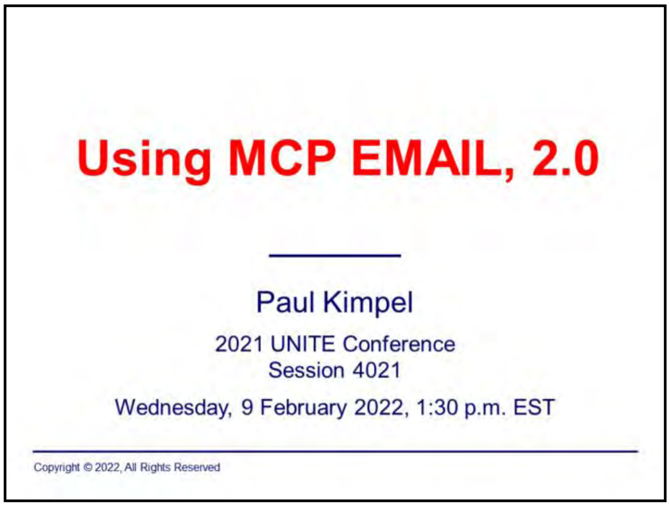


Using MCP EMAIL, 2.0

Paul Kimpel

2021 UNITE Conference
Session 4021

Wednesday, 9 February 2022, 1:30 p.m. EST

Copyright © 2022, All Rights Reserved

Using MCP EMAIL, 2.0

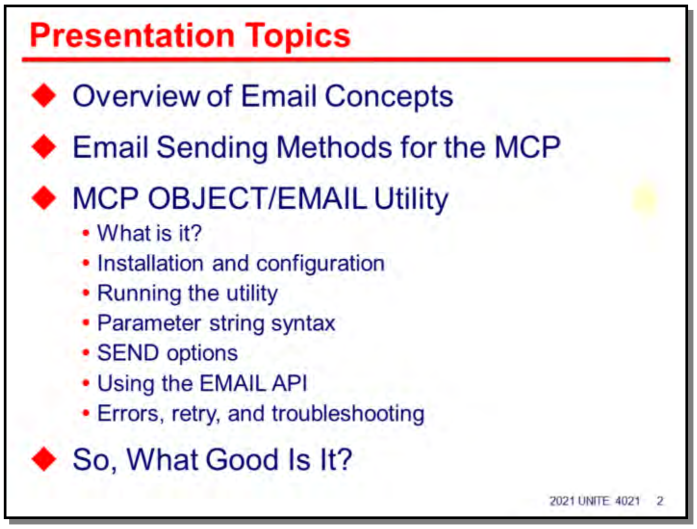
2021 UNITE Virtual Conference
Session 4021

Wednesday, 9 February 2022, 1:30 p.m. EST

Paul Kimpel
San Diego, California

http://www.digm.com

e-mail: paul.kimpel@digm.com



Presentation Topics

- Overview of Email Concepts
- Email Sending Methods for the MCP
- MCP OBJECT/EMAIL Utility
  - What is it?
  - Installation and configuration
  - Running the utility
  - Parameter string syntax
  - SEND options
  - Using the EMAIL API
  - Errors, retry, and troubleshooting
- So, What Good Is It?

2021 UNITE 4021 2

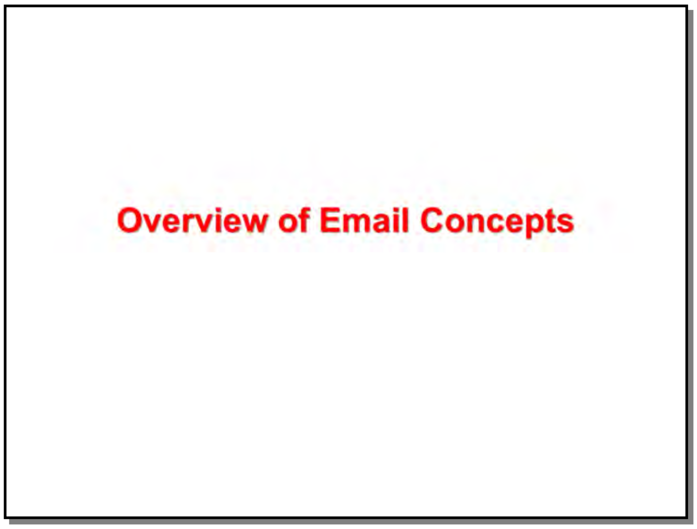
Today I would like to discuss what to me is one of the nicest facilities of the MCP – the ability to send email messages from MCP-resident applications. This is an update of a presentation I made at the 2009 UNITE Conference in Minneapolis.

I'll start with a brief overview of email in general – the components, protocols, and message formats used.

Next, I'll briefly discuss a few methods for sending email from MCP applications, using both bundled and third-party solutions.

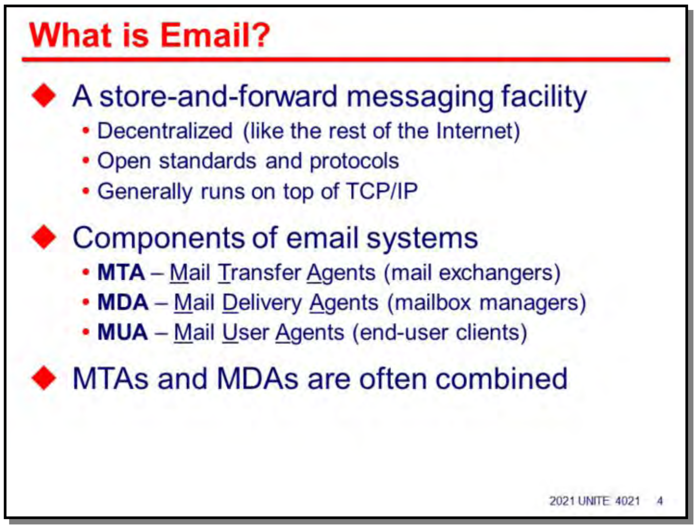
The bulk of this presentation will be devoted to one of those methods, the **`OBJECT/EMAIL`** utility that is bundled with the standard MCP release. I will discuss how to install and run the utility, the components of its command parameter syntax, options it supports for sending email messages, how it can be called from application programs in addition to run as a utility program, and how it handles errors and retry operations.

I will finish with some examples of **`OBJECT/EMAIL`** put to use in a real environment.



Overview of Email Concepts

To begin, let us briefly review the concepts and components that make email in general work.



What is Email?

- A store-and-forward messaging facility
  - Decentralized (like the rest of the Internet)
  - Open standards and protocols
  - Generally runs on top of TCP/IP
- Components of email systems
  - **MTA** – Mail Transfer Agents (mail exchangers)
  - **MDA** – Mail Delivery Agents (mailbox managers)
  - **MUA** – Mail User Agents (end-user clients)
- MTAs and MDAs are often combined

2021 UNITE 4021 4

Email has been described as the "killer app" of the Internet. There are lots of wonderful applications of Internet technology, but email remains the most significant one. We could probably do without the Web, although that would be a great loss. We could probably not do without email at this point.

Email is a store-and-forward messaging facility. It is like teletype and SMS phone messaging in that respect, and unlike telephone voice calls. A great deal of its utility and popularity comes from the fact that the receiver does not need to be available when the sender wishes to communicate. Messages are stored until the receiver gets around to checking for them.

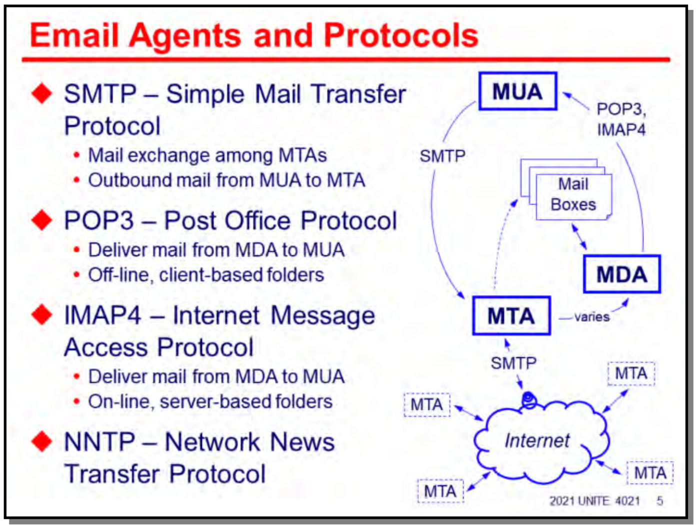
One thing that has made email a success is that it is based on open standards. No one owns email, and there is a vibrant collection of both proprietary and open source solutions for creating, sending, receiving, and reading email messages.

Another thing responsible for its success is that email is awesomely scalable. The whole planet now communicates using message formats and standards that, while they have been enhanced over the years, have not substantially changed since the early 1980s. Part of this success is due to the underlying openness and scalability of TCP/IP and the Internet, but much is due to the original design of the email formats and protocols.

Email systems are composed of three basic components:

- **Mail Transfer Agents**, or **MTA**s. These are also known as "mail exchangers," and are the programs responsible for sending and routing email messages to their destinations. The protocols were originally designed to allow mail to be routed through a web of interconnected MTAs. That can still happen within large organizations, but with the advent of the Internet, the much more common case today is that the sending MTA directly contacts the receiving MTA without needing to route a message through intermediaries.
- **Mail Delivery Agents**, or **MDA**s. These are programs that manage mail boxes and deliver mail to end users. MTAs either deliver mail to MDAs or deposit mail directly in mail boxes for which they are the receiving agent.
- **Mail User Agents**, or **MUA**s. These are end-user client programs used to read and compose email messages. Microsoft Outlook and Mozilla Thunderbird are common examples of MUAs.

It is not uncommon to see the functions of MTAs and MDAs combined in a single package. Microsoft Exchange is an example of this. On Unix/Linux systems, the functions are often implemented separately, with SendMail and Postfix being examples of MTAs, and Dovecot, maildrop, and procmail being examples of MDAs.



Email Agents and Protocols

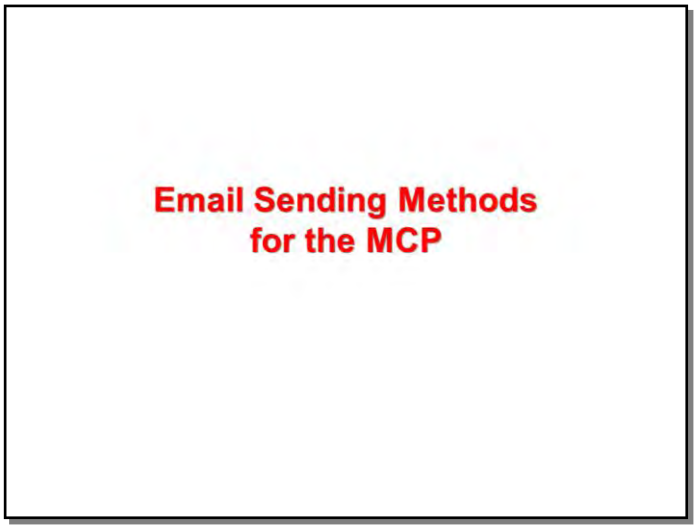
- SMTP – Simple Mail Transfer Protocol
  - Mail exchange among MTAs
  - Outbound mail from MUA to MTA
- POP3 – Post Office Protocol
  - Deliver mail from MDA to MUA
  - Off-line, client-based folders
- IMAP4 – Internet Message Access Protocol
  - Deliver mail from MDA to MUA
  - On-line, server-based folders
- NNTP – Network News Transfer Protocol

MUA
POP3, IMAP4
SMTP
Mail Boxes
MDA
MTA
varies
SMTP
MTA
MTA
Internet
MTA
MTA

2021 UNITE 4021 5

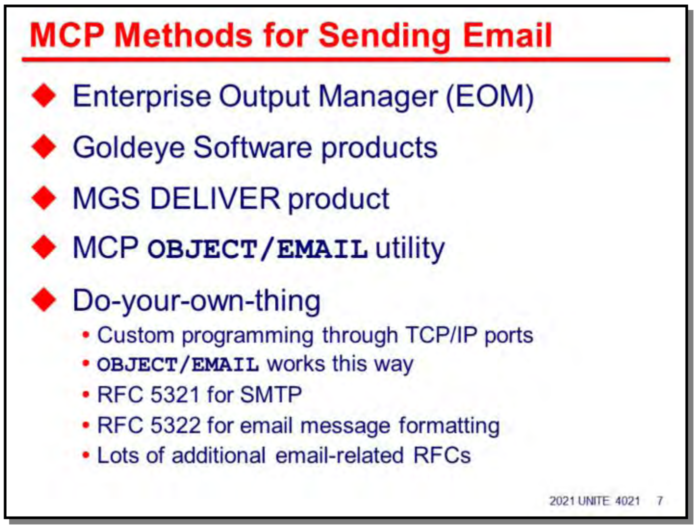
This slide shows the relationship between email protocols and the software components.

- **SMTP** (Simple Mail Transfer Protocol) is the scheme used to communicate among MTAs. As mentioned earlier, most sending MTAs can now contact the destination MTAs directly through the Internet, but SMTP has facilities to allow forwarding of messages through a web of MTAs. Some MTAs are able to deposit incoming messages directly into the receiving user's mail boxes; others pass the messages to a delivery agent for mailbox management. SMTP is also used by user agents to send outgoing emails. Note that SMTP is not used by clients for incoming emails, which is why you usually need to configure the sending and receiving servers separately in your email client.
- **POP3** (Post Office Protocol, version 3) is a scheme used by MDAs and MUAs for delivering incoming messages to end users. Like mail boxes at a post office, users "pick up" their mail when they read it, usually transferring the messages to their local system and removing it from the MDA's storage.
- **IMAP4** (Internet Message Access Protocol) is an alternate scheme used by MDAs and MUAs for delivering incoming messages to end users. Unlike POP3, it maintains mailbox messages on the server, and is commonly used with roaming users or web-mail clients.
- **NNTP** (Network News Transfer Protocol) is strictly speaking not an email protocol but has many features in common with email (particularly message formats), and is often implemented in email server software. This is the protocol used with news groups (e.g., comp.sys.unisys) to distribute and access news postings.



Email Sending Methods for the MCP

With that overview of email concepts, let us now talk about methods available to send email from MCP systems.



MCP Methods for Sending Email

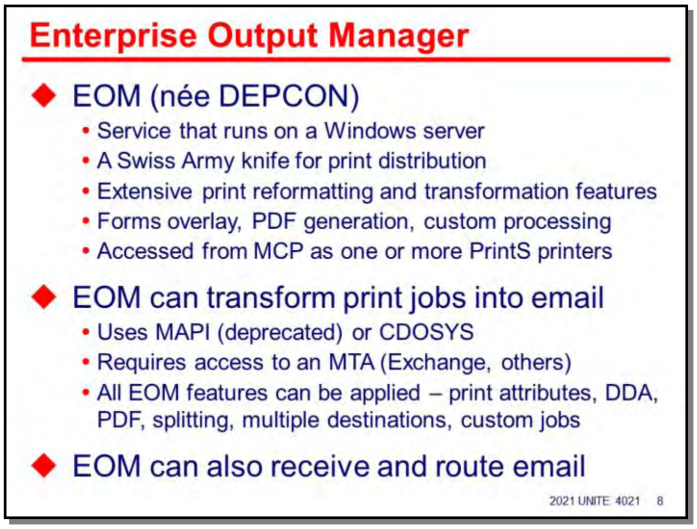
- Enterprise Output Manager (EOM)
- Goldeye Software products
- MGS DELIVER product
- MCP **OBJECT/EMAIL** utility
- Do-your-own-thing
  - Custom programming through TCP/IP ports
  - **OBJECT/EMAIL** works this way
  - RFC 5321 for SMTP
  - RFC 5322 for email message formatting
  - Lots of additional email-related RFCs

2021 UNITE 4021 7

There are currently five methods that you can use to send email from an MCP system. If there are more, I am not aware of them.

I will talk about the first three in this section of the presentation, and the remainder of the presentation will focus on the fourth, the **OBJECT/EMAIL** utility.

The last item, "do your own thing," is there to recognize that email is simply a process of sending data over a TCP/IP connection. The basic SMTP protocol is not that difficult to use, and it's possible to create a solution of your own by programming TCP port files or sockets. In fact, this is exactly what the Goldeye, MGS, and **OBJECT/EMAIL** products do. To give you an idea of the relative simplicity of the basic email protocols, the implementation of **OBJECT/EMAIL** for release 53.1 was only 4K lines of DCALGOL.



Enterprise Output Manager

- EOM (née DEPCON)
  - Service that runs on a Windows server
  - A Swiss Army knife for print distribution
  - Extensive print reformatting and transformation features
  - Forms overlay, PDF generation, custom processing
  - Accessed from MCP as one or more PrintS printers
- EOM can transform print jobs into email
  - Uses MAPI (deprecated) or CDOSYS
  - Requires access to an MTA (Exchange, others)
  - All EOM features can be applied – print attributes, DDA, PDF, splitting, multiple destinations, custom jobs
- EOM can also receive and route email

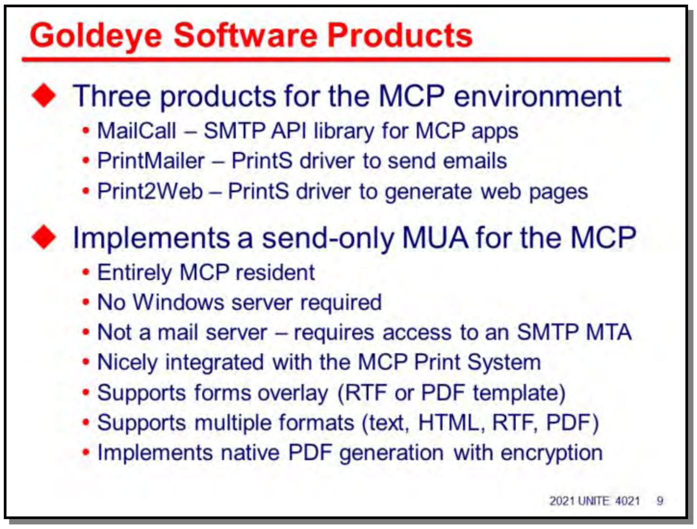
2021 UNITE 4021 8

The second method for sending email from MCP applications involves Enterprise Output Manager (EOM), formerly known as DEPCON. In its current implementation, EOM runs as a Windows service on a separate processor. It is primarily a print distribution facility, but also has extensive features for reformatting and transformation of print data. It can do electronic forms overlay, generate PDF documents, and do custom reformatting of print data. It is typically accessed from MCP applications by defining a virtual printer device in the MCP Print System and directing print requests to that printer.

A basic version of EOM, the Departmental Edition, is bundled with the standard Clearpath MCP software release. The ability to generate PDF documents and do custom reformatting of print data require extra-cost add-ons, but that bundled version does have the ability to transform print jobs into email messages. In order to do that, the Windows server on which EOM runs must have MAPI (which is deprecated, but still works) or CDOSYS (which has been bundled with Windows since Windows 2000). The EOM server must also have access to an email server (an MTA). A server running IIS SMTP is more than adequate for this purpose.

Email output functions as a “physical” printer in EOM. There is also an email job type which can be activated from file masks. All EOM features can be applied to print requests that are output to email destinations – print attributes, data dependent attributes (DDA), PDF output, print stream splitting, multiple destinations, and custom jobs.

EOM has an additional feature that none of the other methods discussed here possess – it can also receive and route email messages as if they were incoming print jobs.



Goldeye Software Products

- Three products for the MCP environment
  - MailCall – SMTP API library for MCP apps
  - PrintMailer – PrintS driver to send emails
  - Print2Web – PrintS driver to generate web pages
- Implements a send-only MUA for the MCP
  - Entirely MCP resident
  - No Windows server required
  - Not a mail server – requires access to an SMTP MTA
  - Nicely integrated with the MCP Print System
  - Supports forms overlay (RTF or PDF template)
  - Supports multiple formats (text, HTML, RTF, PDF)
  - Implements native PDF generation with encryption

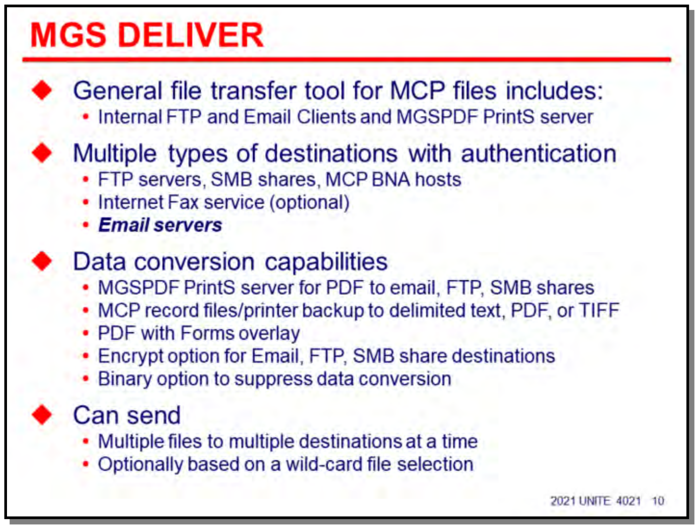
2021 UNITE 4021 9

The third method for sending email from MCP systems uses a software product from Goldeye Software, Inc. The Goldeye product is a suite of three independent but interrelated packages, named PrintMailer, MailCall, and Print2Web.

Each of these packages implements a send-only email client (MUA) that runs in the MCP environment. Unlike EOM, the packages run entirely within the MCP environment, and no separate Windows server is required. The Goldeye packages are not a mail server, however, and still require access to an SMTP MTA in order to work.

The Goldeye packages were written specifically for the MCP and integrate nicely with its facilities, especially the Print System. They can be used to format MCP print data with forms overlays (with the form template coming from either a Microsoft RTF document or a PDF document). It also supports multiple output formats, including HTML, Microsoft RTF, and PDF.

One very impressive feature of the Goldeye suite is that PDF output is built in. The generation of PDF output (including encryption) is done entirely using native MCP code.



MGS DELIVER

- General file transfer tool for MCP files includes:
  - Internal FTP and Email Clients and MGSPDF PrintS server
- Multiple types of destinations with authentication
  - FTP servers, SMB shares, MCP BNA hosts
  - Internet Fax service (optional)
  - ***Email servers***
- Data conversion capabilities
  - MGSPDF PrintS server for PDF to email, FTP, SMB shares
  - MCP record files/printer backup to delimited text, PDF, or TIFF
  - PDF with Forms overlay
  - Encrypt option for Email, FTP, SMB share destinations
  - Binary option to suppress data conversion
- Can send
  - Multiple files to multiple destinations at a time
  - Optionally based on a wild-card file selection

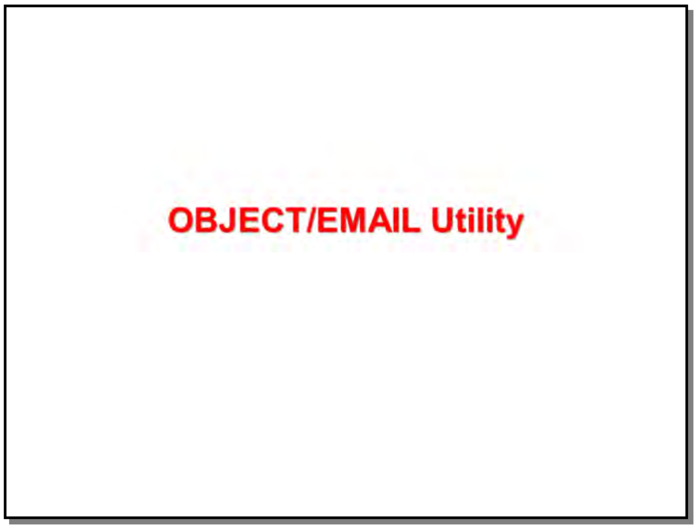
2021 UNITE 4021 10

MGS DELIVER is a general-purpose file transfer tool for MCP files. It runs in the MCP environment.

DELIVER supports sending files to multiple types of destinations, including FTP server, SMB shared directories, BNA hosts, and optionally, an Internet fax service.

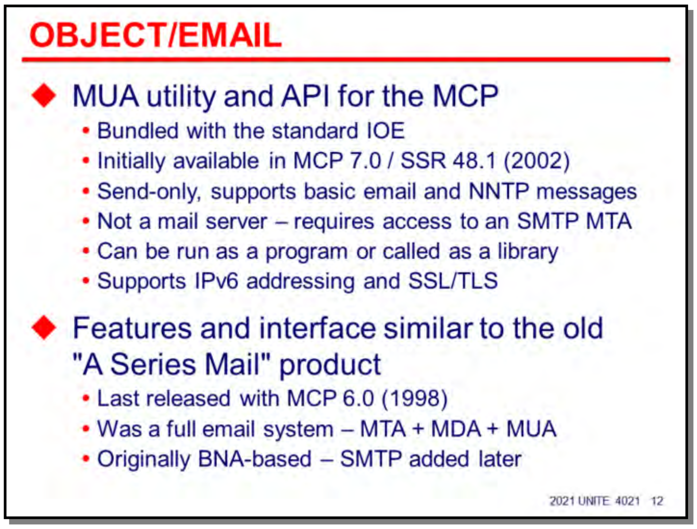
Of interest in the context of this presentation is the software's ability to send files as email messages and email attachments. It can convert textual MCP record-oriented files to line-delimited text files. It can also convert MCP printer backup files to text files. Options exist to format the attachment as PDF or as TIFF image files. There is a mechanism to apply a form overlay to the MCP data. The data can be encrypted in transit. There is also a binary option to suppress translation and data conversion altogether and send the data transparently.

Perhaps DELIVER's strongest point is its universality – a single application and command syntax can handle a wide variety of file distribution requirements, and a single command can transfer one or more files (including wildcard selections from directories) to multiple destinations of differing types.



OBJECT/EMAIL Utility

With that overview of methods for sending email from MCP applications, the rest of this presentation will focus on our featured method, the **`OBJECT/EMAIL`** utility program.



OBJECT/EMAIL

- MUA utility and API for the MCP
  - Bundled with the standard IOE
  - Initially available in MCP 7.0 / SSR 48.1 (2002)
  - Send-only, supports basic email and NNTP messages
  - Not a mail server – requires access to an SMTP MTA
  - Can be run as a program or called as a library
  - Supports IPv6 addressing and SSL/TLS
- Features and interface similar to the old "A Series Mail" product
  - Last released with MCP 6.0 (1998)
  - Was a full email system – MTA + MDA + MUA
  - Originally BNA-based – SMTP added later

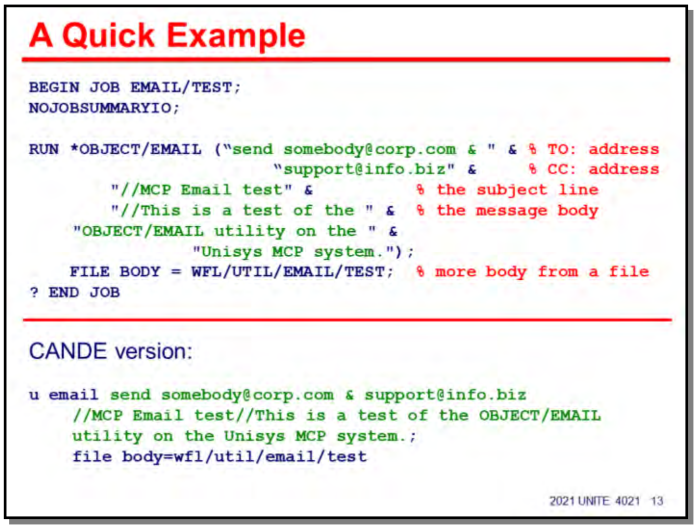
2021 UNITE 4021 12

`OBJECT/EMAIL` is a codefile that functions as both a utility program and an API (library) for MCP applications. It is bundled with the standard Clearpath IOE. It was initially released with MCP 7.0 / SSR 48.1 in 2002.

`OBJECT/EMAIL` is a send-only email client (MUA). It supports the composition of basic email messages. It also has the ability to post messages to a news group using NNTP. It is not a mail server, and like the other email sending methods we have discussed thus far, requires access to an SMTP server (MTA) in order to function.

Starting with MCP 53.1, **`OBJECT/EMAIL`** supports IPv6 addressing. It also supports SSL/TLS if that has been enabled for TCP/IP using the **`NW TCPIP OPTIONS+SSL`** command. See the discussion on the **`*INSTALLATION/OPTIONS`** file on how to configure **`OBJECT/EMAIL`** to use SSL.

The features and interface for this program are based on an older product, A Series Mail, which was last released with MCP 6.0. That product was a full email system, supporting MTA, MDA, and MUA roles. A Series Mail was originally BNA-based, with an SMTP interface being added later. Only the SMTP MUA sending capabilities have been carried forward into **`OBJECT/EMAIL`**. If you used A Series Mail, you may recognize several features in **`OBJECT/EMAIL`**.



A Quick Example

```
BEGIN JOB EMAIL/TEST;
NOJOBSUMMARYIO;

RUN *OBJECT/EMAIL ("send somebody@corp.com & " & % TO: address
                        "support@info.biz" &      % CC: address
        "//MCP Email test" &           % the subject line
        "//This is a test of the " &  % the message body
    "OBJECT/EMAIL utility on the " &
                "Unisys MCP system.");
    FILE BODY = WFL/UTIL/EMAIL/TEST;  % more body from a file
? END JOB
```

CANDE version:

```
u email send somebody@corp.com & support@info.biz
    //MCP Email test//This is a test of the OBJECT/EMAIL
    utility on the Unisys MCP system.;
    file body=wfl/util/email/test
```

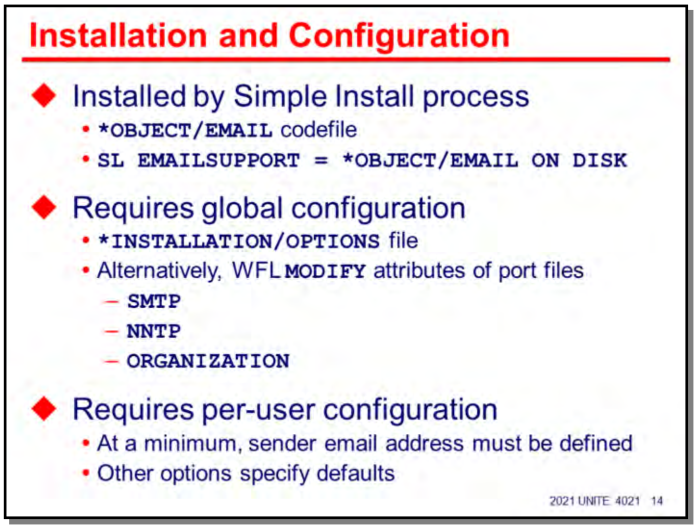
2021 UNITE 4021 13

Here is a quick illustration of **`OBJECT/EMAIL`** in action. The WFL job on this slide runs **`OBJECT/EMAIL`** to send a simple message. The utility is controlled by a string parameter. The components of this parameter are:

- The **`SEND`** command indicates that the program is to compose and send an email message (the alternative to this is **`SUBMIT`**, which posts a message to a news group).
- The next item in the parameter string (before the first "**`&`**") is a list of To: addresses. In this example, there is only one To: address.
- Following the first "**`&`**" (the one inside the quotes – the ones outside the quotes are of course WFL string concatenation operators) is the CC: list. Again, this example shows only one address in the list.
- The first "**`//`**" terminates the addressing portion of the parameter string. Following that is the subject line of the message.
- Following the second "**`//`**" is the body of the message.
- The file equation for **`BODY`** specifies that additional text for the body of the message is to be taken from the indicated file. This text will be appended to the body text in the parameter string.

Using this data, **`OBJECT/EMAIL`** will compose an RFC 822/5322 message and send it to its MTA. How it knows what that MTA is will be discussed shortly.

The second half of the slide shows an equivalent version of the WFL job using the CANDE Utility command. In this form, it's a little easier to see the structure of the parameter string, as the quotes and string concatenation operators have been eliminated. Email addresses are case-insensitive, so this second execution of the utility performs identically to the first.



Installation and Configuration

- Installed by Simple Install process
  - `*OBJECT/EMAIL` codefile
  - `SL EMAILSUPPORT = *OBJECT/EMAIL ON DISK`
- Requires global configuration
  - `*INSTALLATION/OPTIONS` file
  - Alternatively, WFL `MODIFY` attributes of port files
    - `SMTP`
    - `NNTP`
    - `ORGANIZATION`
- Requires per-user configuration
  - At a minimum, sender email address must be defined
  - Other options specify defaults

2021 UNITE 4021 14

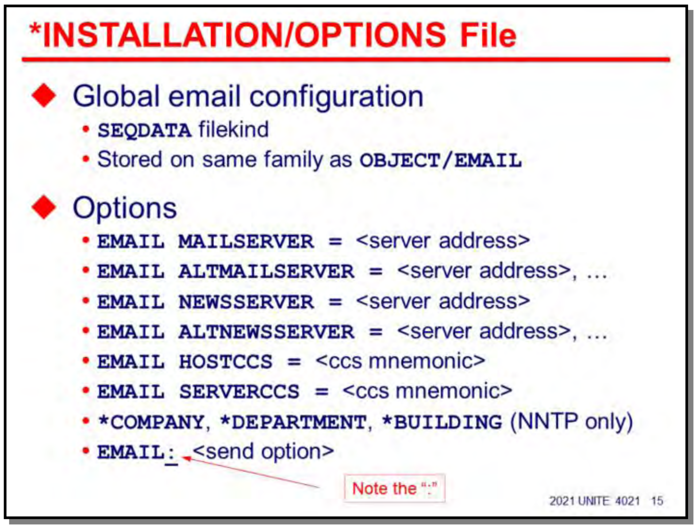
OBJECT/EMAIL requires a small amount of configuration. Before describing the rest of its capabilities, let us discuss how it is installed and configured in the MCP environment.

The program is typically installed by the Simple Install mechanism. That installation process simply copies the **OBJECT/EMAIL** codefile to the system and configures the system library function name **EMAILSUPPORT** to reference that codefile.

Once installed, **OBJECT/EMAIL** requires a set of global configuration settings for your email environment. There are two ways to do this:

- Create a ***INSTALLATION/OPTIONS** file. This is the recommended method, and the only one that will be discussed here, starting on the next slide.
- Modify the attributes of the **SMTP**, **NNTP**, and **ORGANIZATION** files in the **OBJECT/EMAIL** codefile using WFL **MODIFY**. See the documentation for instructions on how to do this.

After the global configuration is established, a small amount of configuration is required for each user who will be sending emails. At a minimum, the sender's email address must be defined to **OBJECT/EMAIL**. There are a number of other per-user options that can be specified, as we will discuss later.



*INSTALLATION/OPTIONS File

- Global email configuration
  - **SEQDATA** filekind
  - Stored on same family as **OBJECT/EMAIL**
- Options
  - **EMAIL MAILSERVER =** <server address>
  - **EMAIL ALTMAILSERVER =** <server address>, …
  - **EMAIL NEWSSERVER =** <server address>
  - **EMAIL ALTNEWSSERVER =** <server address>, …
  - **EMAIL HOSTCCS =** <ccs mnemonic>
  - **EMAIL SERVERCCS =** <ccs mnemonic>
  - ***COMPANY**, ***DEPARTMENT**, ***BUILDING** (NNTP only)
  - **EMAIL:** <send option>

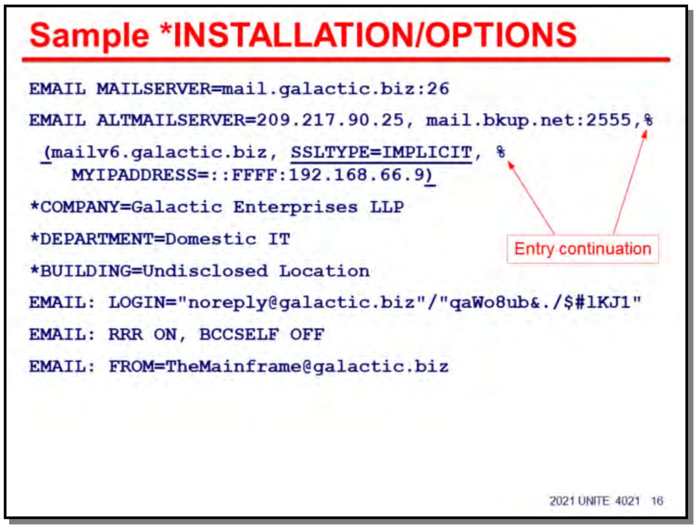
Note the ":"

2021 UNITE 4021 15

The **`*INSTALLATION/OPTIONS`** file supplies global configuration parameters to **`OBJECT/EMAIL`**. It must have a filekind of **`SEQDATA`** and be stored on the same disk family as the **`OBJECT/EMAIL`** codefile.

The configuration data consists of name/value pairs, one per record in the file. A configuration entry can be continued across records by placing a "**`%`**" as the last non-blank character on each line of the entry except the last. The entries in this file used by **`OBJECT/EMAIL`** are:

- **`EMAIL MAILSERVER`** – specifies the primary SMTP server (MTA) to which the utility will send the email messages it composes. This entry is required. It may specify an IP address or domain name.
- **`EMAIL ALTMAILSERVER`** – specifies a list of SMTP servers to be used as alternate MTAs if the primary one is not available.
- **`EMAIL NEWSSERVER`** – specifies the primary NNTP news server to which the utility will send news group postings.
- **`EMAIL ALTNEWSSERVER`** – specifies a list of alternate news servers to be used if the primary one is not available.
- **`EMAIL HOSTCCS`** – overrides the default internal character set (the character set used by MCP applications).
- **`EMAIL SERVERCCS`** – overrides the default character set for outgoing messages (those send to the MTA).
- **`*COMPANY`**, **`*DEPARTMENT`**, **`*BUILDING`** – specify descriptive strings to be included in outgoing message headers. These apply to NNTP news group postings only.
- **`EMAIL:`** <send option> – specifies default message sending options. Sending options will be discussed later in the section on the program's parameter string syntax. Note the use of a colon (**`:`**) instead of an equal sign (**`=`**) in this entry.



Sample *INSTALLATION/OPTIONS

```
EMAIL MAILSERVER=mail.galactic.biz:26
EMAIL ALTMAILSERVER=209.217.90.25, mail.bkup.net:2555,%
 (mailv6.galactic.biz, SSLTYPE=IMPLICIT, %
   MYIPADDRESS=::FFFF:192.168.66.9)
*COMPANY=Galactic Enterprises LLP
*DEPARTMENT=Domestic IT
*BUILDING=Undisclosed Location
EMAIL: LOGIN="noreply@galactic.biz"/"qaWo8ub&./$#1KJ1"
EMAIL: RRR ON, BCCSELF OFF
EMAIL: FROM=TheMainframe@galactic.biz
```

This example shows a somewhat contrived global configuration file. The only required entry is the one for **`MAILSERVER`**. Note the use of a percent sign (**`%`**) to indicate continuation of an entry across records of the file.

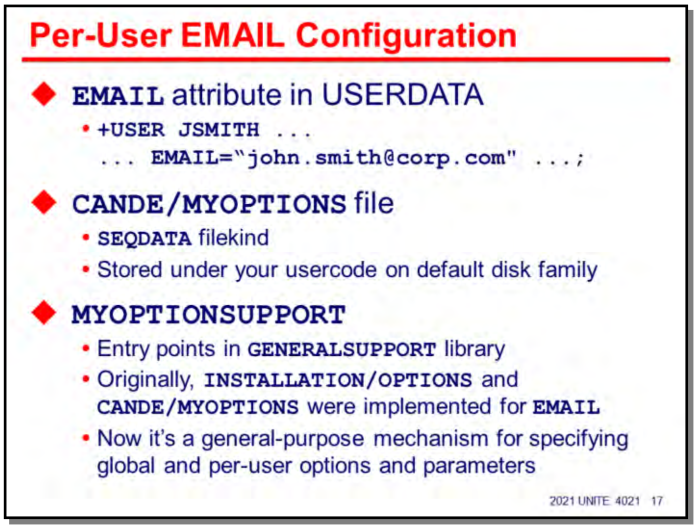
A server address can be specified as an IP address or a domain name. It can be optionally followed by a colon and a number to indicate the MTA server is listening on a port other than the default SMTP port 25. By enclosing a server address in parentheses, you can include additional attributes for the connection. At present the additional attributes supported are:

- **`MYIPADDRESS`**, which specifies the network interface on the MCP system through which connections to the MTA will be made.
- **SSLTYPE**, with values **OFF** and **IMPLICIT**.
- **SECALLOWSELFSIGNED** with values **TRUE** and **FALSE**. Setting this to **TRUE** allows **OBJECT/EMAIL** to accept SSL connections involving a self-signed certificate.

The example above shows SSL/TLS being enabled and an IPv6 address being used with the **`MYIPADDRESS`** attribute.

Also note the **`EMAIL:`** entries specifying global sending options. Note the colon (“**`:`**”). These options will be applied by default to all messages send by **`OBJECT/EMAIL`** unless they are explicitly overridden by user-level configuration or the send options in an individual message. As mentioned earlier, these options will be discussed later in the section on parameter syntax.

Of particular importance is the **`EMAIL:LOGIN`** option, which supplies log-in credentials to the email server. If either the user name or password contains special characters, those values must be enclosed in either single- or double-quotation marks. Note that “**`@`**” in user names is considered to be a special character.



Per-User EMAIL Configuration

- **EMAIL** attribute in USERDATA
  - `+USER JSMITH ...`
    `... EMAIL="john.smith@corp.com" ...;`
- **CANDE/MYOPTIONS** file
  - **SEQDATA** filekind
  - Stored under your usercode on default disk family
- **MYOPTIONSUPPORT**
  - Entry points in **GENERALSUPPORT** library
  - Originally, **INSTALLATION/OPTIONS** and **CANDE/MYOPTIONS** were implemented for **EMAIL**
  - Now it's a general-purpose mechanism for specifying global and per-user options and parameters

2021 UNITE 4021 17

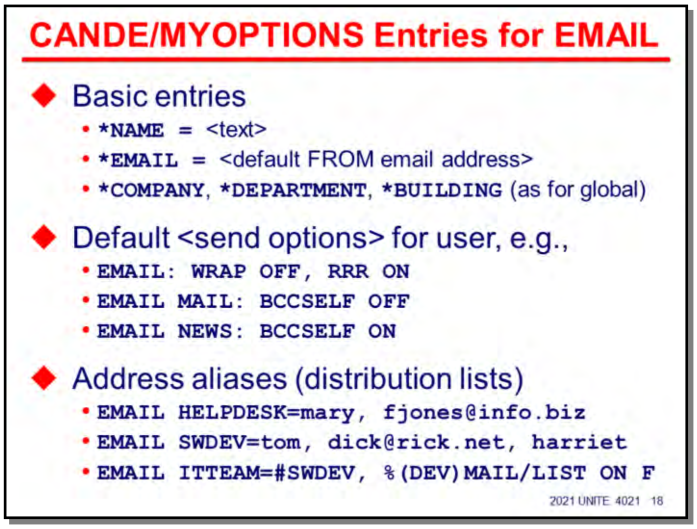
Having dealt with global configuration for **OBJECT/EMAIL**, we now turn to the configuration for individual users. Since the utility runs within the MCP environment, a "user" is identified by an MCP usercode.

At a minimum, each user must be associated with an email address. This is used to annotate outbound messages with a **From:** header. There are two ways to make this association:

- By establishing the **EMAIL** attribute in the user's **USERDATA** record. This is the only bit of email configuration data that is stored in a standard **USERDATAFILE**.
- As a ***EMAIL** entry in the user's optional **CANDE/MYOPTIONS** file. Like the **CANDE/STARTUP** file, this file (if it exists) must reside under the user's usercode on their default disk family. **OBJECT/EMAIL** automatically looks for and processes the entries in **CANDE/MYOPTIONS** whenever it sends a message for a user.

There are a number of other entries that can be established in the **CANDE/MYOPTIONS** file for email and other uses. We will discuss the email and news options over the next several slides.

This file was initially introduced to provide per-user configuration options for **OBJECT/EMAIL**, but a few releases ago was made more general purpose through a series of entry points in the **GENERALSUPPORT** library collectively referred to as **MYOPTIONSUPPORT**. You can use this facility to add your own configuration items to **CANDE/MYOPTIONS** and retrieve the data without having to parse the file yourself by calling the **MYOPTIONSUPPORT** entry points. The key **EMAIL** is reserved for use by the MCP, but you can create entries that use other keys. The capabilities and use of **MYOPTIONSUPPORT** are discussed in the section on **GENERALSUPPORT** in Appendix A of the *System Software Utilities Operations Reference Manual*.



CANDE/MYOPTIONS Entries for EMAIL

- Basic entries
  - ***NAME = <text>**
  - ***EMAIL = <default FROM email address>**
  - ***COMPANY, *DEPARTMENT, *BUILDING** (as for global)
- Default <send options> for user, e.g.,
  - **EMAIL: WRAP OFF, RRR ON**
  - **EMAIL MAIL: BCCSELF OFF**
  - **EMAIL NEWS: BCCSELF ON**
- Address aliases (distribution lists)
  - **EMAIL HELPDESK=mary, fjones@info.biz**
  - **EMAIL SWDEV=tom, dick@rick.net, harriet**
  - **EMAIL ITTEAM=#SWDEV, %(DEV)MAIL/LIST ON F**

2021 UNITE 4021 18

The two basic entries you should put in your **CANDE/MYOPTIONS** file are

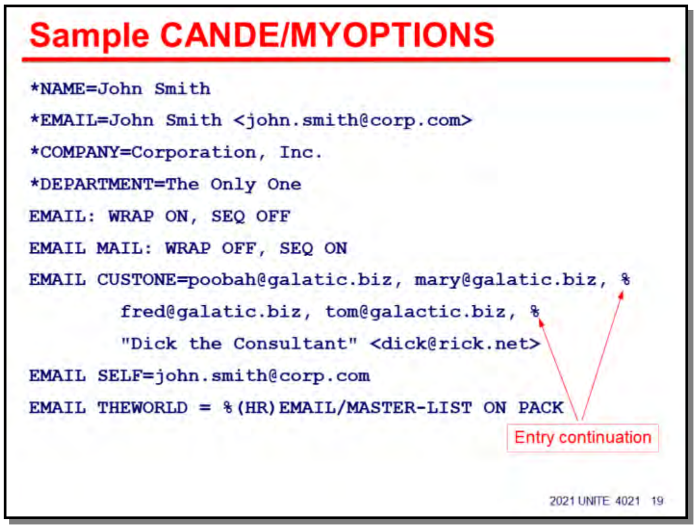
- ***NAME =** <some text intended to be your display name>
- ***EMAIL =** <your default From: email address>

You can also include ***COMPANY**, ***DEPARTMENT**, and ***BUILDING** entries for NNTP news submissions as discussed with the global options.

A second class of entries you can make represent default sending options for your messages. These options will override any specified in the global configuration file, and in turn may be overridden by options specified in the parameter string to **OBJECT/EMAIL**. As shown on the slide, you simply write the key **EMAIL** followed by a colon (**:**) and then include the options in a comma-delimited list. You can have multiple entries for the same key.

You can qualify entries so that they will apply either to email messages or NNTP news group submissions by following the **EMAIL** keyword with either **MAIL** or **NEWS**. The options specified on this type of entry will apply only to that type of message in **OBJECT/EMAIL**.

A third class of entries you can make in **CANDE/MYOPTIONS** are address aliases, also termed distribution lists. To define an alias, enter the **EMAIL** keyword followed by the name you wish to assign to the alias and then an equal sign (**=**). After that, enter a comma-delimited list of email references. We will discuss these email references in detail when discussing parameter syntax, but in brief, these references can be an email address in the standard name@host.domain form, an MCP usercode, another alias name prefixed by a crosshatch (**#**), or the title of a file prefixed by a percent-sign (**%**). Thus, aliases can be defined recursively, with one alias including the email addresses defined by another alias. There are some other options for defining these email reference lists, which will be discussed shortly.



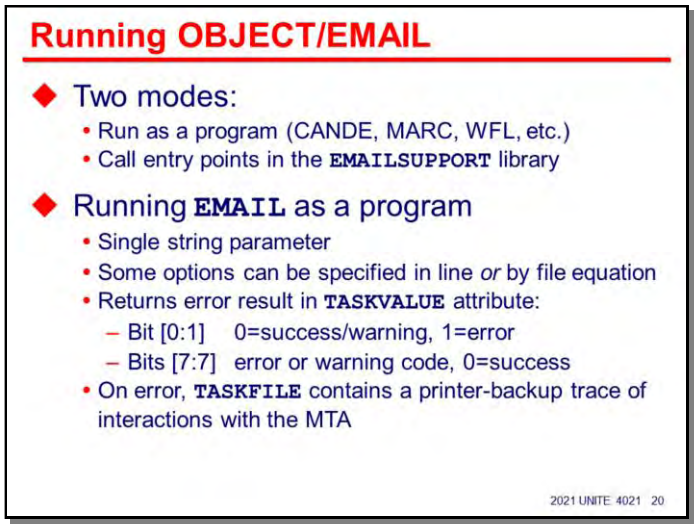
Sample CANDE/MYOPTIONS

```
*NAME=John Smith
*EMAIL=John Smith <john.smith@corp.com>
*COMPANY=Corporation, Inc.
*DEPARTMENT=The Only One
EMAIL: WRAP ON, SEQ OFF
EMAIL MAIL: WRAP OFF, SEQ ON
EMAIL CUSTONE=poobah@galatic.biz, mary@galatic.biz, %
        fred@galatic.biz, tom@galactic.biz, %
        "Dick the Consultant" <dick@rick.net>
EMAIL SELF=john.smith@corp.com
EMAIL THEWORLD = %(HR)EMAIL/MASTER-LIST ON PACK
```

Entry continuation

This slide shows a sample **CANDE/MYOPTIONS** file. It includes

- Some identifying entries for the user.
- A couple of default send options, one that applies to all messages, and one that only applies to email messages.
- Three alias definitions, **CUSTONE**, **SELF**, and **THEWORLD**. Note that, as with the global configuration file, entries can be continued across multiple records using a "**%**" at the end of all but the last record for an entry.



Running OBJECT/EMAIL

- Two modes:
  - Run as a program (CANDE, MARC, WFL, etc.)
  - Call entry points in the **`EMAILSUPPORT`** library
- Running **`EMAIL`** as a program
  - Single string parameter
  - Some options can be specified in line *or* by file equation
  - Returns error result in **`TASKVALUE`** attribute:
    - Bit [0:1] 0=success/warning, 1=error
    - Bits [7:7] error or warning code, 0=success
  - On error, **`TASKFILE`** contains a printer-backup trace of interactions with the MTA

2021 UNITE 4021 20

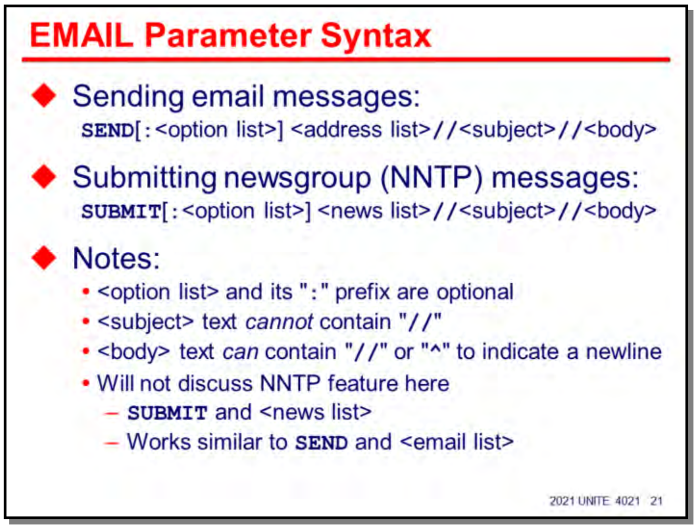
That concludes the discussion on configuring **`OBJECT/EMAIL`**. The next topic concerns running the program. As mentioned earlier, **`OBJECT/EMAIL`** has two modes – it can be run as a standard program (task) or it can be called as a library with a function name of **`EMAILSUPPORT`**. We will first discuss running the utility as a program.

When run as a program, **`OBJECT/EMAIL`** requires a single string parameter. All options for the program can be specified through the parameter string. Some options can be activated by equating files to certain **`INTNAME`**s. The parameter syntax and file equation options will be discussed over the next several slides.

`OBJECT/EMAIL` returns a result code in its **`TASKVALUE`** attribute. This result word consists of two bit fields:

- Bit [0:1] indicates whether there was an error. A value of zero indicates the run was successful or generated a warning. A value of one indicates there was an error that prevented the message from being sent. Note that an indication of success or error concerns the program's ability to contact and deliver the message to the MTA, not to the ultimate addressees. Delivery to the addressees is the MTA's job.
- Bits [7:7] indicate an error code. If this code is zero (and bit 0 is zero), the run completed successfully. If bit 0 is zero and this field is non-zero, the field indicates a warning code. If bit 0 is a one, this field indicates an error code. The codes are defined in the documentation.

In addition to this result word, **`OBJECT/EMAIL`** writes to its **`TASKFILE`** a printed trace of the network interactions it has with its MTA. After a successful send, this file is purged unless the **`TRACE`** option is set. If there is an error, you can use this trace data to help understand what went wrong.



EMAIL Parameter Syntax

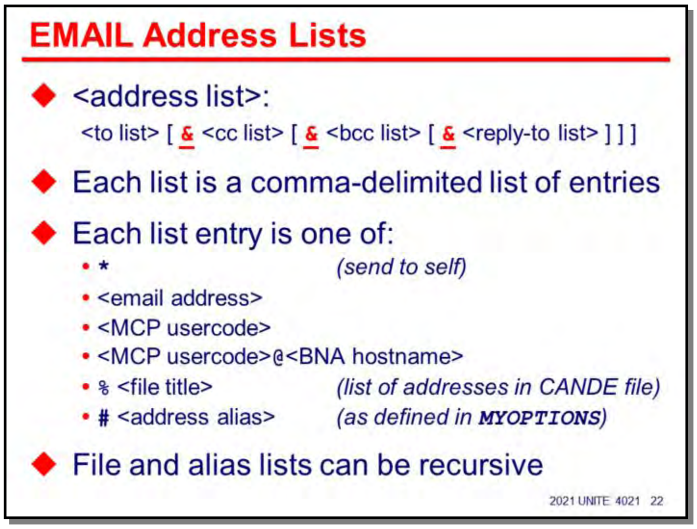
- Sending email messages:
  `SEND[:<option list>] <address list>//<subject>//<body>`
- Submitting newsgroup (NNTP) messages:
  `SUBMIT[:<option list>] <news list>//<subject>//<body>`
- Notes:
  - <option list> and its ":" prefix are optional
  - <subject> text *cannot* contain "//"
  - <body> text *can* contain "//" or "^" to indicate a newline
  - Will not discuss NNTP feature here
    - **SUBMIT** and <news list>
    - Works similar to **SEND** and <email list>

2021 UNITE 4021 21

The basic command syntax for **OBJECT/EMAIL** came almost directly from the old A Series Mail product. The parameter string for the program has five main parts:

- A command keyword of **SEND** or **SUBMIT**.
  - **SEND** indicates you are sending an email message.
  - **SUBMIT** indicates you are submitting a news group posting.
- An optional list of sending options. If present, these options must be preceded by a colon (**:**). The options are written in a comma-delimited list. Spaces may be present around the commas and between tokens in the option syntax.
- A list of email addresses, or email references that eventually resolve to email addresses. For the **SUBMIT** command, this list takes a slightly different form, as submissions are made to news groups rather than email addresses.
- The subject line of the message, preceded by a double-slash (**//**). The text of this line may not contain a double-slash, as that is what delimits it from the next part.
- The body of the message, again preceded by a double-slash (**//**). This part (and its preceding double-slash) can be omitted if desired. The body can be obtained from a file, as will be discussed shortly. The body text *can* contain double-slashes. If present these are translated into carriage-return/line-feed sequences in the outgoing message. A caret (**^**) in the body text will have the same effect.

We will not delve into news groups and the **SUBMIT** command any further in this presentation. Except for a difference in the way that messages are addressed, it works similarly to the **SEND** command.



EMAIL Address Lists

- <address list>:
  <to list> [& <cc list> [& <bcc list> [& <reply-to list>]]]
- Each list is a comma-delimited list of entries
- Each list entry is one of:
  - * *(send to self)*
  - <email address>
  - <MCP usercode>
  - <MCP usercode>@<BNA hostname>
  - % <file title> *(list of addresses in CANDE file)*
  - # <address alias> *(as defined in **MYOPTIONS**)*
- File and alias lists can be recursive

2021 UNITE 4021 22

Perhaps the most important part of the parameter syntax is that which routes the message to recipient addresses. The list of addresses is actually four lists, separated by ampersands (**&**).

- The first list is for **To:** addresses.
- The second list is for **CC:** addresses.
- The third list is for **BCC:** (blind copy) addresses.
- The fourth list is for **Reply-to:** addresses. This defines the default list of addresses to which a reply to the message will be sent by a recipient.

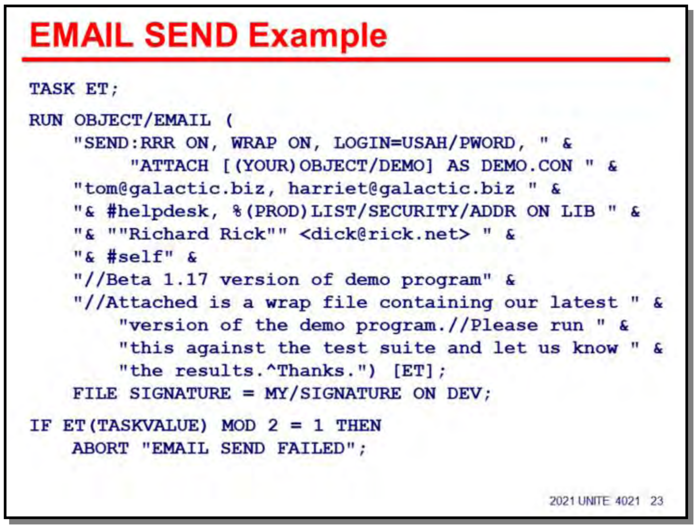
All lists are optional (i.e., they may be empty), but the ampersands are necessary to delimit one list from another. The ampersands for any trailing empty lists may be omitted. Thus, a message with only To: and CC: addresses would only require one ampersand, the one separating those two lists.

Each of these four lists is composed of comma-delimited email reference entries.

Each reference entry in the list can be one of the following:

- An asterisk (*****). This implies the message will be sent to the sending user, provided an email address is configured in their **USERDATA** record.
- A standard email address, which can have one of these forms (the "**<**" and "**>**" are literal in this case):
  - name@host.domain
  - Display Name **<**name@host.domain**>**
  - **"**Display Name**"** **<**name@host.domain**>**
- An MCP usercode. The message will be sent to the email address configured in that user's **USERDATA** record.
- An MCP usercode at a BNA hostname. The message will be sent to the email address configured for the specified user at the specified BNA host.
- A percent sign (**%**) followed by an MCP file title. This is a form of distribution list. The file contains a list of email addresses. Multiple addresses in one record must be separated by commas. The file can be of any CANDE-editable filekind.
- A crosshatch (**#**) followed by an alias name, which is another form of distribution list. The alias must have been defined previously in the user's **CANDE/MYOPTIONS** file or the global ***INSTALLATION/OPTIONS** file.

Note that the **%**<file title> and **#**<alias> distribution lists are fully recursive. Files may reference other files and aliases. Aliases may reference files and other aliases.



EMAIL SEND Example

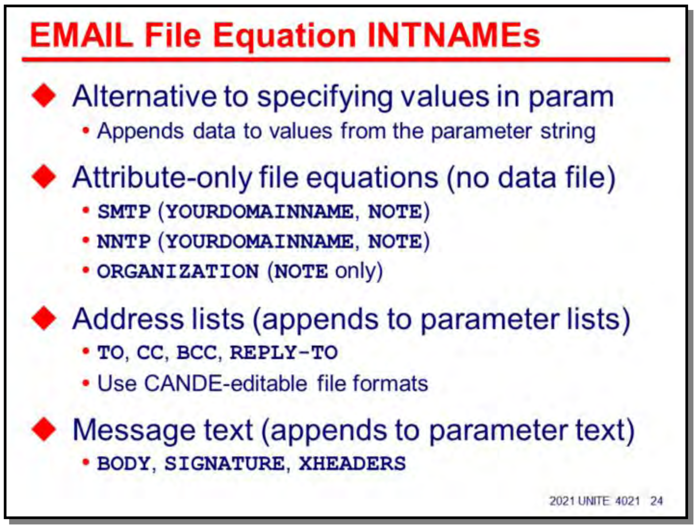
```
TASK ET;
RUN OBJECT/EMAIL (
    "SEND:RRR ON, WRAP ON, LOGIN=USAH/PWORD, " &
         "ATTACH [(YOUR)OBJECT/DEMO] AS DEMO.CON " &
    "tom@galactic.biz, harriet@galactic.biz " &
    "& #helpdesk, %(PROD)LIST/SECURITY/ADDR ON LIB " &
    "& ""Richard Rick"" <dick@rick.net> " &
    "& #self" &
    "//Beta 1.17 version of demo program" &
    "//Attached is a wrap file containing our latest " &
        "version of the demo program.//Please run " &
        "this against the test suite and let us know " &
        "the results.^Thanks.") [ET];
    FILE SIGNATURE = MY/SIGNATURE ON DEV;
IF ET(TASKVALUE) MOD 2 = 1 THEN
    ABORT "EMAIL SEND FAILED";
```

2021 UNITE 4021 23

This slide shows the parts of a fairly full parameter list.

- The **SEND** command indicates we are sending an email message rather than a news group submission.
- The colon (**:**) following **SEND** introduces an option list, which continues for two lines, until a comma delimiter is no longer encountered.
- The next line is the To: list with two email address entries.
- The next line (beginning with **&**) is the CC: list, also with two entries, an alias and a file distribution list.
- The next line (beginning with the second **&**) is the BCC: list, which has one email address.
- The next line (beginning with the third **&**) is the Reply-To: list, containing a single alias entry.
- The next line (beginning with **//**) is the subject line for the message.
- The remainder of the parameter (beginning with the second **//**) is the body of the message. Note that it contains another **//** and a **^** to indicate embedded newlines.
- The file equation on the **SIGNATURE INTNAME** copies the contents of the specified file at the end of the body as a signature block.

The WFL **IF** statement at the end isolates bit [0:1] of the **TASKVALUE** result and checks it for an error condition.



EMAIL File Equation INTNAMEs

- Alternative to specifying values in param
  - Appends data to values from the parameter string
- Attribute-only file equations (no data file)
  - **SMTP** (**YOURDOMAINNAME**, **NOTE**)
  - **NNTP** (**YOURDOMAINNAME**, **NOTE**)
  - **ORGANIZATION** (**NOTE** only)
- Address lists (appends to parameter lists)
  - **TO**, **CC**, **BCC**, **REPLY-TO**
  - Use CANDE-editable file formats
- Message text (appends to parameter text)
  - **BODY**, **SIGNATURE**, **XHEADERS**

2021 UNITE 4021 24

OBJECT/EMAIL has a number of files that can be equated at run time to supply additional data for use in composing a message. All of these files can be specified in the parameter string syntax, but file equation is often more convenient, especially when running the utility program from WFL jobs.

There are three **INTNAMES** which can be file-equated simply to supply parameters to the program. These internal files are never used to open a physical file.

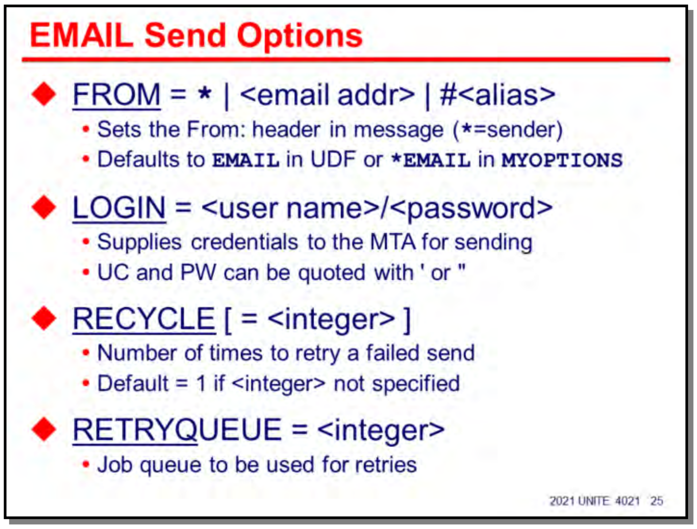
- **SMTP** uses the **YOURDOMAINNAME** and **NOTE** attributes. **YOURDOMAINNAME** overrides the primary email server address in the global configuration file. **NOTE** overrides the list of alternate email servers.
- **NNTP** uses **YOURDOMAINNAME** and **NOTE** in the same fashion, but these attributes provide overrides for the primary and alternate news servers.
- **ORGANIZATION** uses only the **NOTE** attribute to supply an override ***ORGANIZATION** string for news group submissions.

OBJECT/EMAIL also supports file equation for each of the four mail address lists. These files have the same format as the **%**<file title> distribution lists, and can be prepared using any CANDE-editable filekind:

- **TO**
- **CC**
- **BCC**
- **REPLY-TO**

Finally, there are three files that can supply additional text used in composing the message:

- **BODY** supplies text for the body of the message. If the parameter string includes body text (after the second **//**), the contents of this file are simply appended to any body text contained in the parameter string.
- **SIGNATURE** supplies a signature block for the message. The text of this file is appended to the end of any body text.
- **XHEADERS** supplies additional "user" headers for the message. The records from this file are appended to the end of the message heading. This file is for advanced situations and is seldom used.



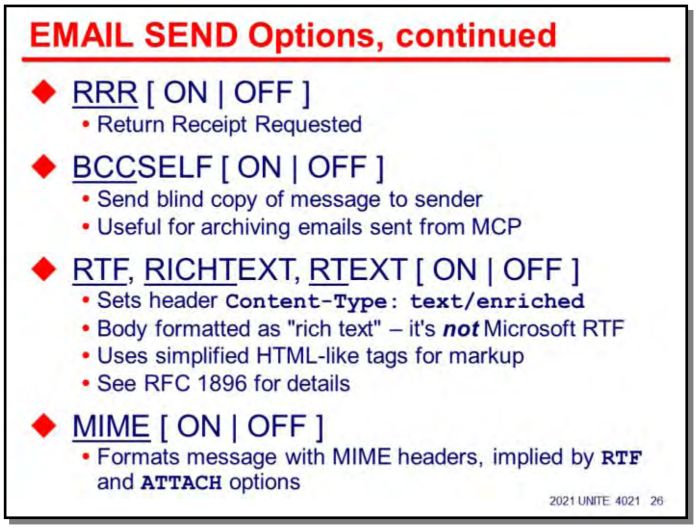
EMAIL Send Options

- FROM = * | <email addr> | #<alias>
  - Sets the From: header in message (*=sender)
  - Defaults to **EMAIL** in UDF or ***EMAIL** in **MYOPTIONS**
- LOGIN = <user name>/<password>
  - Supplies credentials to the MTA for sending
  - UC and PW can be quoted with ' or "
- RECYCLE [= <integer>]
  - Number of times to retry a failed send
  - Default = 1 if <integer> not specified
- RETRYQUEUE = <integer>
  - Job queue to be used for retries

2021 UNITE 4021 25

I have mentioned sending options several times thus far, and we have finally arrived at the point where we can talk about them in detail. These options can appear as defaults in the global and user configuration files. They can also be included in the parameter string as a comma-delimited list, preceded by a colon, following the **SEND** or **SUBMIT** command. There are almost 20 of these options, which I'll discuss over the next several slides.

- **FROM** provides an email address for the sender. This option overrides any default address in the user's **USERDATA** record or ***EMAIL** entry in **CANDE/MYOPTIONS**. It can be specified as
  - An asterisk (*****, meaning use the email address in the user's **USERDATA** record).
  - A standard email address
  - A **#**alias name.
- **LOGIN** provides user name and password credentials to access the email server (MTA) for sending messages. A default can be specified in the global or user configuration file. At present, there is no way to secure the password.
- **RECYCLE** indicates the number of times to retry a failed attempt to send a message. If the <integer> is not specified, the message will be retried once. The retry mechanism used by **OBJECT/EMAIL** will be discussed in more detail a little later in the presentation.
- **RETRYQUEUE** indicates the WFL queue to be used for retry attempts.



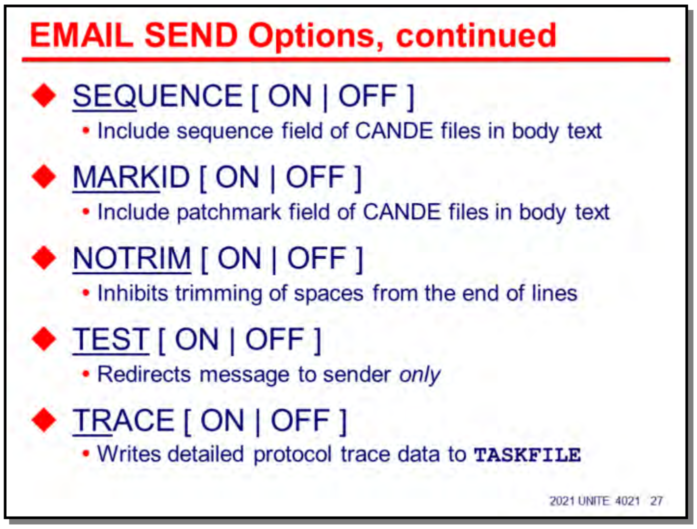
EMAIL SEND Options, continued

- RRR [ON | OFF]
  - Return Receipt Requested
- BCCSELF [ON | OFF]
  - Send blind copy of message to sender
  - Useful for archiving emails sent from MCP
- RTF, RICHTEXT, RTEXT [ON | OFF]
  - Sets header `Content-Type: text/enriched`
  - Body formatted as "rich text" – it's ***not*** Microsoft RTF
  - Uses simplified HTML-like tags for markup
  - See RFC 1896 for details
- MIME [ON | OFF]
  - Formats message with MIME headers, implied by **`RTF`** and **`ATTACH`** options

2021 UNITE 4021 26

Continuing the discussion of sending options…

- **`RRR`** places a header in the message that requests the receiving MDA or MUA to send a return receipt notice. Note that you can only request the receiver to send this notice. Some users configure their email clients not to return receipts.
- **`BCCSELF`** instructs **`OBJECT/EMAIL`** to add the sender's email address to the message's BCC: list. Since **`OBJECT/EMAIL`** does not maintain message folders, it does not retain any record of having sent a message. This option allows you to send a copy to your standard mailbox, from which presumably you will archive it.
- **`RTF`**, **`RICHTEXT`**, and **`RTEXT`** are all synonyms for an option that will cause the message to be sent as "rich text." Note that this is *not* the same as Microsoft's RTF for Office documents. This form is a simplified HTML-like markup notation that is defined in RFC 1896. You must compose the message using the markup yourself. All that this option does is indicate the receiver is to interpret the body according to RFC 1896 by including a **`Content-Type:`** header of "**`text/enriched`**" in the message.
- **`MIME`** indicates that the message is to be formatted with MIME (Multi-purpose Internet Message Extensions) headers. This option is set automatically when you specify the **`RTF`** or **`ATTACH`** options.
  - With MIME disabled, no Content-Type: header will be inserted into the message by **`OBJECT/EMAIL`**, and most email clients will interpret the body of the message as plain text
  - With MIME enabled, the program will examine the type of data in the body (particularly its file name extension) and insert an appropriate Content-Type: header in the message. **`OBJECT/EMAIL`** understands some common extensions, such as **`.TXT`**, **`.CSV`**, **`.RTF`** (for RFC 1896 enriched text), **`.HTM`**, **`.HTML`**, and the symbol file extensions used with Client Access Services.



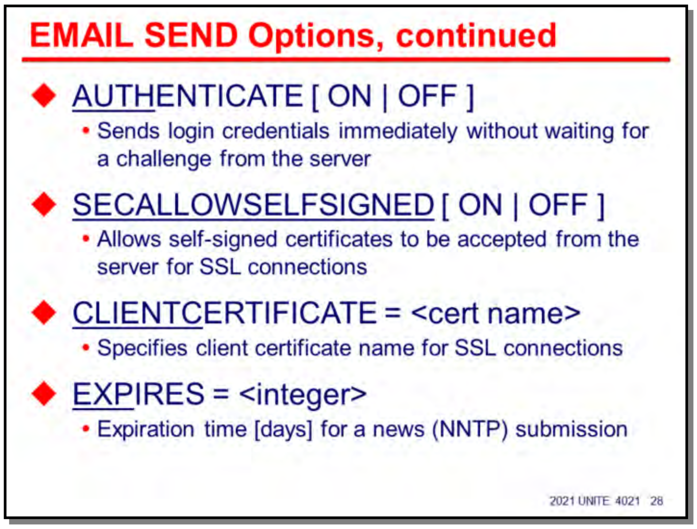
EMAIL SEND Options, continued

- SEQUENCE [ON | OFF]
  - Include sequence field of CANDE files in body text
- MARKID [ON | OFF]
  - Include patchmark field of CANDE files in body text
- NOTRIM [ON | OFF]
  - Inhibits trimming of spaces from the end of lines
- TEST [ON | OFF]
  - Redirects message to sender *only*
- TRACE [ON | OFF]
  - Writes detailed protocol trace data to **TASKFILE**

2021 UNITE 4021 27

Continuing the discussion of sending options…

- **SEQUENCE** indicates, when equating a file with CANDE-style sequence numbers for the body, whether the sequence numbers are to be included in the message.
- **MARKID** similarly indicates whether the mark field of a file used for the body should include the patchmark data.
- **TEST** is useful for testing message composition. It ignores the address lists and sends the message only to the sending user's email address.
- **TRACE** causes the detailed trace of message traffic between **OBJECT/EMAIL** and the MTA to be saved when the program terminates. This trace is always created and written to the program's **TASKFILE**, but by default it is purged upon a successful send or on any retry attempts after an unsuccessful send.



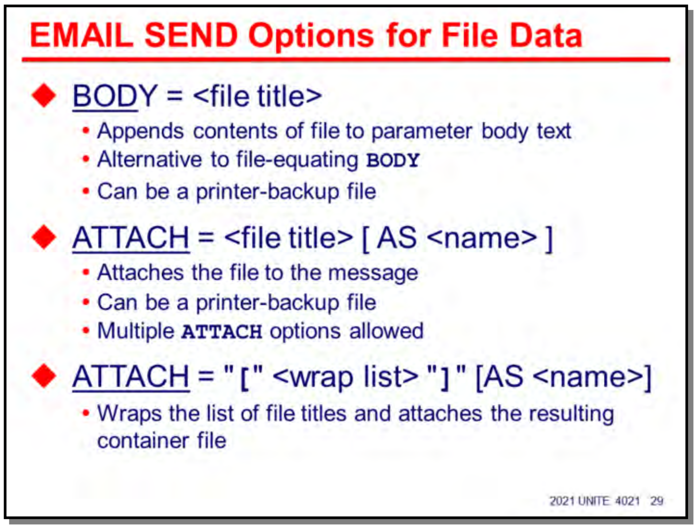
EMAIL SEND Options, continued

- AUTHENTICATE [ON | OFF]
  - Sends login credentials immediately without waiting for a challenge from the server
- SECALLOWSELFSIGNED [ON | OFF]
  - Allows self-signed certificates to be accepted from the server for SSL connections
- CLIENTCERTIFICATE = <cert name>
  - Specifies client certificate name for SSL connections
- EXPIRES = <integer>
  - Expiration time [days] for a news (NNTP) submission

2021 UNITE 4021 28

Continuing the discussion of sending options…

- **`AUTHENTICATE`** controls whether **`OBJECT/EMAIL`** will unconditionally initiate an authentication sequence with the MTA. Normally, the authentication sequence is triggered by a challenge request sent by the server, but some MTAs do not send such challenges, and consequently may refuse the message. This appears to be particularly common when using SSL. Setting this option to **`ON`** will force **`OBJECT/EMAIL`** to send its **`LOGON`** credentials at the beginning of its dialog with the server.
- **`SECALLOWSELFSIGNED`** specifies whether self-signed certificates will be accepted from the MTA for SSL connections. Be default, **`OBJECT/EMAIL`** will not accept such certificates.
- **`CLIENTCERTIFICATE`** specifies the name of the client certificate to be used for SSL connections. If this is not specified, the default certificate will be used.
- **`EXPIRES`** indicates the number of days a message is to be retained by the news group server. It is used only with NNTP and the **`SUBMIT`** command.



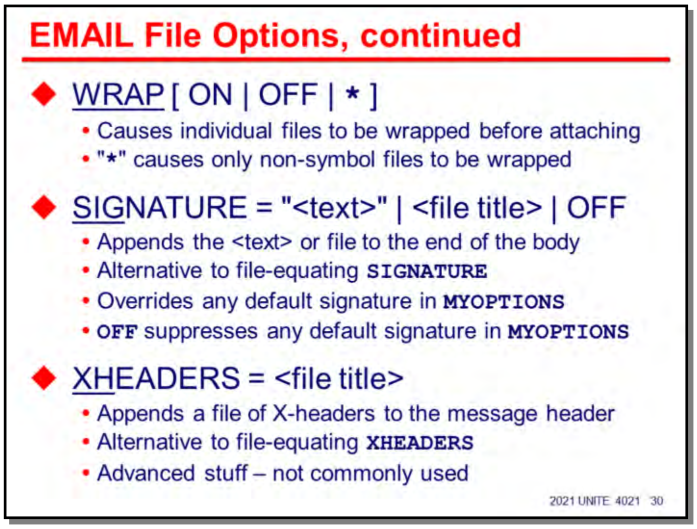
EMAIL SEND Options for File Data

- BODY = <file title>
  - Appends contents of file to parameter body text
  - Alternative to file-equating **BODY**
  - Can be a printer-backup file
- ATTACH = <file title> [AS <name>]
  - Attaches the file to the message
  - Can be a printer-backup file
  - Multiple **ATTACH** options allowed
- ATTACH = "[" <wrap list> "]" [AS <name>]
  - Wraps the list of file titles and attaches the resulting container file

2021 UNITE 4021 29

Continuing the discussion of sending options…

- **`BODY`** specifies a file that will be appended to the body text of the message. This option is an alternative to file-equating the **`BODY INTNAME`**. It is typically used when calling **`OBJECT/EMAIL`** library entry points, since file equation cannot be applied in that case. Note that **`OBJECT/EMAIL`** understands the format of an MCP printer-backup file and will translate such a file to standard text format.
- **`ATTACH`** specifies a file to be attached to the message. The file can be given a different name in the attachment using the AS clause. The format of the attachment varies depending on the **`WRAP`** option, discussed on the next slide. You can attach a printer-backup file and have it converted to standard text format as for **`BODY`**. Unlike the **`BODY`** option, however, you can have multiple **`ATTACH`** options in the option list; each one will attach its file to the message.
- A second form of **`ATTACH`** option will wrap one or more MCP files and attach the resulting container file to the message. This is indicated by enclosing the file name or list of names in square brackets ("**`[`**" and "**`]`**") and is useful if you want to send non-text MCP files, or send the file in such a way that all of its internal formatting and file attributes will be preserved when it is unwrapped back into the MCP environment. This form of attachment is unaffected by the **`WRAP`** option – it always wraps the files before attaching them.



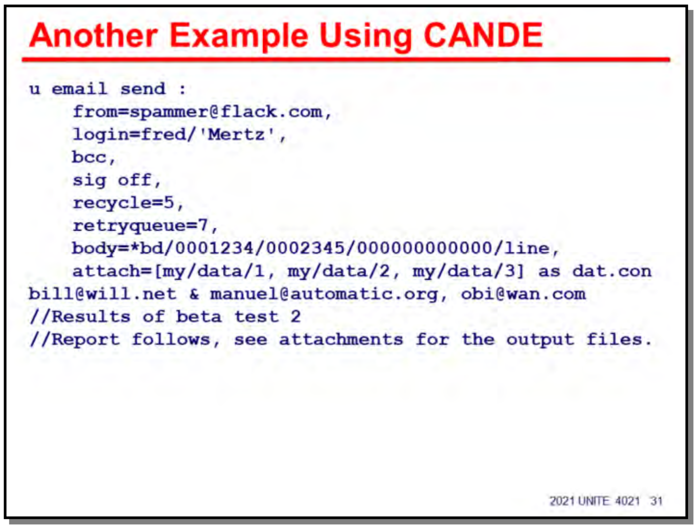
EMAIL File Options, continued

- WRAP [ON | OFF | *]
  - Causes individual files to be wrapped before attaching
  - "*" causes only non-symbol files to be wrapped
- SIGNATURE = "<text>" | <file title> | OFF
  - Appends the <text> or file to the end of the body
  - Alternative to file-equating **SIGNATURE**
  - Overrides any default signature in **MYOPTIONS**
  - **OFF** suppresses any default signature in **MYOPTIONS**
- XHEADERS = <file title>
  - Appends a file of X-headers to the message header
  - Alternative to file-equating **XHEADERS**
  - Advanced stuff – not commonly used

2021 UNITE 4021 30

Completing the discussion of sending options…

- **WRAP** indicates whether single-file attachments (those specified without enclosing square brackets) are to be attached as-is, or wrapped first and the wrapped result attached:
  - **ON** (or just specifying **WRAP** by itself) indicates all attachments are to be wrapped.
  - **OFF** indicates no attachments are to be wrapped.
  - ***** indicates that only non-symbol files are to be wrapped. Symbol files will be attached as plain text.
- **SIGNATURE** can be specified as an alternative to file-equating the **SIGNATURE INTNAME**, but has a couple of additional options.
  - Specifying a file name causes the contents of that file to be appended to the body as a signature block, as for the file-equation alternative.
  - Specifying a string of text causes that string to be appended to the body as the signature block
  - Specifying **OFF** indicates that no signature block is to be appended. This can be used to inhibit a default signature specified in the global or user configuration file.
- **XHEADERS** can be specified as an alternative to file-equating the **XHEADERS INTNAME**. The specified file supplies a list of additional headers that are appended to the heading portion of the email message.



Another Example Using CANDE

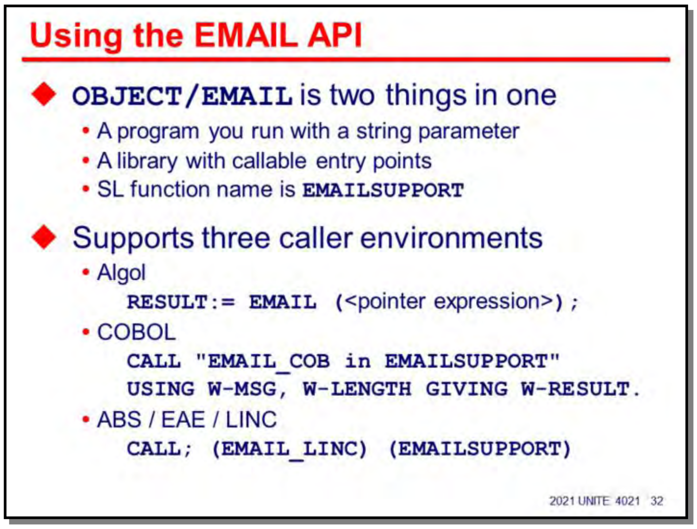
```
u email send :
    from=spammer@flack.com,
    login=fred/'Mertz',
    bcc,
    sig off,
    recycle=5,
    retryqueue=7,
    body=*bd/0001234/0002345/000000000000/line,
    attach=[my/data/1, my/data/2, my/data/3] as dat.con
bill@will.net & manuel@automatic.org, obi@wan.com
//Results of beta test 2
//Report follows, see attachments for the output files.
```

2021 UNITE 4021 31

This slide shows a sample execution of the program from CANDE that illustrates a number of the sending options. This example has been reformatted to highlight the individual options. Note the colon (**`:`**) following the **`SEND`** command. Also note that the keywords are case-insensitive in the parameter string syntax:

- **`FROM`** supplies a sender address that overrides any default that has been configured in **`USERDATA`** or the **`CANDE/MYOPTIONS`** file.
- **`LOGIN`** supplies credentials for the MTA server. Note that special characters and case sensitivity can be preserved by enclosing the password in single quotes (**`'`**).
- **`BCC`** indicates the program should add the sender's email address to the BCC: list. This is an abbreviation for **`BCCSELF`**. For the on/off options, **`ON`** is implied if neither value is specified.
- **`SIG OFF`** indicates no signature block is to be composed, even if a default signature has been specified in one of the configuration files.
- **`RECYCLE`** indicates that **`OBJECT/EMAIL`** is to retry sending the message up to five times.
- **`RETRYQUEUE`** indicates that any retry attempts are to be run from the specified WFL queue.
- **`BODY`** specifies a file to be appended to the body text specified at the end of the parameter string. Presumably this is an MCP printer-backup file, so it will be translated to standard text format.
- **`ATTACH`** specifies that the three files in the bracketed list are to be wrapped into a container and that container file attached to the message with a name of "**`dat.con`**".

Since the final **`ATTACH`** option is not followed by a comma, it terminates the list of options, and the next token in the parameter string is the start of the To: list of email addresses.



Using the EMAIL API

- **OBJECT/EMAIL** is two things in one
  - A program you run with a string parameter
  - A library with callable entry points
  - SL function name is **EMAILSUPPORT**
- Supports three caller environments
  - Algol

```
RESULT:= EMAIL (<pointer expression>);
```

  - COBOL

```
CALL "EMAIL_COB in EMAILSUPPORT"
USING W-MSG, W-LENGTH GIVING W-RESULT.
```

  - ABS / EAE / LINC

```
CALL; (EMAIL_LINC) (EMAILSUPPORT)
```

2021 UNITE 4021 32

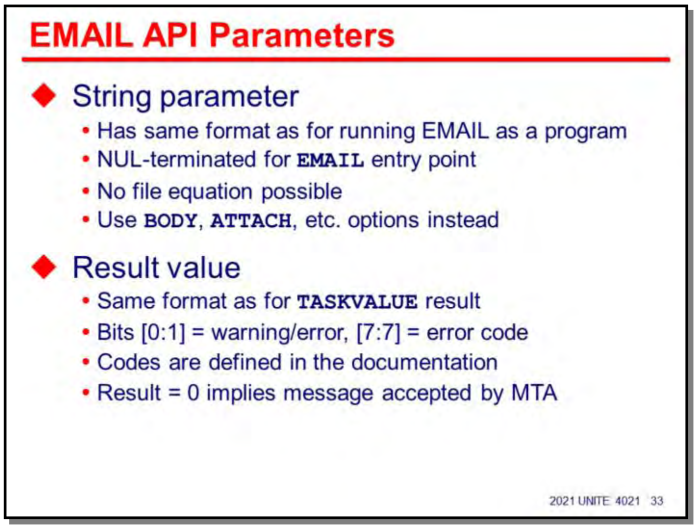
Thus far, we have been discussing the use of **OBJECT/EMAIL** as a utility program you run from WFL, CANDE, or MARC. You run the program, pass it a string parameter, possibly apply some file equation, then it generates the message and sends it to the specified MTA.

OBJECT/EMAIL also functions as a library, and all of the things you can do with it as a running program (except the file equation) can be done by calling one of the library entry points. Simple Install defines the **OBJECT/EMAIL** codefile as a system library (SL) with the function name **EMAILSUPPORT**.

When used as a library, the program has three entry points, each designed for a different language environment.

- **EMAIL** is designed for Algol. It takes a single pointer expression as a parameter. The pointer expression references a parameter string in the same format as discussed previously. The program expects the string to be terminated with a **NUL** (hex 00) character. The entry point returns an integer value in the same format at the **TASKVALUE** result, with an error flag in bit [0:1] and an error code in bits [7:7].
- **EMAIL_COB** is designed for COBOL. It takes two parameters, the name of an 01-level record area, and an integer data item of **USAGE BINARY**. The record area contains the parameter string and the integer item indicates the length of the string. The result value (which must also be an integer with **USAGE BINARY**) is the same as the **TASKVALUE** result.
- **EMAIL_LINC** is designed for the LINC/EAE/ABS environment. The message is in **GLB.PARAM**. The result code is returned in the first three characters of **GLB.PARAM** and any warning code in the second three characters.

The documentation shows examples for each of these entry points.



EMAIL API Parameters

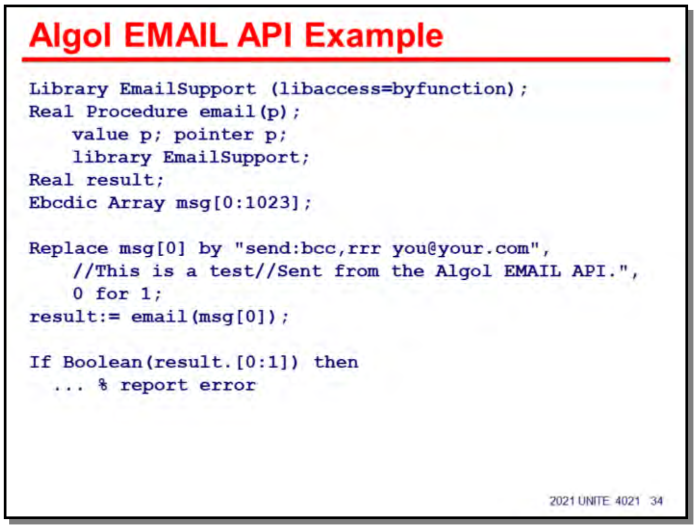
- String parameter
  - Has same format as for running EMAIL as a program
  - NUL-terminated for **EMAIL** entry point
  - No file equation possible
  - Use **BODY**, **ATTACH**, etc. options instead
- Result value
  - Same format as for **TASKVALUE** result
  - Bits [0:1] = warning/error, [7:7] = error code
  - Codes are defined in the documentation
  - Result = 0 implies message accepted by MTA

2021 UNITE 4021 33

All of the **OBJECT/EMAIL** library entry points take a string parameter that has the same format as the one used when running the codefile as a program. Since no file equation is possible with library calls, you must use the <option list> equivalents, **BODY=**, **ATTACH=**, **SIGNATURE=**, etc.

All of the entry points return a result value. For COBOL and Algol, this result is an integer with the same bit-field format as the **TASKVALUE** result discussed earlier for the program. The error/warning codes are defined in the Unisys documentation.

Note that a successful result of zero implies only that the message was accepted for delivery by the MTA. As with all email, actual delivery to the recipient may take place a significant amount of time later. There is no way within the MCP environment to know that a message did or did not reach the intended destination. Since **OBJECT/EMAIL** is a send-only MUA, the From: or Reply-To: address in the message should refer to a "real" mailbox where return receipts and bounce notices can by sent by the MTA.

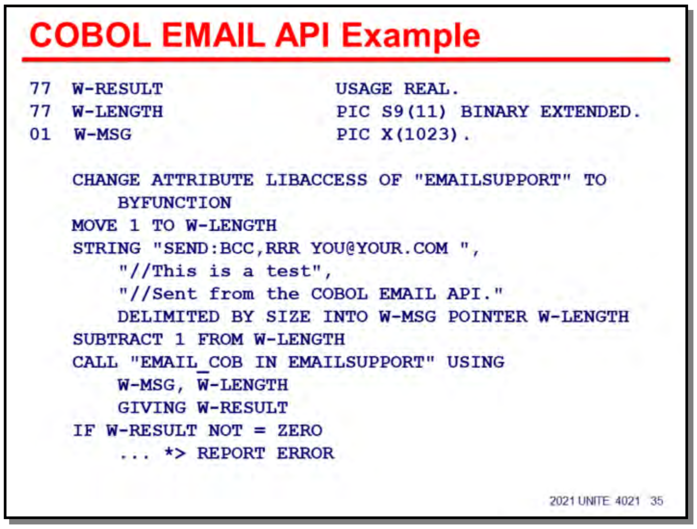


Algol EMAIL API Example

```
Library EmailSupport (libaccess=byfunction);
Real Procedure email(p);
    value p; pointer p;
    library EmailSupport;
Real result;
Ebcdic Array msg[0:1023];

Replace msg[0] by "send:bcc,rrr you@your.com",
    //This is a test//Sent from the Algol EMAIL API.",
    0 for 1;
result:= email(msg[0]);

If Boolean(result.[0:1]) then
  ... % report error
```

2021 UNITE 4021 34

This slide shows a simple example of the Algol **`EMAIL`** entry point. Note that the parameter string must be terminated by a **`NUL`** (hex 00) character.

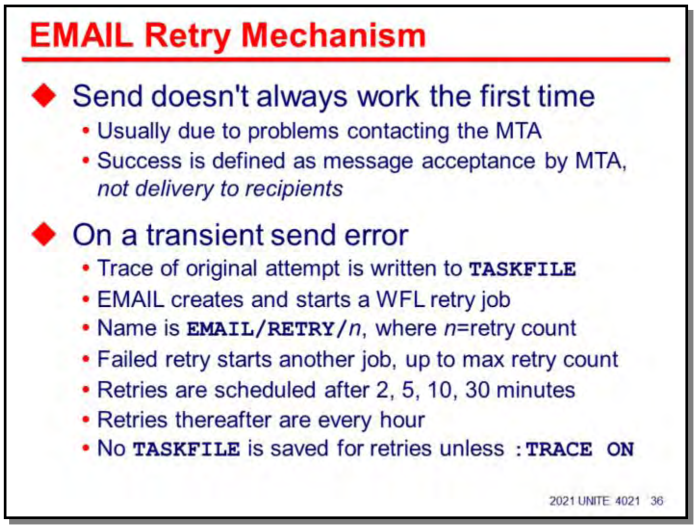


COBOL EMAIL API Example

```
77  W-RESULT               USAGE REAL.
77  W-LENGTH               PIC S9(11) BINARY EXTENDED.
01  W-MSG                  PIC X(1023).

    CHANGE ATTRIBUTE LIBACCESS OF "EMAILSUPPORT" TO
        BYFUNCTION
    MOVE 1 TO W-LENGTH
    STRING "SEND:BCC,RRR YOU@YOUR.COM ",
        "//This is a test",
        "//Sent from the COBOL EMAIL API."
        DELIMITED BY SIZE INTO W-MSG POINTER W-LENGTH
    SUBTRACT 1 FROM W-LENGTH
    CALL "EMAIL_COB IN EMAILSUPPORT" USING
        W-MSG, W-LENGTH
        GIVING W-RESULT
    IF W-RESULT NOT = ZERO
        ... *> REPORT ERROR
```

2021 UNITE 4021 35

This slide shows a simple example of the **`EMAIL_COB`** entry point for COBOL. The first parameter contains the parameter text and the second parameter contains its length.



EMAIL Retry Mechanism

- Send doesn't always work the first time
  - Usually due to problems contacting the MTA
  - Success is defined as message acceptance by MTA, *not delivery to recipients*
- On a transient send error
  - Trace of original attempt is written to **TASKFILE**
  - EMAIL creates and starts a WFL retry job
  - Name is **EMAIL/RETRY/*n***, where *n*=retry count
  - Failed retry starts another job, up to max retry count
  - Retries are scheduled after 2, 5, 10, 30 minutes
  - Retries thereafter are every hour
  - No **TASKFILE** is saved for retries unless **:TRACE ON**

2021 UNITE 4021 36

Because email usually involves exchanging data across a network, messaging does not always work the first time, and sometimes does not work at all. **OBJECT/EMAIL** has a basic retry mechanism it uses when it is unable to successfully send a message to its MTA.

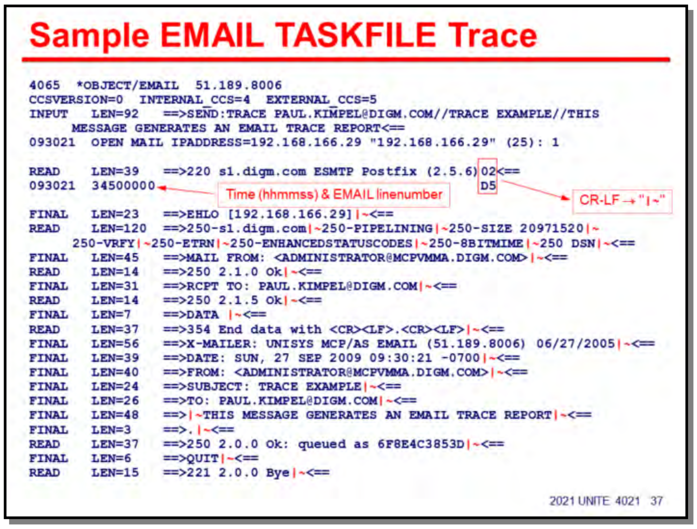
Note once again that success in terms of **OBJECT/EMAIL** is having the message accepted by its MTA, i.e., the primary email server or one of the alternates specified in the global configuration file. Once that first MTA accepts the message, **OBJECT/EMAIL** is done with its job. Delivery to the recipients is the MTA's job.

Therefore, the kinds of problems that typically invoke the **OBJECT/EMAIL** retry mechanism are those that involve contacting the MTA. Common types of problems are that the program cannot connect to the MTA, the connection is interrupted, authentication of credentials is not successful, or that there is a mismatch between the SMTP options supported by the two ends of the connection.

As mentioned earlier, **OBJECT/EMAIL** writes a printer trace of the message traffic between it and its MTA to its **TASKFILE**. The file is by default purged after a successful send, but is retained by default after the first unsuccessful transmission. This trace can prove very useful in determining what went wrong.

When a send to the MTA is unsuccessful, **OBJECT/EMAIL** constructs a WFL job to run the program again using the same parameter string and file equations as for the initial attempt. The name of this job will be **EMAIL/RETRY/***n*, where *n* is the retry count (1, 2, 3, …). The program starts this job with a **STARTTIME** to schedule the retry after a delay. The first four retries are scheduled for 2, 5, 10, and 30 minutes after their predecessor attempt, respectively. Subsequent retries are scheduled every hour. No **TASKFILE** trace is retained for these retries unless you had explicitly set the **TRACE** sending option in the original parameter string.

The number of retry attempts is determined by the **RECYCLE** sending option. The **RETRYQUEUE** option specifies which work flow job queue the retry jobs will be inserted into. The default (when **RECYCLE** is not specified) is that **OBJECT/EMAIL** will retry sending the message once and then give up. If you specify **RECYCLE**>0, the program will start jobs under the schedule described above until the send attempt succeeds or the retry count is exhausted, whichever occurs first.



Sample EMAIL TASKFILE Trace

```
4065   *OBJECT/EMAIL  51.189.8006
CCSVERSION=0  INTERNAL_CCS=4  EXTERNAL_CCS=5
INPUT   LEN=92    ==>SEND:TRACE PAUL.KIMPEL@DIGM.COM//TRACE EXAMPLE//THIS
     MESSAGE GENERATES AN EMAIL TRACE REPORT<==
093021  OPEN MAIL IPADDRESS=192.168.166.29 "192.168.166.29" (25): 1

READ    LEN=39    ==>220 s1.digm.com ESMTP Postfix (2.5.6)02<==
093021  34500000                                          D5
FINAL   LEN=23    ==>EHLO [192.168.166.29]|~<==
READ    LEN=120   ==>250-s1.digm.com|~250-PIPELINING|~250-SIZE 20971520|~
     250-VRFY|~250-ETRN|~250-ENHANCEDSTATUSCODES|~250-8BITMIME|~250 DSN|~<==
FINAL   LEN=45    ==>MAIL FROM: <ADMINISTRATOR@MCPVMMA.DIGM.COM>|~<==
READ    LEN=14    ==>250 2.1.0 Ok|~<==
FINAL   LEN=31    ==>RCPT TO: PAUL.KIMPEL@DIGM.COM|~<==
READ    LEN=14    ==>250 2.1.5 Ok|~<==
FINAL   LEN=7     ==>DATA |~<==
READ    LEN=37    ==>354 End data with <CR><LF>.<CR><LF>|~<==
FINAL   LEN=56    ==>X-MAILER: UNISYS MCP/AS EMAIL (51.189.8006) 06/27/2005|~<==
FINAL   LEN=39    ==>DATE: SUN, 27 SEP 2009 09:30:21 -0700|~<==
FINAL   LEN=40    ==>FROM: <ADMINISTRATOR@MCPVMMA.DIGM.COM>|~<==
FINAL   LEN=24    ==>SUBJECT: TRACE EXAMPLE|~<==
FINAL   LEN=26    ==>TO: PAUL.KIMPEL@DIGM.COM|~<==
FINAL   LEN=48    ==>|~THIS MESSAGE GENERATES AN EMAIL TRACE REPORT|~<==
FINAL   LEN=3     ==>.|~<==
READ    LEN=37    ==>250 2.0.0 Ok: queued as 6F8E4C3853D|~<==
FINAL   LEN=6     ==>QUIT|~<==
READ    LEN=15    ==>221 2.0.0 Bye|~<==
```

2021 UNITE 4021   37

This slide shows a sample **TASKFILE** trace from an **OBJECT/EMAIL** run. The first few lines show the program version, CCS settings, repeat the parameter string, and indicate which MTA was contacted. In this example, the MTA is a Fedora Linux system running Postfix.

Following that is a trace of the message traffic (if any) between the program and the MTA. Each message is shown on two or more lines, surrounded by blank lines. The message text is shown in CANDE "hexplicit" format, where non-graphic characters are displayed in hex, with the first hex digit of the on the line with the printable characters and the second hex digit on the line below.

The following shows an example of one message. The first token indicates the type of operation. **READ** indicates data received from the MTA, **FINAL** indicates the final (or only) chunk of data sent to the MTA by the program for a given protocol message. **LEN=** indicates the message length in characters. The text of the message is shown between the "**==>**" and "**<==**" arrows. On the second line, the first token is the time of day in hours/minutes/seconds. The second token is the line number within **OBJECT/EMAIL** where the operation took place.

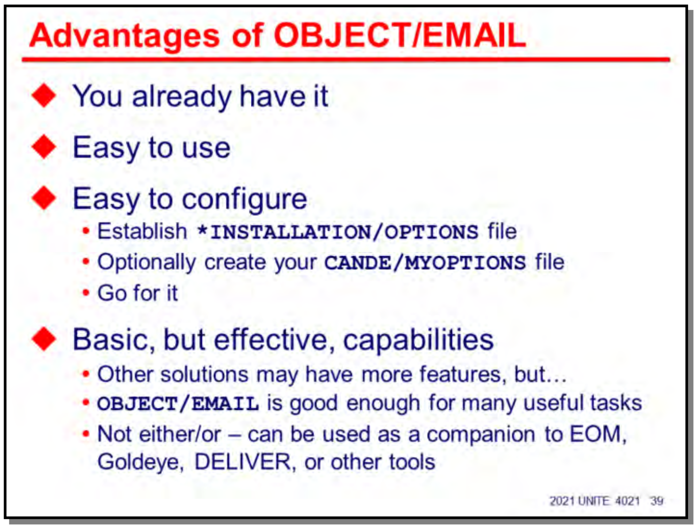
```
READ    LEN=39   ==>220 s1.digm.com ESMTP Postfix (2.5.6)02<==
093021  34500000                                         D5
```

Only the first **READ** operation is shown on the slide in its original format. To save space and allow the entire session to fit on the slide, the remaining blank lines have been removed, along with the second line of trace data. The hex **0D** and hex **25** (carriage-return and line-feed) characters were replaced by "**|**" and "**~**" characters, respectively.

This trace represents a successful send. You can see the SMTP commands being exchanged between **OBJECT/EMAIL** and the MTA, along with the headers and body text of the message. SMTP replies from the MTA begin with a three-digit status code. If there are problems communicating with the MTA, these status codes and their messages are one of the first places to look. These status codes are defined in the SMTP standard, which was originally RFC 821, then 2821, and is now 5321.

So, What Good Is It?

That completes the discussion of the **`OBJECT/EMAIL`** program and how you use it. Having that information, the question is now, what good it? This next section of the presentation discusses some potential applications.



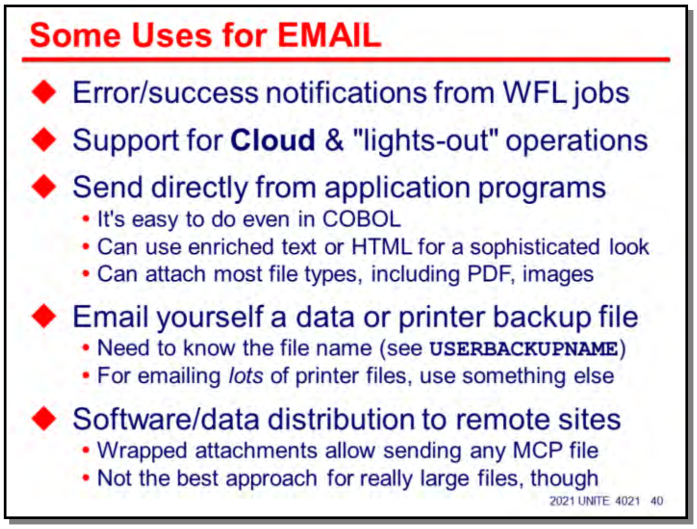
Advantages of OBJECT/EMAIL

- You already have it
- Easy to use
- Easy to configure
  - Establish ***INSTALLATION/OPTIONS** file
  - Optionally create your **CANDE/MYOPTIONS** file
  - Go for it
- Basic, but effective, capabilities
  - Other solutions may have more features, but…
  - **OBJECT/EMAIL** is good enough for many useful tasks
  - Not either/or – can be used as a companion to EOM, Goldeye, DELIVER, or other tools

2021 UNITE 4021 39

OBJECT/EMAIL has a number of advantages.

- First, you already have it. The program has been bundled with every ClearPath MCP release since 7.0 (SSR 48.1).
- Second, it is quite easy to use. The program has a lot of options, but you don't need to know many of them in order to send a simple message.
- Third, it's easy to configure. You need one line in the ***INSTALLATION/OPTIONS** global configuration file to define the MTA, and an email address either in your **USERDATA** record or an entry in your **CANDE/MYOPTIONS** file. With just that setup, you can send messages, unless the MTA requires authentication or something else special.
- Fourth, **OBJECT/EMAIL** has very basic email composing and sending capabilities, but they are effective ones in the server-oriented MCP environment. Other solutions have more features, but **OBJECT/EMAIL** is suitable for many useful tasks. It is also not an either/or situation – you can combine the use of **OBJECT/EMAIL** with other email solutions, picking the one that is most suitable for each task.



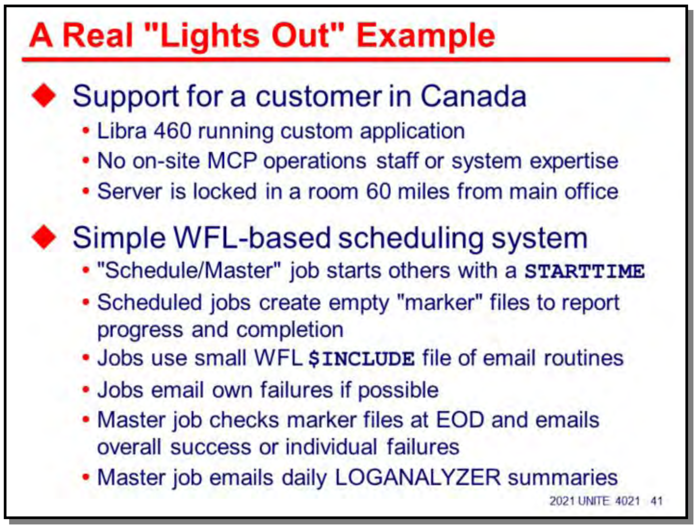
Some Uses for EMAIL

- Error/success notifications from WFL jobs
- Support for **Cloud** & "lights-out" operations
- Send directly from application programs
  - It's easy to do even in COBOL
  - Can use enriched text or HTML for a sophisticated look
  - Can attach most file types, including PDF, images
- Email yourself a data or printer backup file
  - Need to know the file name (see **USERBACKUPNAME**)
  - For emailing *lots* of printer files, use something else
- Software/data distribution to remote sites
  - Wrapped attachments allow sending any MCP file
  - Not the best approach for really large files, though

2021 UNITE 4021 40

Here are some examples of uses for **OBJECT/EMAIL**:

- **Error/success notifications from WFL jobs**.
  This is the perhaps primary use for the program. Given WFL's ability to detect success or failure of the tasks in a job, it is easy to run **OBJECT/EMAIL** as part of the job and send a message to indicate that an activity has completed successfully or has had some problem. You can even attach files and printer output to document a problem.
- **Support for Cloud and "lights-out“ operations**.
  Many data centers run unattended, especially during non-prime hours. If you have moved your MCP system into the Cloud, the need for remote, automated operations reporting is even more severe. **OBJECT/EMAIL** can help automate operations by sending status and error notifications, and automatically distributing data in the form of email attachments.
- **Sending email directly from application programs**.
  The API mode of **OBJECT/EMAIL** makes it very easy to compose and send messages from applications, even those written in COBOL.
- **Email yourself a data or printer backup file**.
  We have lots of ways to exchange data between the MCP environment and workstation environments – FTP, shared network directories, Telnet, PWB, etc. **OBJECT/EMAIL** gives us another one. It is very easy to email a data or printer file to yourself or a colleague. You only need to know the name of the file and can then include it in the body of the message or include it as an attachment. This is a little more difficult for spooled printer files, but by setting the file attribute **USERBACKUPNAME=TRUE**, you can control the name of a printer-backup file so that it can be attached to a message programmatically. I do this quite frequently. If you are going to do a lot of printer distribution by email, though, EOM or Goldeye PrintMailer are much more convenient (and probably more efficient) solutions to use.
- **Software/data distribution to remote sites**.
  If you have multiple MCP sites, **OBJECT/EMAIL** gives you a way to distribute data, and even object code, to those sites. This is probably not be best approach for really big files, but the program's ability to wrap files before attaching them to an email message allows you to send virtually any MCP file to anyone who has an email address.



A Real "Lights Out" Example

- Support for a customer in Canada
  - Libra 460 running custom application
  - No on-site MCP operations staff or system expertise
  - Server is locked in a room 60 miles from main office
- Simple WFL-based scheduling system
  - "Schedule/Master" job starts others with a **STARTTIME**
  - Scheduled jobs create empty "marker" files to report progress and completion
  - Jobs use small WFL **$INCLUDE** file of email routines
  - Jobs email own failures if possible
  - Master job checks marker files at EOD and emails overall success or individual failures
  - Master job emails daily LOGANALYZER summaries

2021 UNITE 4021 41

The preceding slide discussed some general ideas for employing **OBJECT/EMAIL**. The next series of slides describes an application of the program I made for a real "lights-out" environment.

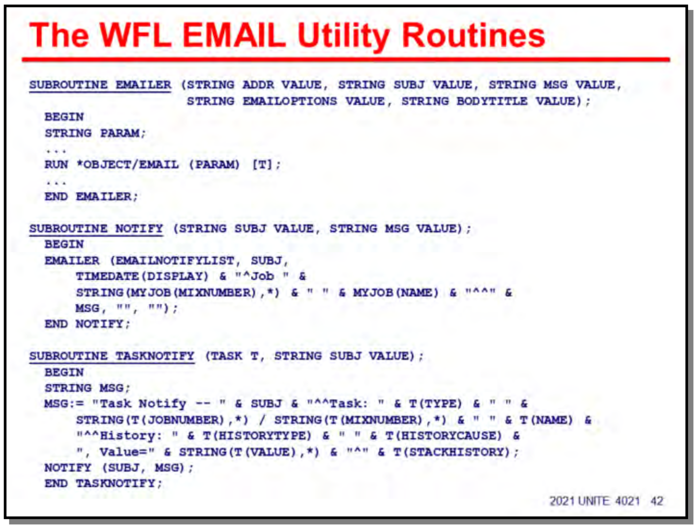
I had a customer in Canada that was running a custom in-house application on a Libra 460, which has since been shut down. They had no on-site MCP operations staff or system expertise. In fact, their Libra sat in a locked room at a satellite facility about 60 miles from the main office where most of the users were located. I was their MCP support, and only visited the site infrequently. I supported them from San Diego over an Internet VPN, using Telnet, Terminal Server, PWB, network shares, FTP, and… **OBJECT/EMAIL**!

Their batch processing requirements were modest. Users initiated batch reports from a menu-based COMS application that built parameter strings and started WFL jobs. Reports printed automatically. They had a few regularly-scheduled jobs – primarily backups and file extracts. The Libra did not have a tape drive, so we used a combination of **SYSTEM/FILECOPY** and **WRAP** to generate container files, which got transferred to the Windows side of the Libra, where they were picked up by a Windows-based network backup system.

I built a "Schedule/Master" job that did a start on itself every weekday just after midnight. This job in turn started the other scheduled jobs with an appropriate **STARTTIME**. To monitor the scheduled jobs, each one generated "marker" files (which are just directory entries; they have no data) to indicate they had finished successfully. The scheduled jobs generated email messages if they detected problems in their processing. The master job checked for these marker file at the end of the day and generated emails if any were missing. This ensured that problems got reported even if one of the scheduled jobs crashed or got DS-ed before it could generate any emails. I wrote a WFL **$INCLUDE** file of email routines to standardize the error reporting and make it easy to use.

The master job also ran some **LOGANALYZER** reports at the end of the day and emailed them to me and another contractor who did application program support. We used these to determine if any programs aborted during the day. If necessary, we analyzed programdumps (which the system is configured to write to **PDUMP** files) by emailing the analyzed printer files. We used EOM to email these reports, because we had email configured in EOM, but with just a little bit of work, we could have used **OBJECT/EMAIL** do this instead.

This was only a couple of days' work to set up, and frankly, it worked great for many years. Instead of constantly monitoring the system, we were been quite successful in getting it to tell us when there was a problem that needed attention. I often knew by the end of my day in California whether a problem occurred, and sometimes could have a correction in place before work at the customer site started the next day.



The WFL EMAIL Utility Routines

```
SUBROUTINE EMAILER (STRING ADDR VALUE, STRING SUBJ VALUE, STRING MSG VALUE,
                    STRING EMAILOPTIONS VALUE, STRING BODYTITLE VALUE);
  BEGIN
  STRING PARAM;
  ...
  RUN *OBJECT/EMAIL (PARAM) [T];
  ...
  END EMAILER;

SUBROUTINE NOTIFY (STRING SUBJ VALUE, STRING MSG VALUE);
  BEGIN
  EMAILER (EMAILNOTIFYLIST, SUBJ,
      TIMEDATE(DISPLAY) & "^Job " &
      STRING(MYJOB(MIXNUMBER),*) & " " & MYJOB(NAME) & "^^" &
      MSG, "", "");
  END NOTIFY;

SUBROUTINE TASKNOTIFY (TASK T, STRING SUBJ VALUE);
  BEGIN
  STRING MSG;
  MSG:= "Task Notify -- " & SUBJ & "^^Task: " & T(TYPE) & " " &
      STRING(T(JOBNUMBER),*) / STRING(T(MIXNUMBER),*) & " " & T(NAME) &
      "^^History: " & T(HISTORYTYPE) & " " & T(HISTORYCAUSE) &
      ", Value=" & STRING(T(VALUE),*) & "^" & T(STACKHISTORY);
  NOTIFY (SUBJ, MSG);
  END TASKNOTIFY;
```

2021 UNITE 4021 42

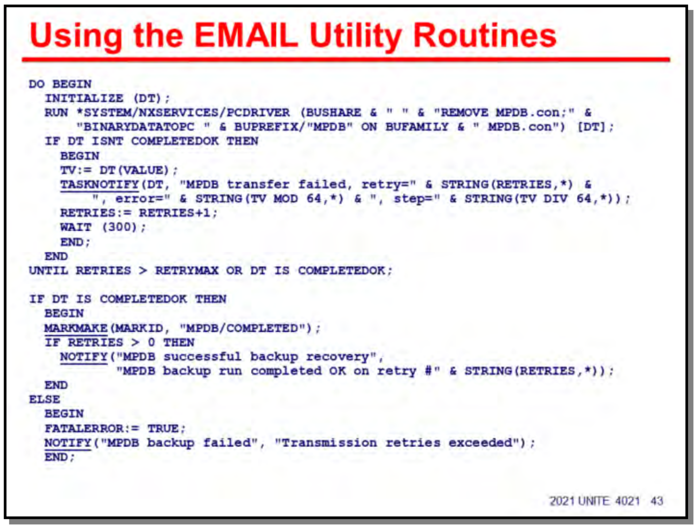
This slide shows an extract from the WFL **`$INCLUDE`** file of email routines I wrote to support email notifications from the scheduled jobs. The source for these routines and some of the scheduled jobs is included in the sample files accompanying the presentation materials.

The file has just three routines:

- **`EMAILER`** formats a parameter string and executes **`OBJECT/EMAIL`** as a program. It accepts string parameters for the email address list, the subject line, message body text, sending options, and an optional file title for more body text. This routine contains mostly string manipulation code to format the parameter for **`OBJECT/EMAIL`**. This formatting code has been excluded from the slide.
- **`NOTIFY`** is a utility routine to call **`EMAILER`** to send a simple message with a heading containing the date, time, job mix number, and job name.
- **`TASKNOTIFY`** calls **`NOTIFY`** to send a more complex message. This routine is intended for use when a task terminates. In addition to the heading formatted by **`NOTIFY`**, this routine reports the task's mix number and name, history attributes (which describe the type of abort), **`TASKVALUE`**, and stack history string (the list of code addresses and line numbers of the most recent procedure calls).

We will see some examples of the output from these routines on the next few slides.

We maintained a list of email addresses where messages were to be sent in a text file, **`CANDE/NOTIFYLIST`**, under the production usercode that was used with the scheduled jobs. Those message are sent to the application support contractor, the IT manager, and me. The name of that file, along with its "%" prefix to indicate a file-based distribution list, is stored in the WFL variable **`EMAILNOTIFYLIST`**, which is part of the **`$INCLUDE`** file.



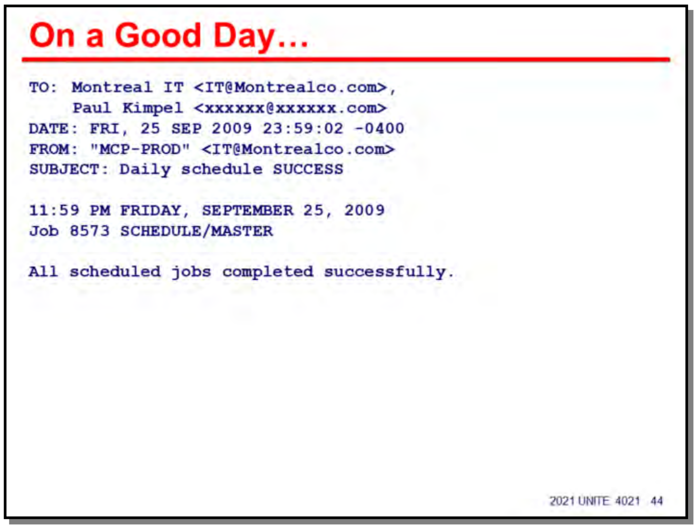
Using the EMAIL Utility Routines

```
DO BEGIN
  INITIALIZE (DT);
  RUN *SYSTEM/NXSERVICES/PCDRIVER (BUSHARE & " " & "REMOVE MPDB.con;" &
      "BINARYDATATOPC " & BUPREFIX/"MPDB" ON BUFAMILY & " MPDB.con") [DT];
  IF DT ISNT COMPLETEDOK THEN
    BEGIN
    TV:= DT(VALUE);
    TASKNOTIFY(DT, "MPDB transfer failed, retry=" & STRING(RETRIES,*) &
        ", error=" & STRING(TV MOD 64,*) & ", step=" & STRING(TV DIV 64,*));
    RETRIES:= RETRIES+1;
    WAIT (300);
    END;
  END
UNTIL RETRIES > RETRYMAX OR DT IS COMPLETEDOK;

IF DT IS COMPLETEDOK THEN
  BEGIN
  MARKMAKE(MARKID, "MPDB/COMPLETED");
  IF RETRIES > 0 THEN
    NOTIFY("MPDB successful backup recovery",
           "MPDB backup run completed OK on retry #" & STRING(RETRIES,*));
  END
ELSE
  BEGIN
  FATALERROR:= TRUE;
  NOTIFY("MPDB backup failed", "Transmission retries exceeded");
  END;
```

2021 UNITE 4021 43

This next slide shows a portion of the code from the daily backup job. This snippet attempts to transfer a wrapped container file produced by the backup to a network share on the Windows side of the Libra 460 (the name of which is in the variable **`BUSHARE`**). The transfer is done in a loop using the SMB file transfer utility **`*SYSTEM/NXSERVICES/PCDRIVER`** that is bundled with the standard MCP release.

If the file transfer utility terminates abnormally, the job calls **`TASKNOTIFY`** to tell us about it and restarts the transfer. After some number of unsuccessful retries, the job gives up and sends us message to say so. If a retry is successful, the job also sends us an email so we know we can ignore the previous error messages. If the transfer succeeds on the first try, no email is sent at all. **`MARKMAKE`** is a routine that will generate a "marker" file (the source for these routines, also in a WFL **`$INCLUDE`** file, are included with the presentation samples as well).



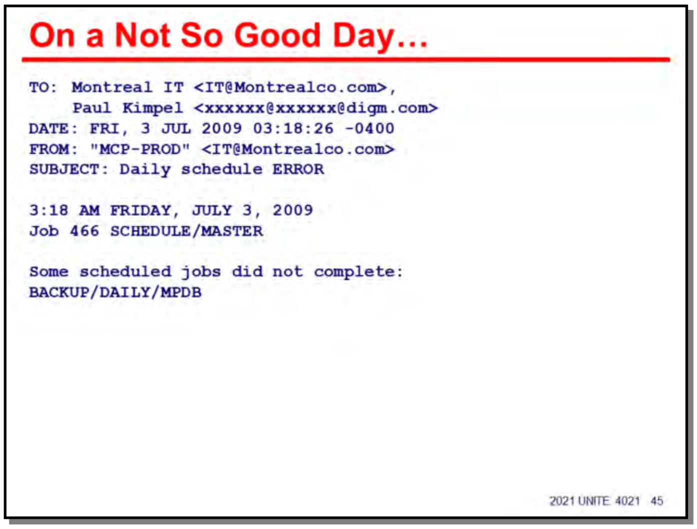
On a Good Day…

```
TO: Montreal IT <IT@Montrealco.com>,
    Paul Kimpel <xxxxxx@xxxxxx.com>
DATE: FRI, 25 SEP 2009 23:59:02 -0400
FROM: "MCP-PROD" <IT@Montrealco.com>
SUBJECT: Daily schedule SUCCESS

11:59 PM FRIDAY, SEPTEMBER 25, 2009
Job 8573 SCHEDULE/MASTER

All scheduled jobs completed successfully.
```

2021 UNITE 4021 44

On a good day, when everything went properly with the scheduled jobs, the master job sent us one email to give us the news.

The message shown on the slide is an example of output from the **`NOTIFY`** routine.

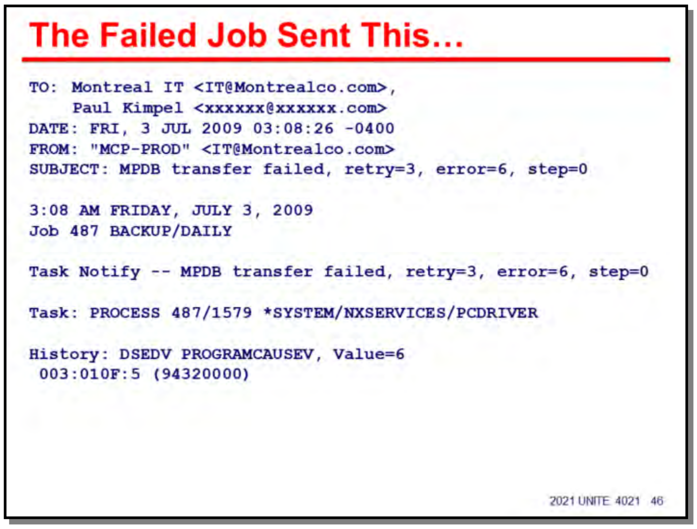


On a Not So Good Day…

```
TO: Montreal IT <IT@Montrealco.com>,
    Paul Kimpel <xxxxxx@xxxxxx@digm.com>
DATE: FRI, 3 JUL 2009 03:18:26 -0400
FROM: "MCP-PROD" <IT@Montrealco.com>
SUBJECT: Daily schedule ERROR

3:18 AM FRIDAY, JULY 3, 2009
Job 466 SCHEDULE/MASTER

Some scheduled jobs did not complete:
BACKUP/DAILY/MPDB
```

2021 UNITE 4021 45

If one of the scheduled jobs did not complete successfully, the master job sent a summary email at the end of the day listing the jobs that had problems. This message was sent instead of the "good news" one on the prior slide. The failed job would typically have sent a more detailed message about the problem prior to this one, unless it aborted or was DS-ed.

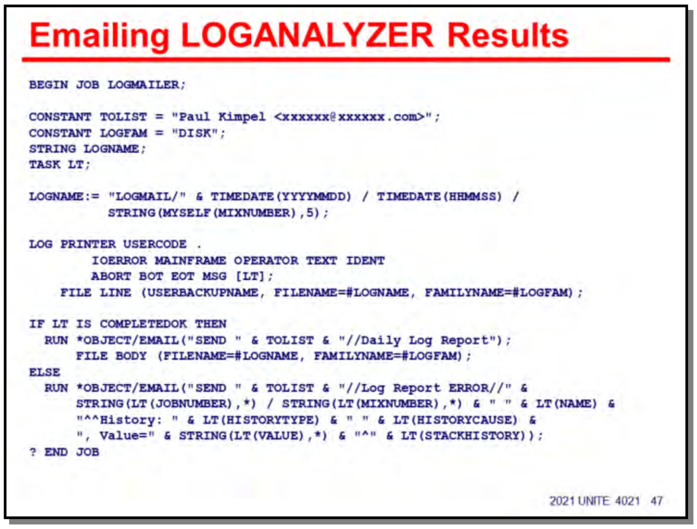


The Failed Job Sent This…

```
TO: Montreal IT <IT@Montrealco.com>,
    Paul Kimpel <xxxxxx@xxxxxx.com>
DATE: FRI, 3 JUL 2009 03:08:26 -0400
FROM: "MCP-PROD" <IT@Montrealco.com>
SUBJECT: MPDB transfer failed, retry=3, error=6, step=0

3:08 AM FRIDAY, JULY 3, 2009
Job 487 BACKUP/DAILY

Task Notify -- MPDB transfer failed, retry=3, error=6, step=0

Task: PROCESS 487/1579 *SYSTEM/NXSERVICES/PCDRIVER

History: DSEDV PROGRAMCAUSEV, Value=6
 003:010F:5 (94320000)
```

2021 UNITE 4021 46

This slide shows an example of a message sent when one of the scheduled jobs detects a problem. This message is an example of the output from the **`TASKNOTIFY`** routine.



Emailing LOGANALYZER Results

```
BEGIN JOB LOGMAILER;

CONSTANT TOLIST = "Paul Kimpel <xxxxxx@xxxxxx.com>";
CONSTANT LOGFAM = "DISK";
STRING LOGNAME;
TASK LT;

LOGNAME:= "LOGMAIL/" & TIMEDATE(YYYYMMDD) / TIMEDATE(HHMMSS) /
          STRING(MYSELF(MIXNUMBER),5);

LOG PRINTER USERCODE .
        IOERROR MAINFRAME OPERATOR TEXT IDENT
        ABORT BOT EOT MSG [LT];
    FILE LINE (USERBACKUPNAME, FILENAME=#LOGNAME, FAMILYNAME=#LOGFAM);

IF LT IS COMPLETEDOK THEN
  RUN *OBJECT/EMAIL("SEND " & TOLIST & "//Daily Log Report");
      FILE BODY (FILENAME=#LOGNAME, FAMILYNAME=#LOGFAM);
ELSE
  RUN *OBJECT/EMAIL("SEND " & TOLIST & "//Log Report ERROR//" &
      STRING(LT(JOBNUMBER),*) / STRING(LT(MIXNUMBER),*) & " " & LT(NAME) &
      "^^History: " & LT(HISTORYTYPE) & " " & LT(HISTORYCAUSE) &
      ", Value=" & STRING(LT(VALUE),*) & "^" & LT(STACKHISTORY));
? END JOB
```

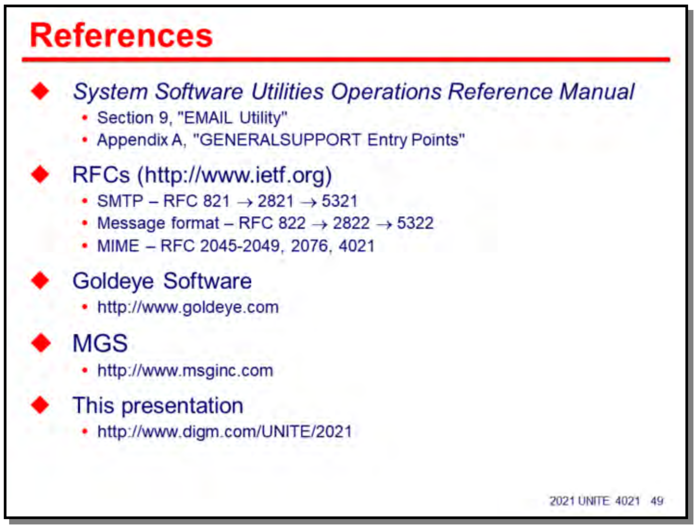
2021 UNITE 4021 47

One final example – this slide shows how to email a printer file to yourself using **`OBJECT/EMAIL`**:

- The job generates a string value and stores it in the variable **`LOGNAME`**. This value will be used to name the printer file and is designed to be reasonably unique.
- The WFL **`LOG`** statement runs **`SYSTEM/LOGANALYZER`**. The file equation for **`LINE`** affects the printer file that **`LOGANALYZER`** produces. The attributes specified are:
  - **`USERBACKUPNAME`** controls how the printer-backup file will be named. The default, when this attribute is **`FALSE`**, is to use the system's **`BD/=`** naming convention. When **`TRUE`**, the name of the printer-backup file is taken from the **`FILENAME`** attribute.
  - **`FILENAME`** indicates the name to be assigned to the printer-backup file. Since you are generating this name, you need to take care that the name is unique and will not cause a previously-created file of the same name to be removed.
  - **`FAMILYNAME`** can be used to specify the disk family where the printer-backup name will be written by the system. If this is not specified, the file will be written to the **`DL BACKUP`** family. *Do not* use a **`TITLE`** or **`LTITLE`** attribute that contains an **`ON`** clause to name the printer-backup file. The presence of the **`ON`** clause changes the file to **`KIND=DISK`**.
- The WFL statements following the **`LOGANALYZER`** run deal with the results. If the run completed successfully, the job executes **`OBJECT/EMAIL`** and file-equates the printer-backup file so that it will become the body of the message (note that no body text is present in the parameter string). If there is an error, the job sends an email message similar to the one for the **`TASKNOTIFY`** routine discussed earlier.

This job does not dispose of the printer backup file that was generated. It could be removed by the job, released for normal printing, or perhaps cleaned up later by some site-developed file retention and purging process.

Time for a Demo?



References

- *System Software Utilities Operations Reference Manual*
  - Section 9, "EMAIL Utility"
  - Appendix A, "GENERALSUPPORT Entry Points"
- RFCs (http://www.ietf.org)
  - SMTP – RFC 821 → 2821 → 5321
  - Message format – RFC 822 → 2822 → 5322
  - MIME – RFC 2045-2049, 2076, 4021
- Goldeye Software
  - http://www.goldeye.com
- MGS
  - http://www.msginc.com
- This presentation
  - http://www.digm.com/UNITE/2021

2021 UNITE 4021 49

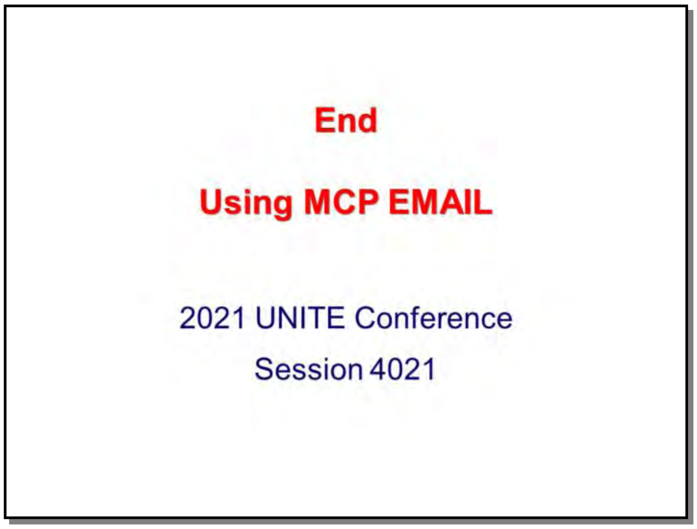
The documentation for **`OBJECT/EMAIL`** is contained in the Unisys *System Software Utilities Operations Reference Manual*. The **`MYOPTIONSUPPORT`** entry points to **`GENERALSUPPORT`** are described in Appendix A of that Document.

The standards for email are contained in Request for Comments (RFC) documents, all of which are available from the IETF web site.

The Goldeye web site provides information on their MCP email products and provides contact and pricing information.

The MGS web site provides information on the features and capabilities of DELIVER.

Finally, a copy of this presentation and a set of demonstration and sample files is available on our web site under the URL shown on the slide.



End

Using MCP EMAIL

2021 UNITE Conference

Session 4021