 'Value';

  // how to modify the template values at run-time
  TextTemplateConverter1.ParamByName('%keyword%').Value := 'value';
  TextTemplateConverter1.ParamByValue('value').Name :=
  '%another_keyword%';

  // also you can enumerate all parameters
  var
    I: Integer;
  begin
    I := TextTemplateConverter1.Count;
    if I <> 0 then
     for I := 0 to I - 1 do
      with TextTemplateConverter1 do
       Caption := Params[I].Name + ' = ' + Params[I].Value;
  end;
```

If you are using this component as plug-in for SendMail component, to replace the keywords in email templates, you don't need to call Convert method. All that you need is just point the TextTemplateConverter to the TemplateConverter property of SendMail component and all the text of outgoing email, the body of message and the message headers will be translated automatically.

See also

SendMail component.

6.2 Properties

6.2.1 IgnoreCaseOfParams

Applies to

TextTemplateConverter component.

Declaration

```
property IgnoreCaseOfParams: Boolean; // True by default
```

Description

The IgnoreCaseOfParams property controls whether the component should ignore the case of

character on convertion, when it checks the text for the keywords described in **Name** property of Params collection.

Set IgnoreCaseOfParams to True, and Convert method will replace the keywords with their values independently of case of characters, so the keyword like **%kEyWoRd%** is the same as **%Keyword%**. Otherwise, set it to False and the Convert method will find only keywords with the same charcase as specified in the Name properties of Params items.

See also
Params property;
Convert method.

6.2.2 Params

Applies to
TextTemplateConverter component.

Declaration

```
type
  TacTemplateParameter = class(TCollectionItem)
  published
    property Name: String;
    property Value: String;
  end;

  TacTemplateParameters = class(TCollection)
  public
    function Add: TacTemplateParameter;
    function Convert(const Value: String): String;

    property Owner: TComponent read FOwner;
    property IgnoreCase: Boolean;
    property Items[Index: Integer]: TacTemplateParameter; default;
  end;

property Params: TTextTemplateParameters; // True by default
```

Description
The Params is the collection of keywords (and their values) which should be converted in the template.

Each item of Params collection is the object which contains 2 properties: **Name** and **Value**. The Name property specifies the template keyword which can be used in the text templates, and Value property is the actual string which sould replace the template keyword.

For example, our Params property contains 2 following items:

```
Item1:
  Name  = %RECIPIENT_NAME%
  Value = Bill Gates
Item2:
  Name  = %RECIPIENT_EMAIL%
  Value = billgates@microsoft.com
```

Plus we have the text template, which contains %RECIPIENT_NAME% and %RECIPIENT_EMAIL% keywords instead of real name and email address. For example, the text like:

```
To: %RECIPIENT_NAME% <%RECIPIENT_EMAIL%>
```

```
Dear %RECIPIENT_NAME%,
etc, etc...
```

After calling the Convert method, all the keywords specified in Name property of each item will be replaced by the words specified in Value property. For example, after convertion, the text displayed above will looks like follows:

```
To: Bill Gates <billgates@microsoft.com>
Dear Bill Gates,
etc, etc...
```

Examples

```
// how to add the the template parameters at run-time
var
  Param: TacTemplateParameter;
begin
  Param := TextTemplateConverter1.Params.Add;
  Param.Name := '%keyword%';
  Param.Value := 'Value';

// how to modify the template values at run-time
TextTemplateConverter1.ParamByName('%keyword%').Value := 'value';
TextTemplateConverter1.ParamByValue('value').Name :=
'%another_keyword%';

// also you can enumerate all parameters
var
  I: Integer;
begin
  I := TextTemplateConverter1.Count;
  if I <> 0 then
   for I := 0 to I - 1 do
    with TextTemplateConverter1 do
     Caption := Params[I].Name + ' = ' + Params[I].Value;
end;
```

See also

IgnoreCaseOfParams property;
ParamByName, ParamByValue and Convert methods.

6.3 Methods

6.3.1 Convert

Applies to

TextTemplateConverter component.

Declaration

```
function Convert(const Value: String): String;
```

Description

The Convert method checks the template string specified in Value parameter, found in that string all the **%keywords%** specified in **Name** property of Params collection, replaces the **%keywords%** with the text specified in **Value** property of Params.

Function returns the translated string, where all template keywords are replaced with their values.

See also
Params property;
ConvertStrings and ConvertStream methods.

6.3.2 ConvertStrings

Applies to
TextTemplateConverter component.

Declaration
```
procedure ConvertStrings(Strings: TStrings);
```

Description
The ConvertStrings method is the same as Convert, but translates the string lists.

It checks the template string specified in Value parameter, found in that string all the **%keywords%** specified in **Name** property of Params collection, replaces the **%keywords%** with the text specified in **Value** property of Params.

After calling the ConvertStrings method, the Strings parameter will contain translated text, where all keywords replaced with their actual values.

Example
```
ConvertStrings(Memo1.Lines);
```

See also
Params property;
Convert and ConvertStream methods.

6.3.3 ConvertStream

Applies to
TextTemplateConverter component.

Declaration
```
procedure ConvertStream(Stream: TStream);
```

Description
The ConvertStream method is the same as Convert, but translates the text in the streams or their descendants (MemoryStream's, FileStream's etc).

It checks the template string specified in Value parameter, found in that string all the **%keywords%** specified in **Name** property of Params collection, replaces the **%keywords%** with the text specified in **Value** property of Params.

After calling the ConvertStream method, the Stream will contain translated text, where all keywords replaced with their actual values.

Example
```
procedure TForm1.Button1Click(Sender: TObject);
var
  fs: TFileStream;
begin
  fs := TFileStream.Create('c:\file.txt', fmOpenReadWrite);
  try
    TextTemplateConverter1.ConvertStream(fs);
```

```
    finally
      fs.Free;
    end;
  end;
```

See also
Params property;
Convert and ConvertStrings methods.

6.3.4 ParamByName

Applies to
TextTemplateConverter component.

Declaration

```
type
  TacTemplateParameter = class(TCollectionItem)
  published
    property Name: String;
    property Value: String;
  end;

function ParamByName(const ParamName: String): TacTemplateParameter;
```

Description
The ParamByName method returns the pointer to TacTemplateParameter object where the *Name* property is equal to the *ParamName* parameter.

You can use this method to quickly find the keyword and modify its *Value*.

⚠ Function returns NIL if the specified *ParamName* was not found in Params collection.

Example

```
acTemplateConverter1.ParamByName('%keyword%').Value := 'New Value for
Keyword';
```

See also
Params property;
ParamByValue and Convert methods.

6.3.5 ParamByValue

Applies to
TextTemplateConverter component.

Declaration

```
type
  TacTemplateParameter = class(TCollectionItem)
  published
    property Name: String;
    property Value: String;
  end;

function ParamByValue(const ParamValue: String): TacTemplateParameter;
```

Description

The ParamByValue method returns the pointer to TacTemplateParameter object where the *Value* property is equal to the *ParamValue* parameter.

You can use this method to quickly find some value in the Params collection and modify its *Name*.

 Function returns NIL if the specified *ParamValue* was not found in Params collection.

Example

```
acTemplateConverter1.ParamByValue('The Value').Name :=
'%another_keyword%';
```

See also

Params property;
ParamByName and Convert methods.

Index

- I -

- L -

- Q -

- R -

- S -

- T -