SERVER FUNCTIONALITY DOCUMENT
EMAIL TRANSPORT

4.3.Email Transport

4.3.1. Requirements

This document covers four implementations of email transports:

· POP3 (receive)

· SMTP (receive)

· SMTP (send)

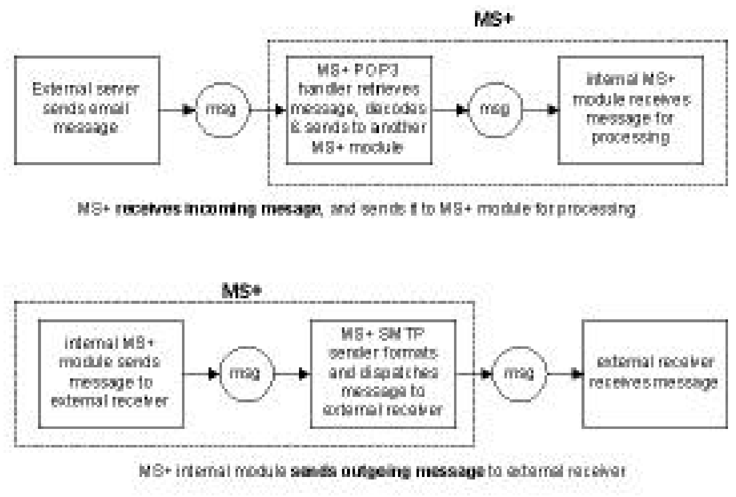
· Log (send)

Each implementation will be discussed in detail later in this section. First, we will discuss matters that affect them all. Generally, the job of an email transport is to either:

· *receive incoming messages* via querymaps and POP3 transport, then send the information to another MS+ module, OR,

· *send outgoing messages* requested by another module via SMTP transport (receive the request, format the message, and send it to the external system)

(See figure below.)


MS+
External server sends email message
msg
MS+ POP3 handler retrieves message, decodes & sends to another MS+ module
msg
internal MS+ module receives message for processing
MS+ receives incoming mesage, and sends it to MS+ module for processing
MS+
internal MS+ module sends message to external receiver
msg
MS+ SMTP sender formats and dispatches message to external receiver
msg
external receiver receives message
MS+ internal module sends outgoing message to external receiver

(email_overview_send_receive)

To do this, the transport must do the following:

- listen for and detect a request
- call the appropriate request handler, which will create an MS++ internal command and link the command to the specified threadpool
- optionally, send (dispatch) a message to a telecommunications carrier

Further, you must properly configure the MS+ email transports, via “config files.”

Each email transport has its own configuration settings, explained in this section.

MS+ interacts with other servers to *send or receive, process, and format email messages only*. In other words, MS+ is *not* a full-service email server that can interact directly with email clients that require communication with a full-service email server. When email comes into MS+, it is meant to be *processed* by MS+ (as opposed to delivery to an email client such as Eudora or Outlook).

However, MS+ can act like an email client by issuing client-like POP3 requests to ask a server, “What email do I have?” Moreover, MS+ can accept email from another server via SMTP.

NOTE: MS+ does not support IMAP – and therefore cannot get messages using IMAP.

You will find the Java programs for the email transports in the `transport/email` folder (see figure below). NOTE: Your ROOT FOLDER will likely be named something other than `server_src Folder`.

4.3.1.1. MS+ Email Transports

As mentioned, MS+ provides **four implementations of email transports** (details later) as follows:

POP3 (receive)	`Pop3Listener`
SMTP (receive)	`SmtpListener`
SMTP (send)	`SmtpDispatcher`

· Log (send) `EmailLogDispatcher`

Each implementation supports **single- and multi-part messages** (example: file attachments) for both sending and receiving.

(**NOTE**: Currently, the SMTP receive transport supports only single-part messages.)

MS+ handles email as a *request*. Action taken by MS+ will differ depending on how the Service Developer writes a service, as follows:

· **Request model**: MS+ receives incoming email, does some processing to fulfill the request then responds to the user with email.

· **Content loading model:** MS+ processes email and stores the email's data as content (most likely stored in database). That data is then available to requests by other transports (such as SMS).

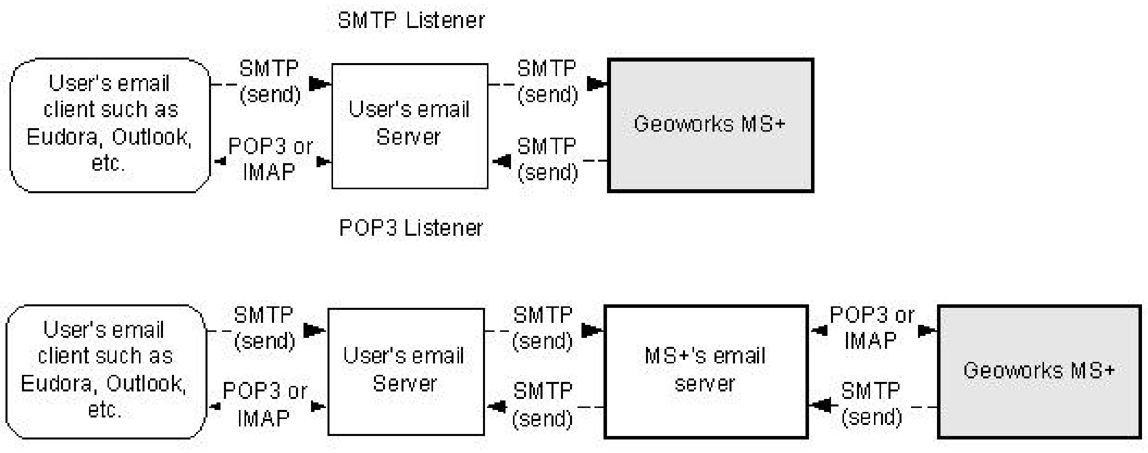
4.3.1.2. Typical Email / MS+ Interactions

From a Service Developer's point of view, the question is:

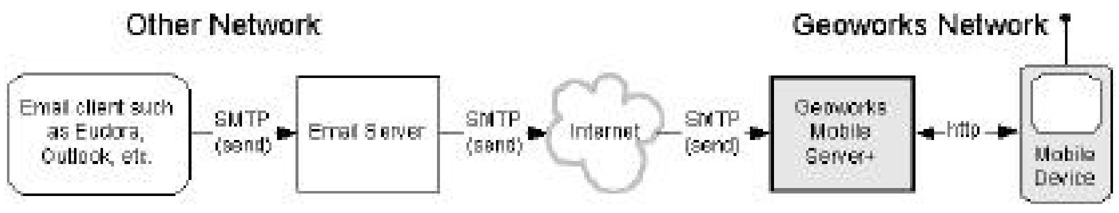
· Does email come to direct to MS+ acting as an SMTP email listener? (or)

· Does email come to MS+ via intermediate, full-service mail server where MS+ polls server for messages via POP3?

Figure below shows these two ways:



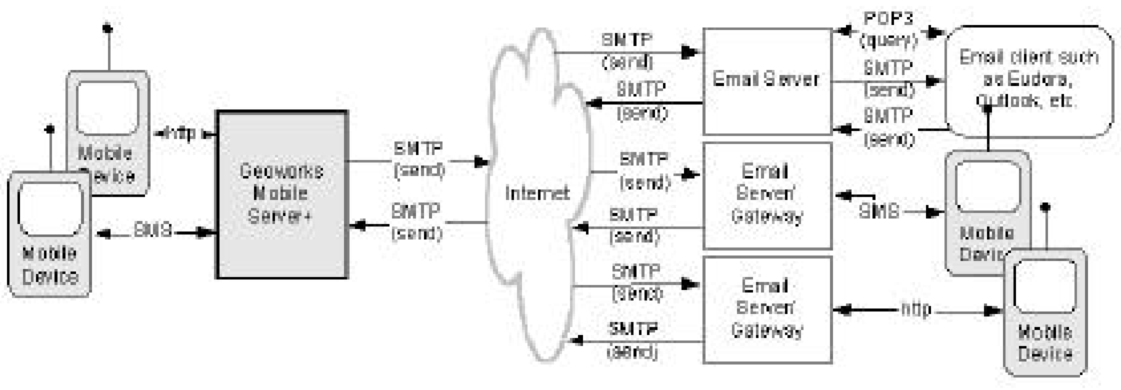
SMTP is used to send messages. Figure below shows how an email client on Other Network sends a message to a mobile device via MS+ presuming viewing through HTTP on device. This requires building a service to do so:



Other Network sends email message to Mobile Device via Geoworks Network

- Email client sends message to its server via SMTP
- Email server receives messages
- Email server then sends message across Internet to MS+ via SMTP (uses address `geoworks.com` to send to `mail.geoworks.com`)
- MS+ server stores message until user prompts, via mobile device via http (or POP3), *"Any new messages?"*

Email clients query a POP3 server for waiting messages using POP3. The figure below shows various ways that messages are routed from/to various devices using HTTP, SMS, SMTP and POP3 protocols.



Various send/receive possibilities using HTTP, SMS,SMTP, and POP3 protocols

From left to right: Notice how mobile devices can send messages via MS+ to other mobile devices or email client.

From right to left: Notice how mobile devices or email client can send messages to other mobile devices via MS+.

4.3.1.3. Request Translators

MS+ currently supports three email *request types*:

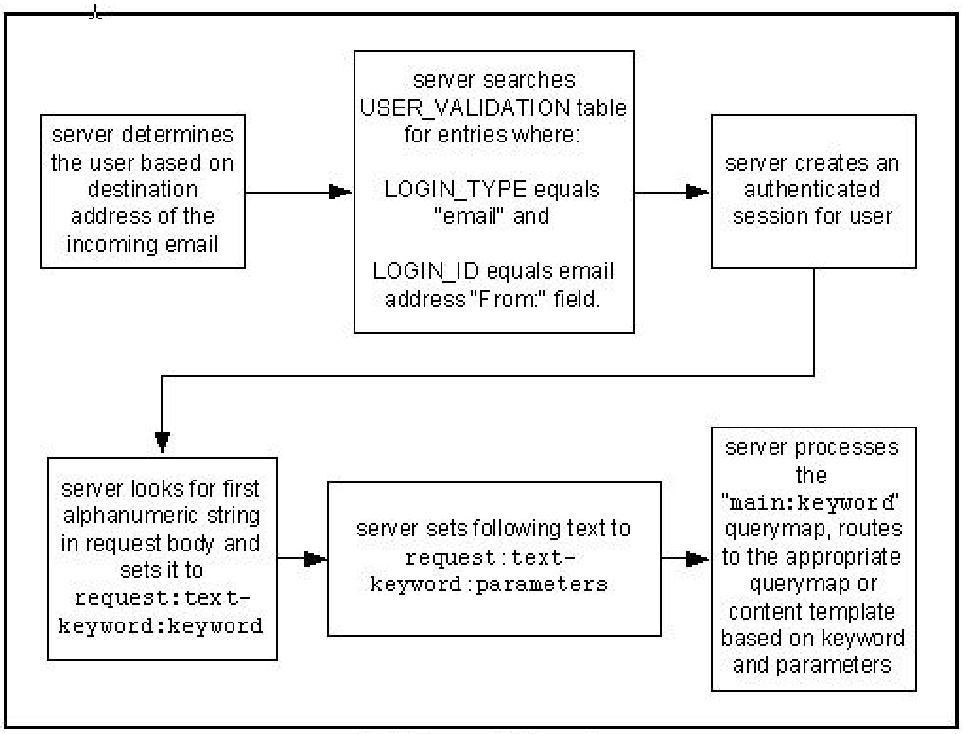
· text keyword

· message

· generic

Remember that a *request type* is an abstract message type that may be relevant across transport types, for example, a *text keyword request type* can be used for SMS or email.

Text Keyword: For a *text keyword request*, MS+ determines the receiving user based on the destination address of the incoming email. (See Text Keyword Request flowchart below.)


server determines the user based on destination address of the incoming email
server searches USER_VALIDATION table for entries where: LOGIN_TYPE equals "email" and LOGIN_ID equals email address "From:" field.
server creates an authenticated session for user
server looks for first alphanumeric string in request body and sets it to request:text-keyword:keyword
server sets following text to request:text-keyword:parameters
server processes the "main:keyword" querymap, routes to the appropriate querymap or content template based on keyword and parameters

Text Keyword Request

(email_text_keyword_request)

MS+ then creates an authenticated session for that user by searching the `USER_VALIDATION` table for entries where:

1. textual message bodies in character sets other than US-ASCII,
2. an extensible set of different formats for non-textual message bodies,
3. multi-part message bodies, and
4. textual header information in character sets other than US-ASCII.

Thus, MIME is a common method for transmitting non-text files via Internet email (email originally designed for ASCII text).

MIME uses one of two techniques to encode non-text files for sending and to decode non-text files back to original format for receiving. (See *base64* and *quoted printable encoding,* below.) A MIME header is added to the non-text file; the header specifies the type of data and encoding technique used.

S/MIME (Secure MIME) adds RSA encryption for secure transmission.

MIME type, a file identification derived from MIME, identifies the contents of the file. MIME types, embedded in email messages, define the content of attachments. Web servers send the MIME type to a requesting browser so the browser can launch the appropriate helper application or plug-in.

MIME "Content Type" separates type and subtype with a slash; for example, `text/plain` and `image/gif`. The major types are `application`, `audio`, `image`, `text`, and `video`. Application refers to a variety of formats; for example, `application/x-pdf` refers to Adobe Acrobat documents, and `application/octet-stream` refers to an .EXE file. (See *base64, quoted printable encoding,* and *Wincode*.)

- **base64** is an encoding technique that converts binary data into ASCII text and vice versa, and is one of two techniques used by MIME. Base64 divides three 8-bit bytes of the original data into four 6-bit units, which it represents as four 7-bit ASCII characters. This typically increases the original file by about a third. (See *quoted printable encoding* below.)

- **quoted printable encoding** is an encoding technique that converts binary data into ASCII text and vice versa and is one of the techniques used by MIME. This technique is good for text that contains an occasional 8-bit character. The 7-bit text is kept the same, and only the 8-bit text is encoded. (See *base64* above.)

4.3.2. POP3 (Receive) Transport (`Pop3Listener` and `Pop3RequestHandler`)

4.3.2.1. Syntax

All transports are setup via Config files by Main during system initialization (see examples herein). After that, transports are invoked automatically as needed by triggers (hardware interrupts).

`Pop3Listener` is called by `Main` via `InitializationBlockade.open`. Once invoked, `Pop3Listener` calls `Pop3RequestHandler` to handle the request.

4.3.2.2. Public Methods

`Pop3Listener`

`Pop3RequestHandler`

`init`

`start`

`listen`

`parse`

`stop`

4.3.2.3. Purpose of POP3 Transport

The purpose of **`Pop3Listener`**, based on JavaMail, is to:

- initialize with direct configuration parameters
- set server and account settings
- poll for incoming messages from a single account
- translate from appropriate character encoding
- dispatch new arriving messages to `Pop3RequestHandler`

The purpose of **`Pop3RequestHandler`** is to handle email requests using POP3 by running the request handler on its own thread.

Notes: POP3 (Post Office Protocol, v 3) is a standard mail server protocol commonly used on the Internet. The mail server stores incoming email until users

pollRate	Specifies the number of seconds to wait before polling the email server, usually 10 (seconds).

Following are four sample config files that receive POP3 email [**bold** emphasis added for ease of reading]:

```
######################################################################
   [transport:core-test-pop3].
   # Always assign a transport to a threadpool.
   threadPool = main
   # listener specifies the class to act as the email listener.
   listener = transport.email.pop3.Pop3Listener
   # service provides lists of stores, often needed by transports.
   service = general-test
   # requestTranslators specifies the type of translation performed
   # on the incoming email.
   requestTranslators = transport.email.EmailToTextKeywordTranslator
   # loginType is required for authenticating email users on an
   # email transport. The user must have an entry in the
   # USER_VALIDATION table with a LOGIN_TYPE of ‘email’ for this
   # to work.
   loginType = email
   # listenerServer provides the POP3 user account, password, and
   # the POP3 host that the above listener will talk to.
   listenerServer = email.server.com
   # commandProcessors specifies the command processors for this
   # transport.
   commandProcessors = command.getcontentProcessor,\
   command.runProcessor
   # If you don’t set deleteMail true, Pop3Listener will keep
   # processing the messages it has already seen. This is the
   # problem with POP3.
   deleteMail = true
   user = myuser
   password = mypassword
   # For email transports, checkPassword should be false. The server
   # has no way to check the password for email requests, at this time.
   checkPassword = false
   # pollRate specifies the number of seconds to wait before polling
```

```
    # the server.
    pollRate = 10
#######################################################################
    [transport:core-test-email]
    # You will need to create an entry for this category in your
    # local.config with appropriate settings for the user and
    # password keys.
    #
    # = myuser
    # = mypassword
    # The following are possible accounts to use:
    # user = jftest1
    # password = sting1
    # user = jftest2
    # password = bla7th
    # user = jftest3
    # password = p09Aq1
    # user = jftest4
    # password = bar?none
    threadPool = main
    listener =
    com.geoworks.premion.server.transport.email.pop3.Pop3Listener
    service = core-test
    requestTranslators =
    com.geoworks.premion.server.transport.email.EmailToTextKeywordTransl
    ator
    dispatcher =
    com.geoworks.premion.server.transport.email.smtp.SmtpDispatcher
    dispatcherServer = almh.geoworks.com
    listenerServer = almh.geoworks.com
    replyDomain = geoworks.com
    commandProcessors =
    com.geoworks.premion.server.command.getcontentProcessor, \
    com.geoworks.premion.server.command.runProcessor
    # If you don't delete mail, Pop3Listener will keep processing the
    # messages it has already seen. This is the problem with POP3.
    deleteMail = true
    # Specifies the type of translation performed on the incoming email.
```

4.3.3. SMTP – Receiving Email (`SmtpListener`)

4.3.3.1. Syntax

`SmtpListener` called by: `Main` (via `InitializationBlockade.open`).

`SmtpListener` calls ??

4.3.3.2. Public Methods

`SmtpListener`

`init`

4.3.3.3. Purpose of `SmtpListener` Transport

The purpose of `SmtpListener` is to:

- wait for connections on the configured port (usually port 25)
- spin off a new thread to run `SmtpRequestHandler`

NOTE: `SmtpListener` currently does not use JavaMail to parse the incoming messages.

SMTP (Simple Mail Transfer Protocol) is the standard email protocol on the Internet. It is a TCP/IP protocol that defines the message format and the message transfer agent (MTA), which stores and forwards the mail. SMTP was originally designed for ASCII text only, but MIME and other encoding methods enable program and multimedia files to be attached to email messages.

SMTP servers route SMTP messages throughout the Internet to a mail server, which stores incoming mail.

> **NOTE**: SMTP receive is a simple implementation, which does not support multi-part messages.

For **receiving**, SMTP does the following:

- directly receives incoming messages through SMTP port.
- can accept mail for an arbitrary number of accounts

4.3.3.5. Output from `smtpListener`

Anything else? syntax parameters, descriptions, files, controls, options, external variables, data types, etc.?

4.3.3.6. Module Details (as appropriate to explain what module DOES – not how it does it)

Anything else? syntax parameters, descriptions, files, controls, options, external variables, data types, etc.?

4.3.3.7. Request Variables

When MS+ receives an email, MS+ sets several request variables according to the incoming message. They are as follows:

Variable Name	Type	Description	Example
`request:email:header`	Map	Name/value pairs of message headers.	(Name:) “`request:email:header:content-type`”
`request:email:body`	List or Map	A `body` may be single- or multi-part. For single-part, `body` will be a Map with attributes of the single part. For multi-part, `body` will be a List of parts, each represented by a Map with its attributes. The `body` part structure is described below.	

Each `body` part has the following structure:

Value name	Type	Description	Examples
`content`	Byte array or String	Content associated with this part.	
`character-encoding`	String	Character set used to encode the content of this body part, if body is a string.	S-JIS, EUC-JP, US-ASCII
`type`	String	MIME type of this part.	“text”
`sub-type`	String	MIME subtype of this part.	“html”
`header`	Map	Name and value pairs for outgoing HTTP headers. The empty map is created at the beginning of request handling.	

4.3.4. SMTP – Sending Email Module

4.3.4.1. Syntax

DISPATCH Tag calls `SmtpDispatcher` and SmtpRequestHandler

4.3.4.2. Public Methods

`init`

`instantiateResource`

`dispatchLow`

`getInputStream`

`getOutputStream`

4.3.4.3. Purpose of `smtpDispatcher`

The purpose of `SmtpDispatcher` is to:

· ensure that transport has been set

· determine server and account settings

· build an email using the output variables (`output:email:*`)

· dispatch content through SMTP email transport (`dispatcherServer` is used because an email transport may be both listener and dispatcher)

Note that the Dispatcher dispatches *one email at a time*. To dispatch different formats for the same message requires a separate dispatch for each format.

SMTP (Simple Mail Transfer Protocol) is the standard email protocol on the Internet. It is a TCP/IP protocol that defines the message format and the message transfer agent (MTA), which stores and forwards the mail. SMTP was originally designed for ASCII text only, but MIME and other encoding methods enable program and multimedia files to be attached to email messages.

SMTP servers route SMTP messages throughout the Internet to a mail server, which stores incoming mail.

For **sending**, SMTP (based on Java Mail) does the following:

4.3.4.8. Dispatching the Email

It is easy to send email from a querymap. You need the message body and destination email addresses (in RFC822 format). The variable name "`output`" is the default output variable, but any variable name can be used for in-querymap dispatching.

Value name	Type	Description	Examples	Required ?
`output:body`	List or Map	A single- or multi-part `body`, as specified in the general dispatch specification.	Y	
`output:email:header`	Map	Name and value pairs for outgoing email headers. The empty map is created at the beginning of request handling.		N
`output:email:header:reply-address`	String	Name from which the mail will appear to be sent.	Daemon	N, if 'user' set in config file
`output:email:address`	String or List	A single String containing a destination address, or a List of Strings each with one address. These will not appear in the outgoing email, similar to using "Bcc:."	"Some Person" `<some@server.com>`, `someone@myserver.org`	N, if 'to' is used
`output:email:`	String	A single String	"Some Person"	N, if

`to`	or List	containing a destination address, or a List of Strings each with one address. These will appear in the "`To:`" field of the outgoing email.	`<some@server.com>, someone@myserver.org`	'address' is used
`output:email:charset`	String	Character encoding for the email headers. If this is not set, the system default will be used.	S-JIS, EUC-JP, US-ASCII	N

Each `body` part conforms to the general dispatch specification, but supports the following attribute:

Value name	Type	Description	Examples	Required?
`header`	Map	Name and value pairs for headers associated with this part.		N

Below is an example of how to set up the request variables before calling DISPATCH.

```
<*dispatch transport='test-email'*>
   <*map*>
      <*varpair name='body'*><*map*>
         <*varpair name='content'*>You have successfully received a
            test message.<*/varpair*>
         <*varpair name='character-encoding'*>Cp1252<*/varpair*>
   <*/map*><*/varpair*>
      <*varpair name='email'*><*map*>
         <*varpair name='address'*><*list*>
            <*param*>nameguy@geoworks.com<*/param*>
            <*param*>tigger@geoworks.com<*/param*>
         <*/list*><*/varpair*>
      <*varpair name='header:subject'*>Here is your test
         message.<*/varpair*>
      <*/map*><*/varpair*>
```

4.3.5.6. Module Details *(as appropriate to explain what module DOES – not how it does it)*

Anything else? syntax parameters, descriptions, files, controls, options, external variables, data types, etc.?

4.3.6. Revision History

Date	**Author**	**Details**
1 February 2000	Paul Canavese	Initial revision.
2 February 2001	Mike Hayden	Edit/format for 4.0
15 February 2001	Mike Hayden	Updated Overview Section
2 April 2001	Mike Hayden	General update with additional information