
E-mail Pre-processor Developer's Guide



S P E E C H I F Y 3 . 0

Document History

Date	Release Name
August 2003	Fourth Edition, Speechify 3.0
May 2002	Third Edition, Speechify 2.1
December 2001	Second Edition, Speechify 2.0
March 2001	First Edition, Speechify 1.1

Notice

Copyright © 2001–2003 by SpeechWorks International, Inc. All rights reserved.

The information in this document is subject to change without notice.

Use of this software is subject to certain restrictions and limitations set forth in a license agreement entered into between SpeechWorks International, Inc. and purchaser of this software. Please refer to the SpeechWorks license agreement for license use rights and restrictions.

SpeechWorks, DialogModules, SADL, SMARTRecognizer, SpeechCare, SpeechCookie, Speechify, Speechify Solo, SpeechSecure, SpeechSite, SpeechSpot, SpeechWorks Here, the SpeechWorks logo and the SpeechWorks Here logo are trademarks or registered trademarks of SpeechWorks International, Inc. in the United States and other countries. All other trademarks are property of their respective owners. Windows NT is a registered trademark of Microsoft Corporation.

Published by:

SpeechWorks International, Inc.
695 Atlantic Avenue
Boston, MA 02111 USA

Table of Contents

Preface	V
 1. Overview of Speechify's E-mail Pre-processor	
Features	1-2
Order of API calls	1-3
 2. Functionality of the E-mail Pre-processor	
Supported message formats.....	2-6
Default behavior	2-8
Header processing.....	2-8
Body processing.....	2-9
Signature processing.....	2-11
MIME format.....	2-12
Modes	2-12
 3. Using the E-mail Substitution Dictionary	
File format.....	3-16
Dictionary entries.....	3-17
Comments and escapes	3-19
Notifications	3-19
 4. API Reference	
Calling convention	4-24
Result codes	4-25
SWIemailInit()	4-26
SWIemailProcess()	4-27
SWIemailTerm()	4-28



Preface

Welcome to SpeechWorks

Speechify™ is SpeechWorks International, Inc.'s state of the art text-to-speech (TTS) system. This guide is written for application developers who want to process e-mail messages before reading the e-mail messages via Speechify. This pre-processing allows you to handle e-mail headers, attachments, and other typical e-mail elements, so that the TTS process can read the e-mail messages more cleanly.

Roadmap

Recommended reading (available documentation)

- ❑ The *Speechify User's Guide* provides installation, programming, and reference information about the Speechify product.
- ❑ The *SpeechWorks Licensing Handbook* describes the process for getting licenses to run the Speechify software, details about configuring your license server, and related topics.
- ❑ The *E-mail Pre-processor Developer's Guide* covers the types of input that the SpeechWorks E-mail Pre-processor handles, how it processes the messages, the modes that the application can take advantage of at run time, the layout and use of the E-mail substitution dictionary, and the API functions.

- ❑ The *Speechify Migration Guide* gives instructions for Speechify 2.x users to migrate their systems and applications to Speechify 3.0.
- ❑ There is a *Speechify Language Supplement* for each supported language. These supplements contain language-specific reference information for application developers.
- ❑ Review the release notes distributed with this product for the latest information, restrictions, and known problems.

SpeechWorks Institute

See the SpeechWorks Institute page at <http://www.speechworks.com> or send e-mail to training@speechworks.com for details about available training courses.

Support services

To receive technical support from SpeechWorks International, Inc.:

- ❑ Visit the Knowledge Base or ask a question at:

http://www.speechworks.com/training/tech_support.cfm

- ❑ Ask for “technical support” at +1 617 428-4444

How this guide is organized

The chapters of this document cover these topics:

Chapter 1 “Overview of Speechify’s E-mail Pre-processor” describes the features, how to use the e-mail pre-processor API, and information about the e-mail substitution dictionary.

Chapter 2 “Functionality of the E-mail Pre-processor” describes the types of input that the E-mail Pre-processor handles, how it processes the messages, and the modes that the application can take advantage of at run time.

[Chapter 3 “Using the E-mail Substitution Dictionary”](#) explains the layout and use of the e-mail substitution dictionary.

[Chapter 4 “API Reference”](#) describes the API function prototypes, types, error codes, and constants.



Overview of Speechify's E-mail Pre-processor

This chapter contains an overview of features, how to use the e-mail pre-processor API, and information about the e-mail substitution dictionary, used to specify and customize text substitutions. For a detailed explanation of the API functions, see “API Reference” on [page 4-23](#).

In This Chapter

- Features on [page 1-2](#)
- Order of API calls on [page 1-3](#)

Features

The SpeechWorks E-mail Pre-processor is provided as a dynamic linked library (.dll) on NT and a static library (.a) on UNIX. It has the following features:

- ❑ Support for standard format e-mail.
- ❑ A powerful e-mail substitution dictionary.
- ❑ Thread-safe API functions.

The SpeechWorks E-mail Pre-processor only processes plain text e-mail messages. It does not process e-mail messages that are in HTML format or are encoded (e.g., base64). The pre-processor does provide the facility to support partially processed e-mail messages, as well as MIME format. This allows the the pre-processor to process e-mail messages that have been pre-filtered for HTML tags (or any other text format that the application chooses to handle). A full description of this is provided in “Functionality of the E-mail Pre-processor” on [page 2-5](#).

The pre-processor has a substitution dictionary that allows the user to specify the way strings are spoken by Speechify. Entries in the dictionary tell the pre-processor to substitute certain pieces of text with other text. For example, you may want to replace “BTW” with “by the way”. Dictionary entries may apply to the whole message or to specific sections of the message, depending on the scope that the user has specified. A full description of the dictionary is provided in “Using the E-mail Substitution Dictionary” on [page 3-15](#).



NOTE

Providing the pre-processor as an independent library rather than embedding the function calls into the full Speechify API gives the application developer the option of processing the e-mail text on the fly (prior to a call to `SWIttsSpeak()` in the Speechify API), or processing e-mail messages off-line, and saving CPU usage at run-time. Very large e-mail messages may take some time to process- long enough to impact time-to-first audio if your TTS application processes them on the fly. For this reason, you may prefer to process e-mail messages as they arrive at the e-mail server, and store the output.

Order of API calls

There are just three API functions in the e-mail pre-processor. You must call them in this order.

1. `SWIemailInit()` initializes the e-mail preprocessor library which includes loading the substitution dictionary and loading it into memory
2. `SWIemailProcess()` processes a single e-mail message in the form of a null-terminated text string.
3. `SWIemailTerm()` frees resources needed by the library.

Pseudo-code for an offline e-mail pre-processor may look like this:

```
SWIemailInit()  
while (more messages to process)  
    SWIemailProcess(message n)  
    Save(message n)  
SWIemailTerm()
```

Pseudo-code for an application that processes and speaks e-mail at runtime may look like this:

```
SWIemailInit()  
SWIttsInit()  
SWIttsOpenPort()  
while (more TTS requests to make)  
    SWIemailProcess()  
    SWIttsSpeak()  
SWIttsClosePort()  
SWIttsTerm()  
SWIemailTerm()
```



NOTE

You only need to call `SWIemailInit()` and `SWIemailTerm()` once per process, and you need to call `SWIemailProcess()` once per message.



Functionality of the E-mail Pre-processor

This chapter explains the types of input that the SpeechWorks E-mail Pre-processor handles, how it processes the messages, and the modes that the application can take advantage of at run time. There is mention of MIME (Multipurpose Internet Mail Extensions) format messages below. A MIME format message is one that conforms to the Internet standards defined in RFCs 822, 2045 and 2046. (<http://www.ietf.org/rfc.html>)

In This Chapter

- Supported message formats on [page 2-6](#)
- Default behavior on [page 2-8](#)
- Modes on [page 2-12](#)

Supported message formats

The SpeechWorks E-mail Pre-processor deals with messages in full MIME format, or partially processed messages. It only processes plain text e-mail messages. It does not process e-mail messages that are in HTML format or that are encoded (e.g., base64), therefore HTML/XML tags or encoded text should be filtered out of the message or decoded before a call to `SWIemailProcess()`. If not, the tags or encoded text are read by Speechify. If you preprocess a message with another application (e.g., to parse or filter HTML tags), and the output is not in MIME format, then the e-mail preprocessor can still deal with the message. The only assumption is that there is an empty line separating the header from the body of the message. Below is an example of a partially processed e-mail.

From: "Dave Burns" <david.burns@speechworks.com>
To: "Daniel Faulkner" <daniel.faulkner@speechworks.com>
Subject: RE: No con call
Date: Mon, 27 Nov 2000 15:44:58 -0500

This is a partially processed e-mail.

If a message is passed into the preprocessor in MIME format, then each MIME boundary is located, and its content type is identified. If there are attachments to the message, the listener is notified what the media type and file name are (e.g., "There is an audio file called hello.wav attached to the message"). File attachments can only be identified if the message is passed in MIME format. Below is an example of a MIME format message with a text file attachment.

From andrew.lowry@speechworks.com Fri Nov 24 04:04:16 2000
Received: from [63.113.17.11] by speechworks.com (3.2) with ESMTP id
MBBE7A428003A4004315F3F71110BB50E0; Fri Nov 24 04:03:52 2000
Received: from vishnu.speechworks.com (mailhost.speechworks.com [206.234.64.17])
by speechworks.com (8.9.3+Sun/8.9.3) with ESMTP id HAA21041
for <dan.faulkner@speechworks.com>; Fri, 24 Nov 2000 07:05:59 -0500 (EST)
Received: from speechworks.com ([10.6.70.30])
by vishnu.speechworks.com (8.9.3+Sun/8.9.3) with ESMTP id HAA00745
for <dan.faulkner@speechworks.com>; Fri, 24 Nov 2000 07:03:50 -0500 (EST)
Message-ID: <3A1E5925.11C0CC@speechworks.com>
Date: Fri, 24 Nov 2000 07:03:49 -0500
From: Andrew Lowry <andrew.lowry@speechworks.com>
X-Mailer: Mozilla 4.7 [en] (WinNT; I)
X-Accept-Language: en
MIME-Version: 1.0
To: dan.faulkner@speechworks.com
Subject: here it is!
Content-Type: multipart/mixed;
boundary="-----D9A5530EC68939F7E9BE4970"

This is a multi-part message in MIME format.

-----D9A5530EC68939F7E9BE4970

Content-Type: text/plain; charset=us-ascii

Content-Transfer-Encoding: 7bit

Here is the file you asked for. Drew

-----D9A5530EC68939F7E9BE4970

Content-Type: text/plain; charset=us-ascii;
name="dan.txt"

Content-Transfer-Encoding: 7bit

Content-Disposition: inline;
filename="dan.txt"

blah blah blah blah blah...

-----D9A5530EC68939F7E9BE4970--

Default behavior

For either message format, the SpeechWorks E-mail Pre-processor identifies the message header and message body, and tries to identify address/signature blocks (although this can be difficult if users employ arbitrary formatting for their address/signature blocks).

Header processing

Discarding header lines

All lines in the header are discarded except the from, date, and subject lines. Each header line is terminated with a period. The expansions of the From, Date and Subject lines are specified in the substitution dictionary. The date in the date line is expanded. The year and time are discarded. For example:

Input	Preprocessed text
Date: Tue, 24 Oct 2000 07:03:49 -0500	<i>Your message arrived on Tuesday, twenty fourth October</i>

Reading From lines

If the From line contains an e-mail address and a real name, the real name is read:

Input	Preprocessed text
From: Dan Faulkner dan.faulkner@speechworks.com	<i>Your message is from Dan Faulkner</i>

Subject line abbreviations

In the subject line, abbreviations like FW and RE can be expanded using the substitution dictionary, e.g.:

Input	Preprocessed text
Subject: RE: Questions from customers	<i>The subject line says the message is a reply about questions from customers</i>

Here “Subject:” was replaced with *The subject line says*, and “RE:” was replaced with *the message is a reply about*.

When expanding the RE: and FW: strings, note that they frequently occur multiple times in a single subject line. For this reason, you should either expand them to something that makes sense when read more than once, or add multiple occurrences to the substitution dictionary, and define a single expansion for them, e.g.:

- ❑ RE:, the message is a reply
- ❑ FW: the message was forwarded
- ❑ FW:FW: the message was forwarded twice
- ❑ RE:FW:FW: this message is a reply to a message that was forwarded twice.

Body processing

Discarding data

In the message body, UUencoded data and octal data are skipped. Within the body, an empty line is considered to be a paragraph break, and if the last line of the paragraph doesn't end with a period, then a period is inserted, e.g.:

Original text	Pre-processed text
Hi	<i>Hi.</i>
How are you?	<i>How are you?</i>

Non-alphanumeric, non-space strings of more than three characters are deleted if they haven't been matched and substituted by the e-mail substitution dictionary. In this example, the lines of dashes have been deleted, so that the listener does not hear “dash dash dash dash dash...”:

Original text	Pre-processed text
----- All or some of this message may be privileged information. If you are not the intended recipient, please discard this message and any attachments now. -----	<i>All or some of this message may be privileged information. If you are not the intended recipient, please discard this message and any attachments now.</i>

Multiple punctuation marks

Multiple punctuation marks resolve to a single punctuation mark, e.g.:

Original text	Pre-processed text
What!???!!!!	<i>What!</i>

Embedded e-mail messages

The listener is notified of embedded e-mail messages (only the from line is read from embedded messages). The listener is also notified at the beginning and end of portions of text that have been indented with greater than (>), e.g.:

Original text	Pre-processed text
>What are you doing on Thursday? I don't know yet.	<i>This next section of text is indented.</i> What are you doing this Thursday? <i>That's the end of the section that was indented.</i> I don't know yet.

These notifications are specified, and can therefore be customized, in the e-mail substitution dictionary. (See “Using the E-mail Substitution Dictionary” on [page 3-15](#) for details.)

Signature processing

The Internet standard for indicating that a signature/address block is next in the text is the sequence `--\n`. If this sequence is found in the message body, the preprocessor assumes that the following text is an address block, until the next empty line. For example:

--

tel. +44 0803 123456

<http://www.speechworks.com>

In addition to this assumption, there is a heuristic process employed that looks for the following strings at the beginning of a line:

- ☐ ph
- ☐ tel
- ☐ fax
- ☐ pgr
- ☐ www
- ☐ mail
- ☐ http
- ☐ work
- ☐ home
- ☐ email
- ☐ e-mail

If any of these are found as the first alphabetic strings on two consecutive lines, then until the next empty line, the text is treated as an address block, e.g.:

Original text	Pre-processed text
+++++	Telephone, +44 0803 123456.
+ Tel. +44 0803 123456	Fax, +44 0803 654321.
+ Fax +44 0803 654321	www.speechworks.com .
+ www.speechworks.com	
+	
+++++	

MIME format

File attachments and media types are only identifiable in MIME format messages. The user can define what should be spoken when a file attachment is found (e.g., “There is an attachment to this message”), and the name of the file is read (e.g., “expenses.xls”). Otherwise, MIME messages are treated the same as partially processed messages.

Modes

You can control the behavior of the e-mail preprocessor through the use of modes. The user can set the following modes using the `SWIemailProcess()` function. (See “`SWIemailProcess()`” on [page 4-27](#).)

Mode	Function
DATE	Read the Date line from the header
FROM	Read the From line from the header
SUBJECT	Read the Subject line from the header
BODY	Read the message body
ADDRESS	Read any address/signature blocks
MIME_FORMAT	The message is in MIME format

At least one of DATE, FROM, SUBJECT, BODY, ADDRESS must be selected. If the message is in MIME format, then MIME_FORMAT should be combined with the other selections. Modes are combined by using bitwise OR. Here are some examples:

Mode	Function
DATE BODY	Read the date line and the message body
BODY ADDRESS	Read the body and any address/signature blocks
FROM SUBJECT	Read the from line and the subject line

Thus, for example, if you don't want to hear the address/signature blocks from a message, don't select ADDRESS.

MIME_FORMAT deserves special mention because it specifies the input rather than the desired output. If you enter a multipart MIME format message and you don't specify MIME_FORMAT, the speech output contains the MIME section boundaries, making it difficult to follow.

If you enter a message that is not in MIME format, and you do specify MIME_FORMAT, the message may be delivered in an unexpected way. (For example: the body may be skipped altogether and encoded data from file attachments is read out character by character.)

If there are no attachments, the message is read identically whether MIME_FORMAT is on or not.



Using the E-mail Substitution Dictionary

This chapter explains the layout and use of the e-mail substitution dictionary.

In This Chapter

- Dictionary entries on [page 3-17](#)
- Comments and escapes on [page 3-19](#)
- Notifications on [page 3-19](#)

File format

The e-mail substitution dictionary file is an ASCII text file with one entry per line. A default named "Email.dic" is supplied in the SDK's bin directory. You may supplement that or replace it entirely. (See "SWIemailInit()" on [page 4-26](#).)

The dictionary is split into five sections:

- ❑ Header Entries for substitutions that should only take place in the header
- ❑ Body Entries for substitutions that should only take place in the message body
- ❑ Address Entries for substitutions that should only take place in the address block
- ❑ Mime Entries for substitutions that should only take place in a Mime section boundary
- ❑ Global Entries for substitutions that should take place everywhere

The user can decide to expand the same string differently depending on whether it is found in the header or the body, or only expand a string if it is found in a specific section of the message, and ignore it if it appears anywhere else.

The entries in the substitution dictionary do not need to be sorted.

Dictionary entries

Dictionary entries are of the form

TARGET, SUBSTITUTION

The target is the string to be replaced, and the substitution is the string to replace the target. The target and the substitution are separated by a comma, e.g.:

RE:, Reply

This entry indicates that the e-mail pre-processor replaces the string *RE:* with the string *Reply*. The entries have the same syntax for every section. If an invalid entry is passed in, it is ignored and an error message is logged to stderr. An invalid entry is a line that does not contain a comma and at least one character on each side of the comma (i.e., a target and a substitution).

A target can be any string of any sequence of characters. Multiple words and tokens separated by white spaces are permitted. For example:

George Bush, the former president
George W. Bush, the new president

The e-mail pre-processor replaces the longest string possible from left to right. For example, given the hypothetical dictionary entries:

John, I
John Smith, my grandfather
John Smith Jr., my father

Input string	Dictionary output
John Smith Jr. entered the room.	my father entered the room.

Although both *John* and *John Smith* match the input, *John Smith Jr.* is the longest match.



NOTE

When a target has been identified, it is replaced by everything in the substitution. Be careful not to add a space at the end of the substitution by mistake, as it may have consequences for subsequent processing.

For example, suppose we had the following dictionary entry, and accidentally placed a space after the expansion:

SPWX, speechworks

In this case, a web address *www.spwx.com* would be expanded to:
www.speechworks .com.

This spurious space character would prevent the web address from being processed correctly at a later stage – it would be read as *www dot speechworks (pause)com* instead of *www dot speechworks dot com*.

By default, the substitution dictionary entries are matched case-insensitively. If you want an entry to be matched case-sensitively, you should append an asterisk to the target. For example, given these dictionary entries:

Tue, Tuesday
Wed*, Wednesday

Input string	Dictionary output
The couple wed on tue.	The couple wed on Tuesday.
The couple wed on Wed.	The couple wed on Wednesday.

If you want to use an asterisk as the last character of a genuine substitution, you must escape it, by preceding it with a backslash, for example, to match the input *gold** (case insensitive), use this dictionary entry:

gold*,gold star



NOTE

If you want a WAV file to be played instead of expanding the text to another piece of text, remember that Speechify supports user defined bookmarks. Bookmarks can be entered as substitutions and – provided that the bookmarks are surrounded by preceding and following white space – they are processed by Speechify. For example:

Date:, \!Mw1

In this example, the text “Date:” is replaced by a bookmark. When that resulting text is passed into Speechify, Speechify tells you the location in the output audio where the bookmark occurs. You could insert other audio (such as a .wav file) at that sample such as a recording of someone saying “The date the message arrived is.”

Comments and escapes

The e-mail substitution dictionary supports comments. Any line that begins with a # is taken as a comment. If you want to use a hash or a comma as part of the TARGET string, escaped it with a backslash. Therefore, the backslash must be escaped too:

Original text	Pre-processor behavior
#### This is a comment	Not loaded into memory
\#3,Number Three	Replace #3 with <i>Number Three</i>
Mon\,,Monday	Replace Mon, with <i>Monday</i>
C:\\Temp, my temp drive	Replace C:\\Temp with <i>my temp drive</i>

Notifications

Certain events can only be spotted inside the code (i.e., a simple dictionary look-up sometimes isn't enough). Examples are:

- ❑ UUencoded data
- ❑ octal data
- ❑ start of indented text
- ❑ end of indented text
- ❑ end of the message

When the e-mail pre-processor spots these, it inserts an upper case constant into the text before substitutions are performed. The constants that are inserted are self-explanatory. They are:

Constant	Meaning
!UUENCODED_DATA_NOTIFY!	UUencoded data found
!OCTAL_DATA_NOTIFY!	Octal data found
!INDENT_NOTIFY!	Start of indented text found
!END_OF_INDENT_NOTIFY!	End of indented text found
!END_MSG_NOTIFY!	End of message found

The following constants are also inserted when file attachments are found in MIME format messages:

- ❑ !IMAGE_FILE_NOTIFY!
- ❑ !AUDIO_FILE_NOTIFY!
- ❑ !APPLICATION_FILE_NOTIFY!
- ❑ !VIDEO_FILE_NOTIFY!
- ❑ !TEXT_FILE_NOTIFY!

Specify substitution text for these in the substitution dictionary. (The supplied Email.dic file contains appropriate defaults.) This means you can define what you want Speechify to say when any of the above events occur. If you want it to say nothing, don't put anything after the delimiter in the dictionary; e.g., if you don't want the listener to be notified that, say, the end of a message had been reached, change the substitution dictionary entry from this:

```
!END_MSG_NOTIFY!*,That's the end of the message.
```

To this:

```
!END_MSG_NOTIFY!*,
```



NOTE

If you remove these constants from the dictionary, the constant names are deleted by the e-mail pre-processor. This prevents the internal notifications from being read out loud.

Here is an example of a legal dictionary.

```
\!HEADER ENTRIES
Date:,The message arrived on
From:,The message is from
FW:,the message was forwarded to you, and it's about
RE:, the message is a reply, and it's about

\!MIMEPART ENTRIES

##### Speechify internal #####
!IMAGE_FILE_NOTIFY!,There's an image file attached.
!AUDIO_FILE_NOTIFY!,There's an audio file attached.
!APPLICATION_FILE_NOTIFY!,There's an application attached.
!VIDEO_FILE_NOTIFY!,There's a video file attached.
!TEXT_FILE_NOTIFY!,There's a text file attached.

##### Generic #####
.txt, dot text.
.doc, dot doc.

\!BODY ENTRIES

\!ADDRESS ENTRIES
email, \!Mw1
e-mail, \!Mw1
fax, \!Mw2
http, \!Mw3
tel, \!Mw4

\!GLOBAL ENTRIES

##### Speechify internal #####
!UUENCODED_DATA_NOTIFY!,I'll have to skip the next section.
!OCTAL_DATA_NOTIFY!,I'll have to skip the next section.
!INDENT_NOTIFY!,This next section of text is indented.
!END_OF_INDENT_NOTIFY!,That's the end of the indented section.
!END_MSG_NOTIFY!,That's the end of the message.

##### Generic #####
ASAP,as soon as possible
BTW,by the way
FYI,for your information,
WRT,with regard to
```




API Reference

All API function prototypes, types, error codes, and constants are located in the header file `SWIEmail.h`.

In This Chapter

- Calling convention on [page 4-24](#)
- Result codes on [page 4-25](#)
- `SWIemailInit()` on [page 4-26](#)
- `SWIemailProcess()` on [page 4-27](#)
- `SWIemailTerm()` on [page 4-28](#)

Calling convention

The calling convention is dependent on the operating system, and is defined in the SWIEmail.h header file.

Under NT:

```
#define SWIAPI __stdcall
```

Under Unix:

```
#define SWIAPI
```


Result codes

The following result codes are defined in the enum `SWIemailResult` in `SWIEmail.h`.

Code	Description
<code>SWIemail_SUCCESS</code>	The function completed successfully
<code>SWIemail_BAD_FILE_FORMAT</code>	The substitution dictionary doesn't contain all 5 section headings
<code>SWIemail_EMPTY_MESSAGE</code>	The input string is empty
<code>SWIemail_ERROR</code>	There was an error in the API
<code>SWIemail_FATAL_EXCEPTION</code>	(NT only.) A crash occurred within the <code>SWIemail</code> library. You should shut down the application.
<code>SWIemail_FILE_NOT_FOUND</code>	Unable to locate the substitution dictionary
<code>SWIemail_MEM_REQUEST</code>	The output string is bigger than the input buffer
<code>SWIemail_UNINITIALIZED</code>	<code>SWIemailProcess()</code> was called before <code>SWIemailInit()</code>

SWIemailInit()

Mode: Synchronous

Initializes the substitution dictionary.

```
SWIemailResult SWIAPI SWIemailInit (
    const char *FilePath
);
```

Parameter	Description
-----------	-------------

FilePath	Path to the substitution dictionary
----------	-------------------------------------

Notes

This must be the first API function called for the e-mail pre-processor.

If you pass in your own FilePath, you can call the dictionary whatever you want. If FilePath is NULL, then SWIemailInit() checks for the existence of the environment variable SWITTSSDK. If it exists, SWIemailInit() tries to load Email.dic from the path SWITTSSDK/bin. If the variable doesn't exist, SWIemailInit() tries to load Email.dic from the current working directory.

Return code	Meaning for SWIemailInit()
SWIemail_SUCCESS	SWIemailInit() returned successfully
SWIemail_BAD_FILE_FORMAT	The file does not contain the section headings for the HEADER, BODY, ADDRESS, MIMEPART, and GLOBAL sections
SWIemail_ERROR	Error opening the file
SWIemail_FILE_NOT_FOUND	SWIemailInit() did not find a file at all

SWIemailProcess()

Mode: Synchronous

Processes an e-mail message.

```
SWIemailResult SWIAPI SWIemailProcess (
    const char *Message,
    int *BufferSize,
    unsigned char Modes
);
```

Parameter	Description
Message	The e-mail message as a null-terminated string. The output is copied into this buffer.
BufferSize	Size of the input buffer. Its contents are changed if the output string is bigger than the input buffer.
Modes	Modes through combinations of the mode variables defined in SWIEmail.h. (See “Modes” on page 2-12.)

Notes

Return code	Meaning for SWIemailProcess()
SWIemail_SUCCESS	SWIemailProcess() returned successfully
SWIemail_EMPTY_MESSAGE	Input string is empty.
SWIemail_ERROR	Memory error in the API.
SWIemail_MEM_REQUEST	The output string is longer than the input buffer, and the value of the contents of BufferSize is changed to the required size.
SWIemail_UNINITIALIZED	SWIemailProcess() called before SWIemailInit().

The parameter BufferSize tells the function how long the input buffer is. If the function returns SWIemail_MEM_REQUEST, the application should reallocate the input buffer to the appropriate size and call SWIemailProcess() again.

SWIemailTerm()

Mode: Synchronous

Frees resources used by the preprocessor library.

```
SWIemailResult SWIAPI SWIemailTerm ( );
```

SWIemailTerm() must be the last function called. If it is called before calling SWIemailInit(), it returns SWIemail_UNINITIALIZED. It returns SWIemail_SUCCESS on completion.

Index

Symbols

-- 2-11
!APPLICATION_FILE_NOTIFY! 3-20
!AUDIO_FILE_NOTIFY! 3-20
!END_MSG_NOTIFY! 3-19
!END_OF_INDENT_NOTIFY! 3-19
!IMAGE_FILE_NOTIFY! 3-20
!INDENT_NOTIFY! 3-19
!OCTAL_DATA_NOTIFY! 3-19
!TEXT_FILE_NOTIFY! 3-20
!UUENCODED_DATA_NOTIFY! 3-19
!VIDEO_FILE_NOTIFY! 3-20
3-19
, 3-19
> 2-10
\\ 3-19

A

ADDRESS 2-12

B

BODY 2-12
body processing 2-9

C

calling convention 4-24
comments 3-19

D

DATE 2-12
default behavior 2-8
dictionary 3-15
 legal example 3-21
discarding data 2-9
discarding header lines 2-8

E

Email.dic 3-16, 3-20
embedded e-mail 2-10
escapes 3-19

F

features 1-2
FROM 2-12
Functionality 2-5

H

header processing 2-8

M

MIME format 2-12
MIME_FORMAT 2-12
modes 2-12

N

notifications 3-19

O

order of API calls 1-3

P

punctuation marks
 multiple 2-10

R

reading From lines 2-8
result codes 4-25

S

signature processing 2-11
special characters 3-19
SUBJECT 2-12
subject line abbreviations 2-8
substitution dictionary 3-16
support services vi
supported message formats 2-6
SWIEmail.h 4-23, 4-24
SWIemailInit() 1-3, 4-26
SWIemailProcess() 1-3, 2-6, 4-27
SWIemailResult 4-25
SWIemailTerm() 1-3, 4-28
SWIttsClosePort() 1-3
SWIttsInit() 4-26
SWIttsOpenPort() 1-3
SWITTSSDK 4-26
SWIttsSpeak() 1-2, 1-3
SWIttsTerm() 1-3

