CICS® TS support for sending emails

CICS SupportPac CA1Y - Installation and User's Guide

Version 1.7.1 – June 2016

Author:

Mark Cocker – mark_cocker@uk.ibm.com

IBM, CICS Development, Technical Strategy and Planning
IBM United Kingdom Limited. Hursley Park. SO21 2JN. UK

Contents

Chapter 1. Overview

Love it or hate it, electronic mail has become an effective form of communication widely used in our private lives and dealings with businesses, governments, and other organizations. The trend towards on-line purchasing, investments, banking, managing utilities and tax returns is driving further adoption of email. For example customers now expect confirmation of orders, account alerts, and availability of statements to be sent within minutes via email.

As many of today's transactions are hosted in CICS Transaction Server (CICS), it is not surprising the need to send emails from CICS applications has been raised a number of times on the cics-l list and customer surveys.

There are already a number of solutions to send emails from CICS and batch, such as the z/OS Communications Server SMTP application and the spool interface, intermediaries such as process servers, and writing your own SMTP client. These solutions however can be inflexible in creating the email content, lack immediacy, require in-depth knowledge of SMTP, require code to be developed and maintained, or require the purchase of additional products.

This SupportPac provides an SMTP client to send emails and attachments that is quick to setup, easy to call from your CICS application, CICS event, batch JCL or script. It runs in Java and therefore eligible for off-loading onto a System z Application Assist Processor (zAAP) specialty engine.

In addition to sending emails, the SupportPac is able to write to CICS temporary data (TD) queues and temporary storage (TS) queues, submit MVS jobs, issue MVS console messages, and send HTTP requests. TD queues can be used as an audit trail or monitored by systems management tools to initiate automation scripts. TS queues are useful to test the capture, emission and formatting of events. MVS jobs and console messages can be used to automate system activity, issue modify commands, or initiate batch work. HTTP requests could initiate many types of activity such as sending SMS messages or updating dashboards.

Sending an email from CICS

This SupportPac provides a choice of two interfaces to send an email from CICS.

1. Capture an event and emit it to the SupportPac event adapter.
   - Use the CICS Explorer event binding editor to define when and what information to capture from your application or system event.

     You can also use the CICS Explorer policy definition editor to define policy actions that you want to emit as a system event.
   - Use the CICS Explorer event adapter editor to define SupportPac transaction CA1Y as the custom event adapter and specify the email headers, content, attachments, etc. using name=value properties.
   - This interface is unlikely to require changes to the application, and because it can process the event asynchronous it is unlikely to change the applications' response time or qualities of service.
   - This is the recommended approach.
2. Write an application to call the SupportPac.
   - From your application, create one or more CICS containers to specify the email headers, content, attachments, etc., then issue a LINK CHANNEL command to program CA1Y, or START CHANNEL command to transaction CA1Y.

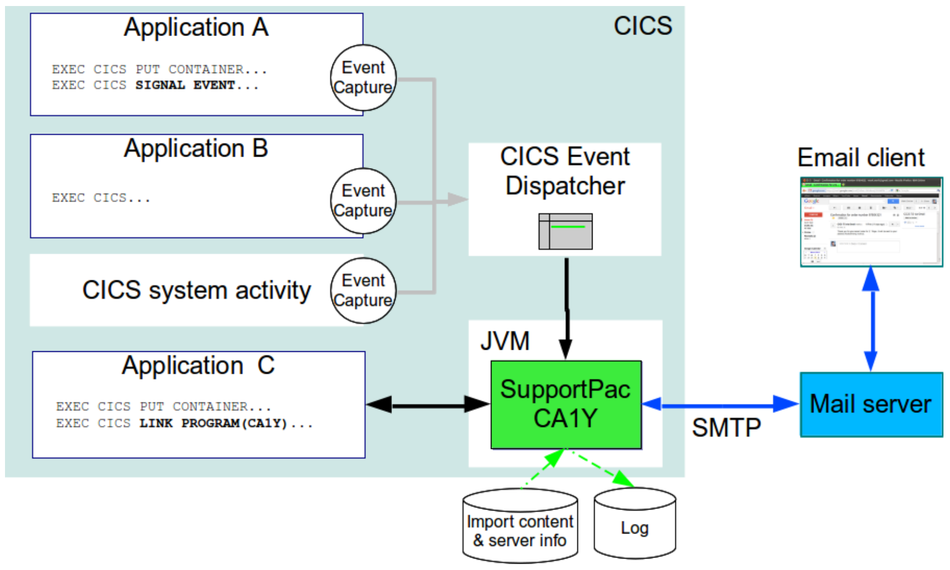
     Alternatively, you can use a CICS communications area (commarea) or START FROM area instead of a container, but these are restricted in size to less than 32KB and the content are required to be encoded using the CICS region local CCSID in which CA1Y executes.

- The LINK CHANNEL and LINK COMMAREA commands will synchronously send an email from an application. Your application will wait for the email to be sent and could react if there is an error.
- The START CHANNEL command will asynchronous send an email from an application. CICS will schedule the CA1Y transaction for execution and your application will not be aware of the success or failure of the request.

Both of these interfaces are easy to use and flexible, allowing you to specify:

- **Email address headers for recipients** including; from, to, carbon copy (cc), blind carbon copy (bcc), and reply to. These headers are specified using the Internet Engineering Task Force (IETF) RFC 822 format, for example; <sam.smith@company.com>,
  “John Doe” <j.doe@example.com>
- **Email subject** formatted as a single line of plain text.
- **Email content** typically formatted as plain text or HTML.
- **Email attachments** including a name and Multipurpose Internet Mail Extensions (MIME) type as defined by RFC 2046 and IANA. For example photos, Portable Document Format (PDF), or spreadsheets could be attached. The SupportPac can compress attachments into a zip to save network bandwidth and mail server storage.
- **Tokens** placed anywhere in the subject, content and other properties are automatically replaced with CICS event information items, current time & date, CICS containers, DOCTEMPLATE resources, MVS files, or zFS files. This token replacement is similar in concept to mail merge in word processors and mass mail systems.
- **Conversion of content and attachments** from XML into other document types. The SupportPac processes the XML in combination with your XML stylesheet (XSLT) to generate HTML, XHTML, or other XML documents. In addition the SupportPac can work in combination with the open source Apache™ FOP Project (Formatting Objects Processor) to convert XSL formatting objects (XSL-FO) into PDF, Rich Text Format (RTF), and other printed documents. For example this enables CICS event information to be combined with other XML and a stylesheet to create an invoice in PDF format that is emailed to a customer.
- **SMTP server configuration** including IP name or address, IP port, and security credentials.
- **Importing of common configuration** from zFS and other locations enabling SMTP server configuration to be secured and managed separate to the application.

The following figure shows how the SupportPac can receive requests from a variety of application and system events and from applications issuing LINK commands. The requests are sent using SMTP to any mail server on any platform, ready for retrieval by an email client, or relaying on to external email systems.



Software requirements

To use the SupportPac in CICS, you must have one of the following versions of CICS installed and configured to use the Java server environment:

- CICS Transaction Server for z/OS V4.2 or later
- CICS Transaction Server for z/OS Developer Trial
- CICS Transaction Server for z/OS Value Unit Edition

To use the SupportPac in batch, any supported version of IBM 31-bit SDK for z/OS, or IBM 64-bit SDK for z/OS is required.

To send emails the JavaMail™ API is required. Although a version is provided with CICS TS, to use the latest version, or to use the SupportPac in batch, download the JavaMail API from their project site.

To create PDF documents, the Apache FOP Project is required.

Restrictions

The SupportPac uses the JavaMail API to interact with the mail server. The SupportPac has been tested with the Simple Mail Transfer Protocol (SMTP) service provider. The JavaMail API provides additional protocol service providers such as IMAP and POP3 that may work, but they have not been tested.

What's new

Version 1.7.1 released June 2016

- The `doctemplate` token has a new addPropertiesAsSymbols option to add the properties passed to CA1Y as document symbols. The symbols can then be used in the template and will be substituted by CICS.
- New `trim` token can be used to removed leading and trailing spaces from the property. This is useful to remove unwanted spaces from COBOL strings that are typically padded with spaces.

Version 1.7.0 released February 2016

- New support to call the SupportPac from a z/OS JCL batch job, a USS script, or using the CICS command START FROM().

- New `texttable` token will create a hexadecimal dump of the conents of all properties and containers, or only those whos name matches a specified regular expression. The `htmltable` token provides similar data but in an HTML table form suitable for use in an email.
- Fix to set the email header time and date to the CICS local time and date (ABSTIME) when the event was captured. If you wish to set an alternative date and time, set the new `mail.sentdate` property.
- Fix to not issue an abend when CICS is unable to capture a data item. For example if your event binding specified a FILE ENABLE STATUS system capture point and made use of DSNAME, but the file resource did not specify a data set name then DSNAME will not be captured.

Version 1.6 released February 2014

- New support for specifying properties in a CICS commarea.
- New support to read JVM system properties.
- Optimization to reduce the number of JNI calls to get CICS containers.
- The property `mail.propstable` has been replaced with token `htmltable` to allow it to be used in more situations.
- Fixes to allow UTF-8 characters in the mail subject and content, and to allow emails to be sent when there is a subject but no content.
- Fix to not overwrite a property alternate name with a container name.

Version 1.5 released December 2013

- New support to link to a CICS program with the current channel using the `link` token.
- New support to copy the data from a property into a container using the `putcontainer` token.
- New support for using the Saxon XSLT and XQuery processor to transform XML into another XML or xHTML document.
- New support to send an HTTP request using the `http.content` property that provides options to retry after network failures.
- New support to LINK to a CICS program if there is a failure to process the request using the `onfailure` property.
- New `commonbaseevent`, `commonbaseeventrest`, and `json` tokens to create XML and JSON documents that represent a CICS event.
- New `nomoretokens` and `noinnertokens` tokens to avoid unnecessary searching for tokens in large properties.
- Fix. In V1.3 and V1.4 when a `file` token was used to read an MVS file it remained open until the JVMSERVER was disabled. In V1.5 the file is closed after being read.

Version 1.4 released September 2013

- New support to submit an MVS job using the `mvsjob.content` property.
- New support to issue an MVS console message using the `mvswto.content` property.
- New support to get files from an FTP server using the `ftp` token.
- New support to insert z/OS system symbols using the `systemsymbol` token.
- New support to insert a hexadecimal representation of a property using the `hex` token.

- `name` token can now set the name of a property to the value of another property, useful to dynamically name attachments. For example, you can include a date stamp in the name.

Version 1.3 released May 2013

- New support to read sequential datasets and partitioned datasets (PDS) using tokens `file=//'dataset'` and `file=//DD:ddName`.
- New support to compress one or more properties into a zip file using tokens `zip=zipFilename` and `zip=zipFilename:include=propertyList`.
- New support to write content to a TD or TS queue using the queue properties.
- New support to transform application data into XML using the CICS XMLTRANSFORM resource with the token `transform=`*property*`:xmltransform=`*resource*.
- Added example logging properties.

Version 1.2 released April 2013

- New support to read DOCTEMPLATE resources using the `doctemplate` token.
- Fix for error "`java.lang.NoSuchMethodError: com/ibm/cics/server/Container.putString(Ljava/lang/String;)`" when using LINK to call SupportPac V1.1 with CICS TS V4.2.

Version 1.1 released February 2013

- New support for converting XSL Formatting Objects (XSL-FO) into PDF or other print documents. This support requires the Apache FOP Project.
- New support for converting XML using XSLT (Extensible Stylesheet Language Transformations) into other document types.
- New support for returning properties to the application using the token `returncontainer`. For example the application can use the SupportPac to generate a PDF that is returned to the application for archiving or returning to a client.
- The MIME type for mail contents and other properties is now set using token `mime`. The properties `mail.contentmime` and `mail.attmime.n` from version 1 are no longer used.
- A property is now automatically attached to the email if it includes the `mime` token. The property `mail.att.n` from version 1 is no longer used to specify email attachments.
- A property is now named using the `name` token. The property `mail.attname.n` from version 1 is no longer used.
- Log messages are now written using the Java SE java.util.logging package for greater flexibility to select the log records of interest, directing log records to sets of files, and integration with other Java applications.

  The property `mail.log.success` from version 1 is no longer used. When an email is successfully sent a log record is written if the log level is set to INFO.

  The property `mail.log.fail` from version 1 is no longer used. When a failure occurs a log record is written if the log level is set to WARNING.

  The properties `log.tokens` and `log.epconversions` from version 1.0 are no longer used. The processing of tokens and adding event processing information items are written if the log level is set to FINE.

Version 1 released October 2012

- Support for sending emails and attachments.

Feedback

This SupportPac is provided as-is and is not supported by IBM service. However the author welcomes your comments and bug reports by email to mark_cocker@uk.ibm.com or by appending to the CICS developer center Q&A.

For new features please submit a Request For Enhancement and select; brand as *WebSphere*, product family as *Transaction Processing*, product as *CICS Transaction Server*, and component as *Other*.

Chapter 2. Installation

The following tasks guide you through downloading and installing the SupportPac, configuring CICS, and optionally installing the Apache FOP and Saxon projects.

The tasks assume you are familiar with managing CICS resources, zFS files, and z/OS UNIX commands, and that the software requirements mentioned on page 5 are installed.

Download, copy and decompress the SupportPac

From a workstation:

1. Use a browser to navigate to the SupportPac CA1Y site.
2. Download `ca1y.zip` by clicking the appropriate link under **Download package**.
3. Providing you agree to the terms and conditions shown, click **I agree**.
4. Save the file `ca1y.zip`.
5. Create an installation directory on zFS for the SupportPac, for example `/usr/lpp/ca1y`, and copy `ca1y.zip` in binary into it.

From a z/OS UNIX shell:

1. Decompress `ca1y.zip` into the zFS installation directory, for example:
   ```
   mkdir /usr/lpp/ca1y
   cd /usr/lpp/ca1y
   jar xvf ca1y.zip
   ```
2. Decompress the .jar file. This is only required to use the SupportPac in batch, for example from a shell script, as the JVM outside of CICS does not provide an OSGi framework.
   ```
   jar xvf com.ibm.cics.ca1y_1.7.1.jar
   ```

Set the permissions for the installation directory and the files contained in them, ensuring in particular the CICS default user ID has read access to the file `/usr/lpp/ca1y/com.ibm.cics.ca1y_1.7.1.jar`.

Update the CICS JVM server profile

The SupportPac is written in Java and requires a CICS JVM server configured with two OSGi bundles.

1. If you do not already have a CICS JVMSERVER resource and JVM profile with OSGi support, follow the guidance in topic Configuring a JVM server for an OSGi application in the CICS Knowledge Center to create and configure them.
2. Modify the OSGI_BUNDLES property in the JVM profile to include the JavaMail file mail.jar supplied with CICS, and the SupportPac .jar file. For example:

   ```
   OSGI_BUNDLES=/CICS_install_dir/lib/pipeline/mail.jar,/usr/lpp
   /ca1y/com.ibm.cics.ca1y_1.7.1.jar
   ```

   Replace `CICS_install_dir` with the path of the CICS installation.

   Note: CICS TS V4.2 and V5.1 provide JavaMail version 1.4.3. Later versions are available from Java.net that contain fixes and enhancements. As of JavaMail version 1.5, the file name was changed from `mail.jar` to `javax.mail.jar`. If you decide to use this version, upload it to zFS and specify its location on the OSGI_BUNDLE property and remove reference to the CICS provided version.

3. Install the JVMSERVER resource and check it is enabled. The OSGi bundles will be installed as part of enabling the JVM server.
4. Optionally, add the JVMSERVER resource to a group in the CICS startup list so it will be installed automatically when CICS is next started using COLD or INITIAL options.

Define and install CICS resource definitions

The SupportPac requires a PROGRAM and a TRANSACTION resource to be installed in the CICS region in which either the events will be captured, or your applications will issue the LINK or START commands to send an email.

1. Define a PROGRAM resource with the following attributes, replacing DFH$JVMS with the name of the JVMSERVER resource installed previously:
    - **Name**: `CA1Y`
    - **Group**: `CA1Y`
    - **Description**: `CICS SupportPac CA1Y program`
    - **JVM**: `Yes`
    - **JVM class**: `com.ibm.cics.ca1y.Emit`
    - **JVM server**: `DFH$JVMS`
    - **Data location:** `ANY`
    - **Concurrency:** `THREADSAFE`
2. Define a TRANSACTION resource with the following attributes. The transaction must be a local transaction as described in topic Custom EP adapter. You may receive warnings saying transaction IDs starting with “C” are reserved for CICS. These can be ignored or choose a different ID.
    - **Name**: `CA1Y`
    - **Group**: `CA1Y`
    - **Description**: `CICS SupportPac CA1Y transaction`
    - **Program**: `CA1Y`
    - **Task data location**: `ANY`
    - **Dynamic**: `NO`
3. Install group `CA1Y`.
4. Optionally, add group `CA1Y` to a CICS startup list so these resources will be installed automatically when CICS is next started using COLD or INITIAL options.

Optionally create an email server properties file

In order for the SupportPac to connect to the mail server you need to provide information such as the server host name and port, and credentials such as a userid and password.

It is recommended you create a properties file on zFS with this information so it can be managed and secured separate to application artifacts such as event adapter configurations, event bindings, or programs. This file can then be read at runtime by specifying an import token as shown in the examples in later chapters.

1. Copy the example properties file `/usr/lpp/ca1y/examples/emailServer.properties` to a suitable directory.
2. Edit the new properties file to specify your mail server SMTP information. Note this file is supplied in UTF-8 format, but you could choose to use an EBCDIC format if required. The properties allowable in this file are described in SMTP mail properties on page 29.

   It is likely the SMTP server will be located within your company network. The SupportPac will communicate with the SMTP server using a TCP/IP socket from the JVM server to send the email, therefore ensure there are no firewall rules in place that prevent this access.

   For illustration purposes only, examples are included for Google Gmail, Yahoo! Mail Plus, Microsoft Windows Live and Hotmail. The author does not endorse these services, nor implies usage of them meets the terms and conditions from the service providers.

   Some mail providers such as Google Gmail require the user ID used to send the email is setup to enable SMTP access.
3. Set the permissions of this file such that only the user ID under which the SupportPac is going to be run has read access, and security administrators have read and write access. To set the user ID under which the SupportPac will run, use the options in the CICS event processing event adapter configuration.

Optionally install Apache Formatting Objects Processor (FOP)

Apache FOP V1.1 or later is only required if you need to transform XML to a PDF, PS, AFP, RTF, PNG or another format. Use the following steps to download the project, package it into an OSGi plugin project, and deploy it into the CICS JVM server using a CICS bundle.

Download and unzip Apache FOP:

1. Download Apache FOP by browsing to http://xmlgraphics.apache.org/fop/download.html and following the links to the binary download for FOP 1.1, or later.
2. Unzip the downloaded file `fop-1.1-bin.zip`. The directory `fop-1.1` will be created.

Create an OSGi plug-in project to contain Apache FOP:

1. In CICS Explorer SDK, select **File** > **New** > **Other** > **Plug-in from Existing JAR Archives** > **Next**
2. Select **Add External**. Navigate to directory `fop-1.1/build` and select `fop.jar` > **OK**
3. Select **Add External**. Navigate to directory `fop-1.1/libs` and select all the jar files > **OK**
4. Select **Next.**
5. In the "Enter the data required to generate the plug-in." panel, set the following:
   - **Project name:** `org.apache.fop`
   - **Plug-in Version:** `1.1.0`
   - **Plug-in name:** `FOP`
   - **Plug-in Provider:** `Apache`

- **Execution Environment:** `J2SE-1.5`
- **This plug-in is targeted to run with:** an OSGi framework: Equinox
- Untick **Unzip the JAR archives into the project**
- Select **Finish**

Create a CICS bundle project to deploy the OSGi plug-in project to CICS:

1. In CICS Explorer SDK, select **File** > **New** > **Other** > **CICS Bundle Project** > **Next** and set:
   - **Project name:** `Apache_FOP`
   - **Version:** `1.1.0`
2. Select the new CICS bundle project, then select **File** > **New** > **Other** > **Include OSGi Project in Bundle** > **Next** > **org.apache.fop** and set the following, replacing DFH$JVMS with the name of the JVMSERVER resource in which the SupportPac was configured to run in:
   - **JVM Server:** `DFH$JVMS`
   - Select **Finish**
3. Right mouse click on the CICS bundle project then select **Export Bundle Project to z/OS UNIX File System** and follow the prompts to export the bundle project to zFS.
4. Define a BUNDLE resource with the following attributes:
   - **Name:** `FOP`
   - **Group:** `CA1Y`
   - **Description:** `Apache FOP`
   - **Bundle Directory:** `<directory>/Apache_FOP_1.1.0`
5. Install the FOP BUNDLE resource and ensure it is enabled.
6. You will need to disable, then enable the JVMSERVER resource to ensure the SupportPac resolves its optional references to this new OSGi project.

Configure the SupportPac to use Apache FOP by adding the following property to either:

1. The JVM server profile:
   `-Djavax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactoryImpl`
2. Or the configuration passed to the SupportPac:
   `javax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactoryImpl`

Do not configure Apache FOP and Saxon to run in the same JVM server instance.

Optionally install Saxon XSLT and XQuery processor

Saxon is an open source project that provides an implementation of the XSLT 2.0, XPath 3.0, XQuery 3.0, and XSD 1.1 standards. This is only required if you need to transform an XML source-tree into another format. For example your application could create an XML document and use CA1Y, Saxon and a stylesheet to convert that into an HTML document.

Use the following steps to download an OSGi packaged version of Saxon:

1. Browse the Saxon HE (OSGi Bundle) site and download the latest .jar file, eg. saxon-he-9.5.1.5.jar.

2. Copy the .jar file to zFS in binary.
3. Add the fully qualified .jar file to the JVM server profile OSGI_BUNDLES setting.
4. Modify the OSGI_BUNDLES property in the JVM profile to include the fully qualified Saxon .jar file.
5. Restart the JVM server.

The version of Saxon available from http://saxon.sourceforge.net/ is not OSGi packaged and therefore not suitable for use in the JVM server.

Configure the SupportPac to use Saxon by adding the following property to either:

1. The JVM server profile:
   ```
   -Djavax.xml.transform.TransformerFactory=net.sf.saxon.TransformerFactoryImpl
   ```
2. Or the configuration passed to the SupportPac:
   ```
   javax.xml.transform.TransformerFactory=net.sf.saxon.TransformerFactoryImpl
   ```

Do not configure Apache FOP and Saxon to run in the same JVM server instance.

Chapter 3. Examples to send an email using the CA1Y event adapter

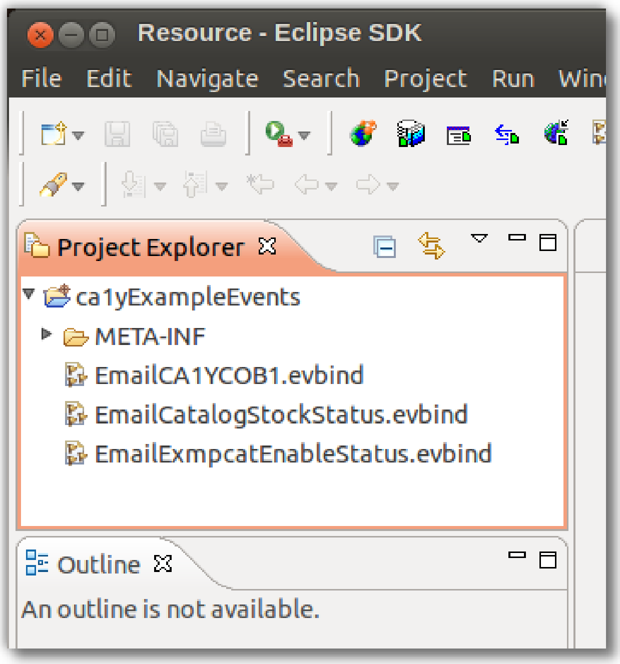
The tasks in this chapter assume you work through the examples in the order presented.

Send an email when the EXMPCAT file changes status

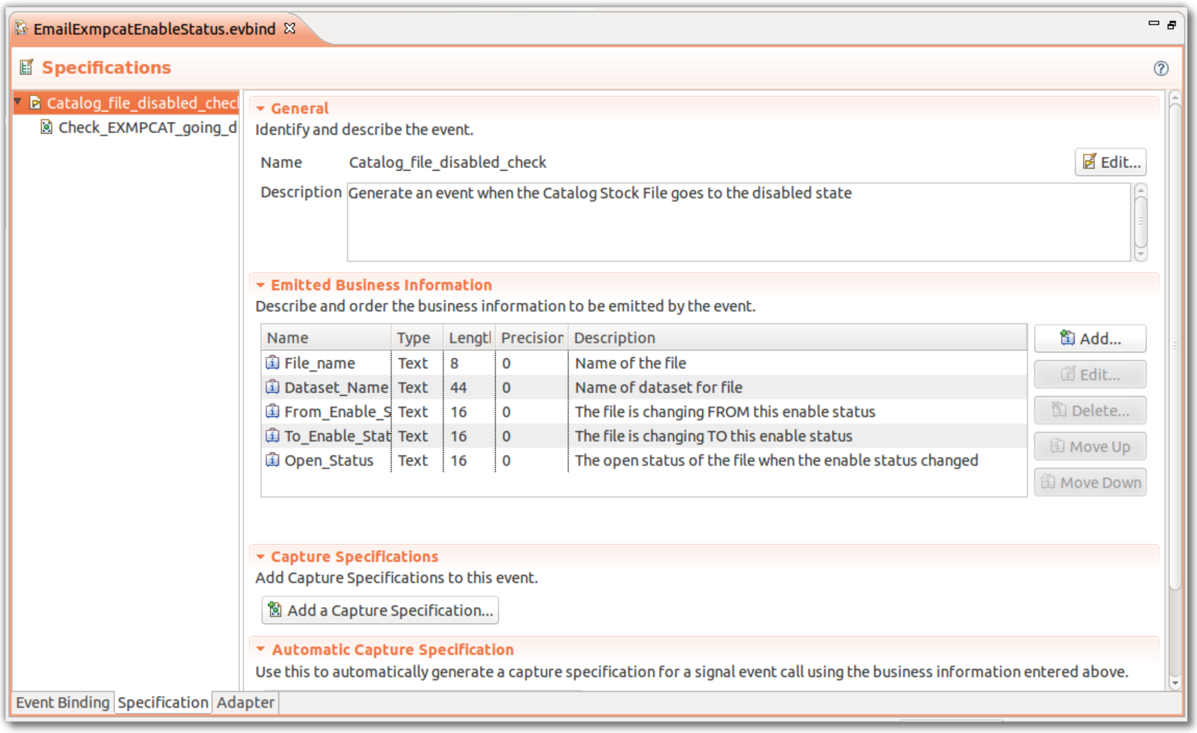
This example shows how to use the CICS Explorer to create an event binding file that captures a system event, in this case a change status to the file EXMPCAT, and to send an email containing data from the event.

Import the SupportPac example CICS bundle project into the CICS Explorer:

1. Start the CICS Explorer.
2. Change to the Resource perspective.
3. Select **File** > **Import** > **General > Existing Projects into Workspace** > **Next**
4. Select `Select archive file:` > **Browse**
5. Navigate to and select the `ca1y.zip` downloaded previously > **OK**
6. Select **ca1yExampleEvents** > **Finish**
7. Expand the ca1yExampleEvents CICS bundle project.


Resource - Eclipse SDK
File Edit Navigate Search Project Run Win
Project Explorer
ca1yExampleEvents
META-INF
EmailCA1YCOB1.evbind
EmailCatalogStockStatus.evbind
EmailExmpcatEnableStatus.evbind
Outline
An outline is not available.

8. Edit `EmailExmpcatEnableStatus.evbind` to start the Event Binding editor.
9. Select the **Specification** tab.
10. In the Emitted Business Information section is a list of fields that will be passed to the SupportPac event binding adapter. As we will see later, each of these fields can be inserted into the email.


EmailExmpcatEnableStatus.evbind

Specifications

Catalog_file_disabled_chec
Check_EXMPCAT_going_d

General
Identify and describe the event.

Name Catalog_file_disabled_check

Description Generate an event when the Catalog Stock File goes to the disabled state

Emitted Business Information
Describe and order the business information to be emitted by the event.

Name	Type	Lengtl	Precision	Description
File_name	Text	8	0	Name of the file
Dataset_Name	Text	44	0	Name of dataset for file
From_Enable_S	Text	16	0	The file is changing FROM this enable status
To_Enable_Stat	Text	16	0	The file is changing TO this enable status
Open_Status	Text	16	0	The open status of the file when the enable status changed

Capture Specifications
Add Capture Specifications to this event.

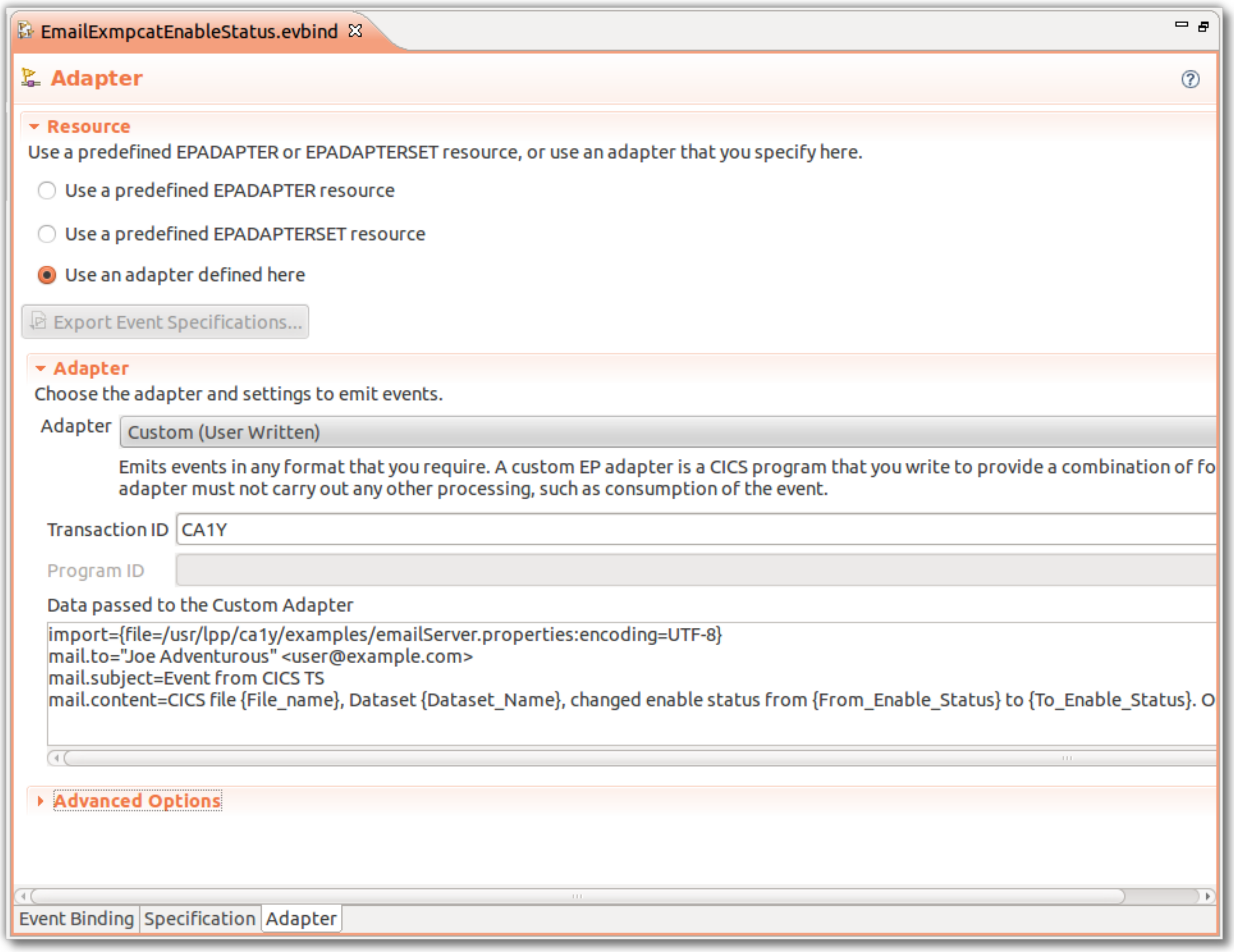
Automatic Capture Specification
Use this to automatically generate a capture specification for a signal event call using the business information entered above.

11. On the left hand pane, select `Check_EXMPCAT_going_disabled`
12. Select the **Capture Point** tab. Notice under the Capture Point section that this event is for `FILE ENABLE STATUS`.
13. Select the **Filtering** tab. Notice next to FILE* the operator is set to Equals and value set to `EXMPCAT`. This file will have been created if you set up the CICS catalog manager example application. You can change this to the name of any FILE resource in your system.

    Also notice TO_ENABLESTATUS is set to `Equals DISABLED`. Once these filter conditions are met the event will be captured by CICS.
14. Select the **Information Sources** tab to see how the emitted business information is obtained – ie. where the data is captured from.
15. Select the **Adapter** tab. This is where you define which adapter is to emit the event and its configuration. Note you can define adapters in separate configuration files to enable you to share common adapter configurations across many event bindings. For simplicity it is defined here.
16. Under the **Adapter** section, notice `Custom (User Written)` is selected. CICS provides a number of adapters to send the event via a WebSphere MQ message, to an HTTP server, etc. but we are going to call the SupportPac transaction CA1Y that was installed previously in Define and install CICS resource definitions on page 10.
17. The area under `Data passed to the Custom Adapter` is where you configure how the SupportPac should process this event. The properties are formatted as name=value pairs and are detailed in Properties on page 28.
18. The `import` property loads further properties from a file on zFS. Change the file path and name to match the mail server properties file created earlier as part of Optionally create an email server properties file on page 10. Alternately you could remove this import and define them all here.
19. Change the `mail.to` property to your email address.

20. Notice the `mail.content` property uses tokens to embed the business information items. The SupportPac will process the tokens and replace them when the email is prepared. The valid tokens are listed in Tokens on page 36.

21. Save the event binding and close the editor.



Use a predefined EPADAPTER or EPADAPTERSET resource, or use an adapter that you specify here.

- Use a predefined EPADAPTER resource
- Use a predefined EPADAPTERSET resource
- [✓ Use an adapter defined here]

Choose the adapter and settings to emit events.

Adapter: Custom (User Written)

Emits events in any format that you require. A custom EP adapter is a CICS program that you write to provide a combination of fo
adapter must not carry out any other processing, such as consumption of the event.

Transaction ID: CA1Y

Program ID:

Data passed to the Custom Adapter

```
import={file=/usr/lpp/ca1y/examples/emailServer.properties:encoding=UTF-8}
mail.to="Joe Adventurous" <user@example.com>
mail.subject=Event from CICS TS
mail.content=CICS file {File_name}, Dataset {Dataset_Name}, changed enable status from {From_Enable_Status} to {To_Enable_Status}. O
```

Deploy this CICS bundle to zFS and install it:

1. With the mouse positioned over the `ca1yExampleEvents` project, right click and select menu item **z/OS UNIX File System as Bundle Project**. Follow the wizard to deploy the project to a directory on zFS.

2. Create a BUNDLE resource and set the following:
   - **Name**: `CA1YEXAM`
   - **Description**: `SupportPac CA1Y example events`
   - **Bundle Directory**: `<zFS directory>`

3. Install the BUNDLE resource and check it is enabled.

CICS will now capture the event when the status of file EXAMPCAT changes to disabled, and emit the event to the SupportPac that will send it as an email.

If the email does not arrive check Troubleshooting on page 43.

Send an email when ordering from the catalog manager application

This example sends an email from the CICS provided catalog manager application when it issues a REWRITE command to update the VSAM file EXMPCAT, providing the stock levels are less than 24 and the value of back orders is 0.

Setup the application and customize the event:

1. Setup the CICS catalog manager example application provided with CICS TS.
2. Use the CICS Explorer to expand the `ca1yExampleEvents` project and edit `EmailCatalogStockStatus.evbind`.
3. Select the **Adapter** tab.
4. The `import` property loads further configuration from a file on zFS. Change the file path and name to match the mail server properties file created as part of Optionally create an email server properties file on page 10.
5. Change the `mail.to` property to your email address.
6. Notice the `mail.content` property is loaded from the file `/usr/lpp/ca1y/examples/emailTemplate.html`. You can use the CICS Explorer z/OS perspective, view z/OS UNIX Files, to navigate to this directory and edit the file.
7. Save the event binding and close the editor.
8. Deploy the updated `ca1yExampleEvents` project to zFS.
9. If the BUNDLE resource CA1YEXAM is currently enabled:
   - Disable BUNDLE `CA1YEXAM`.
   - Discard BUNDLE `CA1YEXAM`.
10. Install BUNDLE `CA1YEXAM`.

You are now ready to run the CICS catalog application.

1. At a 3270 terminal, run transaction `EGUI`.
2. Choose the action **List items**.
3. Select an item in the list to order.
4. Take note of Stock and On Order quantities. The event will only be captured when you raise an order and the stock level is below 24 and the On Order quantity is 0. You may need to make more than one order to achieve this state.

Once an order is made that matches the filter, you should receive an email.

Send an email when a program issues a SIGNAL EVENT

The example COBOL program CA1YCOB1 creates two containers with data, one with customer data and another with order information. The program then issues a SIGNAL EVENT command. An excerpt of the program is below:

```
 Identification Division.
 Program-id. CA1YCOB1.
 Environment division.
 Data division.
*****************************************************************
 Working-storage section.
 01 EVENT       PIC X(32) VALUE 'OrderPlaced                     '.
 01 CHANNEL-INFO.
     02 EVENT-CHANNEL          PIC X(16) VALUE 'MyChannel       '.
     02 CONTAINER-CUSTOMER     PIC X(16) VALUE 'Customer        '.
     02 CONTAINER-ORDER-PLACED PIC X(16) VALUE 'Order           '.
 01 CUSTOMER.
     02 CUST-NAME         PIC X(20) VALUE 'Joe Adventurous     '.
     02 CUST-ADDR1        PIC X(20) VALUE 'Rockclimbing Avenue '.
     02 CUST-EMAIL        PIC X(20) VALUE 'user@example.com    '.
 01 ORDER-PLACED.
     02 ORDER-NUMBER      PIC 9(08) VALUE 12345678.
     02 ITEM-QUANTITY     PIC 9(03) VALUE 1.
     02 ITEM-DESCRIPTION  PIC X(20) VALUE 'Rope                '.
*****************************************************************
 Linkage section.
 Procedure division.
 Main-program section.
* ---------------------------------------------------------------
* Create the container for customer information.
* ---------------------------------------------------------------
     EXEC CICS PUT CONTAINER(CONTAINER-CUSTOMER)
         CHANNEL(EVENT-CHANNEL)
         FROM(CUSTOMER) CHAR
     END-EXEC.
* ---------------------------------------------------------------
* Create the container for order information.
* ---------------------------------------------------------------
     EXEC CICS PUT CONTAINER(CONTAINER-ORDER-PLACED)
         CHANNEL(EVENT-CHANNEL)
         FROM(ORDER-PLACED) CHAR
     END-EXEC.
* ---------------------------------------------------------------
* Signal the event has occurred.
* ---------------------------------------------------------------
     EXEC CICS SIGNAL EVENT(EVENT)
         FROMCHANNEL(EVENT-CHANNEL)
     END-EXEC.

     EXEC CICS RETURN END-EXEC.
```

Use the following steps to edit, compile and install the program:

1. Copy the program source from `/usr/lpp/ca1y/examples/CA1YCOB1` to a COBOL source data set.

   For example in a UNIX System services shell to convert the source to IBM-1047 code page and copy it to the data set HLQ.CA1Y.COBOL:

   ```
   cd /usr/lpp/ca1y/examples
   iconv -f ISO8859-1 -t IBM-1047 CA1YCOB1 > CA1YCOB1.E
   cp CA1YCOB1.E "//'HLQ.CA1Y.COBOL(CA1YCOB1)'"
   ```
2. Edit the program source and change the value of field CUST-EMAIL to be your email address.
3. Ensure that special characters, such as @ { } ( ) in the STRING command, are correct for the program compile options.
4. If the program is compiled with a CCSID that is different than is specified in the CICS SIT parameter LOCALCCSID, you will need specify the programs' CCSID by adding the FROMCCSID parameter to the PUT CONTAINER command, and adding the INTOCCSID parameter to the GET CONTAINER command.
5. Compile the program using your local COBOL compile procedures.
6. Make the object library available to the CICS region if it is not already.
7. Define and install a PROGRAM definition with:
   - **Name**: `CA1YCOB1`

   - **Task data location**: `ANY`

8. Define and install a TRANSACTION definition with:
   - **Name**: `COB1`
   - **Program:** `CA1YCOB1`

Customize the event:

1. Use the CICS Explorer to expand the `ca1yExampleEvents` project and edit `EmailCA1YCOB1.evbind`.
2. Select the **Specification** tab, then select the `OrderPlaced` specification.
   - Notice the **Capture Point** tab shows this is a SIGNAL EVENT command.
   - The **Filtering** tab shows the event name is `OrderPlaced`.
   - The **Information Sources** tab shows how each of the business information items are obtained from the programs' two containers.
3. Select the **Adapter** tab.
4. The `import` property loads further properties from a file on zFS. Change the file path and name to match the mail server file created as part of Optionally create an email server properties file on page 10.
5. Notice the `mail.to, mail.subject`, and `mail.content` properties all have values that include tokens to embed the business information items extracted from the containers created by the program.
6. Save the event binding and close the editor.
7. Deploy the `ca1yExampleEvents` project to zFS.
8. If the BUNDLE resource CA1YEXAM is currently enabled:
   - Disable BUNDLE `CA1YYEXAM`.
   - Discard BUNDLE `CA1YYEXAM`.
9. Install BUNDLE `CA1YEXAM`.

Your can now run transaction `COB1` at a terminal to execute program `CA1YCOB1` and issue the event. The event will be captured and the SupportPac custom adapter will format the email and send it to the SMTP server. Your email application will then pick up the email from the SMTP server.

Chapter 4. Examples to send an email by linking to program CA1Y

These examples illustrates how to send an email by using the LINK command to start SupportPac program CA1Y.

Send an email with an attachment using a single container

This example creates a container named `CA1Y` with properties for the email headers, body, an attachment, and mail server. The example then issues a LINK command to program `CA1Y`. The SupportPac sends the email and returns a container named `CA1YRESPONSE` with `true` if the email was sent, otherwise `false`.

```
 Identification Division.
 Program-id. CA1YCOB2.
 Environment division.
 Data division.
*****************************************************************
 Working-storage section.
 01 CONFIG.
     02 CONFIG-CHANNEL-NAME    PIC X(16) VALUE 'MyChannel       '.
     02 CONFIG-CONTAINER-NAME  PIC X(16) VALUE 'CA1Y            '.
     02 CONFIG-DATA-LENGTH     PIC 9(8) COMP VALUE 0.
     02 CONFIG-DATA            PIC X(2048) VALUE SPACES.
 01 CR                         PIC X(1)  VALUE X'25'.
*****************************************************************
 Linkage section.
 Procedure division.
 Main-program section.
* ---------------------------------------------------------------
* Create a container with the email headers, body, attachment,
* and import for the email server properties.
* ---------------------------------------------------------------
     STRING
         'mail.to="Joe Bloggs" <joe.bloggs@example.com>' CR
         'mail.subject=Email from {REGION_APPLID}' CR
         'mail.content=This email was sent '
         'on {datetime=EEE, d MMM yyyy HH:mm:ss Z} '
         'from transaction id {TASK_TRANID}, '
         'user id {TASK_USERID}, '
         'program {TASK_PROGRAM}, '
         'task number {TASK_NUMBER}, '
         'CICS SYSID {REGION_SYSID}, '
         'CICS APPLID {REGION_APPLID}.' CR
         'attachment={file=/usr/lpp/ca1y/examples/'
         'picture.png:binary}' CR
         'import.private={file=/usr/lpp/ca1y/examples/'
         'emailServer.properties:encoding=UTF-8}' CR
         X'00'
         DELIMITED BY SIZE INTO CONFIG-DATA.

     INSPECT CONFIG-DATA TALLYING CONFIG-DATA-LENGTH
         FOR CHARACTERS BEFORE INITIAL X'00'.

     EXEC CICS PUT CONTAINER(CONFIG-CONTAINER-NAME)
         CHANNEL(CONFIG-CHANNEL-NAME)
         FROM(CONFIG-DATA) FLENGTH(CONFIG-DATA-LENGTH) CHAR
     END-EXEC.

     EXEC CICS LINK PROGRAM('CA1Y')
         CHANNEL(CONFIG-CHANNEL-NAME)
     END-EXEC.

     EXEC CICS RETURN END-EXEC.
```

Use the following steps to edit, compile and install the program:

1. Copy the program source from `/usr/lpp/ca1y/examples/CA1YCOB2` to a COBOL source data set.
2. Edit the program and change the `mail.to` property to your email address, and `import.private` property to match the mail server file created as part of Optionally create an email server properties file on page 10.
3. Ensure that special characters, such as @ { } ( ) in the STRING command, are correct for the program compile options.
4. If the program is compiled with a CCSID that is different than is specified in the CICS SIT parameter LOCALCCSID, you will need specify the programs CCSID by adding the FROMCCSID parameter to the PUT CONTAINER command, and adding the INTOCCSID parameter to the GET CONTAINER command.
5. When linking to program CA1Y it expects the CONTAINER named CA1Y to include the initial set of properties.
6. Notice the property `attachment` that will result in the email having an attachment loaded from the named file.
7. Compile the program using your local COBOL compile procedures.
8. Make the object library available to the CICS region if it is not already.
9. Define and install a PROGRAM definition.
    - **Name**: `CA1YCOB2`
10. Define and install a TRANSACTION definition.
    - **Name**: `COB2`
    - **Program**: `CA1YCOB2`

You can now run transaction COB2 that will prepare the containers and LINK to program CA1Y that will prepare and send the email.

Send an email with an attachment using multiple containers

This example creates separate containers for the email recipient, subject, content and attachment that may be more convenient for the application developer than creating a single configuration container.

It also creates a container named `CA1Y` that includes tokens that will be replaced with the contents of the named containers. The example then issues a LINK command to program `CA1Y`. The SupportPac sends the email and returns a container named `CA1YRESPONSE` with `true` if the email was sent, otherwise `false`.

```
 Identification Division.
 Program-id. CA1YCOB3.
 Environment division.
 Data division.
******************************************************************
 Working-storage section.
 01 CONFIG.
     02 CONFIG-CHANNEL-NAME   PIC X(16)  VALUE 'CA1Y            '.
     02 CONFIG-CONTAINER-NAME PIC X(16)  VALUE 'CA1Y            '.
     02 TO-CONTAINER-NAME     PIC X(16)  VALUE 'TO              '.
     02 SUBJECT-CONTAINER-NAME PIC X(16) VALUE 'SUBJECT         '.
     02 CONTENT-CONTAINER-NAME PIC X(16) VALUE 'CONTENT         '.
     02 ATTACH1-CONTAINER-NAME PIC X(16) VALUE 'ATTACH1         '.
 01 WORKAREA.
     02 WORKAREA-DATA-LENGTH  PIC 9(8) COMP VALUE 0.
     02 WORKAREA-DATA         PIC X(1024) VALUE SPACES.
 01 CR                        PIC X(1)   VALUE X'25'.
*****************************************************************
 Linkage section.
 Procedure division.
 Main-program section.
* --------------------------------------------------------------
* Create container for mail configuration
* --------------------------------------------------------------
     STRING
         'import.private={file=/usr/lpp/ca1y/examples/'
         'emailServer.properties:encoding=UTF-8}' CR
         'mail.to={' TO-CONTAINER-NAME '}' CR
         'mail.subject={' SUBJECT-CONTAINER-NAME '}' CR
         'mail.content={' CONTENT-CONTAINER-NAME '}' CR
         'attachment={mime=application/octet-stream}'
         '{' ATTACH1-CONTAINER-NAME '}' CR
         X'00'
         DELIMITED BY SIZE INTO WORKAREA-DATA.

     MOVE 0 TO WORKAREA-DATA-LENGTH.
     INSPECT WORKAREA-DATA TALLYING WORKAREA-DATA-LENGTH
         FOR CHARACTERS BEFORE INITIAL X'00'.

     EXEC CICS PUT CONTAINER(CONFIG-CONTAINER-NAME)
         CHANNEL(CONFIG-CHANNEL-NAME)
         FROM(WORKAREA-DATA) FLENGTH(WORKAREA-DATA-LENGTH) CHAR
     END-EXEC.
* --------------------------------------------------------------
* Create container for mail recipient
* --------------------------------------------------------------
     STRING '"Joe Bloggs" <joe.bloggs@example.com>' X'00'
         DELIMITED BY SIZE INTO WORKAREA-DATA.

     MOVE 0 TO WORKAREA-DATA-LENGTH.
     INSPECT WORKAREA-DATA TALLYING WORKAREA-DATA-LENGTH
         FOR CHARACTERS BEFORE INITIAL X'00'.

     EXEC CICS PUT CONTAINER(TO-CONTAINER-NAME)
         CHANNEL(CONFIG-CHANNEL-NAME)
         FROM(WORKAREA-DATA) FLENGTH(WORKAREA-DATA-LENGTH) CHAR
     END-EXEC.
* --------------------------------------------------------------
* Create container for subject
* --------------------------------------------------------------
     STRING 'Email from {REGION_APPLID}' X'00'
         DELIMITED BY SIZE INTO WORKAREA-DATA.

     MOVE 0 TO WORKAREA-DATA-LENGTH.
     INSPECT WORKAREA-DATA TALLYING WORKAREA-DATA-LENGTH
         FOR CHARACTERS BEFORE INITIAL X'00'.

     EXEC CICS PUT CONTAINER(SUBJECT-CONTAINER-NAME)
         CHANNEL(CONFIG-CHANNEL-NAME)
         FROM(WORKAREA-DATA) FLENGTH(WORKAREA-DATA-LENGTH) CHAR
     END-EXEC.
* --------------------------------------------------------------
* Create container for content
* --------------------------------------------------------------
     STRING 'This email was sent '
         'on {datetime=EEE, d MMM yyyy HH:mm:ss Z} '
         'from transaction id {TASK_TRANID}, '
         'user id {TASK_USERID}, '
         'program {TASK_PROGRAM}, '
         'task number {TASK_NUMBER}, '
         'CICS SYSID {REGION_SYSID}, '
         'CICS APPLID {REGION_APPLID}.'
         X'00'
         DELIMITED BY SIZE INTO WORKAREA-DATA.
```

```
       MOVE 0 TO WORKAREA-DATA-LENGTH.
       INSPECT WORKAREA-DATA TALLYING WORKAREA-DATA-LENGTH
           FOR CHARACTERS BEFORE INITIAL X'00'.

       EXEC CICS PUT CONTAINER(CONTENT-CONTAINER-NAME)
           CHANNEL(CONFIG-CHANNEL-NAME)
           FROM(WORKAREA-DATA) FLENGTH(WORKAREA-DATA-LENGTH) CHAR
       END-EXEC.
      * ---------------------------------------------------------------
      * Create container for attachment
      * ---------------------------------------------------------------
       STRING X'0102030405060708090A0B0C0D0E0F'
           X'00'
           DELIMITED BY SIZE INTO WORKAREA-DATA.

       MOVE 0 TO WORKAREA-DATA-LENGTH.
       INSPECT WORKAREA-DATA TALLYING WORKAREA-DATA-LENGTH
           FOR CHARACTERS BEFORE INITIAL X'00'.

       EXEC CICS PUT CONTAINER(ATTACH1-CONTAINER-NAME)
           CHANNEL(CONFIG-CHANNEL-NAME)
           FROM(WORKAREA-DATA) FLENGTH(WORKAREA-DATA-LENGTH) BIT
       END-EXEC.
      * ---------------------------------------------------------------
      * Emit the mail message
      * ---------------------------------------------------------------
       EXEC CICS LINK PROGRAM('CA1Y')
           CHANNEL(CONFIG-CHANNEL-NAME)
       END-EXEC.

       EXEC CICS RETURN END-EXEC.
```

1. Copy the program source from `/usr/lpp/ca1y/examples/CA1YCOB3` to a data set.
2. Edit the program and change the mail recipient to your email address, and import.private property to match the mail server file created earlier as part of Optionally create an email server properties file on page 10.
3. Ensure that special characters, such as @ { } ( ) in the STRING command, are correct for the program compile options.
4. If the program is compiled with a CCSID that is different than is specified in the CICS SIT parameter LOCALCCSID, you will need specify the programs CCSID by adding the FROMCCSID parameter to the PUT CONTAINER command, and adding the INTOCCSID parameter to the GET CONTAINER command.
5. Notice the property `attachment` that will result in the email having an attachment loaded from the container named `ATTACH1`.
6. Compile the program using your local COBOL compile procedures.
7. Make the object library available to the CICS region if it is not already.
8. Define and install a PROGRAM definition.
    - **Name**: `CA1YCOB3`
9. Define and install a TRANSACTION definition.
    - **Name**: `COB3`
    - **Program**: `CA1YCOB3`

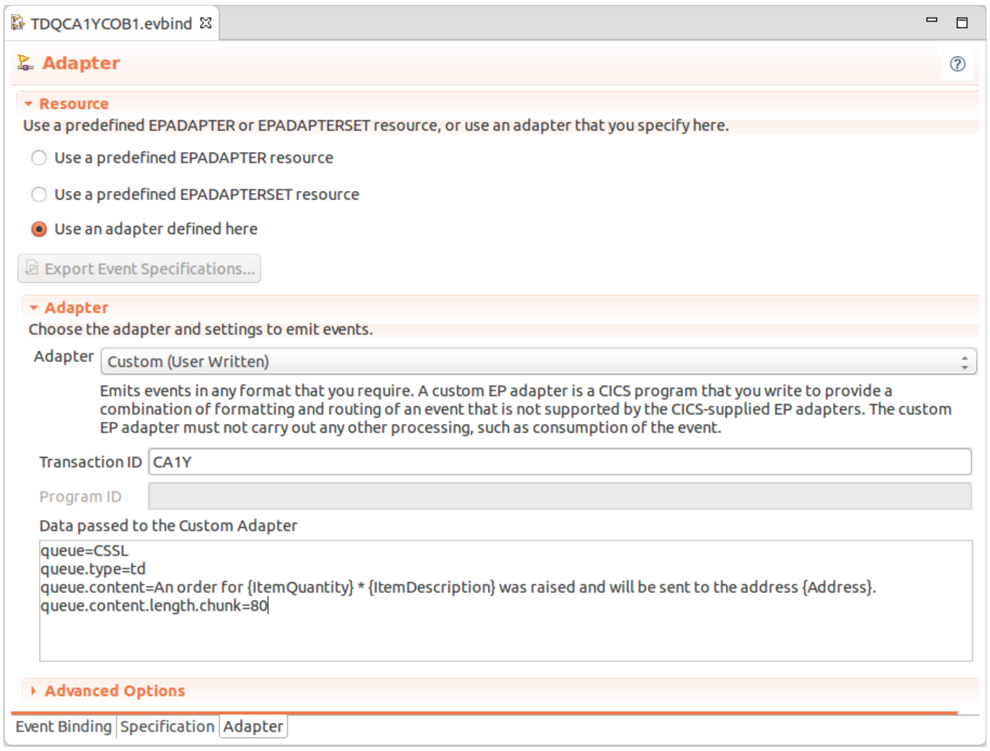
You can now run transaction COB3 that will prepare a set of containers and LINK to program CA1Y that will prepare and send the email.

Chapter 5. Example to write to a TD queue using the CA1Y event adapter

This example illustrates how to emit a CICS event to the temporary data (TD) queue CSSL that is typically directed to the CICS MSGUSR log. This could be useful for automation tools to make use of the event or to simply audit events.

First follow Examples to send an email using the CA1Y event adapter on page 14.

Within the **ca1yExampleEvents** project, edit the event binding sample `TDQCA1YCOB1.evbind` then select the **Adapter** tab.



Use a predefined EPADAPTER or EPADAPTERSET resource, or use an adapter that you specify here.

( ) Use a predefined EPADAPTER resource

( ) Use a predefined EPADAPTERSET resource

(●) Use an adapter defined here

Choose the adapter and settings to emit events.

Adapter: Custom (User Written)

Emits events in any format that you require. A custom EP adapter is a CICS program that you write to provide a combination of formatting and routing of an event that is not supported by the CICS-supplied EP adapters. The custom EP adapter must not carry out any other processing, such as consumption of the event.

Transaction ID: CA1Y

Program ID:

Data passed to the Custom Adapter

```
queue=CSSL
queue.type=td
queue.content=An order for {ItemQuantity} * {ItemDescription} was raised and will be sent to the address {Address}.
queue.content.length.chunk=80
```

Notice in the **Data passed to the Custom Adapter** section the following properties will set the name of the queue to CSSL, the type of queue to TD, the content to write to the queue, and to write multiple records to the queue if the content is over 80 characters.

```
queue=CSSL

queue.type=td

queue.content=An order for {ItemQuantity} * {ItemDescription}
was raised and will be sent to the address {Address}.

queue.content.length.chunk=80
```

Run transaction at COB1 at a terminal. The following two records will be written to CSSL that will likely be redirected to MSGUSR:

```
An order for 1 * Rope was raised and will be sent to the
address Rockclimbing Av

enue.
```

Chapter 6. Example to send an event to an HTTP server

CICS provides an HTTP adapter that is able to format an event as XML and send it to an HTTP server. If you have a robust network between CICS and the HTTP server, and the adapter provides suitable formatting for your event, it is recommended you use this support as its performance will be better than this SupportPac.

The SupportPac however provides a more flexible approach to formatting the event and is able to retry the emission if there are network failures.

The example below shows how to format the event as JSON (JavaScript Object Notation), and if there are network failures retry up to 3 times.

1. Create a URIMAP named MYSERV. Specify the usage as client, and the HTTP server address, path, port, scheme, and security details.
2. Copy the ca1yExampleEvents CICS bundle project used in Examples to send an email using the CA1Y event adapter on page 14 and give it the name ca1yExampleEventsHTTP.
3. Rename EmailExmpcatEnableStatus.evbind to HTTPExmpcatEnableStatus.evbind and delete the other .evbind files.
4. Edit HTTPExmpcatEnableStatus.evbind and in event binding editor Adapter tab, enter the following into the **Data passed to the custom adapter** section:

```
http.urimap=MYSERV
http.content={noinnertokens}{json}
http.retry=3
http.retrydelay=1000
```

5. Save the event binding.
6. Export the CICS bundle to zFS and create and install a BUNDLE definition.

Now when the status of file EXMPCAT is changed to disabled, the event will be captured, and the SupportPac started. The SupportPac will format the event into a JSON format and sent it to the HTTP server configured in the URIMAP.

Chapter 7. Example to convert XML to a PDF document by linking to program CA1Y

This example illustrates how to convert an XML document to a PDF document using an XSLT stylesheet.

The container named `CA1Y` is created with three properties:

1. `org.apache.xalan.processor.TransformerFactoryImpl` to specify the Xalan-Java XSLT processor should be used for XSLT transformation.
2. `MyXSLT` that loads the XSLT document from zFS.
3. `MyPDF` that specifies the XML to convert into a PDF and return it back to the program in the container named `PDF`. The PDF documented is created by converting the XML from type text/xml to application/pdf using the XSLT.

The SupportPac program CA1Y is then called to create the PDF document. The PDF document is then retrieved from the `PDF` container.

```
 Identification Division.
 Program-id. CA1YCOB4.
 Environment division.
 Data division.
*****************************************************************
 Working-storage section.
 01 CONFIG.
     02 CONFIG-CHANNEL-NAME   PIC X(16) VALUE 'MyChannel       '.
     02 CONFIG-CONT-NAME      PIC X(16) VALUE 'CA1Y            '.
     02 CONFIG-DATA-LENGTH    PIC 9(8) COMP VALUE 0.
     02 CONFIG-DATA           PIC X(2048) VALUE SPACES.
 01 PDF-DOC.
     02 PDF-CONT-NAME         PIC X(16) VALUE 'PDF             '.
     02 PDF-DATA-LENGTH       PIC 9(8) COMP VALUE 0.
     02 PDF-DATA              PIC X(10240) VALUE SPACES.
 01 RESPONSE.
     02 RESPONSE-CONT-NAME    PIC X(16) VALUE 'CA1YRESPONSE    '.
     02 RESPONSE-DATA-LENGTH  PIC 9(8) COMP VALUE 0.
     02 RESPONSE-DATA         PIC X(16) VALUE SPACES.
 01 CR                        PIC X(1)  VALUE X'25'.
*****************************************************************
 Linkage section.
 Procedure division.
 Main-program section.
* --------------------------------------------------------------
* Create a container with properties used by the SupportPac.
* --------------------------------------------------------------
     STRING
      'javax.xml.transform.TransformerFactory='
      'org.apache.xalan.processor.TransformerFactoryImpl' CR
      'MyXSLT={file=/usr/lpp/ca1y/examples/helloWorld.xslt'
      ':encoding=UTF-8}' CR
*
      'MyPDF={responsecontainer=' PDF-CONT-NAME '}'
      '{mime=text/xml:to=application/pdf:xslt=MyXSLT}'
      '<?xml version="1.0" encoding="UTF-8" ?>'
      '<name>Joe Bloggs</name>' CR
*
      X'00'
      DELIMITED BY SIZE INTO CONFIG-DATA.

     INSPECT CONFIG-DATA TALLYING CONFIG-DATA-LENGTH
         FOR CHARACTERS BEFORE INITIAL X'00'.

     EXEC CICS PUT CONTAINER(CONFIG-CONT-NAME)
         CHANNEL(CONFIG-CHANNEL-NAME)
```

```
               FROM(CONFIG-DATA) FLENGTH(CONFIG-DATA-LENGTH) CHAR
           END-EXEC.
      * ---------------------------------------------------------------
      * Link to the SupportPac.
      * ---------------------------------------------------------------
           EXEC CICS LINK PROGRAM('CA1Y')
               CHANNEL(CONFIG-CHANNEL-NAME)
           END-EXEC.
      * ---------------------------------------------------------------
      * Get the SupportPac response.
      * ---------------------------------------------------------------
           COMPUTE RESPONSE-DATA-LENGTH = LENGTH OF RESPONSE-DATA.
           EXEC CICS GET CONTAINER(RESPONSE-CONT-NAME)
               CHANNEL(CONFIG-CHANNEL-NAME)
               INTO(RESPONSE-DATA) FLENGTH(RESPONSE-DATA-LENGTH)
           END-EXEC.
      * ---------------------------------------------------------------
      * Get the PDF.
      * ---------------------------------------------------------------
           COMPUTE PDF-DATA-LENGTH = LENGTH OF PDF-DATA.
           EXEC CICS GET CONTAINER(PDF-CONT-NAME)
               CHANNEL(CONFIG-CHANNEL-NAME)
               INTO(PDF-DATA) FLENGTH(PDF-DATA-LENGTH)
           END-EXEC.

           EXEC CICS RETURN END-EXEC.
```

1. Copy the program source from `/usr/lpp/ca1y/examples/CA1YCOB4` to a COBOL source data set.
2. Edit the program source if you need to modify the property MyXSLT with a different location for the XSLT file.
3. Compile the program using your local COBOL compile procedures.
4. Make the object library available to the CICS region if it is not already.
5. Define and install a PROGRAM definition.
    - **Name**: `CA1YCOB4`
6. Define and install a TRANSACTION definition.
    - **Name**: `COB4`
    - **Program**: `CA1YCOB4`

You can now run transaction COB4 to create the PDF document. Use CEDF to step through the CICS API calls. You could modify the program to send the PDF as an email attachment, send it to a web services client, or save it to DB2.

Chapter 8. Properties

The SupportPac acts upon information passed to it as properties.

Place the properties in one of the following locations:

- If the SupportPac will be started as a CICS event custom adapter, place the properties in the **Data passed to the Custom Adapter** field in the adapter tab in the CICS Explorer event adapter editor or event binding editor.
- If the SupportPac will be started as a result of a CICS policy being triggered, place the properties in the **Data passed to the Custom Adapter** field in the adapter tab in the CICS Explorer event adapter editor.
- If the SupportPac will be started by a LINK CHANNEL() or START CHANNEL() command, place the properties in a character container named `CA1Y`. Ensure the container is created with the CCSID of your program by specifying FROMCCSID or FROMCODEPAGE parameters on the PUT CONTAINER command.
- If the SupportPac will be started by a LINK COMMAREA() or START FROM() command, place the properties in the commarea. The content is required to be in the CICS region local CCSID in which CA1Y executes.

Properties are specified using the format name=value and are case sensitive. Each property should end with a new line or line feed character.

To split a value over several lines, use the \ continuation character at the end of the line. To insert a new line character in a property value use \n. Blank lines, and lines where the first non-blank character is # or ! are used for comments and are ignored. Further formatting rules and an alternative XML notation are described in the Java Properties class load method at java.util.Properties load.

If a property is specified more than once, the last instance will take effect.

It is advisable to avoid property names that clash with the names of tokens.

Property values can contain tokens that are replaced as described in Tokens on page 36. Properties are processed in alphanumeric name order to resolve the tokens.

Additional properties can be imported from an external source such as a file that can be maintained independent to the event adapter configuration or program.

A combination of properties could be used to, for example, send an email and write to a temporary data queue. However, when using CICS events you may prefer to define these in separate event adapter configurations to maintain their independence in failure scenarios. Also from CICS TS V5.1 you can emit one captured event to multiple adapters using adapter sets.

General properties

Use these properties to import properties from other sources, to link to a program when there is a failure to process the request, and to define the regular expression to identify tokens.

Property	***Usage and examples***	
`import=`*`value`*	Import additional properties.	
	`import={file=/path/standard.properties}`	Properties are imported from the named file using the JVM default file encoding.
	`import={file=/usr/lpp/ca1y/examples/emailServer.properties:encoding=UTF-8}`	Properties are imported from the named file using the UTF-8 encoding.

Property	***Usage and examples***	
	`import={doctemplate=MyTemplate}`	Properties are imported from a CICS DOCTEMPLATE resource that is cached in memory upon first use.
`import.private=`*value*	Import additional properties and prevent the values of these properties from appearing in log files, or being copied using tokens. This is useful if the tokens include passwords such as for mail servers. `import.private={file=/usr/lpp/ca1y/examples/emailServer.properties:encoding=UTF-8}`	
`onfailure=`*program*	Specify the CICS program the SupportPac will link to if there is a failure to process the request. Your program can access the containers passed to the SupportPac plus those created by specifying the `responsecontainer` token in a property.	

	`http.content=Hello from CICS. {responsecontainer=HTTPCONTENT}`
	`http.urimap=HTTPSERV`
	`onfailure=ERRLOG`
		An event is sent to an HTTP server defined in a URIMAP. If there is a failure, the SupportPac links to program ERRLOG to log the HTTPCONTENT container to DB2.

`token.regex=`*pattern*	Specify a regular expression pattern to identify the start and end of a token. Valid patterns are detailed in the Java Pattern class. The default pattern starts a token with an opening curly bracket { and ends with a closing curly bracket } For example if you need to replace tokens in JSON data, you will need to change this pattern as JSON uses curly brackets to denote arrays.	
	`token.regex=\\$\\{(.+?)\\}`	A token is defined to start with ${ and end with }

SMTP mail properties

Use these properties to send an email to a SMTP mail server. Most of the properties are used by the JavaMail API to construct email headers and content, and to interact with a mail server. These and other advanced properties to secure the connection are detailed in Package com.sun.mail.smtp.

You are required to set at least one of the mail.to, mail.cc, or mail.bcc properties.

To add an attachment to the email, define a property with the attachment contents and include a mime token to set the MIME type. The attachment name can be set using the name token as shown in Examples using tokens for email attachments on page 42.

If the email could not be delivered to one of the recipients, for example due to an incorrect email address, the SMTP server will send a failure report to the senders' email address specified in the mail.from property. Delivery failures may take some time to arrive due to interactions with relay servers.

Property name	***Usage and examples***
`mail.bcc`	Email address list in RFC 822 syntax of who should receive the email as a blind carbon copy. `email.bcc=user@example.com`
`mail.cc`	Email address list in RFC 822 syntax of who should receive the email as a carbon copy. `email.cc=user@example.com`
`mail.content`	Content of the mail, typically plain text or HTML. `mail.content=Hello from CICS.`

Property name	***Usage and examples***
	`mail.content=<html><h1>Hello from CICS.</h1></html>`
`mail.from`	Email address in RFC 822 syntax of who sent the email. `email.from=cics@example.com`
`mail.sentdate`	CICS ABSTIME, returned via a EXEC CICS ASKTIME ABSTIME(), to use for the email sent date and time. If not specified and CA1Y is started by a CICS event, the date and time of the CICS event is used, otherwise the Java date and time used. `mail.sentdate=003609661860917`
`mail.host`	SMTP server host name. `mail.host=smtp.example.com`
`mail.password`	SMTP user password. `mail.password=mypassword`
`mail.reply.to`	Email address in RFC 822 syntax of who replies should be sent. `email.reply.to=helpdesk@example.com`
`mail.smtp.auth`	If true, the userid and password will be used to authenticate with the mail server. `mail.smtp.auth=true`
`mail.smtp.port`	SMTP server port number. The default is 25. `mail.smtp.port=25`
`mail.subject`	One line of text describing the email subject. `mail.subject=Example email`

`mail.to`	Email address list in RFC 822 syntax of who should receive the email.
	`mail.to=user@example.com`
	`mail.to="Joe Bloggs" <j.bloggs@example.com>`
	`mail.to="Joe Bloggs" <j.bloggs@example.com>, "A N Other" <a.n.other@example.com>`

`mail.transport.protocol`	Mail server protocol. `mail.transport.protocol=smtp`
`mail.user`	SMTP user name. `mail.user=joe.bloggs@example.com`

Saxon and Apache FOP properties

Use these properties to specify the Java class to use for XSLT transformations, and to specify additional configuration for the Apache FOP.

Property name	***Meaning and examples***	
`fop.config.file=`*value*	Apache FOP configuration file as defined at http://xmlgraphics.apache.org/fop/1.1/configuration.html `fop.config.file=/path/config.xml`	
`javax.xml.transform.TransformerFactory=`*class*	Specify the Java class name to use to transform XSLT. `javax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactoryImpl`	Use the Apache FOP transformer for XSLT processing.

Property name	*Meaning and examples*	
	`javax.xml.transform.TransformerFactory=net.sf.saxon.TransformerFactoryImpl`	Use the Saxon transformer for XSLT processing.

Queue properties

Use these properties to write content to a CICS temporary storage (TS) or temporary data (TD) queue.

Property	***Usage and examples***	
`queue=`*qName*	Required. The queue name. Follow the queue naming rules defined by CICS. For TS queues see QNAME(name). For TD queues see QUEUE(name).	
	`queue=MyQueue`	
`queue.content=`*value*	Required. Text content to write to the queue.	
	`queue.content=Hello from CICS.`	
	`queue.content={file=//DD:MYDATAS(MEMBER)}`	The text is retrieved from the named dataset member to write to the queue.
	`queue.content={file=/path/picture.png:binary}`	Binary content is retrieved from the named zFS file to write to the queue.
`queue.content.encoding=`*value*	Content is written to the queue using the specified encoding. If not specified the CICS region local CCSID will be used.	
	`queue.content.encoding=Cp1047`	
`queue.content.length.chunk=`*maxLength*	The maximum length of content to write in a single queue record. If the content is beyond this length, multiple queue records will be written. In this situation it is possible other CICS tasks may write records to the same queue and become interspersed with records written by this task. If this property is not specified, the content will be written as a single record and truncation may occur.	
	`queue.content.length.chunk=32763`	
`queue.content.length.max=`*maxLength*	The maximum length of content to write to the queue. Content over this length is truncated.	
	`queue.content.length.max=32763`	
`queue.sysid=`*sysID*	The remote system identifier (SYSID) where the queue resides. If this property is not specified, the queue is assumed to be local to the CICS region that is executing the SupportPac.	
	`queue.sysid=SYS1`	
`queue.ts.storage=`*type*	Specify the queue is to be created in main or auxiliary storage if it does not already exist. If not specified, main is assumed.	
	`queue.ts.storage=main`	
	`queue.ts.storage=auxiliary`	
`queue.type=`*value*	Specify the queue type. If not specified, temporary storage is assumed.	
	`queue.type=ts`	Write to a temporary storage queue
	`queue.type=td`	Write to a temporary data queue

Batch job submission property

Use this property to submit content as a Job Control Language (JCL) batch job to the MVS internal reader facility using the JZOS MvsJobSubmitter class.

Use the escape sequence \n to insert a new line, and the backslash character \ to continue onto the next line. The maximum length of each line in the batch job is 80 characters once all tokens have been resolve.

The SupportPac does not wait for the batch job to start executing. Use MVS Job Entry Subsystem (JES) facilities to check progress, successful execution, and output of the batch job.

If the SupportPac is started by a LINK command, the submitted job ID is returned in the container named *mvsjob.jobid*.

Property	***Usage and examples***

`mvsjob.content=`*value*	Required. The contents of the property are submitted to MVS as a batch job.
	
	In this example a modify command is issued against the CICS region to set a file enabled. The `EPCX_APPLID` and `File_name` tokens are replaced with the captured items from file system event.
	
	`mvsjob.content=//MODIFY JOB CLASS=M,MSGCLASS=H \n\`
	`//IEFBR EXEC PGM=IEFBR14 \n\`
	`// F {EPCX_APPLID},'CEMT SET FILE({File_name}) ENABLE'`

	This example is similar to the above except the content of the batch job is retrieved from a dataset member, with the property token `command` replaced with the contents of the property of the same name.
	
	`command=CEMT SET FILE({File_name}) ENABLE`
	`mvsjob.content={file=//'MYJOBS.JCL(SKELETON)'}`
	
	The contents of MVS file MYJOBS.JCL(SKELETON):
	
	`//MODIFY JOB CLASS=M,MSGCLASS=H`
	`//IEFBR EXEC PGM=IEFBR14`
	`// F {EPCX_APPLID},'{command}'`

HTTP properties

Use these properties to send a request to an HTTP server using the CICS web facilities.

The SupportPac uses the CICS WEB OPEN and WEB SEND commands to connect to the HTTP server and send the request. The timeout for these commands is specified in the DTIMOUT attribute of the TRANSACTION definition under which the SupportPac is started. If started as a custom event adapter or START CHANNEL command, the default transaction ID is CA1Y. If started by a LINK command, the transaction ID is that of the caller.

It is recommended to define a CICS URIMAP resource to specify the HTTP server details, then specify the resource name with the `http.urimap` property. CICS provides useful monitoring data and can reuse the connection when a URIMAP resource is used.

If the HTTP server returns a 401 response to indicate credentials are required, and you specify `http.urimap`, or you specify `http.uri` without a user and password, then CICS will call the XWBSNDO and XWBAUTH exits as described in topic Providing credentials for basic authentication to obtain the credentials and resend the request.

Property	***Usage and examples***
`http.certificate=`*label*	Set the label of the X.509 certificate to use if connecting to the HTTP server using SSL. For more details see the WEB OPEN command. `http.certificate=MyCertificate`
`http.characterset=`*codepage*	Set the character set of the HTTP content. The default is iso-8859-1. For more details see the WEB SEND command. `http.characterset=UTF-8`
`http.content=`*value*	Required. Specify the content to send to the HTTP server. It is important that the MIME is set appropriately, otherwise the HTTP server will fail to understand the content. The default MIME text/plain will be used if not specified using the mime token. `http.content=Hello` — Send the text Hello to an HTTP server. `http.content={json}` — Send a JSON document. The json token will set the MIME type to application/json. `http.content={commonbaseeventrest}` — Send a Common Base Event Rest document. The commonbaseeventrest token will set the MIME type to text/xml.
`http.method=`*method*	Set the HTTP request method as described in RFC2616. The default is PUT. For more details see the WEB SEND command. `http.method=POST` — Use POST to create a new resource. `http.method=PUT` — Use PUT to add content to an existing resource.
`http.retry=`*number*	Set the number of times the request should be retried. The default is 0, meaning do not retry. The request will only be retried when: 1. A socket error is received. 2. A response is not received due to a timeout. 3. An HTTP error response is received and the header Retry-After is present. In this case the value in seconds specified in this header is used to delay the retry. However if the delay retry is greater than 60 seconds the request is failed immediately. `http.retry=3`

Property	***Usage and examples***
`http.retrydelay=`*milliseconds*	Set the time in milliseconds to wait after each HTTP failure before retrying. The default is 500ms. `http.retrydelay=10000` Delay for 10 seconds before retrying.

`http.uri=`*uri*	Set the scheme, hostname, and optionally user name, password, port, path and query string to be used to connect to the HTTP server. The URI format is described by RFC3986.
	
	This property will be ignored if `http.urimap` is also specified.
	`http.uri=http://myserver.example.com/resource`
	`http.uri=http://username:password@server.example.com:8080/path?name=fred`

`http.urimap=`*urimap*	Set the CICS URIMAP resource name to use to connect to the HTTP server. The URIMAP resource should have usage set to CLIENT, and include the scheme, hostname, path and optionally port and query string. `http.urimap=MyServer`

MVS console message properties

Use these properties to write an MVS console message using the write to operator (WTO) macro via the JZOS MvsConsole class.

Property	***Usage and examples***
`mvswto.content=`*value*	Required. The contents of the property are submitted to the MVS console. `mvsjob.content=WTO message content`
`mvswto.descriptor=`*code*	An integer for the MVS descriptor code. A description and rules governing these codes are documented in the Descriptor codes topic in z/OS MVS System Messages. `mvswto.descriptor=16` `mvswto.descriptor=DESC_IMPORTANT_INFORMATION_MESSAGES` As above, but using the constants that start DESC_ defined in class com.ibm.jzos.WtoConstants rather than an integer.
`mvswto.route=`*code*	Specify an integer for the MVS routing code. See the Routing codes topic in z/OS MVS System Messages for a description and rules governing these codes. `mvswto.route=32` `mvswto.route=ROUTCDE_PROGRAMMER_INFORMATION` As above, but using a constant defined in class WtoConstants rather than an integer.

Chapter 9. Tokens

Tokens are used to dynamically create or retrieve content from a variety of sources.

Each property passed to the SupportPac has a value, and the value can contain tokens that are replaced with content as described in the tables below.

The SupportPac identifies a token by searching the property value for a regular expression. By default the regular expression requires a token is prefixed with a { character and suffixed with a } character. You can change the regular expression using the token.regex property.

Tokens and their parameters are case sensitive, and the parameters are required to be in the order specified. Some parameters are optional.

A token cannot be placed within another token.

If the token is invalid, for example it specifies a file name that does not exist, it will be removed and may result in messages being written to the Java stderr.

Token	***Usage and examples***	
property	Token is replaced with the contents of the property. Properties considered private, i.e. loaded using the `import.private` property, will not be replaced. `mail.content=This email is being sent to {mail.to}`	
container	If the container is a character (CHAR) type, the token is replaced with the contents of the container. If the container is a binary (BIT) type, the contents are attached to the property, or appended if one already exists. This will also set the attachment name to the container name if it is not already set using the token `name`.	
	`{MyContainer}`	Replace the token with the contents of MyContainer.
jvmmproperty	The token is replaced with the value of the specified JVM system property. You can set a JVM system property in the CICS JVM profile using the convention -Dproperty=value.	

	In the JVM profile:
	`-DmyProperty=Greetings`
	
	Example token:
	`{myProperty}`
		The JVM property myProperty is set in the JVM profile, and the {myProperty} token is replaced with “Greetings”.

	`{com.ibm.cics.jvmserver.configroot}`	The token is replaced with the JVM server configuration route directory. This property was new at CICS TS V5.2.
`commonbaseeventrest`	Token is replaced with an XML document referred to as the common base event REST format. In addition to header elements, all event business information items are automatically included. Additional properties can be added using the `emit` token. The property MIME is set to text/xml. `{commonbaseeventrest}`	
`commonbaseevent`	Token is replaced with an XML representation referred to as the common base event format. In addition to header elements, all event business information items are automatically included. Additional properties can be added using the `emit` token. The property MIME is set to text/xml. `{commonbaseevent}`	

Token	Usage and examples

`datetime=`*`format`*	Token is replaced with the date and time using the Java SimpleDateFormat format as detailed under heading *Date and Time Patterns*. If CA1Y is started as a custom event adapter, the date and time is taken from when the event was captured. Otherwise the current date and time as known by the JVM server is used.
	`{datetime=yyyy.MM.dd 'at' HH:mm:ss z}`
	`{datetime=}` — Token is replaced with the default Java date and time format. For Java 7 this is: `dow mon dd hh:mm:ss zzz yyyy`

`doctemplate=`*`name`*	Token is replaced with the contents of the CICS DOCTEMPLATE resource named by `name`. The DOCTEMPLATE type should be EBCDIC. Note for improved performance CICS will cache DOCTEMPLATE resources once first used. The cached copy can be discarded by disabling then re-enabling the resource. `{doctemplate=MyTemplate}` — Replace the token with the contents of MyTemplate.
`doctemplate=`*`name`*`:addPropertiesAsSymbols`	As above except each property passed to CA1Y is added as a symbol to the document. CICS will replace symbols in the template with the values of the property. Symbols used in document templates must follow the naming rules outlined in topic Using symbols in document templates. `{doctemplate=MyTemplate:addPropertiesAsSymbols}`
`doctemplate=`*`name`*`:binary`	As above except the contents are treated as binary and the DOCTEMPLATE type should be binary. `{doctemplate=MyTemplate:binary}`
`emit`	Include the value of this property when processing tokens `commonbaseeventrest`, `commonbaseevent`, and `json`. `MyProperty={emit}Hello` — Emit this property when creating one of the above document types.
`file=`*`filename`*	Token is replaced with the contents of the fully qualified file on zFS, where the contents are loaded using the default Java character set encoding. Note the contents are not cached. If caching is required use a DOCTEMPLATE resource. `{file=/path/name.ext}`

`file=`*`filename`*`:encoding=`*`codepage`*	As above except the contents are read using the specified Java character set encoding.
	`{file=/path/name.ext:encoding=US-ASCII}`
	`{file=/path/name.ext:encoding=Cp1047}`
	`{file=/path/name.ext:encoding=UTF-8}`

`file=`*`filename`*`:binary`	As above except the contents are read as binary. `{file=/path/name.ext:binary}`

`file=//'`*`datasetName`*`'`	Token is replaced with the contents of the fully qualified dataset. If the dataset is a PDS, a member must be specified. See the JZOS Zfile constructor for more examples. Note the contents are not cached. If caching is required use a CICS DOCTEMPLATE resource.
	`{file=//'HLQ.MYSEQ'}`
	`{file=//'HLQ.MYPDS(SAMPMEM)'}`

Token	***Usage and examples***
`file=//'`*datasetName*`':encoding=`*codepage*	As above except the contents are read using the specified Java character set encoding. `{file=//'HLQ.MYSEQ':encoding=Cp1047}`
`file=//'`*datasetName*`':binary`	As above except the contents are read as binary. `{file=//'HLQ.MYSEQ':binary}`
`file=//DD:`*ddName*	Token is replaced with the contents of the dataset specified by the DD card in the CICS JCL. If the dataset is a PDS a member can be specified. Note the contents are not cached. If caching is required use a DOCTEMPLATE resource. `{file=//DD:MYCARD(SAMPMEM)}`
`file=//DD:`*ddName*`:encoding=`*codepage*	As above except the contents are read using the specified Java character set encoding. `{file=//DD:MYCARD(SAMPMEM):encoding=Cp1047}`
`file=//DD:`*ddName*`:binary`	As above except the contents are read as binary. `{file=//DD:MYCARD(SAMPMEM):binary}`

`ftp=`*fileName*`:server=`*hostName*`:username=`*userName*`:userpassword=`*password*`:transfer=`*ascii|binary*`:mode=`*localactive|localpassive*`:epsv=`*false|true*`:protocol=`*protocol*`:trustmgr=`*trustManager*`:datatimeout=`*seconds*`:proxyserver=`*hostName*`:proxyusername=`*proxyUserName*`:proxypassword=`*password*	Token is replaced with the contents of the specified file retrieved from the FTP server. The Apache Creative Commons commons-net FTPClient framework is used to interact with the FTP server and documents these parameters.
	
	`fileName` and `hostName` are required and all other parameters are optional.
	
	`fileName` can include a relative or absolute path as understood by the FTP server.
	
	`hostName` can include : followed by a port number to override the default of 21.
	
	The default for:
	• `userName` is anonymous.
	• `transfer` is ascii.
	• `mode` is localactive.
	• `epsv` (extended passive mode) is false.
	`{ftp=/path/file1.txt:server=my.host.com}`
	`{ftp=/path/file1.txt:server=my.host.com:username=johndoe:password=secret}`

`hex=`*property*	Token is replaced with a hexadecimal string representation of the contents of the specified property. `{hex=MyAttachment}`
`htmltable:properties=`*pattern1*`:containers=`*pattern2*	Token is replaced by an HTML document containing one or two tables. The first table has a row detailing each property whos name matches the Java regular expression pattern1. The second table has a row detailing each container in the default channel whos name matches the Java regular expression pattern2. Java regular expressions are details in the Java Pattern class. **Note:** Properties loaded from import.private will have their values excluded. `{htmltable}` Include all properties and containers. `{htmltable:properties=.*}` Include only properties.

Token	***Usage and examples***
	`{htmltable:properties=(?!mail.*).*` — Include only properties whos name does not start with “mail”. `{htmltable:containers=MyContainer1|MyContainer2}` — Include only containers named MyContainer1 and MyContainer2.
`json`	Token is replaced with a JSON representation of the CICS event and the MIME for the property is set to application/json. All event business information items are automatically included. Additional properties can be added using the `emit` token. **Note:** JSON makes use of the { and } characters as start and end tags for the document and arrays. Add the token noinnertokens to prevent the JSON being parsed for tokens. Alternatively use the tokens.regex property to change that characters used to mark the start and end of tokens. `{noinnertokens}{json}`
`link=`*program*	Link to the specified CICS program with the current channel. `{link=MYPROG}`

`mime=`*mediatype*	Set the MIME media type of the property as defined by IANA media types. When sending an email, all properties with a MIME are attached to the email. Using token `file=filename:binary` automatically sets the MIME based on the file name extension.
	`{mime=text/plain}`
	`{mime=text/html}`
	`{mime=text/xml}`
	`{mime=image/jpeg}`
	`{mime=application/octet-stream}`

`mime=`*mime*`:to=`*mime*	Set the MIME type of the property, and the MIME type the property should be converted to. `{mime=text/xsl:to=application/pdf}`
`mime=`*mime*`:to=`*mime*`:xslt=`*property*	Set the property MIME type, the MIME type the property should be converted to, and the XSL Transformations (XSLT) to be used for the conversion. `{mime=text/xml:to=application/pdf:xslt=MyStyleSheet}`
`name=`*name*	Set the name of the attachment. If the token `file` is used in the property, the default name is the file name and extension. `{name=My photo.jpg}`

`name=`*property*	Set the name of the attachment to the contents of the named property.
	`MyFileName=Invoice - {datetime=yyyy-MM-dd}.pdf`
	`MyAttachment={name=MyFileName}...` — The property name is set to the contents of the MyFileName property that could evaluate to: `Invoice - 2013-07-09.pdf`

`noinnertokens`	Tokens for this property will be processed, however content inserted as a result of resolving tokens will not be searched for tokens. Use this to avoid searching within the content from files, doctemplates, or JSON documents for tokens. `MyHTML={noinnertokens}{doctemplate=MyTemplate}` — Content is loaded from the document template, but that content is not itself searched for tokens.
`nomoretokens`	Token processing is stopped for the remainder of the property. Use this when you know the property does not require token replacement.

Token	**Usage and examples**	
	`MyHTML={normoretokens}<HTML><h1>...`	The MyHTML property is not searched for tokens.
`putcontainer=`*`container`*	Copies the content from the current resolved property into a CICS container. If the property contains binary data the container will be type BIT, otherwise type CHAR.	
	`{putcontainer=MyContainer}`	
`responsecontainer=`*`container`*	Copy the contents of the property to the named CICS container once all properties have been processed. The container will be type BIT for binary content, otherwise will be type CHAR.	
	`{responsecontainer=MyPDF}`	
`systemsymbol=`*`pattern`*	Token is replaced with the resolved MVS system symbol pattern specified. The pattern takes the form &SYMBOL. or &SYMBOL(n:m). for a substring as described by substituteSystemSymbols. Topic What are System Symbols? in the z/OS MVS JCL Reference describes how to set and display system symbols.	
	`{systemsymbol=&SYSNAME.}`	
	`{systemsymbol=&SYSPLEX.}`	
	`{systemsymbol=&SYSCLONE.}`	
`texttable:properties=`*`pattern1`*`:containers=`*`pattern2`*	Token is replaced by a hexadecimal dump of the properties and containers whos name match the Java regular expression pattern. Java regular expressions are details in the Java Pattern class. **Note:** Properties loaded from import.private will have their values excluded.	
	`{texttable}`	Include all properties and containers.
	`{texttable:properties=.*}`	Include only properties.
	`{texttable:properties=(?!mail.*).*`	Include only properties whos name does not start with “mail”.
	`{texttable:containers=MyContainer1|MyContainer2}`	Include only containers named MyContainer1 and MyContainer2.
`transform=`*`property`*`:xmltransform=`*`resource`*	Token is replaced with the results from passing the specified property and XMLTRANSFORM resource to the CICS command TRANSFORM DATATOXML. For information on creating the required XMLTRANSFORM resource see Mapping and transforming application data and XML.	
	`{transform=CustomerData:xmltransform=CustomerXML}`	
`trim`	The leading and trailing spaces are removed from the property.	
`zip=`*`zipName`*	Compress the contents of the property into a zip named by `zipName`.	
	`{zip=MyZip.zip}`	
	`{zip=}`	If `zipName` is not specified, the property name and extension .zip is used.
`zip=`*`zipName`*`:include=`*`propertyList`*	As above except compress all of the properties named in `propertyList` into a single zip. Use a comma and no spaces to separate properties in the list.	
	`{zip=MyZip.zip:include=property1}`	
	`{zip=MyZip.zip:include=property1,property2}`	

Additional tokens available with event processing

The following tokens are only available when the SupportPac is started as an event adapter.

Token	**_Usage and examples_**
_event_information_	The named event business information item, as defined using the CICS Explorer event binding editor on the Specifications tab. `{in_stock}`

`EPCX_VERSION`
`EPCX_SCHEMA__VERSION`
`EPCX_SCHEMA__RELEASE`
`EPCX_EVENT__BINDING`
`EPCX_CS__NAME`
`EPCX_EBUSERTAG`
`EPCX_BUSINESSEVENT`
`EPCX_NETQUAL`
`EPCX_APPLID`
`EPCX_TRANID`
`EPCX_USERID`
`EPCX_ABSTIME`
`EPCX_EVENT_TYPE`
`EPCX_PROGRAM`
`EPCX_RESP`
`EPCX_UOWID`
	The event binding or task information. See topic EPCX - Event Processing Context Container in the CICS Information Center for their definitions.
	`{EPCX_BUSINESSEVENT}` Token is replaced with the business event name.

`EPAP_VERSION`
`EPAP_ADAPTER_NAME`
`EPAP_RECOVER`
	Contents of the specified field from EP adapter configuration. Refer to topic EPAP - Event Processing Adaptparm Container in the CICS Information Center for their definitions.
	`{EPAP_RECOVER}`

Additional tokens available with LINK or START commands

The following tokens are only available when the SupportPac is started by the LINK or START CHANNEL commands.

Token	**_Usage and examples_**
`TASK_TRANID`	The current task transaction ID. `{TASK_TRANID}`
`TASK_USERID`	The current task user ID. `{TASK_USERID}`
`TASK_PROGRAM`	The current program name. `{TASK_PROGRAM}`
`TASK_NUMBER`	The current task number. `{TASK_NUMBER}`
`REGION_SYSID`	The CICS region system ID. `{REGION_SYSID}`
`REGION_APPLID`	The CICS region application ID. `{REGION_APPLID}`

Examples using tokens for email attachments

You can add one or more properties to an email as attachments. Email clients can typically preview and save attachments, or start other applications to handle them. Tokens can be used in combination to specify the attachment contents, name and mime type.

Examples of how to use tokens to add a property as an attachment:

- Attach a file from zFS as a picture with the default name `picture.png` and default MIME type image/png:

  ```
  picture={file=/path/picture.png:binary}
  ```

- Attach a terms and conditions PDF with the default MIME type `application/pdf` and specify the name `Terms and Conditions.pdf:`

  ```
  terms={name=Terms and conditions.pdf} {file=/path/terms
  .pdf:binary}
  ```

- Attach a zip file `YourDocuments.zip` that contains a PDF and picture:

  ```
  picture={file=/path/picture.png:binary}
  terms={file=/path/terms.pdf:binary}
  attach={zip=YourDocuments.zip:include=picture,terms}
  ```

- Attach an invoice PDF created from the XML in the property and a stylesheet:

  ```
  myXSLT={file=/path/stylesheet.xslt:encoding=UTF-8}

  invoice={name=Invoice.pdf} {mime=text/xml:to=applicatio
  n/pdf:xslt=myXSLT}<?xml version="1.0" encoding="UTF-8"
  ?> <name>Joe Bloggs</name>
  ```

Chapter 10. Troubleshooting

Use the following checklist to diagnose problems using the SupportPac.

1. The SupportPac will abend CA1Y if it is started as a custom event adapter and the event could not be emitted.
2. Ensure the resources defined when the SupportPac was installed are correct, installed and enabled:

   JVMSERVER named DFH$JVMS
   TRANSACTION named CA1Y
   PROGRAM named CA1Y
   BUNDLE named CA1YEXAM for the examples - optional
   BUNDLE named FOP for Apache FOP - optional
3. If starting the SupportPac with events, ensure that CICS event processing is enabled and the event is being captured. To view the status of event processing and the number of captured events, use the CICS Explorer views available in the CICS SM perspective, under menu Operations → Event Processing.
4. Review the contents of the JVM server stderr and stdout files.

   For example if the mail server could not be reached there will likely be javax.mail.MessagingException entries with further exception information and a backtrace.
5. Review the contents of the OSGi framework log files, as described in Diagnostics for Java.

   Verify the JavaMail API bundle `mail.jar` or `javax.mail.jar` and SupportPac bundle `com.ibm.cics.ca1y_1.7.1.jar` are installed.
6. Enable Java Logging for the package com.ibm.cics.ca1y. For example, add the following line to the JVM server profile to log all message levels using the ConsoleHandler that directs logging output to stderr.

   Note the log is written using the JVM server default file encoding that may be an EBCDIC codepage. If a character cannot be converted the codepage substitution character will be used instead.

   ```
   -Djava.util.logging.config.file=/usr/lpp/ca1y/examples/
   logging.properties
   ```

   After making changes to the JVM server profile you will need to disable and enable the JVMSERVER, then attempt to resend the email.
7. If there are issues transforming XML using an XML stylesheet, check the output from JAXP by adding the following to the JVM server profile, disable and enable the JVMSERVER, then attempt to resend the email:

   ```
   -Djaxp.debug=1
   ```
8. Ensure the TCP/IP stack on the LPAR in which the SupportPac is running is able to reach the target SMTP server and there are no firewalls that block access. For example, log onto z/OS with telnet and use:

   ```
   traceroute <mail_server_hostname>
   ```
9. Once the SupportPac prepares the email it uses the JavaMail API to interact with the mail server. You may therefore find useful diagnostic information in the JAVAMAIL API FAQ.
10. Check the email is not being removed as junk by your email provider.

11. If the Java stderr contains `javax.mail.MessagingException: Could not connect to SMTP host: localhost, port: 25` then it is likely the mail server properties are not defined or being imported correctly. Check the encoding of the file is the same as that used on the file token.

    For example, if the file is stored at `/usr/lpp/ca1y/examples/emailServer.properties` in EBCDIC codepage 1047, the event adapter configuration should import the properties with;

    ```
    import={file=/usr/lpp/ca1y/examples/emailServer.properties:
    encoding=Cp1047}
    ```

12. If the Java stderr contains `java.io.FileNotFoundException: File '/usr/lpp/ca1y/examples/emailServer.properties' does not exist` then it is likely the file does not exist, or the file or directory permissions are not set correctly. Note that the directory needs to have execute bit on for files to be read.
13. Check the site SupportPac CA1Y for later versions that may contain fixes to your issue.
14. Email the SupportPac author for guidance – see Feedback on page 8.
15. If you are having issues creating a Java regular expression, as used in the property token.regex and tokens htmltable and texttable, you may find site http://www.regexplanet.com/advanced/java/index.html useful to try and evaluate expressions.

Chapter 11. Notices

License

Trademarks