CICS Transaction Server for z/OS
Version 5 Release 4

Front End Programming Interface (FEPI) User's Guide

Note

Before using this information and the product it supports, read the information in "Notices" on page 209.

This edition applies to the IBM CICS® Transaction Server for z/OS® Version 5 Release 4 (product number 5655-Y04) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this PDF

This PDF describes how you can use the Front End Programming Interface (FEPI) to write programs for CICS Transaction Server for z/OS.

For details of the terms and notation used in this book, see Conventions and terminology used in the CICS documentation in IBM Knowledge Center.

Date of this PDF

This PDF was created on January 20th 2020.

Chapter 1. FEPI concepts and facilities

This topicpart of the book gives an overview of FEPI, and some general information about functions, services, and implementing applications.

- "Introduction to FEPI" on page 1 explains what FEPI is and what problems it solves; it also describes some planning considerations.
- "FEPI functions and services" on page 5 describes the various types of FEPI commands and introduces the concepts and functions used by FEPI applications.

Introduction to FEPI

The Front End Programming Interface (FEPI) is an integral part of CICS. You can use the interface to write CICS applications that simulate terminals to access other CICS or IMS programs. This type of application is called a front end application.

- "Problems FEPI can solve" on page 1
- "How FEPI fits into your system" on page 2
- "Planning to use the Front End Programming Interface" on page 4.

Problems FEPI can solve

Many users have CICS and IMS applications that they want to use differently; for example, to extend their use by incorporating them into other applications. But they cannot change the way the applications are used because they cannot change the application programs.

FEPI allows existing CICS and IMS application programs to be used in different ways, in different combinations, in different environments, and on different systems, without changing them, because it provides a simple integrated interface to these programs. FEPI also lets you write new programs that add function to old programs.

There are many reasons why existing application programs can't be changed. Perhaps the application was bought in a package, so that you don't have the source. Perhaps someone else owns the application; perhaps it runs on someone else's system. Perhaps the source has been lost, and there's no one around who knows the program well enough. Perhaps the program logic is so complex that any changes are considered too dangerous.

Or perhaps it is an application that was written for one specific environment, such as IBM® 3270 information display systems, and you want to use it for another, or you want to extend its function. You don't want to change the application, because it must still work with the 3270s.

To get around this, you can run the existing application unchanged and provide a front-end program to interface to it. Using FEPI, a front-end program can simulate a terminal. This means the program can gain access to applications written to support that terminal. That program can then use the existing applications, and the existing application is unaware that anything has changed.

Therefore, the existing application can be used differently without being changed in any way. The changes are in the simulating program. For example, newly written applications can collect data from several existing applications. The existing applications can be on the same system as the simulating program, or on a different system.

Advantages over alternative solutions

There are other ways of accessing existing programs differently, but they all have their drawbacks.

Can CICS multiregion operation (MRO) or intersystem communication (ISC) be used to access remote applications?

Yes, but using MRO or ISC often requires some changes to the existing application—for example, to change the type of terminal supported or to provide an interface that uses a communication area.

Can z/OS Communications Server for SNA program-to-program support be used?

Yes, if your programmers can write an access program to issue the appropriate z/OS Communications Server calls. But these z/OS Communications Server calls cannot be part of a CICS application program.

How FEPI fits into your system

FEPI allows CICS front-end application programs to communicate with unchanged back-end applications running on CICS or IMS systems that are local or remote. The back-end applications continue to work just as if they are being accessed from the type of terminal they were originally written for.

A *FEPI application* is a CICS application that is designed to use FEPI to communicate with existing back-end applications. It is also known as a terminal front-end program.

The *front end* is the system on which the FEPI application runs, and the *back end* is the system on which the existing application runs. (They can run in the same CICS region.)

Figure 1 on page 3 shows how FEPI links the unchanged back-end applications to the new CICS FEPI applications. To an existing application, the front-end application looks like a terminal.

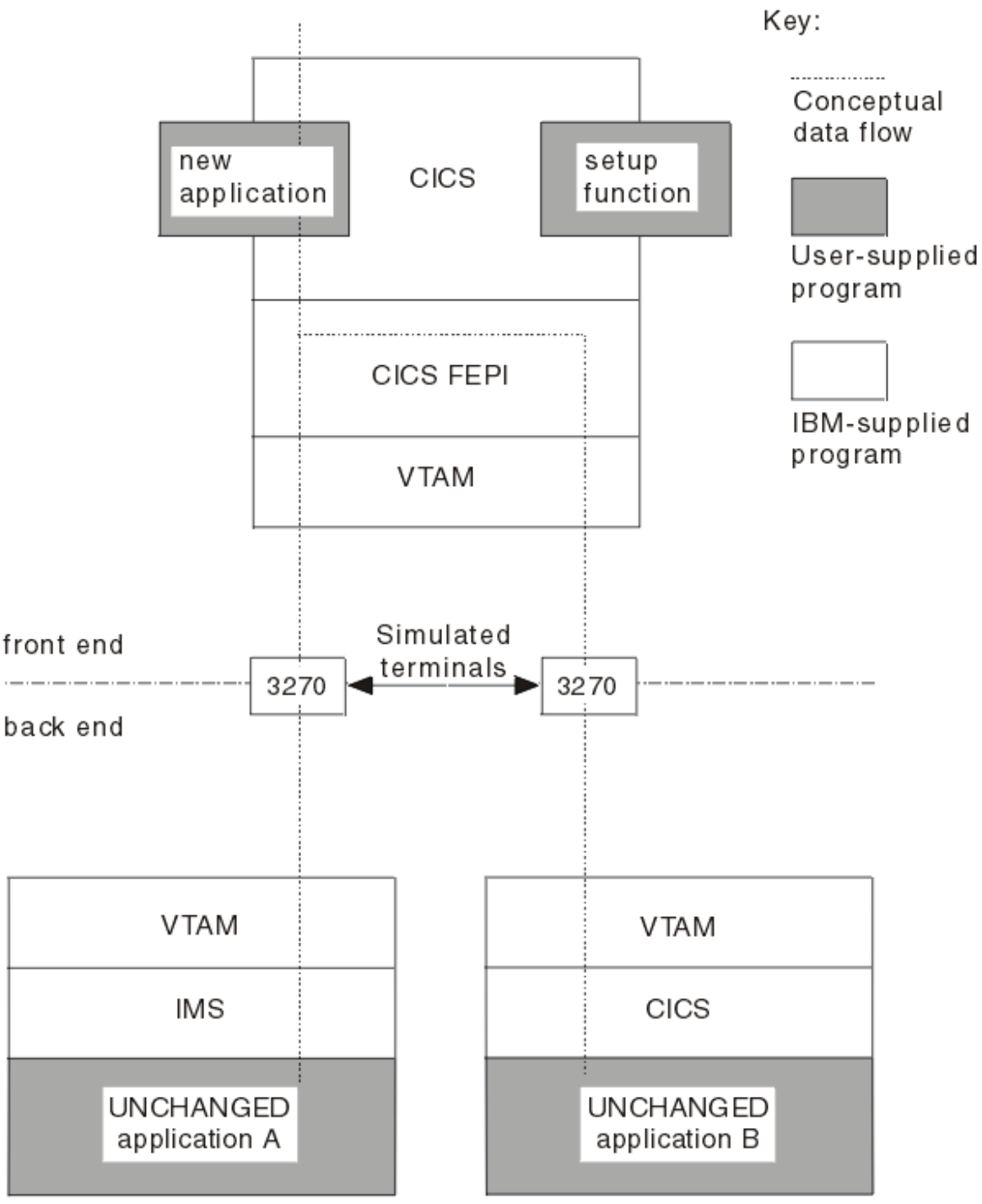


Figure 1. Structure of FEPI and application programs

Planning to use the Front End Programming Interface

This section explains what hardware and software you need to use the Front End Programming Interface (FEPI), what MVS system integrity is involved, what resources you need, and what to consider when installing FEPI and customizing your system.

Hardware and software requirements

There are different requirements for the front-end and the back-end.

Front-end requirements

For front-end systems, FEPI is an integral part of CICS Transaction Server for z/OS. Other hardware and software requirements are the same as for CICS Transaction Server for z/OS.

Extra 37x5 controllers and network control programs (NCPs) may be needed to provide the necessary intersystem connections.

Back-end requirements

Applications running on the following, and subsequent compatible releases, are supported:

- CICS Transaction Server for z/OS
- IMS Version 9.1 or higher

FEPI provides simulation for two very common classes of terminals on these systems:

- 3270-types for CICS and IMS applications (using LU 2 protocol)
- A family of programmable terminals, including the 4700, accessed through an LU 0 protocol (called SLU P), for IMS applications.

System integrity

All application programs that use FEPI run in problem-program mode in user-key storage. No part of FEPI needs to be authorized to run.

IBM accepts authorized program analysis reports (APARs) where the installation of the FEPI function introduces an exposure to the system integrity of MVS. Refer to the *MVS Integrity Programming Announcement* dated 21 October 1981.

Storage

Some storage below the 16 MB line is required, but the bulk resides above the 16 MB line in storage managed by CICS.

For details, see Planning FEPI storage.

There are no inherent resource limits in FEPI. It is limited only by what is configured and the available system storage.

Installation

FEPI is distributed through normal IBM Program Library channels.

It is a part of CICS and cannot be ordered separately. See Getting started with FEPI for more information.

Configuring your system for FEPI

You must configure your system specifically for CICS FEPI, for new application programs, and possibly for existing applications.

About this task

You might need to adapt your z/OS Communications Server setup, your CICS system, and CICS FEPI to use the interface effectively.

Procedure

1. Change the default value of the **FEPI** system initialization parameter to YES (the default is NO).

   The **FEPI** system initialization parameter controls whether FEPI is available or not.

   When the CICS region starts, FEPI runs as a system transaction that is started automatically. You do not need to start or stop it independently.
2. Write a setup program to configure FEPI with the necessary resources when commands are issued from a front-end application program.

   FEPI does not use a configuration file or CICS RDO.

   The setup program can get the configuration data from a file or from whatever source it identifies.
3. Define CICS FEPI applications to CICS in the normal way.
4. Optional: Define simulated terminals for FEPI to use.

FEPI functions and services

The Front End Programming Interface (FEPI) function provides access, by means of simulated terminals, to CICS and IMS applications available through a communication network.

The functions and services provided by FEPI enable you to write application program to provide a front end to other CICS and IMS applications without having to change those applications in any way. This is done by simulating a terminal in session.

Introducing FEPI functions

An application program using FEPI can provide a front end to other CICS or IMS applications. Because this is done by simulating a terminal in session with the non-FEPI application, that application does not have to be changed in any way.

Thus you can write FEPI applications that provide a single integrated interface to previously disparate applications. The scope and usability of your CICS and IMS applications can be extended by using them in combination, in different environments, or on different systems.

Because a FEPI application communicates with other applications that can run in different systems, it is necessary to distinguish between systems and identify the direction of data flows. The convention is:

Front-end
: The front-end system is the one in which the FEPI application runs.

Back-end
: Back-end identifies the system in which the other CICS or IMS applications run. (This is equivalent to “partner” system, used elsewhere by CICS.)

Outbound
: Identifies data sent by the FEPI application to the back-end application.

Inbound
: Identifies data received by the FEPI application from the back-end application.

FEPI provides a programming interface. Its functions can be invoked only through that interface, which is an extension to the EXEC CICS programming interface. All FEPI requests are made by issuing **EXEC CICS**

FEPI commands; all the commands have the qualifier FEPI. The languages supported by the EXEC CICS programming interface (Assembler, COBOL, C, PL/I) can be used. For educational and initial development purposes, you could use CECI, rather than formally writing a program.

All functions are available in the normal way to all applications, except that some functions are intended for system programmers, and their use can be restricted. All the other facilities that you can use with CICS applications, such as the execution diagnostic facility (EDF) and the command interpreter transaction, CECI, are available.

FEPI samples

To help you develop your own CICS FEPI applications, and to show you what FEPI can do, FEPI includes detailed samples. They form an integrated set, and include a program that sets up the FEPI configuration needed to run the other samples.

The samples are supplied in source format in the SDFHSAMP library, and include two back-end application programs, that show many of the principles and techniques discussed in the FEPI programming section. Although the samples are copyrighted, you may use and copy them freely for educational purposes to help you write FEPI applications.

The names of the samples have the form DFH0*xZyy*. *Z* shows that the sample is a FEPI sample and *x* identifies the source language of the sample. A for Assembler language, C for C, P for PL/I, and V for COBOL. *yy* identifies the specific program.

For more information, see FEPI samples.

FEPI programming commands

EXEC CICS FEPI commands provide several ways of developing CICS FEPI applications.

The commands are at three logical levels:

High-level:
a straightforward interface for normal 3270 applications

Data stream-level:
for use with IMS SLU P applications and more complicated 3270 applications

Specialized-level:
for access to complex z/OS Communications Server communication functions and events, designed for use by vendors and experienced CICS FEPI application developers.

High-level FEPI commands

The high-level front-end programming interface consists of two interfaces for everyday use: *key stroke* and *screen-image*, collectively known as *formatted* data. They allow programmers to build their own CICS FEPI applications in a straightforward manner. However, the programmer must understand data communication and protocols.

See FEPI key stroke and screen-image applications for details.

The key stroke interface

The key stroke interface allows programmers writing in any of the CICS-supported languages, to specify the keys that an operator might press while using an existing application. The key strokes are specified using easily coded mnemonics; no hexadecimal values are required.

The screen-image interface

The screen-image interface allows programmers writing in any language supported by CICS, to define the contents of a 3270 screen, using a data structure appropriate to the programming language.

It uses a buffer with one byte for each screen position (for example, 1920 bytes for a 24 × 80 character screen). This buffer can be defined in any way that suits the application program and the programming language. It is passed as a complete screen buffer to the back-end application.

In both cases, key stroke and screen-image, the data received from the back-end application is presented as a screen image.

Data-stream-level commands

For many applications, the key stroke and screen-image interfaces should be quite adequate.

However, where they are not, FEPI data-stream-level commands give an application complete control of the 3270 data stream. These commands are also needed for SLU P applications, which can use only this interface. FEPI does not buffer or interpret the data stream; it is presented as it arrives from the back-end application, and the front-end application must be prepared to handle whatever is presented. Similarly, data sent by the front-end application is transmitted without verification.

A detailed knowledge of data communication and protocols and of data stream format is required.

See FEPI data stream applications for details.

Specialized-level commands

A number of specialized functions can be accessed through FEPI.

These specialized functions are as follows:

STSN for SLU P applications:
Set and test sequence number (STSN) is a communication protocol used to check and control transmissions. FEPI normally handles all necessary STSN processing automatically. However, FEPI also provides access to STSN information for those applications that need to control sequence number data.

Application access to definite responses:
When a flow is received, the receiving LU can choose what response to return to the sending LU. FEPI normally handles this automatically, but also provides facilities for applications to determine this flow.

Other z/OS Communications Server facilities:
Some applications use a z/OS Communications Server facility known as CLSDST(PASS); this can be used in more sophisticated CICS FEPI application programming.

See Specialized FEPI functions for details.

List of commands

All the logical levels use more or less the same set of commands, though the options used may vary.

The EXEC CICS FEPI application programming commands are:

ALLOCATE
Establishes communication with a back-end application

FREE
Frees communication with a back-end application

SEND
Sends data from a CICS FEPI application to a back-end application

RECEIVE
Receives data into a CICS FEPI application from a back-end application

CONVERSE
Sends data to and receives data from a back-end application

ISSUE
Sends control data to a back-end application

EXTRACT
Gets field data and attributes, set-and-test-sequence-number (STSN) data, or conversation status

START
Schedules a CICS transaction to handle inbound data.

Setup and resources

Besides the application programming functions that communicate with back-end applications, FEPI also provides system programming functions that define and inquire about *FEPI resources* and perform control functions.

Defining and configuring FEPI resources is called *setup program*. The **EXEC CICS FEPI** commands that provide these functions are:

INSTALL, ADD
Sets up communication resources

DISCARD, DELETE
Discards communication resources

INQUIRE
Queries FEPI resource status

SET
Controls FEPI resources.

The setup functions are usually performed by a customer-written transaction that is started from a second-phase program list table post initialization (PLTPI) program. See The configuration programs you should write.

FEPI resources can be controlled, like other CICS resources, using the **CEMT SET** and **CEMT INQUIRE** functions. CECI can also be used. See Operator control of FEPI.

FEPI resources

There are four types of FEPI resource: *pool*, *property set*, *target*, and *node*. This topic describes the relationships between them.

Pool
A collection of nodes and targets

Property set
Defines the characteristics of a pool

CICS1, CICS2, IMS1, IMS 2...
Back-end systems

Node 1, Node 2, Node 3...
Simulated terminals

A FEPI pool can have one or more nodes and one or more targets. The same nodes and targets can be in any number of pools, except that the same node-target pair (a *connection*) cannot occur in more than one pool.

A CICS FEPI application can reach a target only by specifying a pool, which defines the set of nodes that can be used to make the connection, and the characteristics of the communication.

A target and an open node in the same pool are ‘connected’; when bound, they are ‘in session’. To *bind* means to establish a *session* on a *connection*, to make it ready to allow communication.

The process of communicating with a back-end system is called a *conversation*; it is the fundamental entity that a FEPI application deals with. Only one conversation can use a connection at one time, although any number can do so consecutively. For efficiency, the session on the connection is kept bound between conversations, unless you choose otherwise. Furthermore, a conversation is owned by the task that establishes it; no other task can use it.

Note: The use of the term conversation does not mean that the back-end or front-end application has to be conversational, in the CICS meaning of the term.

The resources are further explained in Configuring FEPI and the more complex relationships possible between them are illustrated in An example FEPI configuration.

CICS FEPI application programs

A CICS FEPI application consists of several distinct logical functions.

These logical functions are as follows:

Access programs:
Communicate with the back-end applications

Begin-session handler:
Handles begin-session processing

End-session handler:
Handles end-session processing

STSN handler:
Assists message synchronization

Unsolicited-data handler:
Handles unsolicited inbound data

Monitor:
Handles unexpected events such as the loss of a session or errors in setup.

These functions can be in separate programs, or contained in one program. The need for each function depends on the requirements of the application; in many cases default processing is all that you need. You might need several styles of each function, again depending on the requirements of your application.

The application programmer always writes the access programs. The system programmer usually writes the monitors to handle the unexpected events that FEPI reports to transient data queues such as CSZX. As for the other functions, sometimes the system programmer writes them providing, perhaps, just one instance of each, so that they are common to everyone. (This approach has the advantage that adherence to standard procedures—for such things as signon and signoff—is enforced.) In other installations, the application programmers provide them.

In many cases, writing a CICS FEPI application is straightforward. However, some applications need more sophisticated programming. The programmer not only has to understand all the displays and protocols of the back-end application and system (CICS or IMS), but must also understand the detailed data-stream protocols. For further information, see FEPI application design.

Terminals supported

To access back-end applications, FEPI has SNA secondary logical unit (SLU) support, so that CICS FEPI applications can simulate certain logical unit (LU) types. FEPI uses the z/OS Communications Server program-to-program support to provide this function, and to communicate between front-end and back-end applications.

Note: FEPI cannot send z/OS Communications Server logon data.

FEPI provides simulation support for two families of terminals. The names **SLU2** and **SLU P** are used to identify the two types of support:

SLU2
for the 3270 family of terminals, used in many CICS and IMS applications. See the IBM 3270 Data Stream Programmers Reference.

SLU P
for a family of programmable terminals, including the 4700, accessed through an LU 0 protocol, for IMS applications. This protocol is defined in *IMS/VS Programming Guide for Remote SNA Systems* (for IMS/VS Version 2) or *IMS Customization Guide* (for IMS/ESA® Version 3 and later).

Data-stream-level and specialized-level commands can be used with both families of terminals, but the high-level commands, which use formatted data, are only for SLU2.

The mode of a conversation must be either SLU2 or SLU P; it cannot be mixed. For SLU2 conversations, formatted data or data stream data can be used, but cannot be mixed in the same conversation. The mode and data type are controlled by the pool used, which is set up by the system programmer.

These terminals are supported only when they are used to communicate with CICS or IMS systems.

FEPI Security

This section introduces FEPI security.

Because FEPI is a terminal emulator, the back-end system “sees” the front-end as a terminal rather than a system; it cannot differentiate between FEPI emulation and a real device. For details, see Signon security.

You can restrict access to the FEPI system programming commands by defining operator profiles to your ESM. For details, see Command-level security. All application programming commands are generally available.

Problem determination, customization, and performance

The following functions are provided by FEPI.

- Debugging tools, trace, dump routines, and messages are available to help you determine the source of an error. These areas are described in FEPI problem determination.
- Two CICS global user exits are available for you to use with FEPI. They are described in Front End Programming Interface exits XSZARQ and XSZBRQ.
- Data that flows to and from CICS FEPI applications can be journaled for audit trails. For details, see FEPI journaling.
- You can use CICS monitoring and statistics data to help you tune FEPI applications, and to control the resources that they use. For details, see Using CICS monitoring.

Chapter 2. Configuring FEPI

Planning your activities will help you to understand how to set up and configure FEPI in your system.

Analysis and planning

When you are planning to implement FEPI, you must consider the configuration requirements, how to organize your pools, their properties, and the connections.

1. You must consider the following:
   - Details of the back-end applications and systems
   - Names of nodes and targets
   - Operator control requirements
   - Journaling requirements
   - Signon and signoff procedures
   - Special event handling
   - Pools required for control reasons
   - Pools required for functional reasons
   - Number of nodes
   - Setup program organization.
2. Decide how to organize your pools, their properties, and the connections.

Back-end applications and systems

You need to know whether the back-end systems are CICS or IMS, the terminal types they use, and the timing and volume of transactions expected.

Also, are there any restrictions on the use of the terminals? For example:

- Is a specific terminal required, or can any terminal be used?
- Is a specific LU or terminal type defined in the target application; for example, a 3278 model 3?

Names of nodes and targets

Decide on a naming convention for your nodes and targets.

Decide which z/OS Communications Server node names are available for use by FEPI as simulated terminals. Remember that FEPI nodes are z/OS Communications Server APPL definitions, not logical units (LUs). Do not use names starting with “DFH”.

The back-end system already has defined z/OS Communications Server primary PLU names (applids) which you must use. However, you can define your own local target names to associate with these applids. This means that FEPI applications are not affected if an applid is changed; you associate the local name with the new back-end target name. Do not use names starting with “DFH”.

Operator control requirements

The CEMT INQUIRE and SET master terminal transactions can be used to view and amend the state of FEPI resources.

CEMT DISCARD can be used to remove resources from FEPI. This is described in Operator control of FEPI. If you decide you need extra functions for operators, you can write appropriate programs.

Journaling requirements

Journaling is available if you need it.

Among the reasons for using FEPI journaling are:

- To create audit trails
- To monitor performance
- To control message security.

For further information, see FEPI journaling.

Signon and signoff procedures

You need to know if there are any specific requirements for signon and signoff to back-end systems. Central control might be required, or applications could perform signon and signoff individually.

Signon security

Because FEPI is a terminal emulator, the back-end system “sees” the front-end as a terminal rather than a system; it cannot differentiate between FEPI emulation and a real device.

Thus, CICS bind, link, and attach-time security are not applicable to FEPI connections. If security is enabled in the back-end system, in order for your FEPI application to access protected resources the emulated terminal must be signed on to the back-end. The alternative is that you do not use CICS security with FEPI; that is, you make all the back-end transactions accessed by FEPI available to the CICS default user. This option is clearly unacceptable; it means that you must either run a security risk or deprive your FEPI applications of access to sensitive data.

When signing on to a back-end system, FEPI applications can ask the external security manager (ESM) to supply a PassTicket. A PassTicket provides a secure way of signing on to back-end systems. They are valid for one use only and are time-stamped, so the potential damage caused by their being intercepted is minimal. Applications do not have to store passwords (or ask users to reenter them) in order to sign on to back-end systems, and passwords are not transmitted across the network. For information about implementing signon security, see Generating and using PassTickets for secure sign-on.

Special event handling

In addition to signon and signoff, you must consider what special event handling is required and whether it should be handled by central functions or by applications individually.

The special event handling you must consider are as follows:

- The receipt of unsolicited data
- Unexpected events
- Beginning a session
- Ending a conversation or session
- Shutdown of the front-end CICS system.

If some sort of enforcement is required, or you want central provision for convenience, commonality, or the upholding of conventions and standards, you must supply a set of standard handlers. Otherwise, the application programs must handle each event. If you need special back-end processing when CICS shuts down, you need an end-session handler.

Unexpected events (including errors in setup) are reported to a transient data (TD) queue, so that a monitoring transaction can be triggered to handle them; they also send a message to the FEPI message log CSZL. You must decide how to handle these events, and which queues to use.

For more detailed information about the design and structure of applications, including information about using the various event handlers, see FEPI application design.

If you want central control over the range of FEPI commands that applications are permitted to issue, you can use the XSZBRQ global user exit, which is described in Front End Programming Interface exits.

Using pools for control reasons

You can use pools for a number of control purposes.

For example, you could define them so as to:

- Restrict users and applications to particular targets or nodes, or restrict access to some targets to particular times of day.
- Force specific begin-session and end-session effects.
- Split resources among different types of back-end requests, according to (for example) priority, or to the department issuing the request. By doing this, you can ensure that there is always a set of connections to a target for time-sensitive requests, while other connections handle long-running requests that are not time-sensitive.
- Ration the use of connections, especially for long-running requests, so that each set of users has access to only a limited number of connections.
- Ease signon considerations.

Using pools for functional reasons

Pools determine the data format and special event handlers used by your FEPI applications. These attributes may be specified by the application programmer, or they may be imposed by the system programmer for central control, especially of signon and signoff.

If you need several types of special event handling, you might need to define your own pool-specific transient data queues, as well as the default queues.

Number of nodes

The number of nodes required depends on:

- How the pools are structured
- How much storage is available
- How many concurrent sessions are required to a particular target.

The number of concurrent sessions to a particular target may depend on the volumes of data to be transmitted and the speed of the network.

Although a node can have only one session with a particular target at a time, it can communicate with several different targets concurrently, and several nodes can communicate with the same target concurrently.

Setup program organization

You must decide how to organize your setup program; for example, your setup program could consist of a single module, or a set of related modules.

You must also make the following decisions:

- Whether your programs should take replaceable parameters or fixed values. You might use mainly fixed programs, with a flexible program for one-off changes.
- When programs are to be run - started from a second-phase PLTPI program, under operator control, or at set times of the day.
- Where the definitions required by the setup program are to be obtained - from panel entry, from a file, or by other means.

Organizing your pools and property sets

When you have done the analysis work described in the previous section, you can decide how to organize your pools, their properties, and the connections between nodes and targets.

Organizing pools

There are several ways of organizing your pools.

About this task

- If possible, restrict each pool to a single target, but specify as many nodes as you believe you need to satisfy concurrent access to the target. The reasons for taking this approach are:
  - It avoids the need for the front-end application to specify a target.
  - It makes it easier to avoid duplicate connection definitions.
  - Because a connection is created for every node-target combination within a pool, having large numbers of both nodes and targets within the same pool may generate more resources than are required.
  - The overhead associated with a pool is very small. Therefore there is no reason not to define many pools.
  - The expected concurrent usage of each target may be different. If you have more than one target in the pool, it becomes difficult to estimate the number of nodes required.
- You can define a pool containing only one node and one target. This lets a FEPI application allocate a specific session, which is necessary if the target system associates any special qualities with a particular terminal ID. You can use the XSZBRQ global user exit to control access to the pool.
- You can define pools that use different nodes to reference the same target. By making each pool available to a different group of users, you can eliminate competition for resources. Alternatively, you could use each pool to support a different set of properties, according to application requirements.
- If you plan to use the z/OS Communications Server **`VTAM CLSDST(PASS)`** command, other considerations might apply. See “Handling CLSDST(PASS)” on page 32.

Do not use names starting with “DFH” for pools.

Organizing property sets

Property sets allow you to define the properties of pools (such as the data format and special functions they use) separately from the definition of the pool itself.

You can use a single property set to define any number of pools. You must define as many property sets as you need to satisfy every unique pool requirement. Because the overhead associated with a property set is very small, there is no reason why you should not define a large number of them.

The properties are:

Device attributes

This specifies which family the simulated terminal belongs to, SLU2 or SLU P. For SLU2, it also determines the presentation size of the display (24 x 80, 32 x 80, and so on), and whether it supports extended attributes such as color.

Many back-end applications can be run with any terminal type, so you can use the default device type (SLU2, 3278 model 2). But if you have applications that demand particular terminal types, you need to define pools with the appropriate device types.

Data handling

This specifies which command level to use (high-level with formatted data, or data stream), how much data can be handled, and how contention is to be handled.

High-level is simpler to use and suits many front-end applications. Applications that require sophisticated functions or use SLU P, and those performing a simple pass-through, need the more complex data-stream-level. In most cases the default data size of 4096 is adequate; increase it only if you know there are large amounts of data to send and receive in a single command. Set contention

handling so that the front end wins—as for a real terminal—unless you have some particular reason for not doing so.

Session management
This specifies whether begin-session and end-session are to be handled by special transactions, and whether initial inbound data is expected. For SLU P, it also includes whether message resynchronization ("set and test sequence number" (STSN)) is to be handled.

The use of event handlers was introduced in "Signon and signoff procedures" on page 12; it is generally preferable to use specially written transactions for session management, rather than to leave it to be handled individually by applications.

If a back-end system sends initial data (a "good morning" message) you must specify this as a property of the pool, so that FEPI waits for the data to arrive and ensures that the front-end application receives it; otherwise the results will be unpredictable. For SLU2, IMS always sends initial data; CICS might or might not do so, depending on your system definition.

FEPI does all the necessary STSN handling automatically, but you can specify a transaction to handle it yourself.

Unexpected events
This specifies how unsolicited data and other unexpected events (including setup errors) are to be handled.

If you choose not to handle unsolicited data in your own transaction, you can tell FEPI how to handle it for you—positively or negatively; if the back-end system is IMS, you must specify that FEPI should respond positively. All unexpected events are logged in the FEPI message log (CSZL), even if you specify no unexpected event queue.

Journaling
This specifies what sort of data journaling is required, and which journal to use.

Do not use names starting with "DFH" for property sets.

Workload routing in a sysplex

In an MVS sysplex, you can create a CICSplex consisting of sets of functionally equivalent CICS terminal-owning regions (TORs) and application-owning regions (AORs). If the FEPI back end system is a TOR in such a CICSplex, you can use the z/OS Communications Server generic resource function to perform workload routing across the available TORs.

A z/OS Communications Server application program such as CICS can be known to z/OS Communications Server by a generic resource name, as well as by the specific network name defined on its z/OS Communications Server APPL definition statement. A number of CICS regions can use the same generic resource name.

A FEPI application, which starts a session with a CICSplex that has several terminal-owning regions, names a target that you have defined as the generic resource name of the TORs. Using the generic resource name, z/OS Communications Server is able to select one of the CICS TORs to be the target for that session. For this mechanism to operate, the TORs must all be registered to z/OS Communications Server with the same generic resource name. z/OS Communications Server is able to perform dynamic workload routing of the terminal sessions across the available TORs.

For information about defining FEPI targets as z/OS Communications Server generic resource names, see the APPLLIST option of the **FEPI INSTALL TARGETLIST** system programming command. For further information about z/OS Communications Server generic resources, see Configuring z/OS Communications Server generic resources.

Planning FEPI storage

FEPI does not require any additional MVS storage beyond that recommended for basic CICS.

As for dynamic storage, the storage used by FEPI is allocated exclusively from CDSA and ECDSA; CDSA usage is only that required to support z/OS Communications Server processing. The following information allows you to estimate the storage requirements of a particular FEPI configuration.

Table 1. Dynamic storage requirements (in bytes)		
Item	**ECDSA**	**CDSA**
Basic	80K	
For each node	288	180
For each node that is currently available for communication	192	
For each target	236	
For each pool	272 + 64 x (number of nodes in pool) + 64 x (number of targets in pool)	
For each property set	176	
For each connection (note 1)	432 if using data stream data 688 if using formatted data	
For each connection that is currently available for communication	384 + additional value from Table 2 on page 16 if using formatted data	
For each current conversation	128	
For each command in progress	2.5K + size of user data (Note 2)	

Note:

1. The number of connections is (number of nodes in pool) x (number of targets in pool) for each pool.
2. This is the data that is to be sent and received, or used for defining resources. If global user exits are used, twice the data size is needed; similarly if journaling is used.

For each connection that is currently available for communication and that uses formatted data, additional ECDSA storage is required; the amount depends on the device type and capabilities defined, as shown in Table 2 on page 16.

Table 2. Connection storage requirements (in bytes) by device type and function				
Device type	**Basic**	**Additional for color support**	**Additional for extended data stream support**	**Maximum**
327x model 2	3840	1920	5760	11520
327x model 3	5120	2560	7680	15360
327x model 4	6880	3440	10320	20640
327x model 5	7128	3564	10692	21384

Add some contingency (approximately 10%) to your final estimate.

Getting started with FEPI

FEPI is installed automatically when you install CICS. However, to make it operative you must install some additional resources.

The installation process

The FEPI installation process has a number of steps.

These steps are as follows:

1. Updating CICS resource definitions
2. Installing FEPI resource definitions

3. Starting CICS (See)

CAUTION:

About loading FEPI modules into the LPA

Any of the FEPI modules can be loaded in the MVS Link Pack Area (LPA). However, as with CICS modules in general, it is not recommended that you do so. For information about installing modules in the LPA, see Installing CICS modules in the MVS link pack area in Installing.

Updating CICS definitions

The RDO group DFHFEPI, which is on the product tape, contains resource definitions for the FEPI programs and the FEPI transaction CSZI. FEPI programs are prefixed with DFHSZ.

DFHFEPI is included in the default startup group list, DFHLIST.

Use the CEDA transaction:

- To define your FEPI application programs
- If you have installed FEPI modules in the LPA, modify the definitions of the modules in the CICS system definition file (the CSD), so that they specify USELPACOPY(YES).

Transient data queues

Sample definitions for the transient data (TD) queues required by FEPI are supplied in group DFHDCTG. You can use the sample definitions, or create your own, together with any extra queues that you need.

The required queues are:

CSZL

The FEPI message log. You can define CSZL as an intrapartition, extrapartition, or indirect queue. Note that CSZL must be defined as non-recoverable.

It is recommended that you define CSZL as an indirect queue, pointing to CSSL.

CSZX

The queue for information about unexpected events (including setup errors) that do not relate to specific pools. You can define CSZX as an intrapartition, extrapartition, or indirect queue. Note, however, that it must be defined as non-recoverable.

It is recommended that you define CSZX as an intrapartition queue, with a trigger level of 1, so that each event is processed immediately it is reported. (You must also, of course, write and install the event-handling transaction that is to be triggered.)

Any pool-specific TD queues that you require

Such queues receive information about events that affect specific pools. They can be defined as intrapartition, extrapartition, or indirect queues. Note, however, that they must be defined as non-recoverable.

It is recommended that you define pool-specific queues as intrapartition queues with trigger levels of 1, so that each event is processed immediately it is reported.

System initialization parameter, FEPI=YES|NO

Code **FEPI**=YES, to specify that FEPI is available.

The default is **FEPI**=NO. For information about setting system initialization parameters, see FEPI system initialization parameter.

Command-level security

If your installation uses CICS command-level security, you can restrict access to the EXEC CICS FEPI system programming commands (and to the equivalent commands that you can issue with the CEMT master terminal transaction) by defining access authorizations to your external security manager (ESM).

The commands you can protect in this way are those listed in FEPI system programming reference and in the CEMT section of Operator control of FEPI. You cannot restrict access to the FEPI application programming commands (as listed in).

To protect the FEPI system programming commands, use the resource identifier FEPIRESOURCE when defining resource profiles to the ESM. Note that, if you use command security, you must ensure that authorized users of CEMT are also authorized to use the FEPI commands.

For RACF® users, details of how to define resource profiles to the ESM are in the z/OS Security Server RACF Security Administrator's Guide. For information about using RACF with CICS, see Securing overview. Users of other security managers must refer to the documentation for their own product.

Installing FEPI resource definitions
Ensure that the RDO group DFHFEPI is in your startup group list. DFHFEPI is in the DFHLIST startup group list, so this should have been done automatically when you installed CICS.

About this task

Configuring CICS

Before you begin
"Getting started with FEPI" on page 16 covers everything that FEPI itself requires: the RDO group DFHFEPI in the startup group list; definitions of the transient data queues CSZL and CSZX; and any required security access controls.

About this task
Now you have to define your FEPI applications to CICS in the usual way. This includes the setup program, any common functions, and any additional transient data queues that you need for handling pool-specific events. Note that, in an intercommunication environment, FEPI itself must be run in the application-owning region (AOR) and all transactions that FEPI may start must run locally. This is because FEPI commands cannot be function shipped.

Procedure

1. Define transactions that are to be started by FEPI (the event handlers and pseudoconversational access programs) as CICS started tasks, with SPURGE=NO and TPURGE=NO to prevent them from being accidentally canceled by CICS
2. Define any additional transient data queues.

   See "Transient data queues" on page 17 for details of the queues.
3. Before starting CICS, you should ensure that your system has enough storage available to support your FEPI configuration: for details see "Planning FEPI storage" on page 15.
4. If you are using a setup transaction to install your FEPI nodes, targets, and pools that is started by a program list table (PLT) program, you need to include your PLT program in the second part of the program list table post initialization (PLTPI) list. This process is described in "Running setup programs" on page 24. If you use this method, you need to include your PLT program in the second part of the program list table post initialization (PLTPI) list.

Configuring the z/OS Communications Server

For FEPI to communicate with the network, some information must be defined to the z/OS Communications Server.

Procedure

1. Each FEPI node (simulated secondary LU terminal) must have a Communications Server application minor node definition.

   The name of this minor node must be the same as the node name specified on the **FEPI INSTALL NODELIST** command.

For example, the FEPI node called ‘FEPI0001' would require the following application minor node definition in the Communications Server:

```
DG4FEPI1  APPL ACBNAME=FEPI0001
```

2. If your network uses a naming convention to manage network resources, you can allow a network-independent name to be used by specifying it on the ACBNAME keyword of the Communications Server APPL statement.If this is not the case, you can simplify the definition of the Communications Server application minor node by omitting the ACBNAME keyword (which means that the margin-name, DG4FEPI1 in the example, must be the same as the FEPI node name).

   FEPI does not impose any additional restrictions on the naming of nodes, other than that the names should not begin with DFH; apart from this, any values acceptable to the Communications Server are acceptable to FEPI.
3. If you require password protection of the minor nodes, you can use the PRTCT keyword of the Communications Server APPL statement to specify a password of 1–8 characters. The password must then be specified on the corresponding **FEPI INSTALL NODELIST** command.
4. If you are defining multiple FEPI nodes, you can place them all in a single member (also known as a Communications Server application major node) or in several members. You can also add them to an existing Communications Server application major node.

   How you choose to organize the Communications Server definitions can depend on how your installation manages its network resources, or how you plan to manage the FEPI configuration. The Communications Server application minor node definition statements are stored collectively as one or more members of an MVS partitioned data set (usually SYS1.VTAMLST), accessed by the Communications Server via the VTAMLST data-definition statement in the Communications Server startup JCL.

What to do next
For general information about configuring the z/OS Communications Server, see z/OS Communications Server: SNA Network Implementation Guide and z/OS Communications Server: SNA Resource Definition Reference.

Note: VTAM®.

Availability of network resources

For FEPI to communicate with the network using a node, both the application minor node and the defining major node must be active, and the minor node must be in a connectable condition.

If FEPI is initialized before the z/OS Communications Server, and is instructed to acquire this node, it retries the VTAM OPEN request several times. Similarly, if a target application is unavailable, FEPI makes another attempt at session initiation. After this, the operator will need to intervene to establish connectivity.

Note: VTAM is now z/OS Communications Server.

Selection of FEPI session parameters

When FEPI establishes a session with a back-end system, it searches the z/OS Communications Server LOGON mode (logmode) table for an entry that corresponds to the simulated device type specified on the **FEPI INSTALL PROPERTYSET** command used to define the pool to which the node-target connection belongs.

If it finds such an entry, it uses it to set the parameters for the session. Suitable mode table entries for FEPI are in the LOGON mode table ISTINCLM. Table 3 on page 20 shows how entries in ISTINCLM correspond to FEPI device types.

Table 3. Relation of FEPI device-types to ISTINCLM mode table entries		
DEVICE CVDA on FEPI INSTALL PROPERTYSET	**Mode table entry in ISTINCLM**	**Session parameters**
T3278M2	D4A32782	LU2 3278 model 2
T3278M3	D4A32783	LU2 3278 model 3
T3278M4	D4A32784	LU2 3278 model 4
T3278M5	D4A32785	LU2 3278 model 5
T3279M2	SNX32702	LU2 3279 model 2
T3279M3	SNX32703	LU2 3279 model 3
T3279M4	SNX32704	LU2 3279 model 4
T3279M5	SNX32705	LU2 3279 model 5
TPS55M2	SNX32702	LU2 PS/55, 24 lines
TPS55M3	SNX32703	LU2 PS/55, 32 lines
TPS55M4	SNX32703	LU2 PS/55, 43 lines
LUP	IBM3600	Secondary LU P (IMS protocol LU 0)
Note: The mode entries are fixed by FEPI; you cannot use any other entries.		

If ISTINCLM is defined as your default LOGON mode table, no additional definitions are required, and FEPI sessions use the characteristics that these entries specify. If you have defined a different default table, which does not contain the supplied entries, or if you want to associate a different set of characteristics with the names listed above (for example, class-of-service or pacing specifications), then you must provide the required entries in a customized mode table. This must be associated with the node via the MODETAB keyword of the z/OS Communications Server APPL statement used to define the node to z/OS Communications Server. For example:

```
DG4FEPI1  APPL ACBNAME=FEPI0001,MODETAB=mode-table-name
```

Note:

1. If you choose to define your own mode table, it needs to contain only those entries that differ from the set supplied in the default mode table (for example, ISTINCLM). If z/OS Communications Server cannot find a given entry in the node-specific mode table, it automatically searches the system default table for an entry of the same name.
2. FEPI establishes the presentation space size of a terminal, based on the session parameters received in response to the session request, *not* on any fixed dimension implied by the device type specified for the pool (although the device type does establish a default value when a default BIND is received).
3. An externally initiated session (one started by the primary LU or by the operator through the VARY LOGON command) can specify any entry name in the mode table. If you expect to make use of external session initiation, it is advisable to specify the DLOGMOD keyword on the APPL statement used to define the node in question. This keyword identifies the mode table entry to be used in those cases where the session initiation request did not specify session parameters. It can be specified regardless of whether the MODETAB keyword is used. For example:

   ```
   DG4FEPI1  APPL ACBNAME=FEPI0001,
                 MODETAB=mode-table-name,DLOGMOD=mode-table-entry-name
   ```

4. If you define your own mode entries, ensure that all the parameters in an entry are appropriate. These logmode entries must be explicitly named in the APPL statements.

Pacing of FEPI sessions

The pacing values used for FEPI sessions should be consistent with whatever installation standards are in effect for other LU2 and SLU P sessions in the network.

Configuring the back-end systems

The only configuration that you must perform for the back-end systems is to provide and manage simulated terminals (LUs) for FEPI to use. These terminals are defined to the back-end CICS or IMS system just like real terminals. They can be explicitly defined or autoinstalled as required.

About this task

The simulated terminals do not have to be defined to z/OS Communications Server in the back-end system, where they appear as real terminals on that system. z/OS Communications Server uses the various network definitions to determine how and where to route data; it can be routed locally, cross-domain, or cross-network. The LU name corresponds to the front-end node name. Similarly, the z/OS Communications Server applid of the back-end system corresponds to the applid in the FEPI target definition. The diagram of the sample configuration in Figure 2 on page 27 describes these relationships.

Procedure

- If you are configuring CICS as the back-end system, the following terminal definitions (TYPETERMs) are acceptable:
  - DFHLU2E2
  - DFHLU2E3
  - DFHLU2E4
  - DFHLU2E5
  - DFHLU2M2
  - DFHLU2M3
  - DFHLU2M4
  - DFHLU2M5

  These definitions match the z/OS Communications Server mode table entries shown in Table 3 on page 20. You must create your own TYPETERMs for 3279 model 5 and PS/55 devices, if required, because no such definitions are supplied by CICS.

  For information about defining terminals to CICS, see TERMINAL resources.
- If you are configuring IMS as the back-end system, use the following settings on the TYPE or TERMINAL system definition macros:

  a) Required: NAME must match the NODE name specified to and used by FEPI

  b) Required: MODETBL must specify the correct LOGMODE.

  The following nondefault settings are recommended. (FEPI will support the default settings as well.)

  a) Specify OPTIONS=OPTACK for more efficient communication.

  b) Specify OPTIONS=FORCRESP so transactions are run in response mode.

  If you let this default, you might get nonresponse mode regardless of how the transactions are defined.

  c) Specify OPTIONS=NORELRQ to make IMS ignore external requests for the node.

  d) Specify OPTIONS=BID to indicate that the z/OS Communications Server VTAM BID command should always precede output messages that occur while between brackets.

  e) Specify OUTBUF=*nnn* to set a bigger output buffer than the default of 256 bytes.

The following example defines some IMS terminals for use by FEPI. You might need to customize it for use in your own IMS environment.

```
TYPE UNITYPE=SLUTYPEP,MODETBL=IBM3600,          x
OPTIONS=(OPTACK,FORCRESP,NORELRQ,BID),OUTBUF=512
TERMINAL NAME=IMSLUP01
NAME IMSLUP01
TERMINAL NAME=IMSLUP02
NAME IMSLUP02
TERMINAL NAME=IMSLUP03
NAME IMSLUP03
TERMINAL NAME=IMSLUP04
NAME IMSLUP04
```

The configuration programs you should write

You must write a setup program to define your FEPI nodes, targets, property sets, and pools.

You can also choose to write:

- A monitoring program to handle unexpected events (including setup errors)
- Any common functions not provided by individual FEPI applications
- One or more global user exit programs
- Some specialized operator transactions, to simplify the control of FEPI resources.

A number of samples have been provided to give you an example of the types of programs that you can write. See FEPI samples for details.

Writing configuration programs

FEPI programs are CICS applications, and so all aspects of CICS programming apply.

Before you begin

Familiarize yourself with the guidance about writing CICS application programs. See the sections about designing efficient applications and dealing with exception conditions.

About this task

To write a configuration program, use the FEPI system programming commands. They are an extension of the **EXEC CICS** commands and have similar names and similar functions. The FEPI commands also have similar keywords, but they are distinguished by having “FEPI” as a prefix.

Procedure

1. You can use the following system programming commands in your configuration program:

   Definition:

   EXEC CICS FEPI INSTALL
   Define communication resources

   EXEC CICS FEPI ADD
   Add resources to a pool

   EXEC CICS FEPI DELETE
   Remove targets or nodes from a pool

   EXEC CICS FEPI DISCARD
   Remove communication resources completely from FEPI.

   Operations:

   EXEC CICS FEPI INQUIRE
   Query FEPI status and resources

EXEC CICS FEPI SET
Control FEPI resources.

2. When translating your programs, you must specify the FEPI option, which instructs the translator to process FEPI commands.

   You do not need the SP option.

3. Select whether your FEPI configuration programs are AMODE(24) or AMODE(31).

   The configuration programs can issue FEPI commands in either 24- or 31-bit addressing mode, and reside above or below the 16MB line.

4. Consider how your configuration program should handle exception conditions.

Exception conditions
As with all CICS commands, FEPI commands might produce exception conditions that you can check using the RESP option, or capture using HANDLE CONDITION. Most FEPI command errors return INVREQ. The particular error in each case is uniquely identified by the RESP2 value.

All the FEPI exception conditions and RESP2 values are listed in FEPI system programming reference. There are copy books that contain declarations for the RESP2 values:

- DFHSZAPA for Assembler language
- DFHSZAPO for COBOL
- DFHSZAPP for PL/I
- DFHSZAPC for C.

For the system programming commands, errors are reported as unexpected events to the CSZX or other transient data queue, and to the FEPI message log CSZL, as well as by exception conditions on the command.

If there is an error, the command does nothing, and output values are not changed. Some commands operate on a list of resources; an error in one resource does not prevent the command from operating on the other resources in the list.

You can use EDF and CECI to debug FEPI programs. Because FEPI commands can be quite long, you will probably find the NAME field of CECI useful.

All resource names used by FEPI are a fixed length of 8 characters; they must be padded with blanks if necessary. For commands that use lists, make sure that the list field is a multiple of 8 characters long and that the number option is set correctly; neither the translator nor CECI checks these and unpredictable results could occur if they are wrong.

Writing a setup program

There are many considerations in designing set up programs, so there is no single recommended way of writing them. However, there are certain functions that your setup program must perform.

About this task

On the distribution tape, the following sample setup programs are provided:

- An Assembler language sample setup program with filename DFH0AZXS
- A COBOL sample setup program with filename DFH0VZXS
- A C sample setup program with filename DFH0CZXS.

These programs install resources to make FEPI function with the other sample programs. They show you one way of writing a setup program.

Your setup program must:

Procedure

1. Install all node names that are available for FEPI.
2. Install all targets that FEPI is permitted to access.
3. Install properties. See “Organizing property sets” on page 14 for guidance on what choices to make. In defining the properties of connections in pools, the following options must be set:

   Device attributes
   DEVICE

   Data handling
   FORMAT, MAXFLENGTH, CONTENTION

   Session management
   BEGINSESSION, ENDSESSION, INITIALDATA, STSN

   Unexpected events
   EXCEPTIONQ, UNSOLDATA, UNSOLDATACK

   Journaling
   MSGJRNL, FJOURNALNUM, FJOURNALNAME
4. Install pools.
5. Associate nodes and targets with the pools to define connections.

What to do next
In addition to a setup program, you may need a corresponding program to deal with deleting and discarding resources.

By default, FEPI resources are available for use as soon as they are installed or associated with a pool. For control, performance, or other reasons, you might want to override this; if so, you must provide a further program (or operations procedure) to bring the resources into service when you require them.

Many of the FEPI commands used by your setup program can use lists; using lists helps to improve performance. If some items in a list fail, errors (both programming errors and resource problems) are reported to your monitoring program, *not* to the setup program. If you want to track the errors in the setup program itself, without using the monitoring program, restrict your lists to a single item. Errors are then reported on the command itself.

Running setup programs

The setup program is typically initiated by a program list table (PLT) program.

About this task
Using this method, the setup program is run automatically at every CICS startup. Follow this procedure:

Procedure

1. Write your setup program.
2. Define it to CICS, using RDO, and associate it with a transaction.

   You can define your setup program statically, or allow it to be installed automatically (autoinstalled) when it is invoked. For details of the CICS autoinstall facility for programs, see Autoinstalling programs, map sets, and partition sets in Configuring.
3. Write a PLT program containing the command:

   ```
   EXEC CICS START TRANSID(tranid) INTERVAL(1)
   ```

   where *tranid* is the ID of your setup transaction. For programming information about writing PLT programs, see Writing initialization and shutdown programs .
4. Define your PLT program to CICS, and include it in the second part of the program list table post initialization (PLTPI) list.

For information about coding entries in the PLTPI list, see Program list table (PLT).

What to do next
There may be a good reason for you to decide not to use the PLT to start the setup transaction. For example, you might want to have several, time-sensitive, setup programs, each having a corresponding discard program. If you decide not to use the PLT, you must arrange to start the setup transactions manually.

Restrict access to the setup programs, because they are of a sensitive nature.

Varying the resources installed by the setup program

Unless your setup program contains some conditional logic, you always get the same set of FEPI resources installed. This may be exactly what you require, but if not, here are a few techniques that might prove useful.

Checking startup type

Your setup program can determine how the CICS system started by issuing an **EXEC CICS INQUIRE SYSTEM STARTUP** command. It could use this to install different sets of FEPI resources for warm and cold starts.

Recording the status of resources

If you install all your FEPI resources at CICS startup, and then alter their accessibility, consider writing a non-terminal transaction that runs frequently and uses the **FEPI INQUIRE** commands to determine the status of each FEPI resource. Write these to a *recoverable* temporary storage file. (You could, for example, use an XSZARQ global user exit program to log changes to FEPI resources.) At restart time, your setup program can read the file to determine the required access settings.

Using timed actions

You could take advantage of CICS automatic transaction initiation (ATI) at specified times to control FEPI resources. If you want to terminate FEPI access to another system at a specific time each day, schedule a transaction to run at the required time. When this transaction runs it can either make the required FEPI resources unavailable for access, or discard them. Because FEPI resources remain available for use by current tasks in this circumstance, this has no effect on existing FEPI users.

You could use timed initiation in a similar way to make FEPI resources available.

Using event handlers

Another way of controlling FEPI resources is to use the begin-session and end-session event handlers. (See “Other functions” on page 34.)

These handlers are invoked when a conversation starts and ends. Although they are primarily designed to handle signon and signoff to the back-end systems, you can take advantage of the fact that all FEPI functions are available to them. So you can use them to control access to back-end systems by either installing or discarding FEPI resources.

For example, suppose you want to ensure that no FEPI application is waiting for a connection to a back-end system. In the handlers, issue **FEPI INQUIRE POOL** commands, and look at the WAITCONVNUM option, which returns the number of FEPI applications waiting for a connection. If this option exceeds a certain trigger value, issue FEPI commands to increase the number of connections (that is, add nodes, define new pools, and so on).

This technique can be extended to provide tuning of FEPI access to back-end systems.

An example FEPI configuration

An example configuration is given in Table 4 on page 28. Next, the target lists and node lists used in the example are given. Then there are the definitions used to achieve this configuration. Figure 2 on page 27 is a diagrammatic representation of the configuration.

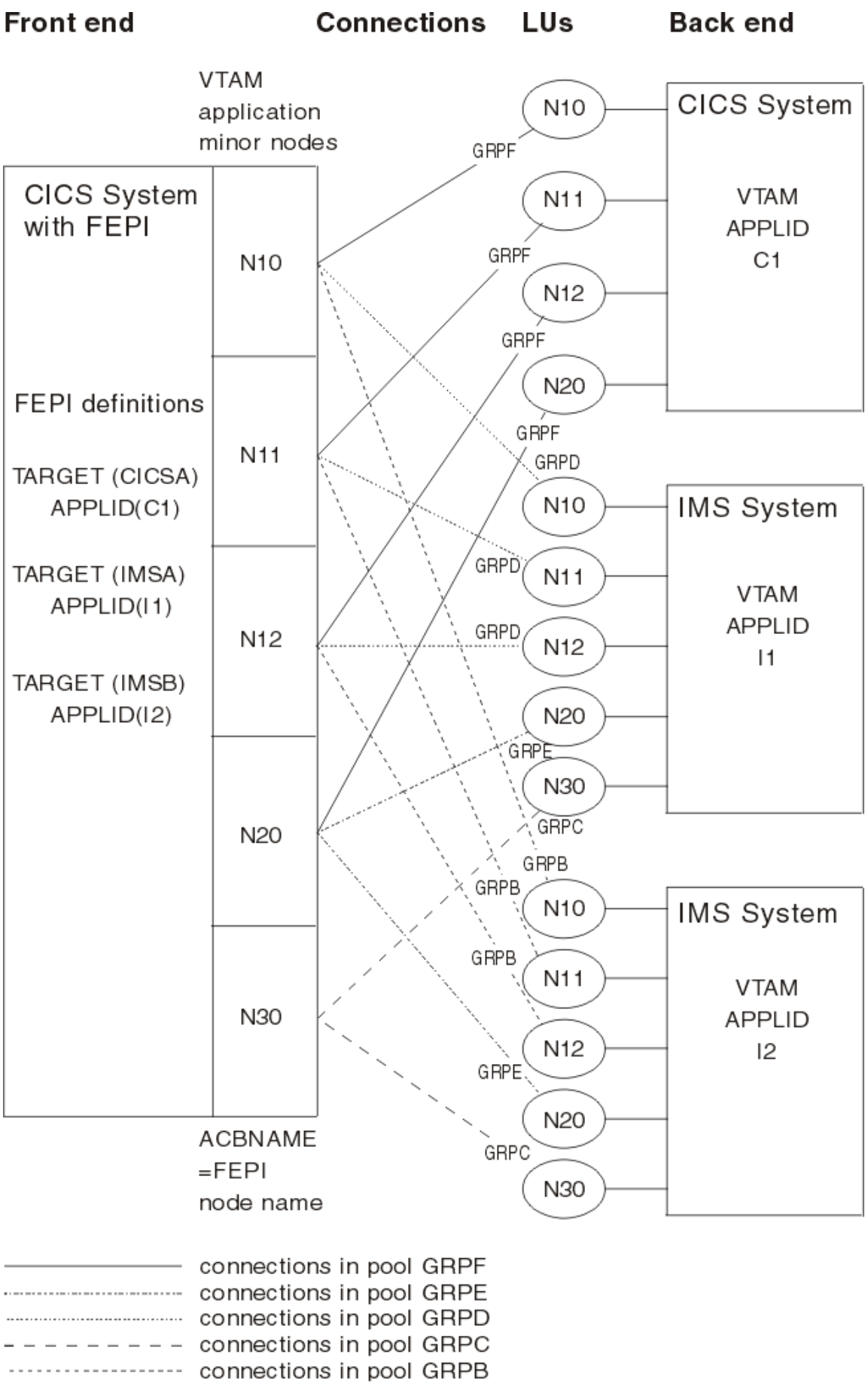


Figure 2. The example FEPI configuration - a diagrammatic representation

Note that this is not the configuration the sample programs use; it illustrates as many aspects of configuration as possible.

Table 4. Resources used in the example FEPI configuration

Resource	**name**	**name**	**name**	**name**	**name**
Pool name	GRPB	GRPC	GRPD	GRPE	GRPF
Property set	SLUP	SLU2M3I	SLU2M3I	SLU2M2I	SLU2M2C
Target names	IMSB	IMSA IMSB	IMSA	IMSA IMSB	CICSA
Node names	N10 N11 N12	N30	N10 N11 N12	N20	N10 N11 N12 N20
Device type	LUP	T3278M3	T3278M3	T3278M2	T3278M2
Logmode name	IBM3600	D4A32783	D4A32783	D4A32782	D4A32782
Exceptional events queue name	IEXEPTP	IEXEPT2	IEXEPT2	IEXEPT2	CEXEPT2
Unsolicited-data transaction name or response	IUP	IU2	IU2	IU2	Negative
Begin-session transaction name	ISIP	ISI2	ISI2	ISI2	CSI2
End-session transaction name	none	IXI2	IXI2	IXI2	CXI2
STSN transaction name	ISTP	n/a	n/a	n/a	n/a
Initial inbound data	No	Yes	Yes	Yes	Yes

Sample lists

Here are the target lists and node lists used in the sample configuration, padded to eight bytes per item.

TLIST
```
'CICSA IMSA IMSB '
```
TLISTA
```
'IMSA '
```
TLISTB
```
'CICSA '
```
TLISTC
```
'IMSA IMSB '
```
TLISTD
```
'IMSB '
```
NLIST
```
'N10 N11 N12 N20 N30 '
```
NLISTA
```
'N10 N11 N12 '
```
NLISTB
```
'N20 '
```
NLISTC
```
'N30 '
```
NLISTD
```
'N10 N11 N12 N20 '
```

The following is the list of z/OS Communications Server application names of the back-end CICS and IMS systems with which FEPI applications will communicate.

PLIST

```
   'C1 I1 I2 '
```

Sample definitions

The following definitions illustrate the various possibilities when defining FEPI resources.

Define the back-end subsystems you want FEPI to access

This defines the logical names (targets) that FEPI uses to refer to back-end systems (in this case CICSA, IMSA, and IMSB as given in TLIST), and relates them to their z/OS Communications Server names (C1, I1, and I2 as given in PLIST).

```
EXEC CICS FEPI INSTALL TARGETLIST(TLIST) TARGETNUM(3)
     APPLLIST(PLIST)
```

Define the z/OS Communications Server minor nodes available to FEPI

The names are N10, N11, N12, N20, and N30, as given in NLIST.

```
EXEC CICS FEPI INSTALL NODELIST(NLIST) NODENUM(5)
```

Define properties

The properties define the characteristics of the connections.

SLU P connections

```
EXEC CICS FEPI INSTALL PROPERTYSET(SLUP)
     LUP                    /* Device type (SLU P) */
     BEGINSESSION(ISIP)     /* Begin session handler */
     STSN(ISTP)             /* STSN transaction */
     EXCEPTIONQ(IEXEPTP)    /* Exception report TD queue */
     UNSOLDATA(IUP)         /* Unsolicited-data transaction */
     NOTINBOUND             /* No "good morning" message */
```

SLU2 24 x 80 connections to IMS

```
EXEC CICS FEPI INSTALL PROPERTYSET(SLU2M2I)
     T3278M2                /* Device type (3278 model 2, 24 x 80) */
     BEGINSESSION(ISI2)     /* Begin session handler */
     EXCEPTIONQ(IEXEPT2)    /* Exception report TD queue */
     UNSOLDATA(IU2)         /* Unsolicited-data transaction */
     INBOUND                /* Initial data */
     ENDSESSION(IXI2)       /* End session handler */
```

SLU2 32 x 80 connections to IMS

```
EXEC CICS FEPI INSTALL PROPERTYSET(SLU2M3I)
     T3278M3                /* Device type (3278 model 3, 32 x 80) */
     BEGINSESSION(ISI2)     /* Begin session handler */
     EXCEPTIONQ(IEXEPT2)    /* Exception report TD queue */
     UNSOLDATA(IU2)         /* Unsolicited-data transaction */
     INBOUND                /* Initial data */
     ENDSESSION(IXI2)       /* End session handler */
```

SLU2 24 x 80 connections to CICS

```
EXEC CICS FEPI INSTALL PROPERTYSET(SLU2M2C)
     T3278M2                /* Device type (3278 model 2, 24 x 80) */
     BEGINSESSION(CSI2)     /* Begin session handler */
```

```
     EXCEPTIONQ(CEXEPT2)      /* Exception report TD queue */
     NEGATIVE                 /* Response to unsolicited data */
     INBOUND                  /* "Good morning" message */
     ENDSESSION(CXI2)         /* End session handler */
```

Define the pools of connections

The pools define connections between targets and nodes; they specify which nodes can be used to access which target, and what properties the connection has.

```
EXEC CICS FEPI INSTALL POOL(GRPB) PROPERTYSET(SLUP)
     TARGETLIST(TLISTD) TARGETNUM(1)
     NODELIST(NLISTA) NODENUM(3)
EXEC CICS FEPI INSTALL POOL(GRPC) PROPERTYSET(SLU2M3I)
     TARGETLIST(TLISTC) TARGETNUM(2)
     NODELIST(NLISTC) NODENUM(1)
EXEC CICS FEPI INSTALL POOL(GRPD) PROPERTYSET(SLU2M3I)
     TARGETLIST(TLISTA) TARGETNUM(1)
     NODELIST(NLISTA) NODENUM(3)
EXEC CICS FEPI INSTALL POOL(GRPE) PROPERTYSET(SLU2M2I)
     TARGETLIST(TLISTC) TARGETNUM(2)
     NODELIST(NLISTB) NODENUM(1)
EXEC CICS FEPI INSTALL POOL(GRPF) PROPERTYSET(SLU2M2C)
     TARGETLIST(TLISTB) TARGETNUM(1)
     NODELIST(NLISTD) NODENUM(4)
```

Writing monitoring programs

You must write a monitoring program to handle unexpected events that are reported by FEPI and errors returned by system programming commands.

FEPI reports these events by writing a record to a transient data (TD) queue. You can define pool-specific TD queues for FEPI, where information about events that relate to specific pools is reported. (There is also a common FEPI TD queue, CSZX, where events that do not relate to specific pools are reported.) Note that, if a pool-specific event occurs, and you have not defined a corresponding queue, information about the event is lost. Also, FEPI TD queues must be defined as NONRECOVERABLE; if a queue is ‘recoverable’, FEPI does not write to it, and discards any information about unexpected events.

Typically, you would arrange for the monitoring program to be triggered whenever an item is placed in a TD queue. (Define the queue with a trigger level of 1.) A single monitoring program can service several queues, by using EXEC CICS ASSIGN QNAME to check which queue triggered it. According to the nature of the event, the monitoring program might write a message, log the event, or embark on a full conversation.

For example, using this method, whenever a session is lost, the monitoring program is invoked. The TD queue data provides information about what happened. Your monitoring program can obtain this in the usual way with EXEC CICS READQ TD. The following copy books describe the structure of the data:

- DFHSZAPA for Assembler language
- DFHSZAPO for COBOL
- DFHSZAPP for PL/I
- DFHSZAPC for C.

Your program may then choose to reestablish the lost session, to reinitialize, and so on. It may also set indicators for the application programs if contact with a target has been lost altogether.

Monitoring programs are written using the techniques and commands discussed in Developing with the FEPI API. See also the overview of the sample monitoring program in FEPI sample program: Monitor and unsolicited data-handler.

Handling unexpected events

This section suggests some actions your monitoring program could take after various types of unexpected event.

The type of event is indicated by the EVENTTYPE area in the TD queue record. In most cases, the EVENTVALUE area gives specific details of the failure; the values are the same as the RESP2 values listed in FEPI RESP2 values.

Events in CSZX TD queue records

The CSZX TD queue records report a number of events relating to FEPI resources and sessions.

INSTALLFAIL

A FEPI resource has failed to be installed. This is probably because you are trying to install a duplicate name. This might indicate either a logic error or a possible security violation.

Recommended action: Report possible application logic error, for investigation.

DISCARDFAIL

A FEPI resource has not been discarded. This is probably because you are trying to discard a nonexistent object. This might indicate a logic error.

Recommended action: Report possible application logic error, for investigation.

SETFAIL

A FEPI resource has rejected a SET request. This is probably because you are trying to manipulate a resource that does not exist. However, there is also the possibility of rejection due to z/OS Communications Server considerations. So SETFAIL might indicate either a logic error or a network failure.

Recommended action: Schedule a transaction to repeat the operation (if not a logic error).

ACQFAIL

A FEPI resource has failed to be acquired. This is probably because of a network failure, and so FEPI automatically retries the acquire request several times at intervals; the count in EVENTDATA shows whether there will be any more retries. However, there is also the possibility of an error in either the z/OS Communications Server definition or the back-end system definition of the object.

Recommended action: After FEPI stops retrying, suggest investigating the condition of the resource from a z/OS Communications Server viewpoint. The z/OS Communications Server sense code describing the problem is in EVENTDATA. See z/OS Communications Server: SNA Messages.

SESSION

An unsolicited bind was received, probably because of a CLSDST(PASS). See “Handling CLSDST(PASS)” on page 32.

Events in pool-specific TD queue records

Pool-specific TD queue records report a number of events relating to problems with connections and pool definitions.

SESSIONLOST

An active connection has failed. A probable reason is that the back-end system has failed. However, this error is also generated when an operator cancels an active connection.

Recommended actions: The following operator actions are recommended:

- Investigate the condition of the connection from a z/OS Communications Server viewpoint. The z/OS Communications Server sense code that describes the problem is in EVENTDATA. For more information about the sense codes, see z/OS Communications Server: SNA Messages and z/OS Communications Server: IP and SNA Codes.
- Check whether the back-end system is still running.
- Check that the back-end system has not "closed" the FEPI simulated terminal.

SESSIONFAIL
A connection has failed to start. Probable reasons are a setup inconsistency or a failure of the back-end system. FEPI automatically retries the acquire request several times at intervals; the count in EVENTDATA shows whether there will be any more retries. However, this failure is also generated when an operator cancels the connection.

Recommended action: After FEPI stops retrying the acquire request, the following operator actions are recommended:

- Investigate the condition of the connection from a z/OS Communications Server viewpoint. The z/OS Communications Server sense code that describes the problem is in EVENTDATA. For more information about the sense codes, see z/OS Communications Server: SNA Messages and z/OS Communications Server: IP and SNA Codes.
- Check whether the back-end system is still running.
- Check that the back-end system has not "closed" the FEPI simulated terminal.
- Check that the terminal type definition in the back-end system matches the FEPI device type.

ADDFAIL
An attempt to add a target or node to a pool has failed. A probable reason is that there was an attempt to add a resource that is already in the pool. This situation indicates a possible logic error.

Recommended action: Report or investigate the possible application logic error.

DELETEFAIL
An attempt to delete a target or node from a pool has failed. A probable reason is that there was an attempt to delete a resource that is not in the pool. This situation indicates a possible logic error.

Recommended action: Report or investigate the possible application logic error.

Handling CLSDST(PASS)

A back-end system can end a network session with a z/OS Communications Server CLSDST(PASS) request.

This indicates that the back-end will reestablish a session with the front-end using a different PLU name (a *third-party* PLU). The front-end system detects reestablishment of the session by receiving an unsolicited bind request; so when the back-end system ends a session, it is important for it to indicate that an unsolicited bind is to be expected.

Note: To determine whether a lost session was caused by a CLSDST(PASS) request, a FEPI application can issue a FEPI INQUIRE CONNECTION command. If the value of LASTACQCODE is X'32020000', the back-end system issued a CLSDST(PASS) to unbind the session.

The three most likely scenarios are described in the following sections.

Unsolicited bind not expected

FEPI unconditionally rejects the bind request.

Third-party PLU name known and unsolicited bind expected

The prospective PLU names must be defined to FEPI as targets.

You might need to restrict access to the pools that include these targets to make sure the connection is not already in use when the CLSDST(PASS) takes place. The simplest way to configure this is to define a pool containing the node and all the targets it can be placed in session with. Install all connections except the initial one with an ACQSTATUS of RELEASED so the back-end system can successfully acquire the session. No other special processing is required and no TD queue record is written in this case.

Third-party PLU name not known and unsolicited bind expected

The necessary resource definitions must be managed dynamically.

Note: Managing the resource definitions dynamically (described under “Conversation in progress” on page 33) is the only method that allows the conversation to persist across the CLSDST(PASS).

When FEPI receives the unsolicited bind, it writes a record to the CSZX TD queue, with an EVENTTYPE of SESSION, and with the third-party PLU name in the TARGET area. At this point, the bind has not been accepted or rejected. A z/OS Communications Server display for either the back-end or the front-end system would show the connection to be in a PSESST/B state. You are responsible for managing these TD queue records and making the necessary FEPI configuration updates so that processing can continue. If no action is taken, the session remains in this state until a z/OS Communications Server VTAM VARY NET,TERM command is issued to terminate the session request.

There are two cases, according to whether or not there is a conversation in progress on the connection when the CLSDST(PASS) occurs. (This can be determined from the STATE option of the FEPI INQUIRE CONNECTION command.) In both cases, you need to determine which pool has the connection that the CLSDST(PASS) applies to, because the TD queue record does not report either the pool or the old target name. If the node is used in only one pool, the old target name can be found easily by browsing connections using FEPI INQUIRE CONNECTION; if not, use some other technique, such as the USERDATA option of the FEPI SET commands.

Conversation in progress
Nodes for which this kind of processing is required should be defined in pools containing only the node and the initial target, because of the nature of the processing involved.

About this task

The monitor program should:

1. Install a new pool with the same properties as the current one.
2. Install a new target whose PLU name is the third-party PLU name given in the TARGET area of the TD queue record.
3. Add the target to the new pool. This should be the only target in that pool.
4. Delete the node identified in the TD queue record from the pool in which it currently exists. If necessary, to ensure continuity, the monitor program can add another node to the pool before deleting the old node.
5. Add the node to the newly created pool. The new connection is now established.

When the session ends, the connection reverts to a RELEASED state. If necessary, use an end-session handler to perform any necessary clean up, such as reversing the process described previously.

The front-end application must also anticipate CLSDST(PASS) processing. See Lost session for more details.

Conversation not in progress
The CLSDST(PASS) occurred as a result of trying to acquire a connection.

About this task

The monitor program should:

1. Install a new target whose PLU name is the third-party PLU name given in the TARGET area of the TD queue record.
2. Add the target to the pool, specifying an intended connection acquire status of ACQUIRED. The new connection is now established.

If necessary, use an end-session handler to clean up the dynamically defined targets. These connections always become RELEASED when the session ends and can be reused, if required.

Writing operator transactions

You might find it useful to write some specialized operator transactions of your own to control FEPI resources.

For more information, see Controlling FEPI resources.

Other functions

The other functions you might need to write for FEPI itself are the begin-session, end-session, and unsolicited-data handlers.

These are extensions of the FEPI application programs, and are described in Developing with the FEPI API. If you write them as common functions, you need to know what the application programs do. Alternatively, the application programmer may write them.

Global user exit programs

There are two global user exits, XSZBRQ and XSZARQ.

XSZBRQ

XSZBRQ is invoked before a FEPI command is executed.

XSZBRQ is passed the parameters input to the command, and can be used to monitor commands, to bypass commands that violate installation conventions, or to change the parameters of a command, subject to the rules applying to global user exits.

XSZARQ

XSZARQ is invoked after a FEPI command is executed.

XSZARQ is passed the parameters output from the command.

For details of the FEPI global user exits, see Front End Programming Interface exits XSZARQ and XSZBRQ. For programming information about writing and using global user exit programs, see Writing global user exit programs.

Chapter 3. Administering FEPI

After you have configured FEPI and deployed FEPI applications, you can use the facilities in CICS to manage the FEPI environment. For example, you can control FEPI resources and manage FEPI with z/OS Communications Server persistent sessions.

The section contains the following topics:

Controlling FEPI resources

The FEPI INQUIRE and SET functions can be carried out by a program, or by using the master terminal transaction, CEMT. You may find it useful to write some specialized operator transactions of your own.

The FEPI INQUIRE command (and its CEMT equivalent) tells you what resources are defined and their statuses. The only thing you cannot do directly is determine which nodes and targets are in a particular pool. Do this using CEMT to inquire about the connections in a particular pool:

```
CEMT I FECONNECTION POOL(poolname)
```

To do this from an application program, browse all connections and select those in the pool you want.

Here are the resource statuses of most interest:

SERVSTATUS

SERVSTATUS is used with connections, nodes, pools, and targets.

It specifies the service status of the resource—that is, whether it can be used for a conversation. The service status can be set to INSERVICE to allow usage, or to OUTSERVICE to stop usage for any *new* conversation. Note that setting OUTSERVICE does not end any existing conversations that are using the resource; the status is GOINGOUT until the existing conversations end.

ACQSTATUS

ACQSTATUS is used with connections and nodes.

ACQSTATUS specifies the "acquire status" of the resource. For a connection, this means whether it should have a session established (bound) or ended (unbound). For a node, it means whether the z/OS Communications Server ACB for the node should be opened or closed. The acquire status can be set to ACQUIRED (a status of ACQUIRING indicates that the acquisition has not yet been completed), or to RELEASED .

Setting RELEASED does not end any existing conversations that are using the resource; the acquire status is RELEASING until the existing conversations end. However, for connections, a conversation that is unowned and in a "pending" state (see “STATE” on page 37) is ended immediately if the acquire state is set to RELEASED ; this means that connections being used by a failed application can be recovered.

ACQUIRING and RELEASING are shown as BEING ACQUIRED and BEING RELEASED by CEMT.

Network and other problems can cause connections to become stuck in a RELEASING or ACQUIRING state, in which case the operator might need to intervene using VTAM operator commands.

Note: VTAM is now the z/OS Communications Server.

If a FEPI connection remains in a RELEASING state for longer than expected, try the following:

1. Note the node and target associated with the connection; use CEMT INQUIRE FETARGET to find the z/OS Communications Server application name that the target represents.
2. Issue the following VTAM command to find out the state of network session associated with the connection:

   ```
   D NET,E,ID=nodename
   ```

3. Note the session status. See z/OS Communications Server: IP and SNA Codes for an explanation of the status. If no session exists and a subsequent INQUIRE of the connection status using CEMT shows the state still as BEING RELEASED, there has been a system failure; you should collect diagnostic information.
4. If the session is in "session takedown processing", you can use the VTAM command

   ```
   D NET,SESSION
   ```

   to find out what signals are needed to complete processing.
5. If you can resolve the problem using commands on the back-end system, attempt to do so.
6. If there is no other way to resolve the session status, you can use the VTAM command

   ```
   V NET,TERM
   ```

   to end the network procedure in progress. FEPI will then be able to complete processing.

If an ACQUIRING state has persisted for too long, and you cannot determine why the session has not been established, follow the same procedure described previously. If no session is active for the connection, FEPI is currently waiting for the retry interval to expire. The system log should contain VTAM messages explaining why the session cannot be established. The LACQCODE option of CEMT INQUIRE FECONNECTION gives the reason code z/OS Communications Server provided for the last session failure.

Also be sure to check that the node on which the connection depends is properly acquired; if not, resolve whatever problem is indicated by the LACQCODE option for the node.

Note that, under normal circumstances, after a FEPI FREE RELEASE command has been issued the session does not remain in RELEASED state, because FEPI automatically tries to reacquire the session. However, if a FEPI SET CONNECTION ACQSTATUS(RELEASED) command is issued before the FREE RELEASE, the session remains in RELEASED state.

LASTACQCODE

The INQUIRE CONNECTION or INQUIRE NODE commands can use the option LASTACQCODE (LACQCODE in CEMT), which returns the result of the last acquire request.

This result is the sense code from the last z/OS Communications Server operation, where zero indicates success. For a full explanation of z/OS Communications Server sense codes, see z/OS Communications Server: SNA Messages for connections and z/OS Communications Server: IP and SNA Codes for nodes.

INSTLSTATUS

INSTLSTATUS is used with connections, nodes, pools, and targets. It specifies whether the resource is installed, or is in the process of being discarded, waiting for the conversations that are using it to end.

WAITCONVNUM

WAITCONVNUM shows how many conversations are currently waiting to start using a connection or pool.

If WAITCONVNUM is nonzero for significant periods of time, it might mean that you need to allocate extra resources to meet the demand. Or it might mean that applications are holding on to resources for too long.

STATE

STATE is used with connections.

It shows the state of the conversation that is using a connection. See State for the values that STATE can have.

If any of the "pending" states (PENDSTSN, PENDBEGIN, PENDDATA, PENDSTART, PENDFREE, PENDRELEASE, PENDUNSOL, or PENDPASS) is shown, it indicates that the conversation is unowned, "pending" the event or task shown. If a pending state persists, it is likely that the application has failed in some way; you should consider resetting the connection by issuing a FEPI SET CONNECTION RELEASED command.

Shutdown

FEPI shutdown is triggered as part of CICS shutdown—you cannot shut down FEPI alone.

There are three forms of shutdown:

- Normal
- Immediate
- Forced.

Normal shutdown

A normal shutdown of CICS causes FEPI to shut down normally - active transactions are allowed to terminate. When all active conversations have ended and all FEPI resources have been discarded, FEPI shuts down.

While FEPI is shutting down, no *new* conversations can be started, but existing owned conversations continue. However, these cannot use the **FEPI START** or **FEPI FREE PASS** commands. Existing unowned conversations are ended immediately. Any FEPI transactions that you want to be able to start during CICS shutdown must be defined in the transaction list table (XLT).

If an end-session handler is invoked at the end of conversations, it is told that the session is to be ended because of CICS shutdown. The handler can choose to perform additional back-end operations that might be needed because of the shutdown. If you require this function, make sure the end-session handler transaction is defined in the transaction list table (XLT), and that it does not adversely affect the performance of CICS shutdown. (For details of how to define entries in the XLT, see Transaction list table (XLT) .)

CICS normal shutdown waits until FEPI shutdown has completed before continuing processing. So if you know when CICS shutdown is to occur, you should initiate FEPI DISCARD operations before starting CICS termination. Removing FEPI resources as they become inactive allows existing FEPI conversations to continue, but prevents new ones from starting. You could achieve the same effect by setting the status of FEPI resources to OUTSERVICE,RELEASED.

If shutdown is not proceeding, then before you force it to continue, consider carefully whether the problem is due to:

- A back-end system taking a long time to respond. In this case, do not attempt to speed things up—you may generate integrity errors in the back-end system.
- A FEPI failure. In this case, issue the following commands, pausing after each step to see whether CICS is still waiting:
  1. CEMT DISCARD FExxxx(*), to remove all FEPI resources
  2. CEMT SET FECONNECTION(*) OUTSERVICE RELEASED, to end any waiting conversations
  3. CEMT SET TASK(nnn) FORCE, to end any running FEPI transactions
  4. Attempt to issue z/OS Communications Server **VTAM VARY NET,INACT,FORCE** commands from the system console to terminate connections.

If CICS shutdown still does not proceed, you cannot perform a warm shutdown. Try issuing a **CEMT P SHUT IMMEDIATE** command. If this fails, you must cancel CICS .

Immediate shutdown

An immediate shutdown of CICS immediately terminates FEPI. There is nothing you can do to influence this process.

Forced shutdown

A forced shutdown of CICS immediately terminates FEPI. There is nothing you can do to influence this process.

Using FEPI with z/OS Communications Server persistent sessions

When creating FEPI applications, you need to be aware of the possible effects of the use of z/OS Communications Server persistent sessions in the front- or back-end systems.

For information about CICS support for z/OS Communications Server persistent sessions, see the Troubleshooting for recovery processing .

Restart of front-end system using persistent sessions

Using persistent sessions in the front-end does not give FEPI any additional recoverability benefits. FEPI is always initialised with a cold start; thus, to FEPI, the effect of restarting a front-end system for which persistent sessions support is enabled is indistinguishable from a cold start of CICS.

Restart of back-end system using persistent sessions

In the back-end system, there are terminal definitions that are used when the FEPI simulated terminals establish sessions with the target.

These definitions may be hard-coded, or may be autoinstall model definitions. If the terminal definitions have been set up to use persistent session support, and the back-end system is restarted within the persistent session delay interval, the terminal sessions are recovered.

Effect on FEPI application programs

It is likely that FEPI application programmers have little say in the way that persistent session support is used in the back-end system.

They therefore need to be aware of the different ways in which terminal sessions can be recovered, so that their applications cater for all possibilities. The way in which a session is recovered depends on the setting of the RECOVOPTION and RECOVNOTIFY options of the TYPETERM definition.

RECOVOPTION(SYSDEFAULT)

On restart within the persistent session delay interval, CICS selects the optimum procedure to recover a session.

For LU2, if the session is busy and CICS is in send mode, CICS sends an end bracket. If the session is busy and CICS is not in send mode, CICS sends an SNA CLEAR request to reset the conversation state.

If a FEPI conversation is in progress when the target system terminates, your application could see one of the following:

- A timeout on a RECEIVE, CONVERSE, or START command, while it waits for the target to restart.

  Deal with this in the normal way for a timeout.
- A FEPI RECEIVE or CONVERSE command completes as a result of the end bracket sent by CICS. The RU on this data flow may be empty or may contain a user-defined message, depending on the value of the RECOVNOTIFY option.

  Your application may need to perform some backout processing.

- An INVREQ response with a RESP2 value of 230 on a FEPI SEND, RECEIVE, CONVERSE, ISSUE, or START command, indicating that an SNA CLEAR was received.

  Your application may need to perform some backout processing.

You must also consider the value specified for RECOVNOTIFY:

RECOVNOTIFY(MESSAGE)
A message (defined in the BMS maps DFHXRC3 and DFHXRC4) is sent to the “terminal”. Your FEPI application must contain logic to deal with this data flow.

If there is no active conversation at the time of restart, the flow is received as unsolicited data at the FEPI front-end.

RECOVNOTIFY(TRANSACTION)
A transaction is initiated in the target. The default is the Good Morning transaction. Your application must contain logic to deal with this data flow.

If there is no active conversation at the time of restart, the flow is received as unsolicited data at the FEPI front-end.

RECOVNOTIFY(NONE)
The “terminal” is not notified that a restart has occurred. Your application need take no special action.

RECOVOPTION(CLEARCONV)
On restart within the persistent session delay interval, CICS sends an SNA CLEAR request to reset the conversation states. The CLEAR is sent only if the session was busy at the time of system restart. If a FEPI conversation is in progress when the target system terminates, your application could see one of the following:

- A timeout on a RECEIVE, CONVERSE, or START command, while it waits for the target to restart.

  Deal with this in the normal way for a timeout.
- An INVREQ response with a RESP2 value of 230 on a FEPI SEND, RECEIVE, CONVERSE, ISSUE, or START command, indicating that an SNA CLEAR was received.

  Your application may need to perform some backout processing.

You must also consider the value specified for RECOVNOTIFY. The possible values are as described for RECOVOPTION(SYSDEFAULT).

RECOVOPTION(RELEASESESS)
On restart within the persistent session delay interval, CICS sends an UNBIND request to release an active session. The request is sent only if the session was busy at the time of system restart.

If a FEPI conversation is in progress when the target system terminates, your application could see one of the following:

- A timeout on a RECEIVE, CONVERSE, or START command, while it waits for the target CICS to restart.

  Deal with this in the normal way for a timeout.
- An INVREQ response with a RESP2 value of 215 on any FEPI command, indicating a 'session lost' condition.

  Deal with this in the normal way for a session loss.

RECOVOPTION(UNCONDREL)
On restart within the persistent session delay interval, CICS sends an UNBIND request to release an active session. The request is sent whether or not the session was busy at the time of system restart.

If a FEPI conversation is in progress when the target system terminates, your application could see either of the symptoms described for RECOVOPTION(RELEASESESS).

RECOVOPTION(NONE)

Even if the system is restarted within the persistent session delay interval, the session is not recovered; it has no persistent session support.

Deal with this in the normal way for a session loss.

Operator control of FEPI

Two CICS -supplied transactions, CEMT and CETR, provide operator control of FEPI. You can also use z/OS Communications Server commands to manage communication with target systems.

FEPI application programs, and the CICS resources they use, are controlled just like other CICS applications and resources.

Using CEMT to control FEPI resources

The CEMT- master terminal transaction has a range of commands (shown in the following list) that support FEPI. You can use the **CEMT INQUIRE**, **SET**, and **DISCARD** commands to control FEPI resources such as nodes, targets, and pools.

The following commands work exactly like other CEMT commands - for example, in supporting resource selection by families (AB*, for example), lists (AB,CD,EF, for example), and by subdefining groups. Note that 4-character option names are used in the display.

- "CEMT DISCARD" on page 40
- "CEMT INQUIRE FECONNECTION" on page 41
- "CEMT INQUIRE FENODE" on page 44
- "CEMT INQUIRE FEPOOL" on page 46
- "CEMT INQUIRE FEPROPSET" on page 48
- "CEMT INQUIRE FETARGET" on page 48
- "CEMT SET FECONNECTION" on page 50
- "CEMT SET FENODE" on page 51
- "CEMT SET FEPOOL" on page 51
- "CEMT SET FETARGET" on page 52

Using CETR to control FEPI trace

You can use the CETR transaction to control tracing activity. For instructions, see CETR - trace control.

CEMT DISCARD

The **CEMT DISCARD** command removes targets, nodes, pools, or property sets completely from FEPI.

Syntax

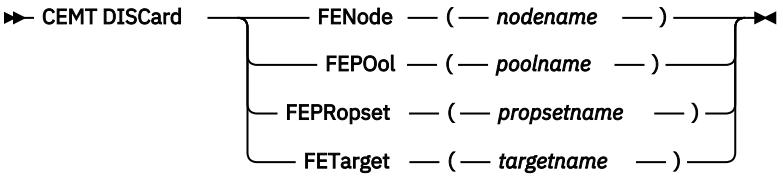
Press the Clear key to clear the screen. Type `CEMT DISCARD` (or suitable abbreviations for the keywords), fololowed by one of the following options:

- `FENODE(` *nodename* `)`
- `FEPOOL(` *poolname* `)`
- `FEPROPSET(` *propsetname* `)`
- `FETARGET(` *targetname* `)` .

For example, `cemt disc fen(fepnode1)` removes the node fepnode1 from FEPI.

Typing ? at the beginning of either the first or second line gives a syntax prompt.

CEMT DISCARD



Options

FENode(*nodename*)
The name of the FEPI node to be discarded.

FEPOol(*poolname*)
The name of the FEPI pool to be discarded.

FEPRopset(*propsetname*)
The name of the FEPI property set to be discarded.

FETarget(*targetname*)
The name of the FEPI target to be discarded.

CEMT INQUIRE FECONNECTION

Display information about FEPI connections.

Description

The INQUIRE FECONNECTION command displays information about the state of FEPI connections. A connection is identified by specifying the target and node. The results are given in order of target within the node. You can use family selection for TARGET and NODE, but you cannot use list selection.

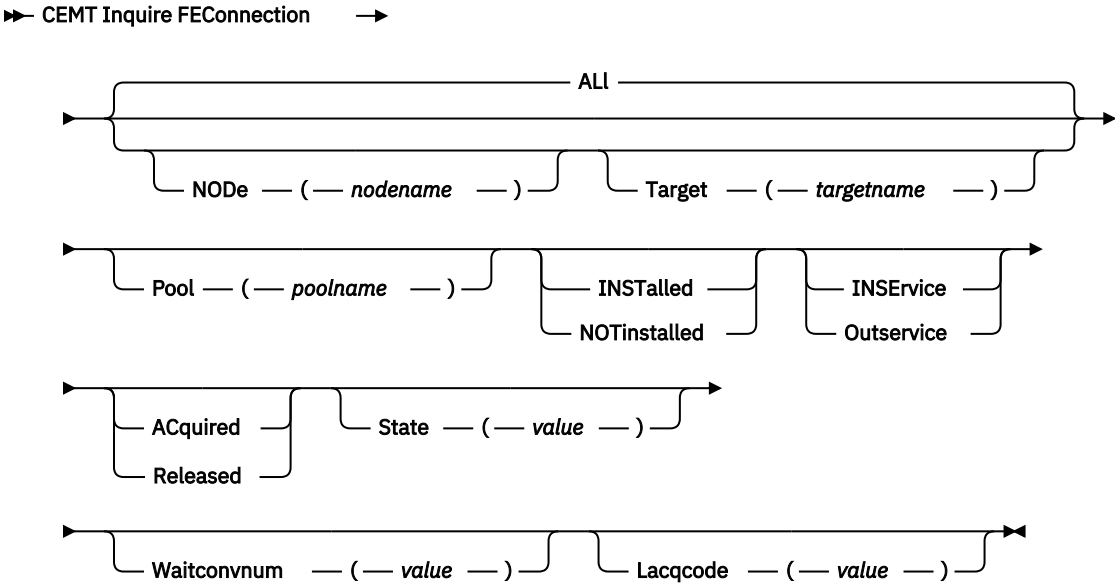
Input

Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT INQUIRE FECONNECTION` (or suitable abbreviations for the keywords). The resulting display lists the current status.
- Type `CEMT INQUIRE FECONNECTION` (or suitable abbreviations for the keywords), followed by the attributes that are necessary to limit the range of information that you require. For example, if you enter `cemt i fec p(pool5) acq` , the resulting display shows the details of all FEPI connections in the pool named pool5 on which sessions are bound.

You can tab to the highlighted fields and overtype them with new values.

CEMT INQUIRE FECONNECTION



ALl
The default. Information about all connections is given, unless you specify a selection.

NODe(nodename)
The 8-character name of a node. Information is restricted to connections of which this node forms part.

Target(targetname)
The 8-character name of a target. Information is restricted to connections of which this target forms part.

Sample screen

```
CEMT IN FEC
STATUS: RESULTS - OVERTYPE TO MODIFY
Node(NODE1 )
Targ(TARGETA ) Pool(POOL5 ) Inst
Inse Rele
Stat(NOCONV ) Wait(00000) Lacq(X'08570002')
Node(NODE1 )
Targ(TARGETB ) Pool(POOL5 ) Inst
Inse Rele
Stat(NOCONV ) Wait(00000) Lacq(X'08570002')
Node(NODE1 )
Targ(TARGET3 ) Pool(POOL3 ) Inst
Inse Rele
Stat(NOCONV ) Wait(00000) Lacq(X'08570002')
```

Figure 3. CEMT INQUIRE FECONNECTION screen

Displayed fields

Node(*value*)
Displays the 8-character name of a node that identifies a connection.

Target(*value*)
Displays the 8-character name of a target that identifies a connection.

Pool(*poolname*)
Displays the 8-character name of a pool of connections.

Installed|Notinstalled
Displays a value that identifies the installation state of the connection. The values are as follows:

Installed
The connection is in a pool that has been defined by INSTALL and is available for use.

Notinstalled
The connection is in a pool, or involves a node or target that is being discarded, but is still in use.

Inservice|Outservice
Displays a value that identifies the service state of the connection. The values are as follows:

Inservice
The connection is in service and can be used in a conversation. If OUTSERVICE state has been requested but has not yet completed, a "GOING OUT" message is shown.

Outservice
The connection is out of service and cannot be used for any conversation.

Acquired|Released
Displays a value that identifies whether a session on the connection is bound. The values are as follows:

Acquired
A session is bound on the connection. If RELEASED state has been requested but has not yet completed, a "BEING RELEASED" message is shown. If this message persists, you might need to use z/OS Communications Server commands to recover the connection.

Released
Sessions involving the connection have been unbound. If ACQUIRED state has been requested but has not yet completed, a "BEING ACQUIRED" message is shown. If this message persists, you might need to use z/OS Communications Server commands to recover the connection.

State(*value*)
Displays a 12-character value that identifies the state of the conversation using the connection. The values are as follows:

APPLICATION
A normal application task owns the conversation.

BEGINSESSION
A begin-session handling task owns the conversation.

FREE
An end-session handling task owns the conversation, following a FEPI FREE command.

NOCONV
No conversation is active on the connection.

PENDBEGIN
A begin-session handling task has been scheduled.

PENDDATA
FEPI is waiting for inbound data, following a FEPI START command.

PENDFREE
An end-session handling task has been scheduled, following a FEPI FREE command.

PENDPASS
The conversation is unowned, following a FEPI FREE PASS command.

PENDRELEASE
An end-session handling task has been scheduled, following an unbind request.

PENDSTART
Inbound data having arrived, a task specified by FEPI START has been scheduled.

PENDSTSN
An STSN-handling task has been scheduled.

PENDUNSOL
An unsolicited-data handling task has been scheduled

RELEASE
An end-session handling task owns the conversation, following an unbind request.

STSN
An STSN-handling task owns the conversation.

UNSOLDATA
An unsolicited-data handling task owns the conversation.

The pending states indicate the conversation is unowned, pending the event or task indicated. If a pending state persists, it is likely that the application has failed in some way; you should consider resetting the connection by issuing a **CEMT SET FECONNECTION RELEASED** command.

Waitconvnum(*value*)
Displays a value that identifies the number of conversations that are waiting to start using a connection. If a conversation could use any one of several connections, it is counted as waiting on each one.

Lacqcode(*value*)
Displays a hexadecimal value that indicates the result of the last acquire request for the node; that is, the sense code from the last z/OS Communications Server REQSESS. A value of zero indicates success. For information about z/OS Communications Server sense codes, see z/OS Communications Server: SNA Messages or z/OS Communications Server: IP and SNA Codes.

CEMT INQUIRE FENODE

Display information about a FEPI node.

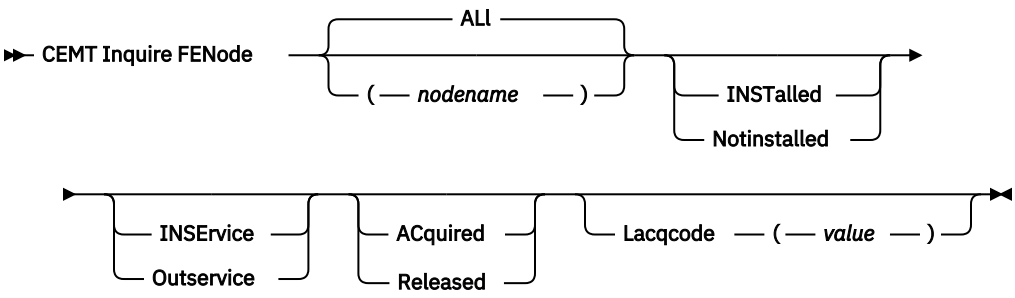
Input

Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT INQUIRE FENODE` (or suitable abbreviations for the keywords). The resulting display lists the current status.
- Type `CEMT INQUIRE FENODE` (or suitable abbreviations for the keywords), followed by the attributes that are necessary to limit the range of information that you require. For example, if you enter `cemt i fen inst`, the resulting display shows the details of all FEPI nodes that have been installed and are ready for use.

You can tab to the highlighted fields and overtype them with new values.

CEMT INQUIRE FENODE



ALL
The default. Information about all nodes is given, unless you specify a node.

nodename
The 8-character name of the node to be queried.

Sample screen

```
CEMT IN FEN
STATUS: RESULTS - OVERTYPE TO MODIFY
Feno(NODE1 ) Inst
Inse Acqu
Lacq(X'00000000')
Feno(NODE2 ) Inst
Inse Acqu
Lacq(X'00000000')
Feno(NODE3 ) Inst
Inse Acqu
Lacq(X'00000000')
Feno(NODE4 ) Inst
Inse Acqu
Lacq(X'00000000')
```

Figure 4. CEMT INQUIRE FENODE screen

Displayed fields

Feno
Indicates that this panel relates to an FENODE inquiry.

(*value*)
Displays the 8-character name of a node.

Installed|Notinstalled
Displays a value that identifies the installation state of the node. The values are as follows:

Installed
The node has been defined by INSTALL and is available for use.

Notinstalled
The node is being discarded, but is still in use.

Inservice|Outservice
displays a value that identifies the service state of the node. The values are as follows:

Inservice
The node is in service and can be used in a conversation. If OUTSERVICE state has been requested but has not yet completed, a "GOING OUT" message is shown.

Outservice
The node is out of service and cannot be used for any conversation.

Acquired|Released
Displays a value that identifies the state of the z/OS Communications Server ACB for the node. The values are as follows:

Acquired
The z/OS Communications Server ACB for the node is open and the z/OS Communications Server **`set logon start`** command has completed. If RELEASED state has been requested but has not yet completed, a "BEING RELEASED" message is shown. If this message persists, you might need to use z/OS Communications Server commands to recover the node.

Rele ased
The z/OS Communications Server ACB is closed. If ACQUIRED state has been requested but has not yet completed, a "BEING ACQUIRED" message is shown. If this message persists, you might need to use z/OS Communications Server commands to recover the node.

Lacqcode(*value*)
Displays a hexadecimal value that indicates the result of the last acquire request for the node; that is, the sense code from the last z/OS Communications Server OPEN ACB. A value of zero indicates

success. For information about z/OS Communications Server sense codes, see z/OS Communications Server: IP and SNA Codes.

CEMT INQUIRE FEPOOL

Display information about the state of FEPI pools of connections.

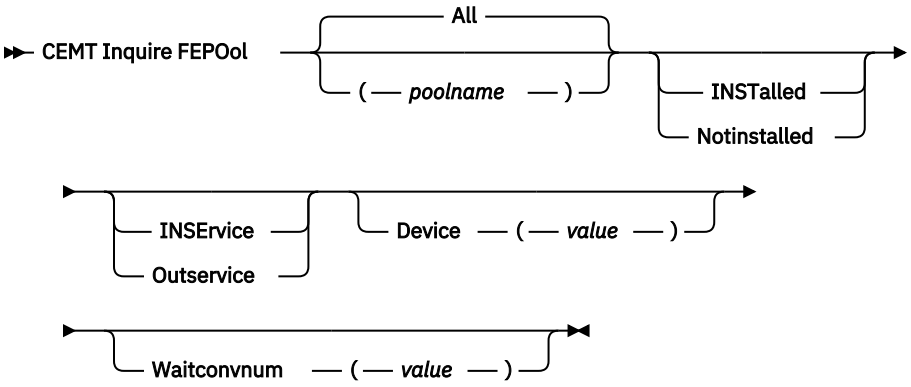
Input

Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT INQUIRE FEPOOL` (or suitable abbreviations for the keywords). The resulting display lists the current status.
- Type `CEMT INQUIRE FEPOOL` (or suitable abbreviations for the keywords) followed by the attributes that are necessary to limit the range of information that you require. For example, if you enter `cemt i fepo inse` , the resulting display shows the details of all FEPI pools that are in service and can be used by conversations.

You can tab to the highlighted service state field and overtype it with a new value.

CEMT Inquire FEPOol



All
The default. Information about all pools is given, unless you specify a pool to be queried.

poolname
Specifies the name of a pool of connections.

Sample screen

```
CEMT IN FEPO
STATUS: RESULTS - OVERTYPE TO MODIFY
Fepo(POOL3 ) Inst
Inse
Devi(T3278M4 ) Wait(00000)
Fepo(POOL5 ) Inst
Inse
Devi(T3278M2 ) Wait(00000)
```

Figure 5. CEMT INQUIRE FEPOOL screen

Displayed fields

Fepo
Indicates that this panel relates to an FEPOOL inquiry.

(value)
Displays the 8-character name of a pool of connections.

Installed|Notinstalled
Displays a value that identifies the installation state of the pool. The values are as follows:

Installed
The pool has been defined by INSTALL and is available for use.

Notinstalled
The pool is being discarded, but is still in use.

Inservice|Outservice
Displays a value that identifies the service state of the pool. The values are as follows:

Inservice
The pool is in service and can be used in a conversation. If OUTSERVICE state has been requested but has not yet completed, a "GOING OUT" message is shown.

Outservice
The pool is out of service and cannot be used for any conversation.

Device(*value*)
Displays a value that identifies the mode of conversation and the type of device. The values are as follows:

T3278M2
SLU2 mode, 3278 Model 2

T3278M3
SLU2 mode, 3278 Model 3

T3278M4
SLU2 mode, 3278 Model 4

T3278M5
SLU2 mode, 3278 Model 5

T3279M2
SLU2 mode, 3279 Model 2B

T3279M3
SLU2 mode, 3279 Model 3B

T3279M4
SLU2 mode, 3279 Model 4B

T3279M5
SLU2 mode, 3279 Model 5B

TPS55M2
SLU2 mode, PS/55, 24 lines

TPS55M3
SLU2 mode, PS/55, 32 lines

TPS55M4
SLU2 mode, PS/55, 43 lines

LUP
SLU P mode, all cases

Waitconvnum(*value*)
Displays a value that identifies the number of conversations that are waiting to start using a connection in the pool.

CEMT INQUIRE FEPROPSET

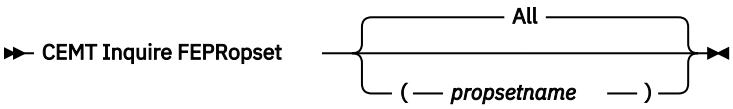
Display information about a set of FEPI properties.

Input

Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT INQUIRE FEPROPSET` (or suitable abbreviations for the keywords). The resulting display lists all FEPI property sets that are currently installed.
- Type `CEMT INQUIRE FEPROPSET` (or suitable abbreviations for the keywords), followed by the name of a specific property set. For example, if you enter `cemt i fepr (feprop1)` , the resulting display shows whether the FEPI property set named feprop1 is installed. If it is not installed, the response is "NOT FOUND" .

CEMT INQUIRE FEPROPSET



All
The default. Information about all property sets is given, unless you specify a particular one.

propsetname
The name of the property set to be queried.

Sample screen

```
CEMT IN FEPR
                    STATUS: RESULTS
                    Fepr(PROP1 )
                    Fepr(PROP2 )
                    Fepr(PROP3 )
                    Fepr(PROP4 )
```

Figure 6. CEMT INQUIRE FEPROPSET screen

Displayed fields

Fepr
Indicates that this panel relates to an FEPROPSET inquiry.

(value)
Displays the 8-character name that identifies a property set.

CEMT INQUIRE FETARGET

Display information about the state of FEPI targets.

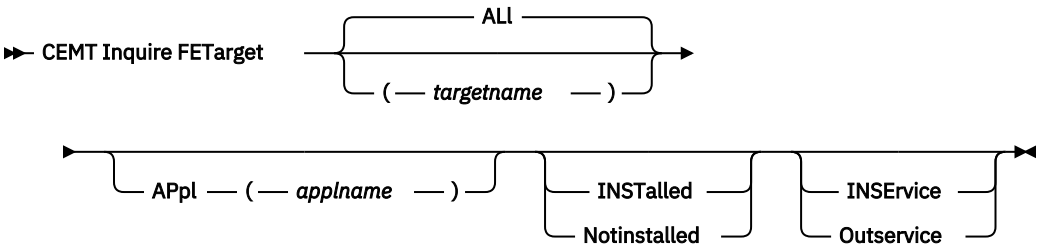
Input

Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT INQUIRE FETARGET` (or suitable abbreviations for the keywords). The resulting display lists the current status.
- Type `CEMT INQUIRE FETARGET` (or suitable abbreviations for the keywords), followed by the attributes that are necessary to limit the range of information that you require. For example, if you enter `cemt i fet inse` , the resulting display shows the details of all FEPI targets that are in service.

You can tab to the highlighted service state field and overtype it with a new value.

CEMT INQUIRE FETARGET



ALL
The default. Information about all targets is given, unless you specify the target to be queried.

targetname
The name of the target to be queried.

Sample screen

```
CEMT IN FET
STATUS: RESULTS - OVERTYPE TO MODIFY
Feta(TARGETA )
Appl(APPL5 ) Inst
Inse
Feta(TARGETB ) Appl(APPL6 ) Inst
Inse
Feta(TARGET1 ) Appl(APPL1 ) Inst
Inse
Feta(TARGET2 ) Appl(APPL2 ) Inst
Inse
Feta(TARGET3 ) Appl(APPL3 ) Inst
Inse
Feta(TARGET4 ) Appl(APPL4 ) Inst
Inse
```

Figure 7. CEMT INQUIRE FETARGET screen

Displayed fields

Feta
Indicates that this panel relates to an FETARGET inquiry.

(*value*)
Displays the 8-character name that identifies a target.

Appl(*applname*)
Displays the 8-character z/OS Communications Server application name of the back-end system that the target represents.

Installed|Notinstalled
Displays a value that identifies the installation state of the target. The values are as follows:

Installed
The target has been defined by INSTALL and is available for use.

Notinstalled
The target is being discarded, but is still in use.

Inservice|Outservice
Displays a value that identifies the service state of the target. The values are as follows:

Inservice
The target is in service and can be used in a conversation. If OUTSERVICE state has been requested but has not yet completed, a "GOING OUT" message is shown.

Outservice
The target is out of service and cannot be used for any conversation.

CEMT SET FECONNECTION

Change the state of FEPI connections. Family selection can be used for TARGET and NODE, but list selection cannot be used.

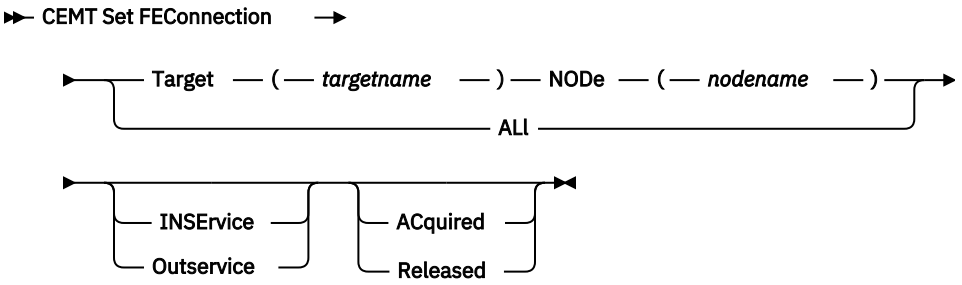
Syntax

Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT SET FECONNECTION` (or suitable abbreviations for the keywords), followed by either `TARGET( targetname ) NODE( nodename )` or `ALL` . The resulting display lists the current status, similar to that obtained by using the **`CEMT INQUIRE FECONNECTION`** command. You can tab to the highlighted fields and overtype them with new values.
- Type `CEMT SET FECONNECTION` (or suitable abbreviations for the keywords), followed by either `TARGET( targetname ) NODE( nodename )` or `ALL` , then followed by one or more attribute settings that you want to change. For example, `cemt s fec al ac` causes sessions to be bound for all FEPI connections.

Typing ? at the beginning of either the first or second line gives a syntax prompt. Resetting the values takes effect immediately.

CEMT SET FECONNECTION



Options

ACquired
Specifies that the connection is to have a session established (that is, bound). The state is ACQUIRING until this action completes.

ALl
Specifies that any change you request is made to all connections that you are authorized to access.

INSErvice
Specifies that the connection is to be put in service and can be used in a conversation.

NODe(nodename)
Specifies the 8-character name of the node that identifies a connection.

Outservice
Specifies that the connection is to be put out of service and not to be used for any new conversations, although existing conversations are unaffected. The service state is GOINGOUT until the existing conversations end.

Released
Specifies that the connection is to have its session ended (that is, unbound), when usage of the connection by all owned conversations ends. An unowned conversation on the connection is ended immediately. The state is RELEASING until this action completes.

Target(targetname)
Specifies the 8-character name of the target that identifies a connection.

CEMT SET FENODE

Change the state of FEPI nodes.

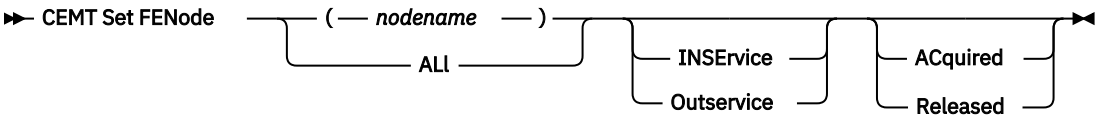
Syntax

Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT SET FENODE` (or suitable abbreviations for the keywords), followed by either a *nodename* or `ALL`. The resulting display lists the current status, similar to that obtained by using the **CEMT INQUIRE FENODE** command. You can tab to the highlighted fields and overtype them with new values.
- Type `CEMT SET FENODE` (or suitable abbreviations for the keywords), followed by either a *nodename* or `ALL`, then followed by one or more attribute settings that you want to change. For example, `cemt s fen al ac` causes the z/OS Communications Server ACBs for all FEPI nodes to be opened, and 'set logon start' to be done.

Typing ? at the beginning of either the first or second line gives a syntax prompt. Resetting the values takes effect immediately.

CEMT Set FENode



Options

ACquired
Specifies that the z/OS Communications Server ACB for the node should be opened, and the z/OS Communications Server set logon start command is to be issued. The state is ACQUIRING until this is completed.

ALl
Specifies that any change you request is made to all nodes that you are authorized to access.

INSErvice
Specifies that the node is in service and can be used in a conversation.

(nodename)
Specifies the 8-character name of the node whose state is to be changed.

Outservice
Specifies that the node is to be put out of service and cannot be used for any new conversations, although existing conversations are unaffected. The service state is GOINGOUT until the existing conversations end.

Released
Specifies that the z/OS Communications Server ACB for the node is to be closed, when usage of the node by any conversation ends. The state is RELEASING until this is completed.

CEMT SET FEPOOL

Change the state of FEPI pools of connections.

Syntax

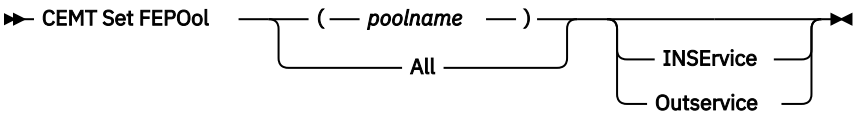
Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT SET FEPOOL` (or suitable abbreviations for the keywords), followed by either a *poolname* or `ALL`. The resulting display lists the current status, similar to that obtained by using the **CEMT INQUIRE FEPOOL** command. You can tab to the highlighted service state field and overtype it with a new value.

- Type `CEMT SET FEPOOL` (or suitable abbreviations for the keywords), followed by either a *poolname* or `ALL` , then followed by a service state setting. For example, `cemt s fepo fepool1 i` specifies that the pool named fepool1 is in service and available for use by a conversation.

Typing ? at the beginning of either the first or second line gives a syntax prompt. Resetting the values takes effect immediately.

CEMT SET FEPOOL



Options

All
Specifies that any change you request is made to all pools that you are authorized to access.

INSErvice
Specifies that the pool is in service and can be used in a conversation.

Outservice
Specifies that the pool is put out of service and cannot be used for any new conversations, although existing conversations are unaffected. The service state is GOINGOUT until the existing conversations end.

(*poolname*)
Specifies the pool of connections to be changed.

CEMT SET FETARGET

Change the state of FEPI targets.

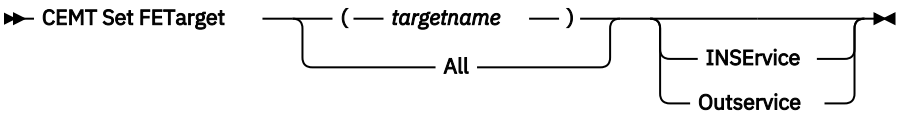
Syntax

Press the Clear key to clear the screen. You can start this transaction in two ways:

- Type `CEMT SET FETARGET` (or suitable abbreviations for the keywords), followed by either a *targetname* or `ALL` . The resulting display lists the current status, similar to that obtained by using the **`CEMT INQUIRE FETARGET`** command. You can tab to the highlighted service state field and overtype it with a new value.
- Type `CEMT SET FETARGET` (or suitable abbreviations for the keywords), followed by either a *targetname* or `ALL` , then followed by a service state setting. For example, `cemt s fet fetarg1 i` specifies that the target named fetarg1 is in service and available for use by a conversation.

Typing ? at the beginning of either the first or second line gives a syntax prompt. Resetting the values takes effect immediately.

CEMT Set FETarget



Options

All
Specifies that any change you request is made to all targets that you are authorized to access.

INSErvice
Specifies that the target is in service and can be used in a conversation.

Outservice
Specifies that the target is out of service and cannot be used for any new conversations, although existing conversations are unaffected. The service state is GOINGOUT until the existing conversations end.

(*targetname*)
Specifies the 8-character name of the target to be changed.

z/OS Communications Server commands

In addition to the resource control facilities provided by FEPI, you can use specific z/OS Communications Server commands to manage communication with target systems.

These commands are useful where there are problems in acquiring or releasing sessions. See "ACQSTATUS" on page 35.

Note: z/OS Communications Server was previously known as VTAM.

The following list summarizes the z/OS Communications Server commands you can use. For a full description of each command, see z/OS Communications Server: SNA Operation.

- – Use the **DISPLAY** command to inquire about the status of the FEPI nodes (acting as SLUs) and the target systems. Normally, you need to use this command only when there are problems communicating with a particular target. To understand the displays, you require some knowledge of how the z/OS Communications Server operates. For explanations of messages, see z/OS Communications Server: SNA Messages .
  - Use the VARY command to control the availability of resources in the network. For FEPI, you can use this command to force the closure of a node regardless of whether it is in use in an active conversation. To do this, make the z/OS Communications Server node inactive. Any pending request to change to a state of RELEASED or OUTSERVICE can complete before the node becomes inactive. A subsequent VARY ACTIVE command makes the node available for use again (if its state is still INSERVICE).
  - Use the VARY TERM command to terminate individual connections; that is, to end the session between a particular PLU (target) and SLU (FEPI) pair.
  - Use the DISPLAY SESSIONS command to diagnose problems in establishing sessions. To use this command, you require an understanding of the z/OS Communications Server session processing.

Chapter 4. Customizing FEPI

CICS provides global user exits for FEPI so that you can customize the features of FEPI for your CICS environment. You can also customize FEPI to write data to journal records and print them.

For more information about customizing CICS, see Developing system programs.

For information about FEPI global user exits, see Front End Programming Interface exits XSZARQ and XSZBRQ.

This section contains Product-sensitive Programming Interface information.

FEPI journaling

This section describes the format of FEPI journal records, and how to print them.

For background information about CICS journaling, you should refer to Reading log streams using batch jobs (DFHJUP).

FEPI journal operation

You can request FEPI to write inbound, outbound, or both inbound and outbound data to a specified CICS user journal; you cannot write to the system log. This is done using the **MSGJRNL**, **FJOURNALNUM**, and **FJOURNALNAME** options in your property set definitions.

Of the various reasons for using CICS journaling, the following are particularly relevant to FEPI processing:

- Creating audit trails
- Monitoring performance
- Controlling message security.

Table 5 on page 55 shows the types of FEPI data that can be journaled.

Table 5. FEPI journaled data

FEPI command	Data flow	Type
SEND	Outbound	Data stream Formatted, screen image Formatted, key stroke
RECEIVE	Inbound	Data stream Formatted, screen image
CONVERSE	Outbound	Data stream Formatted, screen image Formatted, key stroke
CONVERSE	Inbound	Data stream Formatted, screen image
EXTRACT FIELD	Inbound	Extract field data

The records journaled by FEPI are identified in the usual way by module and function identifiers. These are listed in Table 6 on page 55.

Table 6. FEPI journal record identifiers

Identifier-type	Name	Value	Type of data
Module identifier	MODIDFEP	X'5D'	Identifies FEPI records in the journal

Table 6. FEPI journal record identifiers (continued)			
Identifier-type	**Name**	**Value**	**Type of data**
Function identifiers	FIDFEPIN FIDFEPOU	X'F0' X'F1'	Identifies FEPI inbound data Identifies FEPI outbound data

In order to identify the conversation for which the data was journaled, FEPI provides a prefix area in the journal record.

Printing FEPI journal records

Each FEPI journal record contains a prefix area which contains FEPI-related information.

You can select FEPI journal records in the following ways, using a batch job like the CICS-supplied utility program DFHJUP

The FEPI prefix area lies within the API user header, as shown in Figure 8 on page 56.

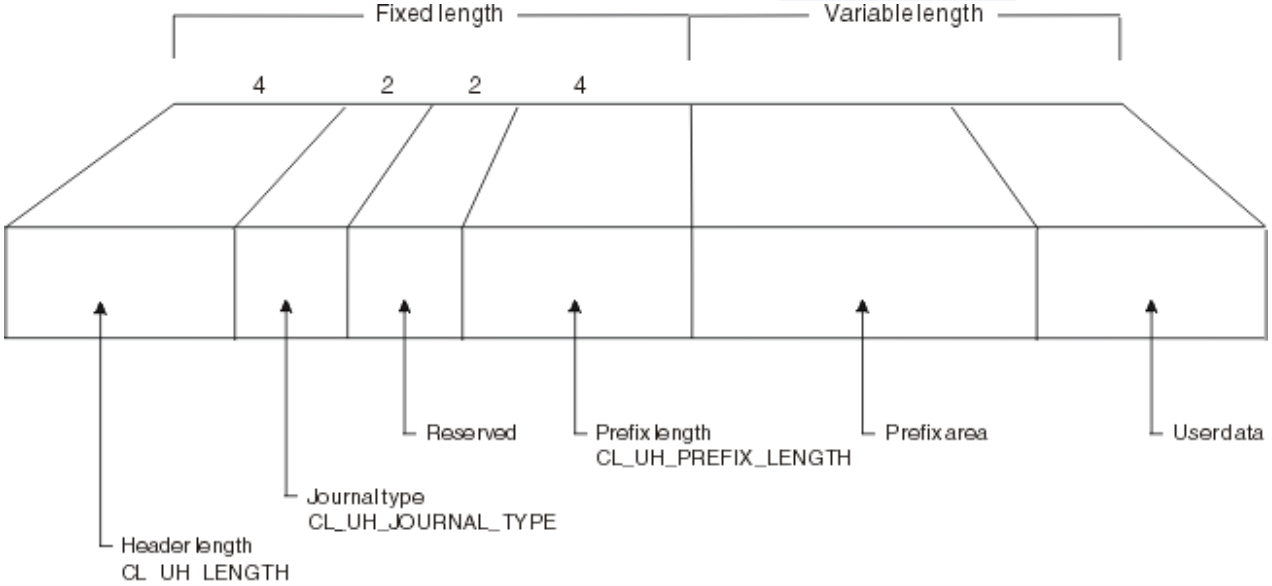


Figure 8. Format of the API user header, showing the position of the prefix area

CL_UH_LENGTH
4-byte length of header

CL_UH_JOURNAL_TYPE
2-byte journal type

Reserved
2-byte reserved field

CL_UH_PREFIX_LENGTH
4-byte length of prefix

Prefix area
The variable length prefix

User data
Variable length user data

The exact format of this FEPI prefix area is shown in Figure 9 on page 57.

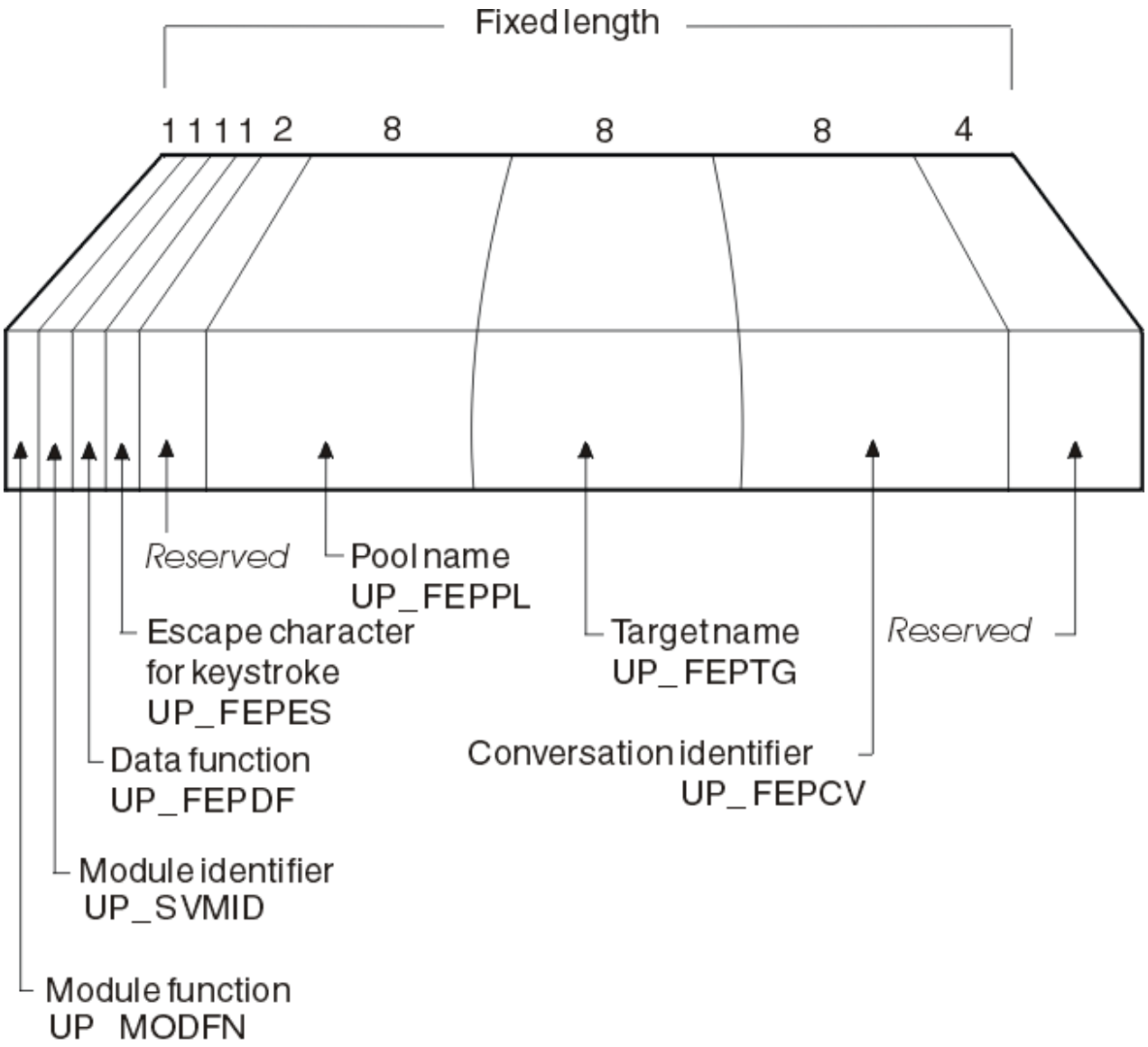


Figure 9. Format of the FEPI prefix area

UP_MODFN
1-byte module function.

UP_SVMID
1-byte module identifier.

UP_FEPDF
1-byte data function.

Field UP_FEPDF can take any of the following values:

Table 7. Values of UP_FEPDF

Field name	Value	Meaning
UP_FEPDD	1	Data stream
UP_FEPDS	2	Formatted, screen image
UP_FEPDK	3	Formatted, keystroke
UP_FEPDE	4	Extract field data

UP_FEPES
1-byte escape character for keystroke.

Reserved
2-byte reserved field.

UP_FEPPL
8-byte pool name.

UP_FEPTG
8-byte target name.

UP_FEPCV
8-byte conversation identifier.

Reserved
4-byte reserved field.

For examples of how you can use the CICS-supplied utility program, DFHJUP, to select FEPI records for printing, see Reading log streams using batch jobs (DFHJUP).

For more information about journal records, see Journal records.

Chapter 5. Developing with the FEPI API

Application developers can use CICS API commands to develop FEPI applications. This information describes how to develop FEPI applications, including the interfaces that are available and how to handle conversations and errors.

- "Basics of FEPI programming" on page 59 introduces FEPI programming and the commands that are used.
- "FEPI key stroke and screen-image applications" on page 62 discusses the high-level interface for FEPI applications.
- "FEPI data stream applications" on page 68 describes the low-level interface for FEPI applications.
- "FEPI application design" on page 73 describes the programs comprising a FEPI application and various design aspects such as conversation ownership, handling errors, and specific requirements for CICS and IMS back-end systems.
- "Specialized FEPI functions" on page 83 describes control functions, normally handled by FEPI, that can be taken over by FEPI applications.
- contains reference information for the FEPI commands that are used in application programs.

Basics of FEPI programming

These topics introduce FEPI programming and the FEPI commands that you can use.

Before reading this information, you should be familiar with the FEPI concepts and facilities described in Introduction to FEPI.

To write FEPI front-end applications, you need to know how to write programs in at least one of the programming languages that CICS supports. More importantly, you also need knowledge of data communication and protocols. And, if you will be accessing IMS back-end systems, you must also be familiar with using IMS and writing IMS applications.

The applications that you write using FEPI are normal CICS transactions with the familiar EXEC CICS commands. These FEPI applications use the FEPI subset of EXEC CICS application programming commands to:

- Allocate a connection from a pool
- Communicate with a back-end application using this connection
- Free the connection when finished.

Communication and conversations

A FEPI application runs in a front-end CICS system and accesses applications in a back-end CICS or IMS system. FEPI lets it do this by simulating a terminal connected to the back-end system; this means that it has to act just like a real terminal and terminal operator.

The back-end systems are known as *targets* and the connections to them are arranged in *pools* that define the properties controlling communication. Targets, pools, and properties are defined by your system programmer, who can tell you which targets and pools to use and what properties they have.

When a connection has been established, on successful completion of a bind, the connection is *in session* and it can be allocated by FEPI for a *conversation* with the back-end system.

Conversations are the basis of all FEPI applications and, depending upon the needs of your application, may be used in several ways (see "FEPI application design" on page 73):

- A single conversation for all transactions on a back-end system
- A different conversation for each transaction or associated series of transactions

- A special conversation to handle unusual events.

The task that started the conversation owns it and other tasks cannot issue commands for it; however, the owning task can transfer ownership to another task. You can have as many conversations as you like at a time with various targets: they can be consecutive or, much more usefully, interleaved.

FEPI simulates a 3270-type terminal (SLU2 mode) for both CICS and IMS systems; it also supports the SLU P mode that is used by IMS for programmable terminals such as the 4700 family. The mode to be used, SLU2 or SLU P, is a property of the pool being used. Your application cannot change the mode of a conversation.

The data that you send and receive can be **formatted** or **data stream** and, as with mode, the data type is a property of the pool being used:

Formatted
A high-level data interface for SLU2 mode. The data sent by the FEPI application can be either **key stroke** format or **screen-image** format; data received by the application is in screen-image format.

Data stream
A low-level data interface for more sophisticated SLU2 mode applications and for use with SLU P mode. The data sent and received by the FEPI application is the data stream; applications using this format have access to some very specialized z/OS Communications Server communication functions.

The same basic set of FEPI commands is used for all modes and data types and protocols, but the command options and keywords are generally different.

Structure and design

In addition to your main access program that handles communication with the back-end system, you might need to provide programs for other functions.

These functions are as follows:

Begin session
Handle begin-session processing.

Unsolicited data
Handle unsolicited inbound data that arrives when there is no conversation.

End session
Handle end of conversation and end of session processing.

These functions could be combined in one program or implemented in separate programs with individual transaction names. There may be any number of each function, again according to your requirements and preferences. Suggestions about the various possibilities are given later.

As the application programmer, you will always write the main access programs. Sometimes the system programmer provides any special functions that are required; otherwise you would be responsible for these. Even if you are writing only the main access program, you need to be aware of what these special functions do and how they affect how you communicate with the back-end system. Because the use of these special functions is controlled by the pools that you use, you need to liase with the system programmers or administrators who set them up.

Several different styles of access program are possible:

One-out one-in conversational
One program performs the complete conversation with the target and each conversation has a single transmission to and from the back-end system.

Conversational
One program performs the complete conversation with the target with multiple transmissions to and from the back-end system, waiting each time for the inbound data.

Pseudoconversational
Here, one program sends data to the target and requests CICS to start another program when the inbound data arrives.

The section beginning with “FEPI key stroke and screen-image applications” on page 62 and ending with “Specialized FEPI functions” on page 83 describes the various features of writing application programs. A set of sample programs is available to help you to get started; these are supplied as source code on the distribution tape. For details, see FEPI samples.

Programming
FEPI programs are CICS applications, so all aspects of CICS programming apply. The FEPI application programming commands are an extension of the EXEC CICS commands. They have similar names and similar functions. The FEPI commands also have similar keywords, but they are distinguished by having FEPI as a prefix.

Your FEPI application programs can be AMODE(24) or AMODE(31) - that is, they can issue FEPI commands in either 24- or 31-bit addressing mode, and reside above or below the 16MB line.

The application programming commands are:

EXEC CICS FEPI ALLOCATE
Starts a conversation with a back-end system.

EXEC CICS FEPI FREE
Ends the conversation with a back-end system.

EXEC CICS FEPI REQUEST PASSTICKET
Requests the external security manager to supply a password substitute.

EXEC CICS FEPI SEND
Sends data to the back-end system.

EXEC CICS FEPI RECEIVE
Receives data from the back-end system.

EXEC CICS FEPI CONVERSE
Sends data to and receives data from the back-end system.

EXEC CICS FEPI ISSUE
Sends control data to the back-end system.

EXEC CICS FEPI EXTRACT
Gets field data and attributes, set-and-test sequence number (STSN) data, or information about a conversation.

EXEC CICS FEPI START
Schedules a CICS transaction to handle inbound data.

Note that, when translating your programs, you must specify the FEPI option; this instructs the translator to process FEPI commands.

Exception conditions
As with all CICS commands, FEPI commands may produce exception conditions that you can check using the RESP option, or capture using HANDLE CONDITION.

Most FEPI command errors return INVREQ. The particular error in each case is uniquely identified by the RESP2 value. All the FEPI exception conditions and RESP2 values are listed in . There are copy books that contain declarations for the RESP2 values:

- DFHSZAPA for Assembler language
- DFHSZAPO for COBOL
- DFHSZAPP for PL/I
- DFHSZAPC for C.

If there is an error, the command does nothing, and output values are not changed. Note, however, that commands such as FEPI SEND may have transferred data before the condition is recognized.

You can use EDF and CECI to debug FEPI programs. Because FEPI commands can be quite long, you will probably find the NAME field of CECI useful.

FEPI key stroke and screen-image applications

These topics cover the key stroke and screen-image data interfaces for FEPI applications.

The examples given here are confined to simple conversational applications. However, you can use this data interface whatever the application structure. See “FEPI application design” on page 73 for further possibilities together with full details of conversations, error handling, and system considerations.

The key stroke and screen-image data interface is suitable for a wide range of applications, and is simpler to use than the alternative data stream interface. However, there are certain types of application for which you cannot use screen-image data. For more details, see “FEPI data stream applications” on page 68.

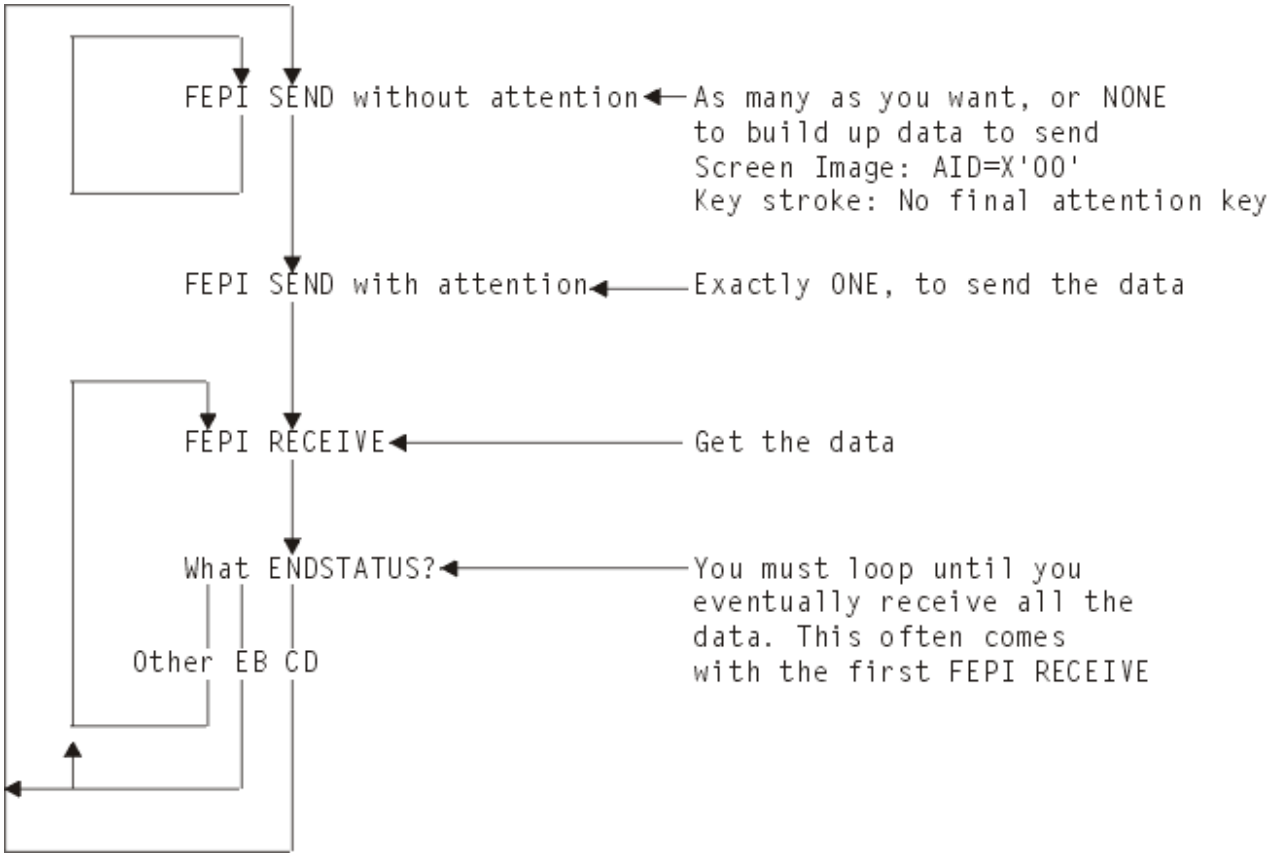
You can send both key stroke and screen-image data in the same conversation. The inbound data format is the same for both: a screen-image, that you can also access field-by-field.

You must have general knowledge of data communication and protocols.

General sequence of commands

The following diagram illustrates the general sequence of FEPI commands that you use with key stroke and screen-image data.

That is, a FEPI SEND, multiple FEPI RECEIVES that complete when all the data has been received, followed by another FEPI SEND.



Note: The diagram does not show any processing of the data, nor where you might enter, or leave, the loop. This information is explained more fully in “FEPI application design” on page 73.

Sending key stroke data

Sending key strokes is the easiest way of sending data.

Your program acts in the same way as the keyboard operator, with FEPI letting the program "press keys" just as the operator does.

A sample program illustrates the techniques used; see FEPI sample program: Key stroke CONVERSE.

The data can contain any combination of data characters together with manipulative, special, and attention key values representing almost every keyboard key. Data characters are represented as themselves. Manipulative, special, and attention key values are represented by *escape* sequences, comprising an escape character followed by a 2-character code. For example, using '&' for the escape character, you might send the following sequence to insert AB in one field, type IJKL into another field, and press PF7 to complete the input operation:

```
&HO&T2&R1&INAB&RS&N4IJKL&EF&07

Home
Tab, twice
Cursor right
Insert
AB
Reset
Newline, 4 times
IJKL
Erase—EOF
PF7
```

If the sequence were in a character string named KEY-SCRIPT, you would send it with:

```
 EXEC CICS FEPI SEND FORMATTED
CONVID(....)
KEYSTROKES
FROM(KEY-SCRIPT)
FLENGTH(30)
```

In full, the escape sequences are:

Manipulative keys

&HO
home

&Ln
cursor left, n times

&Rn
cursor right, n times

&Un
cursor up, n times

&Dn
cursor down, n times

&Tn
tab, n times

&Bn
backtab, n times

&Nn
newline, n times

(n = 1–9)

Special keys

&IN
insert

&DL
delete

&RS
reset

&EF
erase to end of field

&EI
erase input

&FM
field mark

&DU
DUP

&ES
escape character

&SO
shift out

&SI
shift in

&MS
start secure MSR

Attention keys

&AT
attention

&An
PAn (n = 1–3)

&nn
PFnn (nn = 01–24, any leading 0 must be specified)

&CL
clear

&CS
cursor select (light pen)

&EN
enter

&ME
end secure MSR

You can choose an alternative escape character.

Data characters must have values ≥ X'40' , so nulls (X'00') are not supported as such, although they can be generated using the erase or delete keys. Key strokes following an attempt to type into a protected field are ignored until RESET is keyed.

For magnetic stripe reader support, the sequence &MS...data...&ME represents passing a secure magnetic stripe card through the reader. Nonsecure cards have to be simulated by entering the data in the normal way.

The cursor position is set by your key strokes, rather than specifying where the cursor is placed. If your first key stroke is always the HOME key (&HO), you will have the cursor in a known starting position.

You can choose to send all the data with one command, or to use several commands to build up the data. The last (or only) command should have an attention key as its final key stroke, to send the data. There should be no other attention keys.

Alternatively, if you are not interested in the received data, you can ignore it by sending key strokes with multiple attention keys, as described in “Multiple attentions” on page 65.

Error handling
Apart from communication errors caused externally, there are two likely sorts of error that you might get.

These two sorts of errors are as follows:

- Bad command sequencing; that is, you have issued a FEPI SEND when one was not expected. A FEPI SEND must **not** follow a FEPI SEND with a final attention key, or a FEPI RECEIVE that did not indicate ‘change direction’.
- Incorrect data; that is, your key strokes are improper. You may have:
  - Sent data, characters, or escape sequences that are not valid.
  - Got into an ‘input inhibited’ situation and not reset it.
  - Broken the rules for double-byte character set (DBCS) data.
  - Failed a validation test, if there are fields with one of the validation attributes.

Many of these data errors cannot be detected until the data is processed, because they depend on the previous data. This means that any key strokes preceding the error will already have taken effect—they cannot be removed by FEPI.

The FEPI SEND can also fail if, following end bracket, the back-end sends BID to send more data and your pool has CONTENTION(LOSE). You must then receive the new back-end data first.

Receiving field-by-field

Receiving data field-by-field is the easiest way of receiving data.

In the simplest case you would issue a FEPI RECEIVE command without specifying an INTO data area. FEPI gets the data from the back-end system and builds the resulting screen image internally. The cursor position is returned by the CURSOR option. Information about the number of lines, columns, and fields in the screen image is returned by the LINES, COLUMNS, and FIELDS options.

To get the data, you issue the FEPI EXTRACT FIELD command for each individual field that you want. As well as the data, you can find out the attribute settings for the field, and its length and position. The attribute values are defined in the DFHBMSCA copy book, as is used with BMS. You can issue as many FEPI EXTRACT FIELD commands as you need, for whichever fields you want. You can issue more than one for each field, for example, if you want to get the data and attributes separately. It is generally preferable to use the FIELDLOC option rather than FIELDNUM. There may be spurious attributes between each displayed field which make determining field numbers difficult.

A sample program illustrates the techniques used; see FEPI sample program: Screen image RECEIVE and EXTRACT FIELD.

Command completion
The FEPI RECEIVE command completes on 'end of chain'.

This normally coincides with 'change direction' or 'end bracket', meaning that all data has been received. In some cases, however, back-end applications may send data to you in several sections (chains), each causing a screen update, so you must keep on receiving data until 'change direction' or 'end bracket' is indicated.

In all cases, the ENDSTATUS option is set to indicate what the completion conditions were. Where several conditions occur together, ENDSTATUS shows the most significant one. The values of ENDSTATUS and their associated meanings are shown in Table 8 on page 65.

Table 8. ENDSTATUS values and associated meanings for formatted data

ENDSTATUS	End bracket	Change direction	End of chain	Next command expected
EB	Y	-	Y	Any
CD	-	Y	Y	FEPI SEND or CONVERSE
LIC	-	-	Y	FEPI RECEIVE

Note: Y=Condition indicated.

When 'end bracket' is received, the session is in *contention state* , and either end may try to transmit data next. Some back-end systems use 'end bracket' in the middle of a series of transmissions to allow the terminal to break in if it wants, and they may use 'end bracket' instead of 'change direction' at the end of the flow. This is particularly true of IMS. CICS usually sends 'change direction' eventually, although it may send 'end bracket' indicators intermediately.

Using your knowledge of the back-end application and system, you must check the data that you have already received, to determine whether more data is to be expected or the transmission is complete. If more data is expected, you should issue another FEPI RECEIVE command; if the transmission is complete, it is the front-end application's turn to send data.

You should always use the TIMEOUT option on a FEPI RECEIVE command; see "timeouts" on page 78.

Error handling
Apart from communication errors caused externally, the most likely error you might get is due to bad command sequencing. That is, you have issued a **FEPI RECEIVE** command when a **FEPI SEND** command is expected.

A **FEPI RECEIVE** command must not follow a **FEPI SEND** command without attention, or a **FEPI RECEIVE** command that indicated 'change direction'.

Another likely error is 'previous SEND failed'. This might be an external communication error, or the back-end system might have responded negatively—as IMS does, for example, if you try to run an unknown transaction. The sense data which you can get using **FEPI EXTRACT CONV** tells you which error it is, and, where the back-end system has responded negatively, you issue another **FEPI RECEIVE** command to get the data.

Multiple attentions

In certain circumstances you might not have any interest in the immediate result of the data you send, but only in a later result, after you have sent more data.

If this is the case, you can construct a single key stroke sequence, comprising all the sets of data to send, each with its own attention key, and then send the whole lot in one operation.

At each attention key, FEPI sends your data to the back-end system and receives the results internally, until 'change direction' or 'end bracket' is indicated. Then FEPI sends the next set of key strokes. Using multiple attentions improves performance but, if the intermediate results are not what you expect, FEPI

has no way of knowing this and carries on sending your key strokes. This can lead to unexpected effects, or to the failure of the command with a data error. In the latter case, all the key strokes and back-end system interactions preceding the error have already taken effect and you may find it difficult to determine the state of the back-end system. Further, no timeout can be specified for the intermediate receives, and so, if there is a communication problem, your application may be suspended indefinitely.

If the last set of key strokes ends with an attention key, you **must** issue a FEPI RECEIVE command to get the final result. If the last set of key strokes does not end with an attention key, you can issue another FEPI SEND command, with yet more key strokes.

Sending screen-image data

Sending screen-image data is an alternative to sending key stroke data. In general, this would be the screen image that you received modified to reflect the changes that would be the result of an operator action.

A sample COBOL program, DFH0VZTS, illustrates the techniques used; see FEPI sample program: Screen image SEND and START.

The data is exactly what you would expect: an image of the screen that you want to send. That is, 24 rows of 80 bytes (or whatever your screen size is) of data, corresponding byte-for-byte with the screen. For example, in a COBOL program containing this data description:

```
 01 SCREEN-IMAGE PIC X(1920).
01 SCREEN-FIELDS REDEFINES SCREEN-IMAGE.
05 LINE-1 PIC X(80).
05 FILLER REDEFINES LINE-1.
10 FILLER PIC X(20).
10 CUST-NO PIC X(12).
10 FILLER PIC X(48).
05 LINE-2 PIC X(80).
05 LINE-3 PIC X(80).
05 LINE-4 PIC X(80).
05 FILLER REDEFINES LINE-4.
10 FILLER PIC X(12).
10 CUST-NAME PIC X(32).
10 FILLER PIC X(36).
```

you would put the required data into the fields and send the screen image using:

```
 EXEC CICS FEPI SEND FORMATTED
CONVID(....)
FROM(SCREEN-IMAGE) FLENGTH(1920)
AID(PF2)
```

where AID specifies which attention key was pressed on the simulated terminal.

Data bytes are represented as themselves; you must set any nulls (X'00') that are needed to fill a field. In a protected field, the data bytes must be the same as in the current, simulated terminal buffer that FEPI holds. In the case of attribute bytes, it does not matter what values you put, because you have no control over their positions or settings, any more than a terminal operator does. However, if the value is X'01' , FEPI sets the modified data tag (MDT) for the field, even if its data has not changed. (If the data has changed, FEPI sets the MDT automatically.)

You do not have to send a complete screen image. If your changes are confined to the first few lines, you need only send those few lines. The data you send is taken as starting from the top left position of the screen.

Note: If you are using the C programming language, remember that a screen image probably contains null characters. Take care if you are handling the screen image as a string.

The cursor position can be set using the CURSOR option.

You can choose to send all the data with one command, or to use several commands to build up the data. The last (or only) command must have an attention identifier (AID) specified, using the AID option, to send the data. The other commands must have an AID value of X'00' . Definitions for the AID values are in the DFHAID copy book, as is used with BMS.

Note: The COBOL and assembler versions of the DFHAID copybook are different. Therefore, you cannot copy unmodified SEND commands from the DFH0VZTS sample program, which is supplied in COBOL only, to a user-written assembler program.

Error handling

The errors you can get are similar to those for key stroke data. Your screen-image data has other ways of being incorrect. In place of escape sequences not being valid, or 'input inhibited', you might have cursor or AID settings not valid, or changed data in a protected field. Many of these data errors cannot be detected until the data is processed. This means that some of the changes will have taken effect already - they cannot be removed by FEPI.

Receiving screen-image data

If you specify an INTO data area on a FEPI RECEIVE command, the data you receive is the screen image; 24 rows of 80 bytes (or whatever your screen size is) corresponding byte-for-byte with the screen.

Data bytes are represented as themselves. In positions corresponding to attribute bytes, X'FF' appears.

You need only get the first few lines of the screen if that is all that you are interested in.

After you have processed the data, you will probably use the same screen image, modified as required, on a subsequent screen-image send.

Even though you got a screen image, you can use the FEPI EXTRACT FIELD command as well if you want, for any particular fields that you require, just as described in "Receiving field-by-field" on page 64. In particular, the FEPI EXTRACT FIELD command is the only way you can determine the value of the field attributes.

A sample program illustrates the techniques you can use; see FEPI sample program: Key stroke CONVERSE.

Note: If you are using the C programming language, remember that a screen image probably contains null characters. Take care if you are handling the screen image as a string.

Command completion and errors

As far as completion and errors are concerned, a FEPI RECEIVE command with an INTO data area is just like one without.

So, if you do not get 'change direction' or 'end bracket', you have to issue another FEPI RECEIVE command before you can send your screen image back, and even 'end bracket' might require further FEPI RECEIVE commands.

Extracting field data

It is not only after a FEPI RECEIVE command that you can issue a FEPI EXTRACT FIELD command. You can issue this command anywhere in the conversation to find out about the current screen image that FEPI holds for the simulated terminal.

This can be particularly useful where a FEPI SEND command has failed or given unexpected results, to discover what happened.

CONVERSE

FEPI CONVERSE can be used instead of a FEPI SEND with attention and the first (or only) FEPI RECEIVE.

It is more efficient than issuing two separate commands and is allowed anywhere that FEPI SEND is allowed. The effects are exactly as if the two commands had been issued.

The ending conditions are identical to those for FEPI RECEIVE, unless you use the POOL option to get a temporary conversation. In this case, it ends on the first occurrence of either 'Change direction' or 'End bracket' and does not end at 'end of chain' alone.

Error handling

You must plan which command is expected next:

- If the receive part of the FEPI CONVERSE command fails, the send will have already been done, and so a FEPI RECEIVE command is expected next.
- If the send part fails, the receive is not done, and, if the initial send was expected, a FEPI SEND or CONVERSE command is expected next.

FEPI data stream applications

These topics discuss the low-level data stream interface for FEPI applications.

The examples it contains are confined to simple conversational applications. However, you can use this data interface whatever the application structure; see “FEPI application design” on page 73 for all the possibilities, together with details of conversations, error handling, and system considerations.

When to use the data stream interface

You can use the data stream interface for the following types of applications:

- With passthrough; that is where the application passes data through, usually to the user's terminal, without doing anything to it.
- With SLU P.
- Where the formatted interface does not provide the detailed function that you need.
- For handling non-3270 LU2 devices.
- With non-response mode IMS transactions.

The 3270 data stream interface is especially useful when creating FEPI applications that require little or no manipulation of the inbound (screen) data, because it is already in a form suitable for sending to a real terminal. If interpretation or reformatting of the inbound data is required, however, it can be significantly more difficult to operate on a 3270 data stream.

An example of an application suited to the 3270 data stream interface is a passthrough program, as illustrated by the FEPI sample program: 3270 data stream passthrough . Such programs can also be used to determine the flows and screen layouts of back-end systems when you are developing FEPI applications that, for example, drive signon or menu selection sequences and manipulate screens or dialogs.

You must be fully conversant with the data stream and data stream protocols as detailed in the books in the following list, and with how the back-end system uses them:

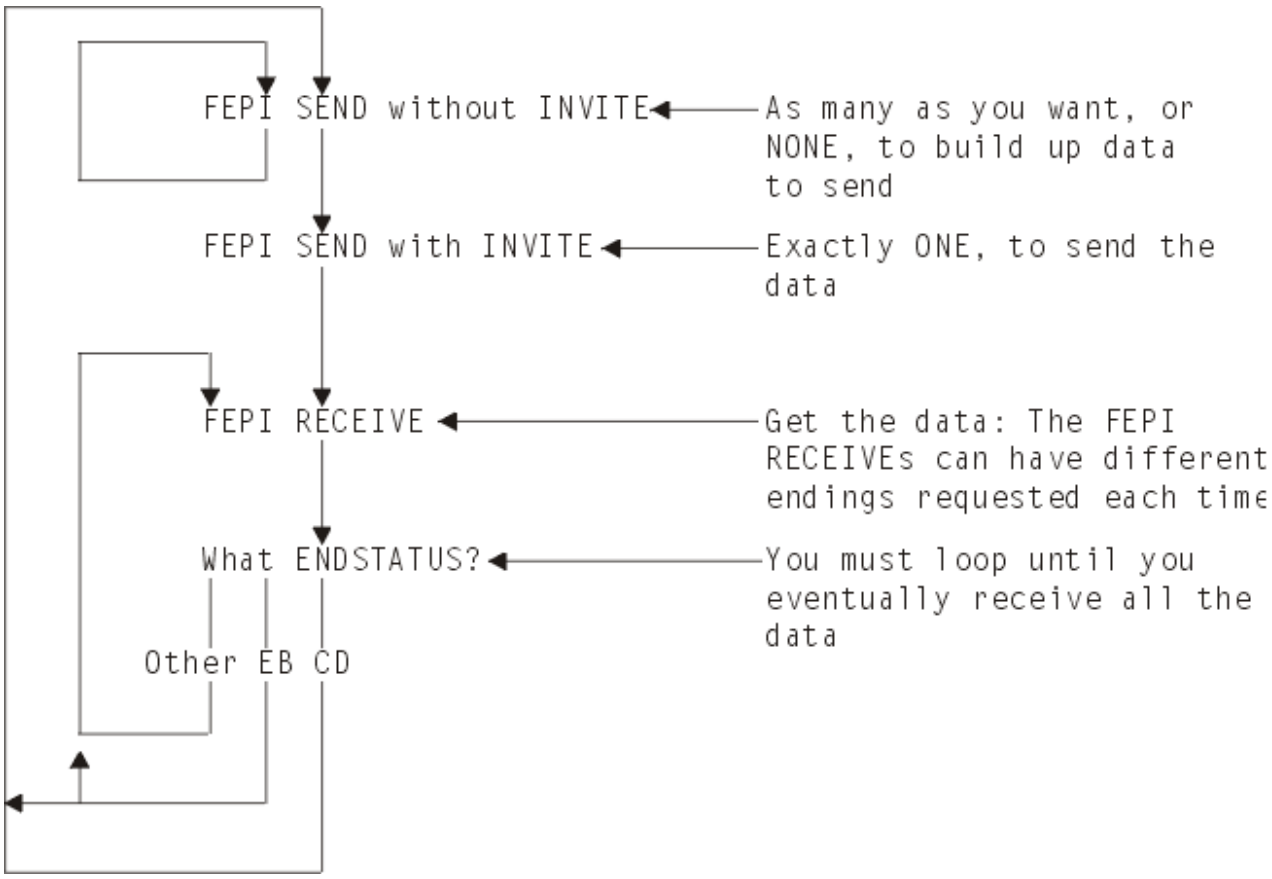
- IBM 3270 Data Stream Programmers Reference
- IBM 3174 Establishment Controller: Functional Description

The application program is entirely responsible for the integrity of the data stream that uses this interface. FEPI performs no checking or interpretation on the data stream that is sent to or received from the back-end system, and makes no attempt to manipulate data into RU sizes that the sender or receiver can handle; the application program must be prepared to handle whatever data is presented to it. For example, with SLU2 mode, it must be prepared to handle READ commands, and WRITE STRUCTURED FIELD commands, in addition to the normal WRITE commands.

General sequence of commands

The following diagram illustrates the general sequence of FEPI commands that you use with data stream.

That is, a FEPI SEND, multiple FEPI RECEIVE commands that complete when all the data has been received, followed by another FEPI SEND.



Note: The diagram does not show any processing of the data, nor where you might enter, or leave, the loop. This information is explained more fully in "FEPI application design" on page 73.

Receiving

You can choose whether to process data in small segments or all at once.

Your choice depends upon various factors including:

- Processing convenience
- The amount of data that you expect
- The size of the data area that you can use
- What you are doing with the data
- How the back-end application operates
- Whether you want to handle responses (see "Specialized FEPI functions" on page 83 for this feature).

The data is a standard inbound data stream, exactly as is sent to the simulated terminal from z/OS Communications Server. It is quite possible that there will be occasions on which you will receive no data; for example, when the back-end system needs to set a protocol indicator.

Command completion

The **FEPI RECEIVE** command can be specified, or defaulted, to end in one of several ways.

You can end the command using one of the following valid values:

RU

On the first to occur of:

- INTO data area full
- End of request unit.

CHAIN
On the first to occur of:

- INTO data area full
- End of chain.

UNTILCDEB
On the first to occur of:

- INTO data area full
- End of chain with definite response request
- 'Change direction' indicated
- 'End bracket' indicated.

Note: Using UNTILCDEB is not recommended, because you may have the difficult task of splitting data back into its constituent chains in order to process it.

In all cases, the ENDSTATUS option is set to indicate what the completion conditions were. Where several conditions occur together, ENDSTATUS shows the most significant one. The values of ENDSTATUS and their associated meanings are shown in Table 9 on page 70.

Table 9. ENDSTATUS values and associated meanings for data stream

ENDSTATUS	Command options	End bracket	Change direction	End chain	End RU	INTO area full	Next command expected
EB	RU, CHAIN, UNTILCDEB	Y	-	Y	Y	-	Any
CD	RU, CHAIN, UNTILCDEB	-	Y	Y	Y	-	FEPI SEND or CONVERSE
LIC	RU, CHAIN, UNTILCDEB	-	-	Y	Y	-	FEPI RECEIVE
RU	RU	-	-	-	Y	-	FEPI RECEIVE
MORE	RU, CHAIN, UNTILCDEB	-	-	-	-	Y	FEPI RECEIVE

Note: Y=Condition indicated.

FEPI RECEIVE commands must continue to be issued until 'change direction' or 'end bracket' is indicated. You cannot start sending data until all inbound data has been received. If an ENDSTATUS of MORE is indicated, the data stream is not necessarily self-contained and should not be processed until the remainder of the information is received. The value returned for REMFLENGTH might indicate how much more information is to come.

When 'end bracket' is received, the session is in *contention state* , and either end may try to transmit data next. Some back-end systems use 'end bracket' in the middle of a series of transmissions to allow the terminal to break in if it wants, and they may use 'end bracket' instead of 'change direction' at the end of the flow. This is particularly true of IMS. CICS usually sends 'change direction' eventually, although it may send 'end bracket' indicators intermediately.

Using your knowledge of the back-end application and system, you must check the data that you have already received, to find out whether more data is to be expected or the transmission is complete. If more data is expected, you should issue another FEPI RECEIVE command; if the transmission is complete, it is the front-end application's turn to send data.

A problem arises where the application is the passthrough type, because it does not look at the received data. There are various ways of handling this:

1. Request data conditionally from both ends–which cannot generally be done, and particularly not in the most typical case where the passthrough is directly to a front-end terminal.
2. Wait for data from both ends at once. This can be done where the passthrough is directly to a front-end terminal and the transaction is pseudoconversational for both CICS and FEPI . See “Started tasks” on page 75.
3. Ask each end at intervals if there is data waiting (for the back-end system by using FEPI RECEIVE with TIMEOUT); this is often not possible, as in the case where the passthrough is directly to a front-end terminal.
4. Forego a strict passthrough technique and check the data.
5. Assume that a transmission with ‘end bracket’ and no data means that more data is to come.
6. Issue another FEPI RECEIVE with TIMEOUT in case more data is to come, which has the disadvantage of introducing a delay.

Note: The last two cases involve an element of risk because the wrong assumptions can be made.

You should always use the TIMEOUT option on a FEPI RECEIVE command; see “timeouts” on page 78.

Error handling

Apart from z/OS Communications Server and back-end communication errors caused externally or, more probably, by errors in the outbound data stream that you sent previously, the most likely cause of an error condition is an incorrect sequence of commands.

That is, you have issued a FEPI RECEIVE when a FEPI SEND is expected. A FEPI RECEIVE must not follow a FEPI SEND without INVITE, or a FEPI RECEIVE that indicated ‘change direction'.

Another likely error is ‘previous SEND failed'. This may be an external communication error, or it may be that the back-end system has responded negatively–as IMS does, for example, if you try to run an unknown transaction. The sense data which you can get using FEPI EXTRACT CONV tells you which error it is, and in the latter case you issue another FEPI RECEIVE to get the data.

See FEPI sample program: 3270 data stream passthrough and FEPI sample program: SLU P pseudoconversational for sample programs illustrating some of the programming techniques.

Sending

You can choose to send an entire stream of data, or you can break it up into smaller units, finishing with a FEPI SEND with INVITE.

INVITE indicates that this is the last data to send, and that inbound data should be expected next. The data is sent with ‘last in chain' and ‘change direction'. Otherwise, further FEPI SENDS are to be expected. It is the application program's responsibility to ensure that the amount of data sent on a request does not exceed the capacity of the receiving LU.

Error handling

Apart from z/OS Communications Server errors caused, most probably, by errors in the outbound data stream that you sent previously, the most likely cause of an error condition is an incorrect sequence of commands. That is, you have issued a FEPI SEND when one was not expected. A FEPI SEND must not follow a FEPI SEND with INVITE, or a FEPI RECEIVE that did not indicate ‘change direction'.

The FEPI SEND can also fail if, following ‘end bracket', the back-end system sends BID to send more data and your pool has CONTENTION(LOSE). You must then receive the new back-end data first.

See FEPI sample program: 3270 data stream passthrough and FEPI sample program: SLU P pseudoconversational for sample programs illustrating some of the programming techniques.

CONVERSE

FEPI CONVERSE can be used instead of a FEPI SEND with INVITE and the first (or only) FEPI RECEIVE. It is more efficient than issuing two separate commands and is allowed anywhere that FEPI SEND is allowed. The effects are exactly as if the two commands had been issued.

The ending conditions are identical to those for FEPI RECEIVE, unless you use the POOL option to get a temporary conversation. In this case, it ends on the first to occur of:

- INTO data area full
- 'Change direction' indicated
- 'End bracket' indicated,

and **not** at 'end of chain' alone. Further, if there is any residual data to receive, it is lost.

With regard to errors, you need to take into consideration which command is expected next:

- If the receive part of the FEPI CONVERSE command fails, the send will have already been done, and so a FEPI RECEIVE command is expected next.
- If the send part fails, the receive is not done, and, if the initial send was expected, a FEPI SEND or CONVERSE command is expected next.

SLU2 mode considerations

It is necessary, when sending outbound 3270 data streams, to ensure that a three-byte prefix containing the attention identifier (AID) and cursor address is inserted at the front of the data.

Similarly, the first two bytes of inbound data typically contain the 3270 command code and write control character (WCC). The lengths supplied or returned on the FEPI SEND, RECEIVE, or CONVERSE DATASTREAM commands include the length of the prefix.

AID values are the same as the CICS values and, in passthrough applications, can be taken from EIBAID. The cursor address however is a buffer address and cannot be taken from EIBCPOSN. 3270 buffer addresses can be 12-, 14-, or 16-bit addresses depending on the device. 12 -bit addressing is the most difficult to convert to or from, but it is very common; an address conversion table and an algorithm are contained in the *3270 Information Display System 3274 Control Unit Reference Summary*.

The inbound 3270 command is most likely to be a WRITE or ERASE WRITE and is, therefore, followed by a WCC then orders and data. However, this is not guaranteed and the inbound command should be inspected to determine what it is, what, if anything, should follow it, and how it should be handled. For example, the application may choose to perform an EXEC CICS SEND TEXT from the inbound data and may, therefore, require to know whether to append the ERASE keyword. The various READ commands (such as READ BUFFER and READ MODIFIED) and all the WRITE STRUCTURED FIELD commands (a common one being READ PARTITION with QUERY) need special handling.

If you receive more than one chain (using the UNTILCDEB option), you have to find each inbound command yourself, so this is not recommended unless you know that the back-end system only sends a single chain.

For further information, see IBM 3270 Data Stream Programmers Reference.

SLU P mode considerations

Two sample programs illustrate some of the programming techniques for SLU P mode.

- FEPI sample program: SLU P one-out one-in
- FEPI sample program: SLU P pseudoconversational

FEPI application design

These topics describe the programs comprising a FEPI application and the basic design aspects. It also discusses signon security, error handling, and system considerations, including performance.

Programs

A FEPI application consists of a number of programs.

These programs are as follows :

- Access
- Begin-session handler
- Unsolicited-data handler
- End-session handler.

Access program

The main purpose of an access program is to start a conversation, communicate with the back-end application, and end the conversation.

It must also be able to handle exception cases such as edit errors, transactions that are not valid, or security violations, and it might need to manage signon/signoff sequences. It might also need to handle begin-session and end-session requirements, if special handlers are not provided. The SESSNSTATUS option of FEPI ALLOCATE tells you if a new session has been started, or if you are reusing an existing session.

For many FEPI applications, particularly where formatted data is used, the access program is not complex. However, you do need to be fully conversant with everything that the back-end application might do. Your application must behave just like the real terminal and operator, and you m ust send and receive data in the correct sequence. Within a conversation, any data received is passed to the application that owns the conversation; FEPI cannot determine whether it is the data or screen image that was expected or, for example, a message reporting an abnormal end. Although the FEPI application needs to handle these cases, the access program need not test for all possibilities. The suggested method is to test only for the expected data or screen image and use a special error-handling program if the test fails.

Other applications may require more sophisticated programming. In some cases, you not only have to understand all the displays and protocols of the back-end application, but must also be conversant with the detailed data stream protocols. Applications may have to be custom-written for each device and type of target that is to be supported.

Syncpoints are not needed and not applicable in FEPI because communication environments do not provide any recoverable units of work. It is up to you to provide the syncpoints and any recovery of data that you need. For particularly critical operations with the back-end applications, you may find that using "definite responses" is helpful; see "DRx responses" on page 83.

Begin-session handler

The begin-session handler transaction is started by FEPI when a connection is acquired. This transaction handles any functions that are required to initialize the session.

Typical tasks are as follows:

- Handling device queries.
- Handling any initial inbound data, or "good morning" message, following the bind.
- Signing on to the back-end system.

Device queries are sent by the back-end system (particularly CICS) if the terminal definitions so demand. You would normally reply 'null' (as illustrated by the begin-session sample program), or with some particular terminal properties that you want. Note, if you want to match the terminal properties to those of the real front-end terminal that an application is using, you cannot use a begin-session handler; each application will have to do its own begin-session handling.

When a back-end system sends a message after a successful bind, the connection should be in a pool where the INITIALDATA property is set to INBOUND. For SLU2, IMS always sends such a message; CICS may or may not do so depending on the way your system is defined. This extends the process of acquiring a connection to include receiving the data. Note that, if INBOUND is specified, the begin-session handler (or each application program, if there is no begin session handler) must issue a FEPI RECEIVE command to get the data and then send a suitable reply to the back-end system.

Remember that handling this initial data is just like handling any other back-end data: you must cope with whatever the back-end system may send, and handle and reply to it accordingly.

Security requirements in the back-end system might make it more appropriate for sign-on to be part of the access program. For information about implementing signon security, see Generating and using PassTickets for secure sign-on.

There is a sample begin-session handler program. For more information, see FEPI sample program: Begin session.

Unsolicited-data handler
The unsolicited-data handler transaction is started by FEPI if inbound data arrives on a connection for which there is no current conversation.

Unsolicited data can occur when:

- A target sends more data than the application expected.
- The access program times out, or the conversation is ended, before the data arrives.
- Asynchronous IMS output such as:
  - Message from previous input that could not be processed at the time of receipt by IMS
  - Reassignment of a logical terminal that has a message queued.

  With IMS, this type of unsolicited data does not usually occur in SLU2 mode because IMS only sends messages in reply to explicit requests from the terminal.
- Asynchronous CICS output such as that sent by ATI.

The unsolicited data should all be received by the handler, even if it is only to be discarded. Otherwise, although FEPI eventually discards the data, it also ends and restarts the session, which is inefficient.

There is a sample unsolicited-data handler program. For more information, see FEPI sample program: Monitor and unsolicited data-handler.

End-session handler
The end-session handler transaction is started by FEPI when a conversation ends or a session is to be unbound.

This could be used as follows:

- To set the session to a known state, perhaps by signing off from the back-end system, ready for the next conversation.
- When the conversation ends, to force (or prevent) unbind and the subsequent starting of a new session (overriding what the access program specified).
- To perform special action on CICS shutdown in the front-end system.

There is a sample end-session handler program; see FEPI sample program: End-session handler.

Note: The end-session handler transaction runs under the CICS region userid.

Application organization

This topic section discusses application styles, started tasks, and conversations.

The three application styles can be mixed. If there are enough connections available, you can have as many conversations as you like at a time with various targets: they can be consecutive or, much more

usefully, interleaved. For example, if you need data from four different applications, you could overlap the processing by sending all four requests for data before you start waiting for a response.

Application style
Your FEPI application can have one of three conversational styles.

One-out one-in conversational

One transaction performs the complete conversation with the back-end application in a single send and receive operation. This is the simplest style, if the required data can be obtained from the back-end application in this way. The transaction can be reduced to a single FEPI CONVERSE command using a temporary conversation.

By freeing the connection between transmissions, the capacity of the connection is increased. However, this style only works where no setup is needed to run the back-end transaction and it does not depend on any prior communication. This is because, unless you have a very strict pool regime, you cannot generally guarantee which simulated terminal FEPI will use–it may not be the same one as in a previous conversation–or that you were the last user of the terminal. Further, if you receive unexpected results from the back-end transaction, you may not be able to recover. Therefore, you should only use this style where it does not matter if the back-end transaction runs or not, for example, for a simple inquiry. A one-out one-in conversational program is unlikely to be suitable for accessing CICS transactions or IMS conversational transactions.

See the FEPI sample program: SLU P one-out one-in.

Conversational

One transaction performs the complete conversation with the back-end application using multiple send and receive operations and waiting for the inbound data to arrive. This style is used for a back-end application that requires several transmissions or complex setup. This style is simple, and if the network performance is good, the time spent waiting for inbound data may not be a problem.

See the FEPI sample program: Key stroke CONVERSE.

Pseudoconversational

One transaction sends data to the back-end application, identifies another transaction that is to be started when the inbound data arrives, and ends. When inbound data arrives, FEPI starts the specified transaction which then receives the data. A typical technique is to have a transaction that, when started to receive inbound data, receives the data, sends the next piece of outbound data, issues FEPI START to start itself, and then ends.

The pseudoconversational style (use of FEPI START commands) results in significant CPU overheads in the front-end region. Further, since the use of FEPI START generates additional flows to and from the real terminal, response times are also significantly increased. As a consequence, FEPI START should be used sparingly when, for example. the receipt of the data from the back-end application takes a long time.

See the FEPI sample program: Screen image SEND and START and the FEPI sample program: Screen image RECEIVE and EXTRACT FIELD.

Started tasks
In the pseudoconversational case, the ‘receive’ program is started by FEPI as a CICS started task, with a start code of 'SZ' (for FEPI) which can be checked using EXEC CICS ASSIGN STARTCODE.

FEPI supplies **start data** that identifies the reason for starting the task and gives information about the FEPI resources, such as the node-target connection, the data mode and format, and the conversation ID involved. The program that processes the transaction issues EXEC CICS RETRIEVE to get this data (the

CICS rules relating to transactions and start data apply; in particular, you must retrieve all of the start data to prevent multiple initiations). Copy books DFHSZAPA, DFHSZAPO, DFHSZAPC, and DFHSZAPP contain declarations of the start data structure. You can provide your own data to be included in the start data, so that your programs can communicate with each other about their processing state and so on.

The first thing such a program must do is get ownership of the conversation using the conversation ID from the start data; it should then use FEPI RECEIVE to get the actual data from the back-end. Then it can do whatever it likes: end the conversation, send more data to the back-end system (and start itself or a new task to receive the reply), and so on.

In addition to inbound data arriving, anything else that would cause a FEPI RECEIVE command to complete causes the 'receive program' to be started. This includes a 'previous SEND failed' error, and a response from the back-end system without any data. The FEPI RECEIVE that you issue shows these cases, as if FEPI START had not been used.

The program is also started if the time limit set by the FEPI START command expires, or if the session is lost. These cases are indicated by the value of EVENTTYPE, in the start data, being TIMEOUT or SESSIONLOST rather than DATA. They should be handled as if a FEPI RECEIVE command had caused the error.

If your 'send' program is associated with a front-end terminal, your FEPI START command would normally specify that the 'receive' program uses the same terminal. You should be aware that it is not possible for FEPI to guarantee that another transaction will not use the terminal while the inbound data is awaited. In the majority of cases, this does not happen or does not matter. If it does happen and it is critical (perhaps for security reasons), you can prevent user input at the terminal by issuing an EXEC CICS SET TERMINAL command specifying NEXTTRANSID(itran) before issuing FEPI START; remember to reset NEXTTRANSID to blank in the started task. *itran* is the name of a transaction that you provide which rejects any user input, and sets NEXTTRANSID(itran) again. If this is unacceptable, you must avoid using pseudoconversational applications.

The handlers mentioned in topics on pages "Begin-session handler" on page 73 , "Unsolicited-data handler" on page 74 and "End-session handler" on page 74 —begin-session, unsolicited data, end-session—are also CICS started tasks. Again, the start data (obtained with EXEC CICS RETRIEVE) tells you why the task was started and the identity of the conversation. The started task must get ownership of the conversation so that it can continue the conversation and so that FEPI knows that the event is being handled.

Conversations
Your entire communication with a particular back-end transaction should be contained in a single FEPI conversation.

This means that you remain in control of the communication; no other program can break in and you keep using the same simulated terminal. Only the task that started the conversation with FEPI ALLOCATE can use the conversation. It "owns" it and no other task can issue any command for it, not even FEPI EXTRACT CONV.

Conversational applications
In the simplest case, an access program starts a conversation with a FEPI ALLOCATE command specifying the pool of connections that is to be used.

The command returns an identifier, the conversation ID, that is used to refer to the conversation subsequently. The program then issues a series of FEPI SEND, RECEIVE (and possibly other) commands for the conversation, each specifying the identifier, so that FEPI knows which conversation—and therefore which connection and target —the command is for. Finally, it ends the conversation with a FEPI FREE command. If it does not, the conversation is ended by FEPI when the task ends.

The FEPI FREE command should normally specify the HOLD option, so that the connection remains ready for use by another conversation. If the RELEASE option is used, or you leave the conversation to be freed by FEPI at the end of task, the session is ended, and a new one must be started for the next conversation; this is inefficient and, therefore, not recommended.

Started tasks
If the access program is pseudoconversational, after sending data it issues a FEPI START command to name the transaction that FEPI is to start when inbound data arrives.

At this point the conversation becomes “unowned” and the first task can no longer use it. However, the conversation does not end; when data arrives, the conversation ID is passed to the started task and that task issues FEPI ALLOCATE with the PASSCONVID option to get ownership of the conversation. Only then can the started task use the conversation to receive the inbound data.

While the conversation is unowned, it can be acquired by any task that knows the conversation ID. Acquiring the connection cancels the pending start request, and the task that acquired ownership has to continue the conversation as if no FEPI START had been issued. This technique is useful in a pass-through application to a front-end terminal to handle contention between inbound data and terminal input. The application issues a FEPI START command, specifying the front-end terminal, and then returns to CICS specifying a ‘next’ transaction. Inbound data arriving first causes FEPI to start the transaction on the front-end terminal, which causes CICS to cancel its wait for terminal input; if terminal input arrives first, the application, after using EXEC CICS ASSIGN STARTCODE to determine why it was started, issues FEPI ALLOCATE with PASSCONVID which cancels the FEPI START request.

Getting ownership also applies to the tasks started by the various handlers. The conversation may have been started by some access program (end-session), or by FEPI itself (begin-session, unsolicited-data). Either way, you must still issue a FEPI ALLOCATE command with PASSCONVID, quoting the conversation ID, to get ownership and continue the conversation.

When a handler has finished processing, it must tell FEPI by issuing a FEPI FREE command for the conversation. For the begin-session handler, this should specify the HOLD option to indicate that the session is ready to be used; if RELEASE is used, the session is ended. The end-session and unsolicited-data handlers can use any of the options according to requirements.

Passing conversations
Besides using FEPI START to have a task for receiving data, any program or handler can explicitly give up ownership of its conversations so that another task can use them.

You do this with the FEPI FREE command and the PASS option. Any task can then get ownership by using FEPI ALLOCATE with PASSCONVID and, if it maintains the command sequence, continue the conversation (for example, if the first task has issued a FEPI SEND with INVITE, the second task would have to issue a FEPI RECEIVE or, perhaps, a FEPI START). It is up to the two tasks to communicate between themselves, using the standard CICS methods (TS queue, COMMAREA, and so on), about the state of the conversation and its ID. FEPI does not offer any application programming facilities for this except that the new task can use FEPI EXTRACT CONV to determine details such as the data format.

If you do not employ a method of passing and saving the conversation across invocations of a pseudoconversational front-end transaction, and instead issue the default FREE command, you lose your connection to the back-end transaction, making it possible for another program to start a conversation and effectively “break into” the active transaction. This can cause the back-end application to abnormally end.

The only other method that can be used to ensure a unique relationship between front-end and back-end transactions, is to have FEPI pools containing a single FEPI node for each user. This ensures that you always get connected to the back-end transaction on the same terminal (FEPI node) to continue your conversation. However, this method can cause administrative problems where there are a large number of users.

Temporary conversations
In a one-out one-in conversational application you can use a single FEPI CONVERSE command that combines an ALLOCATE–SEND–RECEIVE–FREE command sequence.

This combination is selected by using the POOL option of FEPI CONVERSE rather than the CONVID option. In this case, the conversation is a *temporary* conversation that lasts only for the duration of the FEPI CONVERSE command. No conversation ID is returned by FEPI and no other commands can be issued for the conversation; you cannot even use FEPI EXTRACT FIELD to process the returned data.

As with all one-out one-in conversational applications, temporary conversations should be used with care. If more data is received than can be returned on the FEPI CONVERSE command (because, for example, the data is not what you expect), the excess is discarded and cannot be retrieved by the application. Data may be lost if the command fails and, because you cannot receive any more data or guarantee that your next conversation will use the same simulated terminal, it may be difficult to determine the state of the back-end system.

Note:

1. Every conversation started with FEPI ALLOCATE has a unique conversation ID, as does every conversation started for a handler, except in the case of end-session when started after a FEPI FREE. In this case, the ID is the same as in the task issuing the FEPI FREE.

   A task started when inbound data arrives gets the same conversation ID as the task that issued the FEPI START command.
2. The state of a conversation (whether, for example, it is owned by an access program, in a begin-session handler, waiting for inbound data, or being passed) is shown by the STATE option of the CEMT INQUIRE FECONNECTION command (see CEMT INQUIRE FECONNECTION) or the FEPI INQUIRE CONNECTION command (topic page FEPI INQUIRE CONNECTION). This may be useful when you are debugging applications.
3. If your programs are written in C, do not handle conversation identifiers as strings; they may contain null characters.

Error handling

FEPI does not recover any user data when an error condition is raised. Data recovery, if needed, must be performed by the application program. In addition, the output option values on a command are not set if the command fails; your program should not be using these values in such cases.

The recommended method is errors raised by FEPI commands should be handled by your application rather than letting CICS terminate the transaction abnormally. Errors and exceptions can be detected by using the RESP and RESP2 command options, or trapped using HANDLE CONDITION.

timeouts
You should use timeouts with FEPI commands. If there is a problem with the connection to the back-end application, a program without timeouts may wait for ever, you may stop other applications running, and operator intervention may be needed.

timeouts can be used with FEPI ALLOCATE, RECEIVE, START, and CONVERSE commands. In all cases, the timing applies only to the period that FEPI waits for a reply from the back-end system. As soon as anything is received from the back-end, FEPI stops the timer, and then waits for as long as is necessary to receive all the data that is required to complete the command. You **cannot** specify a timeout for FEPI SEND, because the command always completes immediately, without waiting for any data to be transmitted. Any delay or other problem is handled by the following FEPI RECEIVE command. The action to take on a timeout depends on the command that was used:

- For FEPI ALLOCATE, you could retry the initial command and then retry using a different pool or target before going into your error-handling routine.
- For FEPI RECEIVE, you can retry the command and, if that fails, handle the error as if the session with the back-end application had been lost.
- For FEPI START, the timeout is reported to the started task, and not as an error on the command. In other respects, however, it is the same as a FEPI RECEIVE timeout.
- For FEPI CONVERSE with a previously allocated conversation, it is exactly as if a FEPI SEND command and then a FEPI RECEIVE command were issued. That is, the timeout that you specify applies only to the ‘receive’ part of the command, and is treated and handled just like that for a FEPI RECEIVE.

  For a temporary conversation, it is as if the command were preceded by a FEPI ALLOCATE and followed by a FEPI FREE, so in this case the timeout is applied to both the ‘allocate’ and ‘receive’ parts of the command. In this situation, if a timeout occurs, there is no indication as to which part caused it.

Lost session
If a FEPI application loses the session with the back-end application, it should free the conversation. Having done that, the application can take whatever action is required. A typical action would be to recover any data and restore the initial state before retrying the conversation or sending a message to the user.

The loss of a session can also occur because of CLSDST(PASS) processing (as discussed in Handling CLSDST(PASS)). If this is the case, you can find out when the session has been reestablished using the FEPI EXTRACT CONV command. You can then continue processing as required.

Previous SEND failed
This occurs on a FEPI RECEIVE and is indicated by RESP2=216.

It may be an external communication error, or it may be that the back-end system has responded negatively (as IMS does, for example, if you try to run an unknown transaction). Use the FEPI EXTRACT CONV command to get the sense data describing the failure. If this indicates a negative response, you should reissue the FEPI RECEIVE to get the data. If it was not a negative response, it is equivalent to a lost session and the session cannot be recovered.

Communication errors
It is simplest to treat communication and network errors as a lost session, which avoids the need for detailed SNA error protocol handling. However, sophisticated applications may want to handle certain recoverable conditions, for example, SNA CLEAR received (RESP2=230).

Bypass by user exit
A command can be rejected by the FEPI global user exits (RESP2=10). Typically this would be because it violates the rules imposed by your system programmer. Check the rules with your system programmer.

Unknown conversation ID
An unknown conversation ID is most likely to occur because the ID is specified incorrectly.

Alternatively this problem could be caused by the task that issued the command not owning the conversation, because:

- The conversation has been ended
- The conversation has been passed to another task
- FEPI ALLOCATE with PASSCONVID has not been issued.

If the error occurs on a **FEPI ALLOCATE** command with PASSCONVID, the conversation was probably not “unowned”. Where the CONVID was obtained from FEPI start data, it is possible that between FEPI scheduling the task and it starting, a resource used by the conversation has been discarded, or CICS has started shutdown.

Operator/system action
An operator/system error occurs when the operator tries to cancel a FEPI transaction. If, as is likely, it is waiting for a FEPI command to be processed, it is the ‘wait’ for FEPI processing that is canceled, not the transaction.

When a FEPI command fails with an ‘operator action’ error (RESP2=18), first end all the active conversations and then end the transaction as soon as possible.

Shutdown
A normal CICS shutdown waits for currently active tasks to end, but does not allow new tasks to start. FEPI allows existing conversations to continue within a task but does not allow them to be passed to another task (because that task would never be started), nor does it allow new conversations to be started.

Conversations that are “unowned” are ended immediately, because the tasks that would subsequently handle them would never be started. Therefore, **FEPI START** or **FREE PASS** commands issued during shutdown fail (RESP2=214); in this case the error-handling routine, after doing whatever housekeeping is

required, should issue **FEPI FREE** to end the conversation. **FEPI ALLOCATE** commands issued during shutdown fail with RESP2=12.

You might need to take special action on the back-end system, for example, signing off, when the front-end application is going to shut down. For this reason, when conversations end during shutdown, the end-session handler is invoked with SHUTDOWN indicated in the EVENTVALUE field of the start data, so that the back-end system can be restored to a known state before FEPI ends; the FEPI FREE issued by the handler is treated as if RELEASE is specified. If you require this function, make sure the end-session handler is defined in the transaction list table (XLT), so that it can be started, and so that it does not adversely affect the performance of CICS shutdown. Using an end-session handler is the only way to perform special processing on shutdown, because no notification of shutdown is given to normal active transactions and conversations.

An immediate CICS shutdown ends all conversations immediately, and commands in progress fail. No further FEPI commands can be issued, and no end-session handlers are started.

System considerations

You can think of FEPI as a “pipe” through which users access back-end transactions; any peculiarities that exist in the back-end system have to be allowed for in the FEPI application. IMS has special considerations and these are explained in the following text.

This topic section concludes with some notes about performance.

IMS considerations

It is essential that you are familiar with using IMS and writing IMS applications.

When designing access programs that have IMS as a target back-end system, careful consideration must be given to the differences between CICS and IMS under certain circumstances:

- Message protocols
- Use of response mode
- Beginning and end of session
- Effects of IMS restart and recovery features in a FEPI environment. (Because IMS is almost totally recoverable, this can present problems in the design of the FEPI application and some event handlers.)

Message protocols

In SLU2 mode, IMS sends messages only in reply to explicit requests from the terminal.

- Unsolicited data will not usually occur; rather it will be available for the next FEPI conversation to receive. At the start of a FEPI conversation, you should first dequeue all such messages. However, unsolicited data can occur when requested data arrives after a FEPI conversation has been ended by, for example, a timeout.
- Take care if you use the IMS /SET command to preset a destination or put the transaction ID in the SPA to specify which IMS transaction to use next.
- If you are using Message Format Services (MFS), consider the following:
  - Physical paging or operator logical paging:
    - Whether paged output is deleted automatically by an input message or not.
    - For SLU2 mode, sending PA1 to request additional pages of paged output, and sending PA2 to remove paged output from the queue.
  - Unlocking the keyboard after MFS bypass.

Response mode

You are strongly recommended to run all your back-end IMS transactions in response mode where messages to IMS from the simulated terminal are handled synchronously; each message from the

terminal is processed by IMS and the reply is queued before a further message from the terminal is allowed.

Using response mode simplifies the front-end application because the data received is the reply to the data just sent and because the data stream flows from IMS are more straightforward; further, a separate FEPI conversation can be used for each IMS transaction and this allows much better use of the network. You must use the same FEPI conversation throughout an IMS conversational transaction.

If you use non- response mode, the data stream flows might be more complex. If you send multiple messages to IMS, the application has to handle asynchronous messages from IMS and, to keep the same simulated terminal, has to use the same FEPI conversation all the time.

Check with your system programmer that the transactions to be used by FEPI are defined to run in response mode. This requires the terminals for FEPI to be defined either to force response mode or to use the setting for the transaction (which in turn should be defined as response mode).

Beginning of session
For SLU2 , there is always initial data.

You should:

- Dequeue all output messages by sending CLEAR and PA1 after each FEPI RECEIVE, until there are no more messages (there may be 'unsolicited' data as well as the initial data).
- If there is an IMS error message, end the session using FEPI FREE with the RELEASE option.

End of session
When you are designing an application program you must plan what happens when a session ends.

You must ensure the following:

- An IMS conversation does not remain active.
- An IMS /RCLSDST command is issued if appropriate.
- An IMS MFS bypass application does not remain in bypass mode.
- Any preset destination has been reset.
- Any used test mode has been ended.
- No paged output message remains on the IMS message queue.
- All messages have been received.

Physically paged messages are removed from the queue automatically when the last page has been sent and, if they are recoverable, acknowledged. Operator logically-paged messages are not removed and require a PA2 (for SLU2 mode) or a NEXTMSG/NEXTMSGP control function (for SLU P mode) to be sent to IMS to remove the message from the queue if no input message is due.

IMS recovery
After a system failure, IMS recovers following a restart from the last checkpoint it took.

This means that, if the failure occurs when IMS has committed a message to the input queue then, on restart, IMS requeues that message and schedules a transaction to process it. Similarly, IMS will requeue all output messages that it has committed to its output queues and not successfully sent.

When IMS fails, all sessions between FEPI and IMS are ended. This is reported to the FEPI application as a command error ('session lost'). A FEPI application should check this so that it can tidy up before ending and take the appropriate action (such as informing the operator).

FEPI attempts to regain lost connections and, therefore, when IMS restarts, any previously acquired connections are reestablished. If IMS has committed an input or output message, eventually there is going to be an output message to send. With the connection reacquired, IMS attempts to recover its position and ultimately to send any queued output messages to the FEPI node that carried the original FEPI conversation. The process of recovery in this situation is different for each of the two modes:

- **SLU P recovery** When IMS tries to recover SLU P connections, it uses 'set and test sequence numbers' (STSN) in an attempt to resynchronize failed conversations. The STSN flow from IMS carries its version

of the sequence numbers for the node being resynchronized. If there is an STSN handler specified, it is started. If not, FEPI responds POSITIVE, which effectively tells IMS that FEPI is satisfied with the sequence numbers sent. On receiving this, IMS sends all messages queued for the node. FEPI receives the messages, discards them and responds to IMS, completing the resynchronization.

- **SLU2 recovery** The queued message is sent by IMS until there is a request from a front-end application, that application will receive the message as unexpected data interleaved with the data that it expects to receive. This problem can be handled in either of two ways:

  1. By the application issuing a FEPI RECEIVE, with TIMEOUT, before starting its intended task or by dequeuing all output messages using CLEAR and PA1.
  2. By the begin-session handler.

This situation becomes more complex if the back-end transaction is IMS conversational, because the front-end transaction has no way of knowing this, and the IMS conversation will still be active in the back-end system awaiting input.

The potential therefore exists for a front-end FEPI application to allocate a FEPI conversation on a node where an IMS conversation still exists on the back-end system. Any data flowing on this FEPI conversation is viewed by the front-end application as an exchange with a new back-end transaction, but it is viewed by IMS as the next input message to the existing conversation. To prevent this situation occurring, you can use the begin-session handler to issue the IMS /EXIT command, which has the effect of ending an active IMS conversation.

Where the possibility exists of a number of nodes with active IMS conversations following a restart, it is possible to use FEPI to obtain a connection to IMS and control the clean up operation, from a single point. You do this by issuing, again from the appropriate handler:

- An IMS /DISPLAY command to display all active conversations
- The IMS /EXIT command to end all those attached to FEPI nodes.

In the event of a failure that unbinds all the FEPI connections to IMS, the recovery procedure is identical to that described here.

FEPI application techniques for performance
Use the following techniques to get the best performance from your FEPI applications; the main principles are to minimize the number of commands issued and the amount of data transmitted.

Remember, however, that some of these techniques have drawbacks and some conflict with each other; you must choose the best balance to meet your needs.

- Use data area sizes that allow a send or receive to be completed with a single FEPI command.
- Use FEPI CONVERSE where possible. But remember that the send part of CONVERSE can fail for various reasons, so be sure to write your program so that it can issue a subsequent FEPI RECEIVE if necessary.
- The pseudoconversational style (use of FEPI START commands) results in significant CPU overheads in the front-end region. Further, since the use of FEPI START generates additional flows to and from the real terminal, response times are also significantly increased. As a consequence, FEPI START should be used sparingly when, for example, the receipt of the data from the back-end application takes a long time.
- Avoid ending sessions unnecessarily. Use the begin-session and end-session handlers to manage usage of the connections.
- Try to avoid operator dependency in exchanges with a back-end system.

Formatted data

- Unformatted screens (where the terminal character buffer contains no field attributes) require more processing than formatted screens. Where possible use formatted screens from the back-end systems.
- Not clearing a screen results in unnecessary data being transmitted to the back-end system.

- If, when data is received, only a small portion of the resultant screen is of interest, use FEPI EXTRACT FIELD to minimize the amount of data that needs to be transferred to the application.
- When using key stroke data, avoid issuing a FEPI CONVERSE, SEND, or RECEIVE for each attention operation; combine all the operations into one long string.
- When using key stroke data with an unformatted screen, use the HOME and ERASE-EOF keys to clear the screen rather than CLEAR, because the latter requires a network transmission.
- Use key stroke rather than screen-image data where possible, because much less data needs transferring from the application.

Specialized FEPI functions

These topics describe specialized control functions that are handled by FEPI but can be taken over by a FEPI application.

It contains the following topics:

Set and test sequence number (STSN)

In SLU P mode, message sequence numbers are available in the data stream to allow message resynchronization. This can be demanded by a 'Set and Test Sequence Number' (STSN) request when a session is started.

The response that IMS requires, and that FEPI supplies if the system programmer has not defined a transaction to handle the STSN request, depends upon whether the STSN request showed 'SET' or 'TEST and SET':

- For 'SET', the response is always 'TEST POSITIVE'.
- For 'TEST and SET', the response is 'TEST POSITIVE' or 'TEST NEGATIVE'.

Any other response to STSN will cause the session to be unbound.

If an STSN handler is defined, it is started when session resynchronization is requested by the back-end system through an SNA STSN or SDT command. The back-end system sends an SNA STSN command indicating whether the last inbound message was in doubt or not; that is, whether a message had been sent by the back-end system but it had not logged the receipt of a response. The back-end system does not send an SNA STSN command if no traffic has been on the session since the latest cold start of the back-end system, but sends an SNA SDT command directly.

Like other handlers, the STSN handler is a CICS started task that uses EXEC CICS RETRIEVE to get the start data and FEPI ALLOCATE with PASSCONVID to get ownership of the conversation identified in that data. The STSN handler, which can use the FEPI EXTRACT STSN command to determine what response is needed, must use the FEPI ISSUE command to respond to the STSN.

FEPI normally does all the necessary STSN handling automatically, so an STSN handler is required only where you need to handle the sequence number information yourself. The FEPI SEND, FEPI RECEIVE, and FEPI CONVERSE commands return the current sequence numbers for you.

A sample program illustrates the techniques used. See FEPI sample program: STSN handler.

DRx responses

In all cases except those mentioned in the next paragraph, FEPI automatically gives a positive DRx response when the inbound data indicates that a response is required. This response flows on the next FEPI command (SEND, RECEIVE, CONVERSE, FREE, or START).

The automatic response is not issued if the next command for a conversation is a FEPI ISSUE CONTROL or a FEPI FREE PASS. Thus, if you want to send your own response, perhaps for added certainty or

confirmation of particularly sensitive changes, you would do so using FEPI ISSUE CONTROL. The response type that is required can be determined from the RESPSTATUS option of FEPI RECEIVE and FEPI CONVERSE.

You can send your own responses with either formatted data or data stream. But do not use the following because they can cause FEPI to send responses automatically:

- Key stroke formatted data containing an attention key that is not the final key stroke
- FEPI CONVERSE with the POOL option to use a temporary conversation.

If you respond negatively a back-end CICS system will discard the data but an IMS system will resend it.

SNA commands

The FEPI ISSUE command allows you to send various other SNA commands yourself. You should do this only if you have a particular requirement.

Chapter 6. Improving FEPI performance

You cannot tune FEPI itself because it is already optimized for speed of response. However, you can manage the performance of FEPI applications.

FEPI runs under a separate CICS task control block (TCB) and CICS permits only one application program to issue a FEPI command at a time. This is a major influence on FEPI performance. Although many application programs can have FEPI commands being processed at any time, only one application can issue a FEPI command.

In a lightly loaded system, CICS does not run FEPI until a command is issued. Therefore, performance is impacted by the overhead of starting up the TCB so that the FEPI command can be processed. In a heavily loaded system, this overhead is not present, because the TCB is already active processing earlier FEPI commands.

FEPI tries to minimize this overhead by issuing timer requests that ensure that the TCB is not inactive for more than one second.

There are three main principles that can be used in FEPI applications to provide the best performance:

- Minimize the number of commands. Each FEPI command generates a CICS WAIT, even if no network transmission is involved.
- Keep data transmission to a minimum.
- Avoid disconnecting sessions.

Application development techniques are described in FEPI application techniques for performance.

For FEPI system programming, you can reduce the number of commands by using lists of resources on a command where possible. However, if you use a list that results in a z/OS Communications Server operation, the following conditions might occur:

- Flood z/OS Communications Server by requesting too many operations at once
- Flood the back-end system with requests for session initiation
- Flood the front-end system with started begin- or end-session transactions.

So you must carefully evaluate the benefits of using lists.

Using CICS monitoring

CICS monitoring data can help with performance tuning and resource planning for applications that use FEPI.

By default, CICS performance class monitoring records include the following data about the user task:

- The number and type of requests made to FEPI
- The time spent waiting for requests to FEPI to complete
- The number of requests to FEPI that are timed out.

For detailed information about the FEPI-related fields in performance class monitoring records, see Performance class data: listing of data fields.

Using statistics data

CICS statistics can help with performance tuning and resource planning for applications that use FEPI.

The standard CICS statistics reports contain data about usage of:

- FEPI pools

- FEPI connections
- FEPI targets.

To obtain the current statistics for a FEPI pool, connection, or target, a utility program can issue an **EXEC CICS COLLECT STATISTICS** command. For example, the command `EXEC CICS COLLECT STATISTICS SET(pointer) POOL(GRPD)` returns the current statistics for the 'GRPD' pool. To map the returned statistics, your utility program should include the appropriate CICS-supplied copybook:

DFHA22DS
FEPI pool statistics

DFHA23DS
FEPI connection statistics

DFHA24DS
FEPI target statistics.

The copybooks are supplied in COBOL, PL/I, and assembler language.

To cause all FEPI statistics to be written immediately to the SMF statistics data set, you can use either the EXEC CICS or the CEMT version of the **PERFORM STATISTICS RECORD FEPI** command. For details of the **CEMT COLLECT STATISTICS** and **PERFORM STATISTICS RECORD** commands, see CEMT - master terminal; for programming information about the equivalent **EXEC CICS** commands, see the Introduction to System programming commands.

To format and print FEPI-related statistics in the DFHSTATS data set, you can use the CICS-supplied utility program, DFHSTUP. To print only the FEPI statistics, specify the command parameter SELECT TYPE=FEPI. For information about how to use the DFHSTUP program, see Statistics utility program (DFHSTUP). For detailed information about fields in the FEPI statistics records, see FEPI statistics.

Chapter 7. Troubleshooting FEPI

If a problem occurs with a FEPI application, use the guidance information in this section to help you identify the source of the errors.

This information contains diagnosis, modification, or tuning information. It contains the following topics:

Debugging FEPI applications

The CICS execution diagnostic facility (EDF) helps users of the EXEC CICS interface to step through the EXEC CICS commands of an application program.

EDF can be used in just the same way to debug programs that use the EXEC CICS FEPI commands.

FEPI dump

CICS dump routines are available for FEPI. These routines are under the control of the usual CICS selection mechanisms.

You generate interpretation of the FEPI areas of a CICS dump by specifying the SZ keyword from within the interactive problem control system (IPCS). SZ can take the following values:

SZ value
What is printed

0
No FEPI areas are interpreted.

1
All FEPI areas are interpreted, excluding the stacks.

2
All FEPI areas are interpreted, including the stacks.

If you are looking at a FEPI problem, first ensure the SZ TCB is active, and the FEPI Resource Manager is running. Look at the kernel and dispatcher prints to verify their presence.

If the SZ TCB is present, and the FEPI Resource Manager is running, the problem is probably caused by a wait or an abend. In the case of a wait, the dispatcher and kernel prints should show where it is located.

After looking at any FEPI trace entries, you should direct your attention to the output from the 'SZ=2' dump formatting keyword. This displays all known FEPI control blocks. If you think a storage violation has occurred, use the dump storage manager options to display the contents of the FEPI storage subpools.

Here are some things that might help you identify a problem when you read the dump:

- Were any errors reported during interpretation? If so, this may indicate a corrupt address pointer or a broken chain.
- Follow all the pointers to associated control blocks (such as the conversation pointed to by the connection). Is this pointer correct? If not, this probably indicates corruption.

- Are there the expected numbers of nodes, targets, property sets, and pools? If not, this can indicate a broken chain or an unauthorized deletion.
- Does each pool contain the expected number of connections (that is, the number of nodes multiplied by the number of targets)? If not, this may indicate the failure of a FEPI ADD command.
- Has each node been successfully acquired? If not, there is the possibility of z/OZ Communications Server definition errors. The ACB and RPL may contain z/OZ Communications Server sense information —perhaps a z/OZ Communications Server major node is inactive.
- Is there successful communication with a target? If not, have APPLID and PASSWORD been correctly specified? If they are correct, is the back-end system running?
- Are there any queued ALLOCATE commands? If so, this indicates that there are not enough connections for the pool to process FEPI conversations without queuing. This may be acceptable, or not, depending on your configuration.
- Are the event handlers being run? If not, have they been correctly defined to CICS using RDO?
- Are the event handlers being recursively invoked? If so, this indicates a problem with a FEPI FREE command, a storage violation, or an internal logic error.
- Is information being correctly sent to the specified transient data queues? If not, are the queues defined as unrecoverable?
- Are transactions being triggered from the TDQs? If not, are the transactions correctly defined to CICS?
- Is there a current conversation? If so, this conversation may be causing the error. Is the data correct? Is there any z/OZ Communications Server sense information in the RPL?
- Are the surrogate terminals correct? If not, the links between the nodes, pools, and targets may have become corrupted.
- Are FEPI SEND or FEPI RECEIVE commands failing due to state errors? If so, look at the conversation and see if the states are correct. If they are not, the conversation has become out of step with the z/OZ Communications Server flow.
- Is unexpected data being sent or received in formatted conversations? If so, there may be corrupt FEPI data. Look at FEPI's internal terminal character buffer.
- Look at the queues. Are there any requests that look as if they have got stuck? If so, the FEPI work chains may be corrupt. However, it may be that the flow to satisfy the requests has not yet happened. If you think it should have happened, there may be communication problems.
- Look at the FREE queue. The last z/OZ Communications Server event may be shown. If so, does it correspond with what you expected?
- Is the behavior of a pool correct? If not, it is possible that the property set used to define the pool is incorrect. However, if the property set is shown, it could have been re-created since the pool was defined—treat property set definitions with care.
- Are there any outstanding timer events that should have run? If so, this may indicate a chaining failure.
- Has a timer-dependent action been delayed? If so, this could indicate that the TIMEOUT parameter on the command was incorrect.
- Are you receiving all the data you expect? If not, have you set the correct end-of-flow condition on the FEPI RECEIVE (or CONVERSE) command?
- Are there many transactions waiting on FEPI? If so, either back-end systems are not responding, or the FEPI Resource Manager has failed.
- Has a z/OZ Communications Server dump been taken? If so, this may indicate a failure in one of the z/OZ Communications Server exits.

Using CICS dump facilities to investigate FEPI problems

This section describes how FEPI relates to the rest of CICS, and how its presence is revealed by the other CICS dump formatting commands.

The problem determination process for FEPI is driven from the usual CICS dump interpretation routines. The following sections describe what to look for in the major CICS areas.

Dispatcher
You should see a task (CSZI) running under the SZ task control block.

(However, note that CSZI can run under the QR TCB while executing certain CICS functions, such as starting transactions and writing to transient data queues.) If CSZI is not present, then either FEPI is not in the system, or the FEPI Resource Manager has failed.

Application programs waiting for responses from the FEPI Resource Manager are shown as waiting on FEPI.

Interval control
Any transactions that have been started by the FEPI Resource Manager, but not yet run, appear in the interval control section.

Kernel
In the kernel, you should find a running task named KETCB SZ representing the SZ TCB that FEPI uses. If KETCB SZ is not present, then either FEPI is not in the system, or the TCB has abended.

You should find the CSZI task either running or waiting. If CSZI is not present, then either FEPI is not in the system, or the FEPI Resource Manager has failed.

If an abend has occurred, the usual information is available. The location of the abend is indicated by the failing module, as follows:

DFHESZ
The application programming EXEC stub

DFHEIQSZ
The system programming EXEC stub

DFHSZATR
The FEPI adapter

DFHSZRMP
The FEPI Resource Manager.

Storage manager
You can use the storage manager dump facilities to display the contents of the subpools used by FEPI.

If you suspect a storage violation, a comparison of the contents of these subpools with the areas interpreted by a FEPI dump might show where the corruption has occurred.

Table 10. FEPI storage subpools

Name	Type	Chained	Above or below 16MB line?	Usage
SZSPFCAC	Fixed	Yes	Below	ACBs
SZSPFCCD	Fixed	Yes	Any	Connections
SZSPFCCM	Fixed	Yes	Any	Common area
SZSPFCCV	Fixed	Yes	Any	Conversations
SZSPVUDA	VAR	Yes	Any	Various data areas
SZSPFCDS	Fixed	Yes	Any	Device support extensions
SZSPFCDT	Fixed	Yes	Any	Device-type control areas

Table 10. FEPI storage subpools (continued)				
Name	**Type**	**Chained**	**Above or below 16MB line?**	**Usage**
SZSPFCNB	Fixed	Yes	Any	NIBs
SZSPFCND	Fixed	Yes	Any	Nodes
SZSPFCPD	Fixed	Yes	Any	Pools
SZSPFCPS	Fixed	Yes	Any	Property sets
SZSPFCRP	Fixed	Yes	Any	RPLs
SZSPFCRQ	Fixed	Yes	Any	Requests
SZSPFCSR	Fixed	Yes	Any	Surrogates
SZSPFCTD	Fixed	Yes	Any	Targets
SZSPFCWE	Fixed	Yes	Any	DQEs

FEPI trace

There are appropriate trace entries in the CICS trace table which are under the control of the usual CICS mechanisms.

FEPI trace entries are listed in the Trace entries overview manual.

FEPI generates exception and event trace entries - the latter under control of the ‘SZ' component code. Points AP 1200 through AP 16FF are reserved for use by FEPI, although not all of these are used.

Taking and interpreting trace entries

FEPI supports only one level of tracing. At CICS initialization, you can specify the default levels of standard and special tracing using the **STNTR**, **SPCTR**, **STNTRSZ**, and **SPCTRSZ** system initialization parameters.

About this task

Exception trace entries are always taken.

To take trace entries, you can either:

Procedure

1. You can start tracing using either of these methods:
   - Use the CETR transaction. The FEPI component code on the CETR “Component Trace Options” panel is ‘SZ’. Specify ‘SZ 1’ to turn on FEPI tracing.

     You can use the selection features of the CETR transaction to limit tracing to specific transactions rather than using the SZ component. If you do this, you can control the tracing of application programs. The FEPI Resource Manager, running as the CSZI transaction, is unaffected, because trace selection is applied only at transaction start.
   - Use the following command: **SET TRACETYPE SZ**.
2. Check if there are any exception trace entries.

   Their presence indicates that a problem has been detected. Exception trace entries are either initialization errors or storage management errors.

   a) Initialization errors result from checks made when CSZI starts, to prevent a second instance of the FEPI Resource Manager.

b) Storage errors result from GETMAIN or FREEMAIN errors, and are usually caused by a lack of CICS storage.

3. Other trace entries are the usual module entry and exit traces, together with a few points indicating that important processing events have occurred (such as the FEPI Resource Manager becoming idle).

What to do next
If you are using DFHTRAP under the guidance of IBM support, note that the FEPI Resource Manager runs under the SZ TCB. Therefore, do not do anything that could force an MVS task switch to any other TCB.

FEPI messages

Messages produced by FEPI have exactly the same format (DFHSZ...) as other CICS messages. They are all sent to the FEPI message log (the CSZL transient data queue); some are also sent to the operator.

FEPI messages are documented in CICS messages, and are also available through the CMAC transaction.

FEPI abends

FEPI does not (deliberately) issue either CICS transaction abends or MVS abends.

However, an unexpected failure can occur in the following places:

- In a FEPI application program when INVREQ is returned
- In the EXEC stubs
- In the FEPI adapter
- In the FEPI Resource Manager transaction (CSZI) code
- In a z/OS Communications Server exit routine.

These abends have different results, as shown in Table 11 on page 91.

Table 11. Types of abend issued by FEPI

Point of failure	**Result**
Application	The usual transaction abend for the error condition.
EXEC stubs	The usual transaction abend for a failure within CICS management modules. An example of this is an ‘operation' program check, which generates a CICS AKEA abend, which in turn generates an ASRA abend.
FEPI adapter	The usual transaction abend for a failure within CICS management modules. An example of this is an ‘operation' program check, which generates a CICS AKEA abend, which in turn generates an ASRA abend.
FEPI Resource Manager	No direct effect on the application program, because the abend occurs under the CSZI Resource Manager task. This probably results in a DFHSZ4099E message (see “Message DFHSZ4099E” on page 92), and the failure of the Resource Manager. An example of this is an ‘operation' program check, which generates a CICS AKEA abend, which in turn generates an ASRA abend. Any CICS FEPI transactions remain waiting on the FEPI_RQE resource (for details of FEPI waits, see FEPI problem determination.
z/OS Communications Server exit	A z/OS Communications Server abend; a z/OS Communications Server dump is taken. Because the exit lies within the FEPI Resource Manager, the CICS abend handling routines are activated to process a “normal” failure in the Resource Manager.

Restart

An abend in an application program, an EXEC stub, or the FEPI adapter affects only the active CICS task that issued the FEPI command; other FEPI programs continue as normal.

If an abend affects the SZ TCB, CICS makes that TCB unavailable for use, while keeping the other CICS TCBs active and accessible. This means that FEPI functions can be restored only by restarting the CICS system.

Message DFHSZ4099E

This message indicates that the abend exit routine within the FEPI adapter has trapped an abend within the FEPI Resource Manager.

As soon as an abend within the Resource Manager is detected, the FEPI state (in the FEPI static area) is set to 'Failed'. If possible, message DFHSZ4099E is issued, together with a SNAP dump, to indicate that FEPI has failed. However, in some circumstances it is not possible to issue DFHSZ4099E, and a system dump is generated instead.

Any FEPI transactions that are waiting on the FEPI_RQE resource never get posted, so the transactions suspend. For details of FEPI waits, see FEPI problem determination. You must issue a CEMT FORCEPURGE command to remove these suspended transactions from the system.

Attention: It is strongly recommended that the CSZI transaction is initiated only as part of CICS system initialization. *Do not attempt to restart the CSZI transaction after a failure, other than by restarting CICS*.

Message DFHSZ4155I

This message indicates that a connection has ended, and gives a reason code taken from the z/OS Communications Server control blocks. The reason code may be returned in the LASTACQCODE option of a CEMT or FEPI INQUIRE command, depending on the operation which generated DFHSZ4155I.

DFHSZ4155I does not always indicate a problem; if you took positive action to end the connection, DFHSZ4155I merely confirms that z/OS Communications Server did as you requested. However, if the connection ended unexpectedly, the reason code tells you why.

Reporting a FEPI problem to IBM

When reporting a problem to IBM Support, you must provide details of the CICS system in which FEPI is installed.

About this task

You must provide the following information:

- All listings from the CICS job, including the CICS job log and JCL
- A print of all reports sent to the CSZL transient data queue
- A full system dump (including the CSA and LSQA)
- Any relevant transaction dumps
- All trace entries (you may need to re-create the problem with SZ trace active)
- A listing of the application program that detected the problem
- Listings of the programs used to configure your FEPI system
- Listings of any active CICS global user exit programs (not only the FEPI ones)
- Prints of user journals, if FEPI journaling was active when the problem occurred.

The following materials might also be required:

- A z/OS Communications Server trace showing the data flows

- A trace of the back-end system showing what data streams were received from FEPI application programs
- A z/OS Communications Server status display showing the status of FEPI connections
- Any dumps or logs produced by the back-end system.

Appendix A. FEPI application programming reference

These topics define the FEPI application programming commands.

System programming commands such as **INSTALL**, **INQUIRE**, and **SET** are defined in Appendix B, "FEPI system programming reference," on page 137.

For EIB response codes, see Response codes of EXEC CICS commands.

For EIB function codes, see Function codes of EXEC CICS commands.

Overview of the FEPI API commands

The FEPI application programming commands are additions to the set of **EXEC CICS** commands that are available to application programmers and they have the same features and properties as those commands.

The notation used to describe these commands is the same as that used to describe all application programming commands in CICS. The FEPI application programming commands are:

- **ALLOCATE**
- **AP NOOP**
- **CONVERSE**
- **EXTRACT**
- **FREE**
- **ISSUE**
- **RECEIVE**
- **REQUEST PASSTICKET**
- **SEND**
- **START**

You must specify the 'FEPI' translator option when you use FEPI commands. FEPI commands can be issued in either 24-bit or 31-bit addressing mode, by programs that reside either above or below the 16MB line.

No information is passed through the EXEC interface block (EIB) except that, as for all CICS commands, the EIBRESP, EIBRESP2, EIBFN, and EIBRCODE fields are set.

For EIB response codes, see Response codes of EXEC CICS commands.

For EIB function codes, see Function codes of EXEC CICS commands.

Arguments and data types

The text used to identify arguments indicates the type of data represented by the argument and whether it is a value used by the command, or an area in which the command returns data. For example:

- `POOL(8-character data-value)` indicates that the argument is, or identifies, a string of eight characters, and that the string is passed to the command as an input value.
- `ACQNUM(fullword binary data-area)` indicates that the argument is a user-defined fullword data area in which the command can return a binary number as an output value.

Exceptionally, arguments that are lists have to be data areas, even though they are input values.

Command format

The general format of a command is:

```
EXEC CICS FEPI command option(argument)...
```

where:

command
Is the command name (for example, ADD)

option
Is an option name (for example, POOL)

argument
Is the source or destination for data, as required for the specified option, that is passed to or returned from the command.

The way that you terminate the command is determined by the programming language that you use—COBOL, for example, requires an END-EXEC statement.

Errors and exception conditions

All FEPI commands support the RESP and RESP2 options to signal successful completion or an exception condition. Alternatively, you can use HANDLE CONDITION to trap errors.

Most FEPI command errors give the 'INVREQ' exception condition. The particular error in each case is uniquely identified by the RESP2 value.

If there is an error, the command does nothing and the output arguments are not changed. Note, however, that commands such as FEPI SEND may have transferred data before the condition is recognized.

Both RESP and RESP2 take, as an argument, the name of a user-defined fullword binary data area. Possible values of the RESP2 option are given in the description of each of the commands and a full list is given in "FEPI RESP2 values" on page 181. The following copy books provide declarations for the RESP2 values:

- DFHSZAPA for Assembler language
- DFHSZAPO for COBOL
- DFHSZAPP for PL/I
- DFHSZAPC for C.

The INVREQ condition and the following RESP2 values can occur for any application programming command:

RESP2
Meaning

10
Command bypassed by user exit.

11
FEPI not installed, or not active..

12
CICS shutting down, command not allowed.

13
FEPI unavailable.

14
FEPI busy or cannot get storage..

15
Unknown command..

16
Internal error.

17
FEPI cannot get storage for user exit..

18
Command failed through operator or system action..

FEPI ALLOCATE PASSCONVID

FEPI ALLOCATE PASSCONVID acquires ownership of an existing unowned conversation.

Syntax

FEPI ALLOCATE PASSCONVID

Notes:

Options

PASSCONVID(8-character data-value)
specifies the ID of the conversation.

Conditions

RESP2
Meaning

216
Error occurred on previous FEPI SEND.

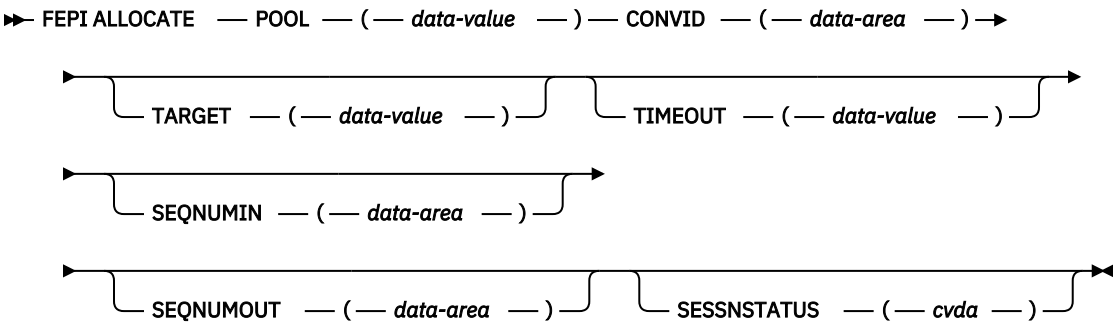
240
Unknown conversation ID.

FEPI ALLOCATE POOL

FEPI ALLOCATE POOL establishes a new FEPI conversation with a target application, acquiring a session from the named pool to use for the conversation. The conversation has the properties, particularly the mode (SLU2 or SLU P) and data format (data stream or formatted), specified for the pool that is used: some of the properties can be queried using FEPI EXTRACT CONV.The command completes immediately if, in the named POOL, a suitable session has been established and is not in use. Otherwise the request waits for a session to become available. A time limit can be set for this wait.

Syntax

FEPI ALLOCATE POOL



Notes:

Options

CONVID(8-character data-area)
returns a unique identifier for the new conversation; this is the ID that must be quoted on all subsequent commands for the conversation.

POOL(8-character data-value)
specifies the name of the pool containing the target for the conversation.

SEQNUMIN(fullword binary data-area)
in SLU P mode, returns the current sequence number for inbound data. (SEQNUMIN has no significance in SLU2 mode.)

SEQNUMOUT(fullword binary data-area)
in SLU P mode, returns the current sequence number for outbound data. (SEQNUMOUT has no significance in SLU2 mode.)

SESSNSTATUS(cvda)
returns a value that indicates whether the session being used for the conversation was newly-bound or not. The relevant CVDA values are:

- NEWSESSION
- OLDSESSION

TARGET(8-character data-value)
specifies the name of the target. TARGET can be omitted if there is only one target in the pool or if all targets are suitable for the intended conversation.

TIMEOUT(fullword binary data-value)
specifies the maximum time in seconds that the command is to wait for a suitable session to become available. If TIMEOUT is not specified or the specified time is zero, the command is not timed out.

Conditions

If an INVREQ condition is returned, it can have the following RESP2 values:

RESP2
Meaning

30
Pool name unknown.

31
Pool name out of service.

32
Target name unknown..

33
Target name out of service..

34
Target name required but not specified.

36
No suitable session available and in service.

213
Command timed out.

241
TIMEOUT value negative or not valid.

FEPI AP NOOP

FEPI AP NOOP has no effect.

Syntax

FEPI AP NOOP

Notes:

Options

None

Conditions

None specific to this command.

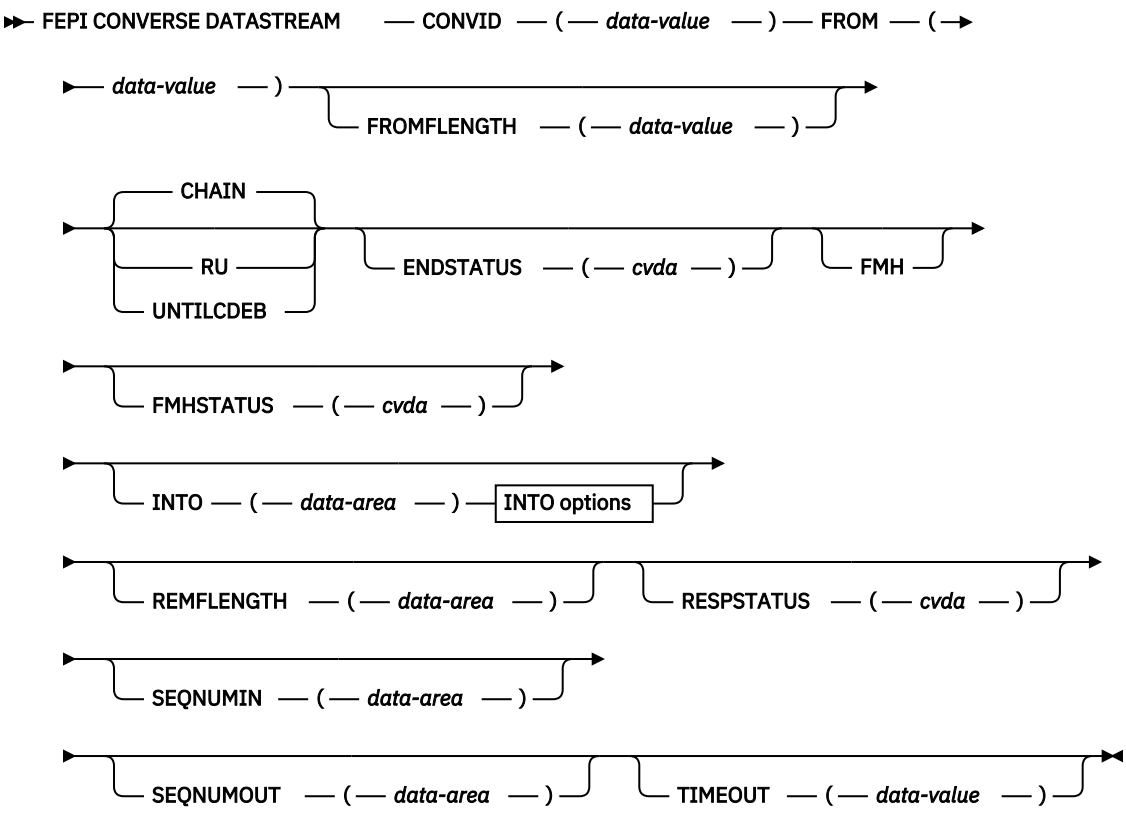
FEPI CONVERSE DATASTREAM

FEPI CONVERSE DATASTREAM sends application data to and receives a reply from a target.

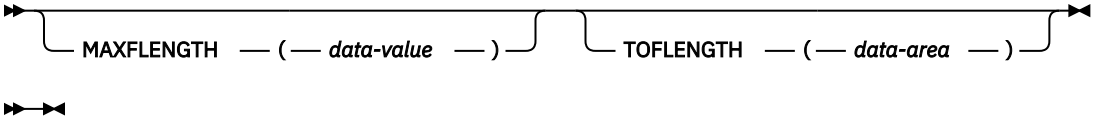
The data supplied by the application must be a currently valid data stream appropriate to the mode of the conversation (SLU2 or SLU P); the data received into the application's data area is also data stream. Full details about the data are given in “Data formats” on page 132

The conversation with the target can be one of two types. A time limit can be set for this command. For more details of ending conditions, see “Ending status” on page 134.

Previously allocated conversation syntax

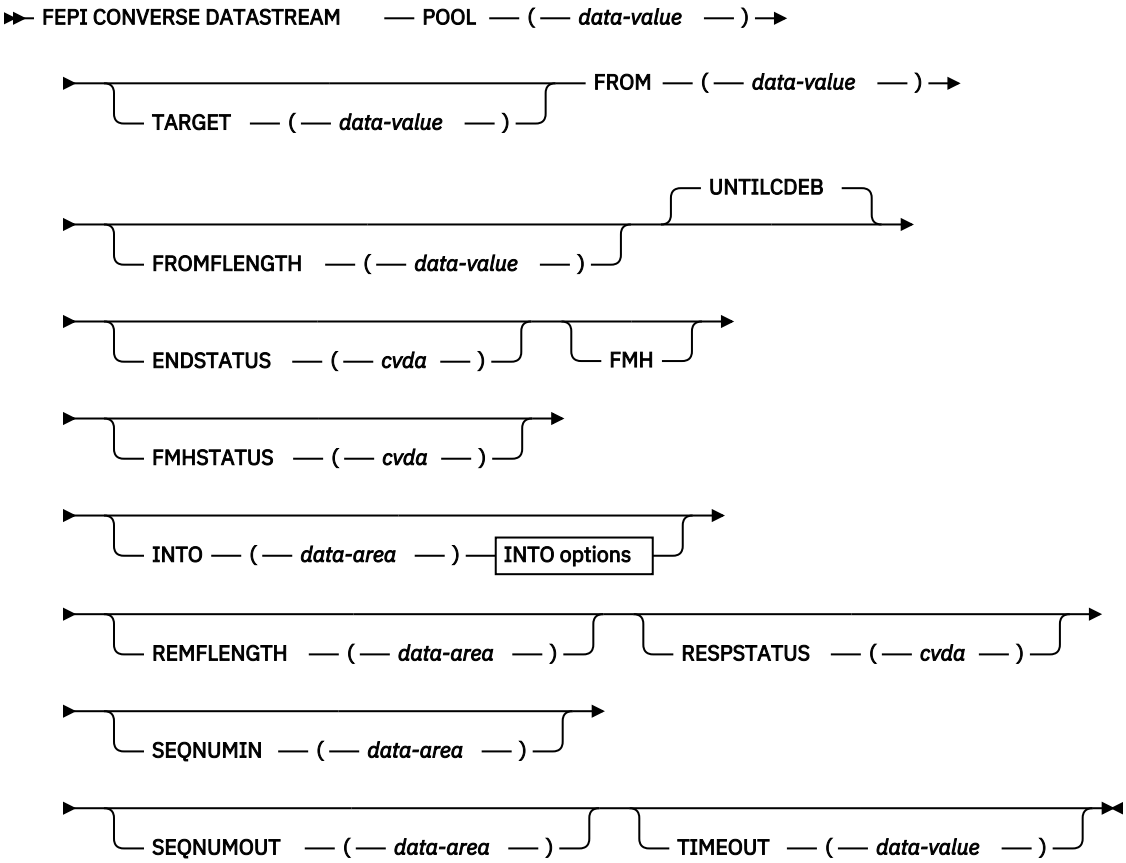


INTO options

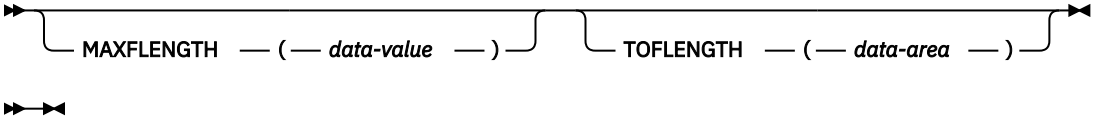


Notes:

Temporary conversation syntax



INTO options



Notes:

Options

CHAIN
specifies that the command should complete when a whole chain has been received. CHAIN is not allowed if the POOL option is specified.

CONVID(8-character data-value)
specifies the ID of the conversation to use. The conversation must be owned by the task issuing the command.

ENDSTATUS(cvda)
returns a value that indicates the ending status for the received data. The relevant CVDA values are:

Value
Meaning

CD
"Change direction" received.

EB
"End bracket" received.

LIC
"Last in chain" received.

RU
RU received.

MORE
The data area identified by the INTO option was too small to receive all the requested data.

For more details of ending status and how additional data is handled, see “Ending status” on page 134.

FMH
indicates that the data to send includes a function management header.

FMHSTATUS(cvda)
returns a value that indicates whether the received data contains a function management header. The relevant CVDA values are:

- FMH
- NOFMH

FROM(data-value)
specifies the data to send to the back-end application. Its length is specified by the FROMFLENGTH option.

FROMFLENGTH(fullword binary data-value)
specifies the length of the data to send; that is, the length of the data area identified by the FROM option. It must not be zero or more than the maximum length allowed for the pool.

INTO(data-area)
specifies the data area in which the received data is to be returned. The length of the area is specified by the MAXFLENGTH option, and the actual length of data written into the area is returned by the TOFLENGTH option.

MAXFLENGTH(fullword binary data-value)
specifies the maximum amount of data that can be returned; that is, the length of the data area identified by the INTO option. It must not be more than the maximum length allowed for the pool.

POOL(8-character data-value)
specifies the name of the pool containing the target for the conversation. Specifying POOL means that the conversation is a temporary one, that exists only for the duration of the FEPI CONVERSE. The CHAIN and RU options are not allowed, and the command completes when "change direction" or "end bracket" is received. If there is more data to receive than fits into the data area identified by the INTO option, the additional data is discarded.

REMFLENGTH(fullword binary data-area)
returns the length, if known, of data remaining after filling the data area identified by the INTO option.

RESPSTATUS(cvda)
returns a value that indicates the type of response that is required at the back-end system. The relevant CVDA values are:

Value
Meaning

DEFRESP1
Definite response 1 required.

DEFRESP2
Definite response 2 required.

DEFRESP3
Definite response 1 and definite response 2 required.

NONE
No response required.

RU
specifies that the command should complete when a request unit has been received. RU is not allowed if the POOL option is specified.

SEQNUMIN(fullword binary data-area)
in SLU P mode, returns the current sequence number for inbound data, as at the completion of the command. (SEQNUMIN has no significance in SLU2 mode.)

SEQNUMOUT(fullword binary data-area)
in SLU P mode, returns the current sequence number for outbound data, as at the completion of the command. (SEQNUMOUT has no significance in SLU2 mode.)

TARGET(8-character data-value)
specifies the name of the target. TARGET can be omitted if there is only one target in the pool or if all targets are suitable for the intended conversation.

TIMEOUT(fullword binary data-value)
specifies the maximum time in seconds that the command is to wait for the requested data to begin to arrive. If TIMEOUT is not specified or the specified time is zero, the command is not timed out.

TOFLENGTH(fullword binary data-area)
returns the actual length of data received in the data area identified by the INTO option.

UNTILCDEB
specifies that the command should complete when "change direction" or "end bracket" is received.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

30
Pool name unknown.

31
Pool name out of service.

32
Target name unknown.

33
Target name out of service.

34
Target name required but not specified.

35
POOL name is unsuitable for temporary conversations. It has CONTENTION(LOSE) or it has INITIALDATA(INBOUND) and no begin-session handler.

36
No suitable session available and in service.

40
FROMLENGTH value negative, zero, or more than the maximum allowed for the current pool.

50
Inbound data with "begin bracket" to be received.

58
z/OS Communications Server VTAM SEND failed.

60
MAXFLENGTH value negative, zero, or more than the maximum allowed for the current pool.

71
z/OS Communications Server VTAM RECEIVE failed.

212
Conversation has wrong data format.

213
Command timed out.

215
Session lost.

216
Error occurred on previous FEPI SEND.

220
FEPI CONVERSE not allowed at this point in the conversation.

224
Only FEPI ISSUE or FEPI FREE commands allowed at this point in the conversation.

230
SNA CLEAR command received. For an explanation of this SNA command, see Systems Network Architecture Formats (GA27-3136).

231
SNA CANCEL command received. For an explanation of this SNA command, see Systems Network Architecture Formats (GA27-3136).

232
SNA CHASE command received. For an explanation of this SNA command, see Systems Network Architecture Formats (GA27-3136).

233
Exception response received.

234
Exception request received.

240
Conversation ID not owned by this task.

241
TIMEOUT value negative or not valid.

FEPI CONVERSE FORMATTED

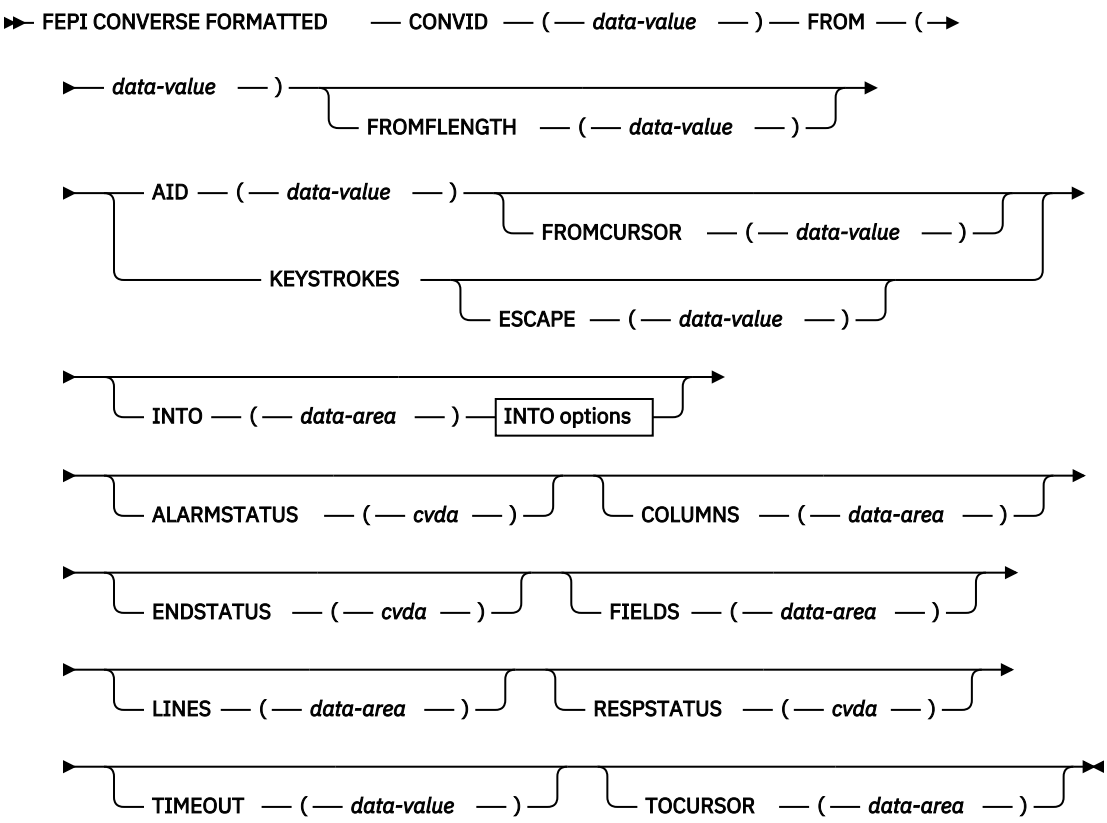
FEPI CONVERSE FORMATTED sends application data to and receives a reply from a target.

The data supplied by the application must be formatted data, as key strokes (with a final attention character) or a screen image; the data received into the application's data area is a screen image.

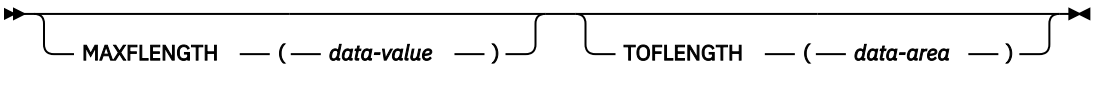
This command is for SLU2 mode only.

Full details about the data are given in "Data formats" on page 132. The conversation with the target can be one of two types. A time limit can be set for this command. For more details of ending conditions, see "Ending status" on page 134.

Previously allocated conversation syntax

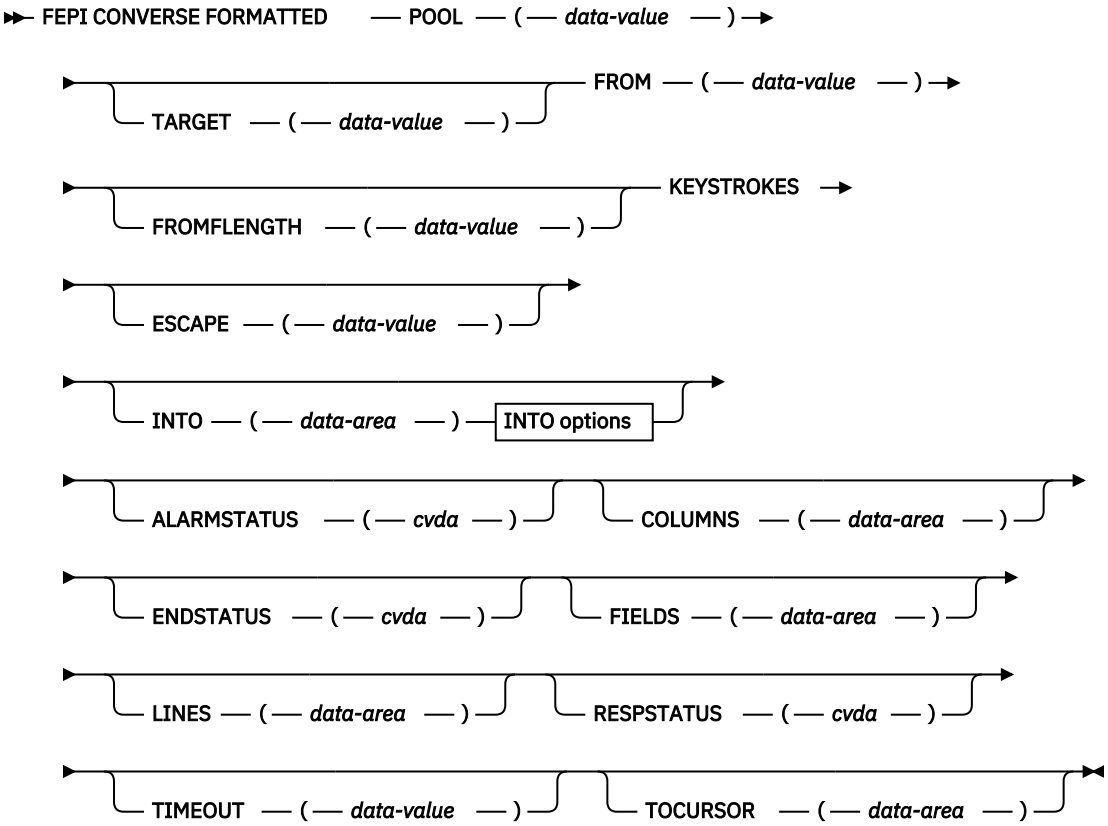


INTO options

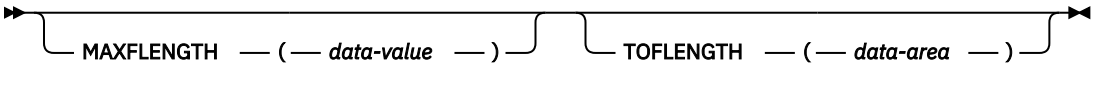


Notes:

Temporary conversation syntax



INTO options



Notes:

Options

AID(1-character data-value)
specifies the attention identifier value to send with the data. Specifying AID also indicates that the data to send is in screen-image format, as described in . The value must not be null (X'00'). AID, and therefore screen-image format data, is not allowed if POOL is specified.

Symbolic names for the AID values are available for the supported languages in the language-specific DFHAID copybooks.

ALARMSTATUS(cvda)
returns a value that indicates whether the received data sounded the alarm. The relevant CVDA values are:

- ALARM
- NOALARM

COLUMNS(fullword binary data-area)
returns the number of columns in the screen image.

CONVID(8-character data-value)
specifies the ID of the conversation to use. The conversation must be owned by the task issuing the command.

ENDSTATUS(cvda)
returns a value that indicates the ending status for the received data. The relevant CVDA values are:

Value
Meaning

CD
'Change direction' received.

EB
'End bracket' received.

LIC
'Last in chain' received.

For more details of ending status and how additional data is handled, see "Ending status" on page 134.

ESCAPE(1-character data-value)
for send data in key stroke format, specifies the escape character used to indicate character combinations representing special keys. You can use any value in the range X'40' through X'FE'. The default escape character is & (X'50').

FIELDS(fullword binary data-area)
returns the number of fields in the screen image.

FROM(data-value)
specifies the data to send to the back-end application. Its length is specified by the FROMFLENGTH option. For send data in screen-image format, if the length is more than the screen image, the additional data is ignored; if it is less, the data is the first part of the screen image, and the last part of the screen image is not changed.

FROMCURSOR(fullword binary data-value)
for send data in screen-image format, specifies the position of the cursor, expressed as an offset from the start of the screen image; offset zero is the top left-hand corner of the screen. If FROMCURSOR is not specified, the cursor remains where it was positioned by the last inbound data.

FROMFLENGTH(fullword binary data-value)
specifies the length of the data to send; that is, the length of the data area identified by the FROM option. It must not be zero or more than the maximum length allowed for the pool.

INTO(data-area)
specifies the data area in which the received data is to be returned. The length of the area is specified by the MAXFLENGTH option, and the actual length of data written into the area is returned by the TOFLENGTH option.

KEYSTROKES
specifies that the data to send is a sequence of key strokes (see "Data formats" on page 132).

LINES(fullword binary data-area)
returns the number of lines in the screen image.

MAXFLENGTH(fullword binary data-value)
specifies the maximum amount of data that can be returned; that is, the length of the data area identified by the INTO option. It must not be more than the maximum length allowed for the pool.

POOL(8-character data-value)
specifies the name of the pool containing the target for the conversation. Specifying POOL means that the conversation is a temporary one, that exists only for the duration of the FEPI CONVERSE. You must also specify the KEYSTROKES option. If the length of the data area identified by the INTO option is less than the size of the screen image, the additional data is discarded.

RESPSTATUS(cvda)
returns a value that indicates the type of response that is required at the back-end system. The relevant CVDA values are:

Value
Meaning

DEFRESP1
Definite response 1 required.

DEFRESP2
Definite response 2 required.

DEFRESP3
Definite response 1 and definite response 2 required.

NONE
No response required.

TARGET(8-character data-value)
specifies the name of the target. TARGET can be omitted if there is only one target in the pool or if all targets are suitable for the intended conversation.

TIMEOUT(fullword binary data-value)
specifies the maximum time in seconds that the command is to wait for the requested data to begin to arrive. If TIMEOUT is not specified or the specified time is zero, the command is not timed out.

TOCURSOR(fullword binary data-area)
returns the position of the cursor in the received screen image, expressed as an offset from the start of the screen image; offset zero is the top left-hand corner of the screen.

TOFLENGTH(fullword binary data-area)
returns the actual length of data received in the data area identified by the INTO option.

Note: On a FEPI CONVERSE FORMATTED command, if MAXFLENGTH is less than the presentation space size, TOFLENGTH returns the value defined in MAXFLENGTH. If MAXFLENGTH is greater than the presentation space size, TOFLENGTH returns the presentation space size.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

30
Pool name unknown.

31
Pool name out of service.

32
TARGET name unknown.

33
TARGET name out of service.

34
TARGET name required but not specified.

35
POOL name is unsuitable for temporary conversations. It has CONTENTION(LOSE) or it has INITIALDATA(INBOUND) and no begin-session handler.

36
No suitable session available and in service.

40
FROMLENGTH value negative, zero, or more than the maximum allowed for the current pool.

41
ESCAPE value not valid.

50
Inbound data with ‘begin bracket’ to be received.

51
AID value not valid.

52
Cursor position not valid.

53
Character values in send data not valid.

54
Attribute positions or values in send data not valid.

55
Key stroke escape sequence in send data is not valid.

56
Field validation (mandatory fill, mandatory enter, trigger) failed.

57
Input inhibited.

58
VTAM SEND failed.

59
DBCS data rules violated.

60
MAXFLENGTH value negative, zero, or more than the maximum allowed for the current pool.

71
VTAM RECEIVE failed.

72
RECEIVE FORMATTED processing found invalid, or unexpected data while interpreting the 3270 data stream for a WRITE, ERASE/WRITE ALTERNATE, or WRITE STRUCTURED FIELD command code.

210
Command not allowed for SLU P mode.

212
Conversation has wrong data format.

213
Command timed out.

215
Session lost.

216
Error occurred on previous FEPI SEND.

220
FEPI CONVERSE not allowed at this point in the conversation.

221
Data cannot be received because no AID or final attention key stroke specified.

224
Only FEPI ISSUE or FEPI FREE commands allowed at this point in the conversation.

230
SNA CLEAR command received. For an explanation of this SNA command, see Systems Network Architecture Formats (GA27-3136).

231
SNA CANCEL command received. For an explanation of this SNA command, see Systems Network Architecture Formats (GA27-3136).

232
SNA CHASE command received. For an explanation of this SNA command, see Systems Network Architecture Formats (GA27-3136).

233
Exception response received.

234
Exception request received.

240
Conversation ID not owned by this task.

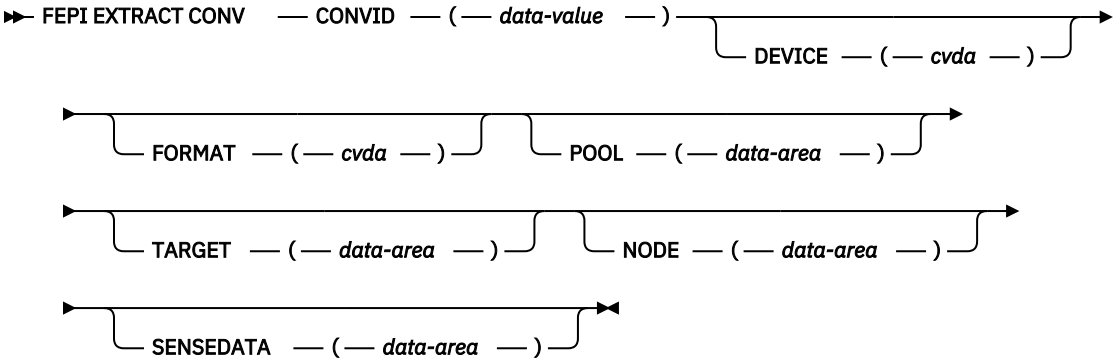
241
TIMEOUT value negative or not valid.

FEPI EXTRACT CONV

FEPI EXTRACT CONV gets general information about a conversation.

Syntax

FEPI EXTRACT CONV



Notes:

Options

CONVID(8-character data-value)
specifies the ID of the conversation for which information is wanted. The conversation must be owned by the task issuing the command.

DEVICE(cvda)
returns a value that identifies the mode of conversation and the type of device. The relevant CVDA values are:

Value
Meaning

T3278M2
SLU2 mode, 3278 Model 2

T3278M3
SLU2 mode, 3278 Model 3

T3278M4
SLU2 mode, 3278 Model 4

T3278M5
SLU2 mode, 3278 Model 5

T3279M2
SLU2 mode, 3279 Model 2B

T3279M3
SLU2 mode, 3279 Model 3B

T3279M4
SLU2 mode, 3279 Model 4B

T3279M5
SLU2 mode, 3279 Model 5B

TPS55M2
SLU2 mode, PS/55, 24 lines

TPS55M3
SLU2 mode, PS/55, 32 lines

TPS55M4
SLU2 mode, PS/55, 43 lines

LUP
SLU P mode, all cases.

FORMAT(cvda)
in SLU2 mode, returns a value that identifies the data mode. The relevant CVDA values are:

- DATASTREAM
- FORMATTED

NODE(8-character data-area)
returns the node name.

POOL(8-character data-area)
returns the pool name.

SENSEDATA(fullword binary data-area)
returns the sense data associated with the last FEPI SEND, FEPI RECEIVE, or FEPI CONVERSE command for the conversation. If there is no sense data, zero is returned.

TARGET(8-character data-area)
returns the target name.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

215
Session lost.

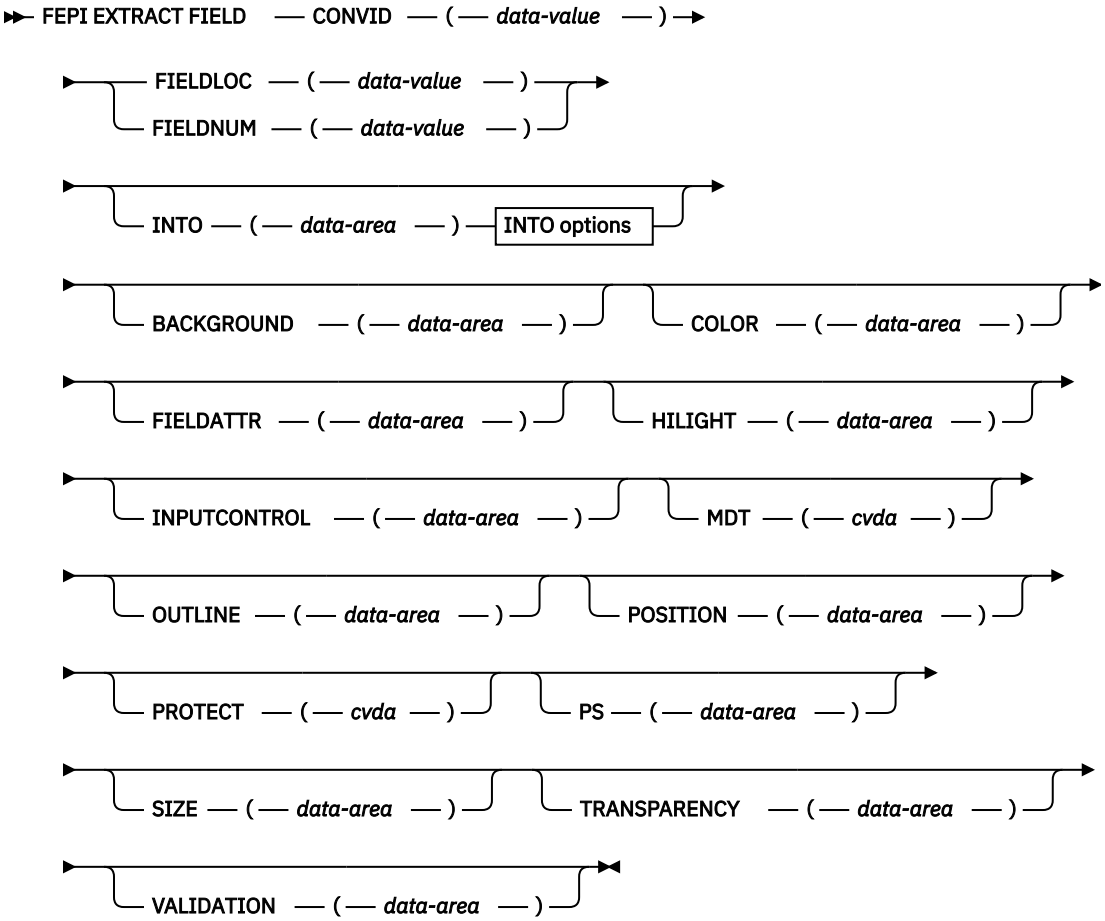
240
Conversation ID not owned by this task.

FEPI EXTRACT FIELD

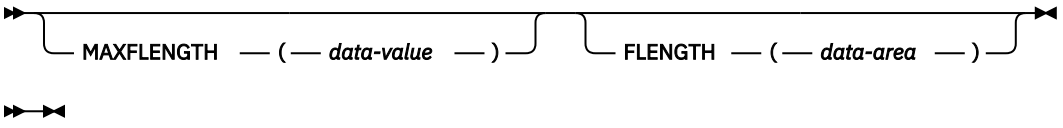
The command is for SLU2 mode only, and for formatted data only.FEPI EXTRACT FIELD gets information about a field in the current character buffer of the simulated terminal. It can be issued at any point in the conversation. More than one FEPI EXTRACT FIELD command can be issued for a given field.For information about field attributes and their values see IBM 3270 Data Stream Programmers Reference. Symbolic names for the various attribute values are available in the DFHBMSCA copybook.

Syntax

FEPI EXTRACT FIELD



INTO options



Notes:

Options

BACKGROUND(1-character data-area)
returns the background color attribute of the field.

COLOR(1-character data-area)
returns the foreground color attribute of the field.

CONVID(8-character data-value)
specifies the ID of the conversation for which information is wanted. The conversation must be owned by the task issuing the command.

FIELDATTR(1-character data-area)
returns the 3270 field attribute of the field.

FIELDLOC(fullword binary data-value)
specifies the location of the required field expressed as an offset from the start of the screen image; offset zero is the top left-hand corner of the screen. The location can refer to any character position in the field, including its attribute byte.

FIELDNUM(fullword binary data-value)
specifies the location of the required field expressed as a field number counting from the top left-hand corner of the screen. The first field is number 1, and starts at the top-left hand corner of the screen, whether or not there is an attribute in that position. The last field ends at the bottom right-hand corner of the screen, and does not wrap back to the top.

FLENGTH(fullword binary data-area)
returns the actual length of data received in the data area identified by the INTO option.

HILIGHT(1-character data-area)
returns the extended highlighting attribute of the field.

INPUTCONTROL(1-character data-area)
returns the DBCS input control attribute of the field.

INTO(data-area)
specifies the data area in which the data in the field is to be returned. The length of the area is specified by the MAXFLENGTH option, and the actual length of data written into the area is returned by the FLENGTH option.

MAXFLENGTH(fullword binary data-value)
specifies the maximum amount of data that can be returned; that is, the length of the data area identified by the INTO option. It must not be more than the maximum length allowed for the pool.

MDT(cvda)
returns a value that identifies the state of the modified data tag for the field. The relevant CVDA values are:

- NOMDT
- MDT

OUTLINE(1-character data-area)
returns the field outlining attribute of the field.

POSITION(fullword binary data-area)
returns the position of the field expressed as the offset of the first data byte from the start of the screen image; offset zero is the top left-hand corner of the screen.

PROTECT(cvda)
returns a value that indicates whether or not the field is protected. The relevant CVDA values are:

- UNPROTECTED
- PROTECTED

PS(1-character data-area)
returns the character set attribute of the field.

SIZE(fullword binary data-area)
returns the size of the field on the screen, excluding the field attribute byte, expressed as a number of bytes.

TRANSPARENCY(1-character data-area)
returns the transparency attribute of the field.

VALIDATION(1-character data-area)
returns the field validation attribute of the field.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

60
MAXFLENGTH value negative, zero, or more than the maximum allowed for the current pool.

70
FIELDLOC or FIELDNUM value negative or not valid.

210
Command not allowed for SLU P mode.

212
Conversation has wrong data format.

224
Only FEPI ISSUE or FEPI FREE commands allowed at this point in the conversation.

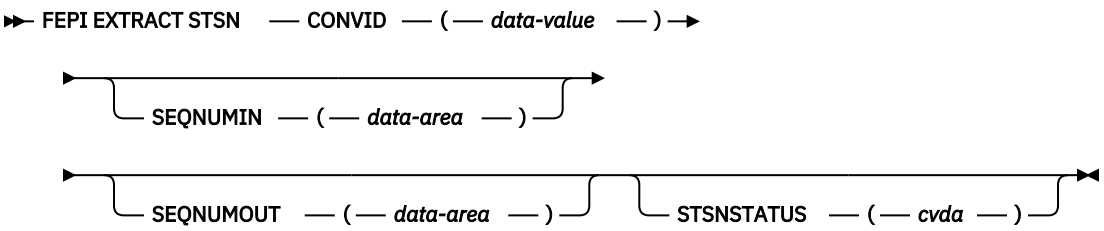
240
Conversation ID not owned by this task.

FEPI EXTRACT STSN

The command is for SLU P mode only. FEPI EXTRACT STSN gets sequence number status information for a conversation.

Syntax

FEPI EXTRACT STSN



Notes:

Options

CONVID(8-character data-value)
specifies the ID of the conversation for which information is wanted. The conversation must be owned by the task issuing the command.

SEQNUMIN(fullword binary data-area)
returns the current sequence number for inbound data.

SEQNUMOUT(fullword binary data-area)
returns the current sequence number for outbound data.

STSNSTATUS(cvda)
returns the current sequence-number set and test status. The relevant CVDA values are:

Value
Meaning

NOSTSN
No ‘set’ or ‘test and set’ issued.

STSNSET
‘Set’ sequence number issued.

STSNTEST
‘Test and set’ sequence number issued.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

211
Command not allowed for SLU2 mode.

240
Conversation ID not owned by this task.

FEPI FREE

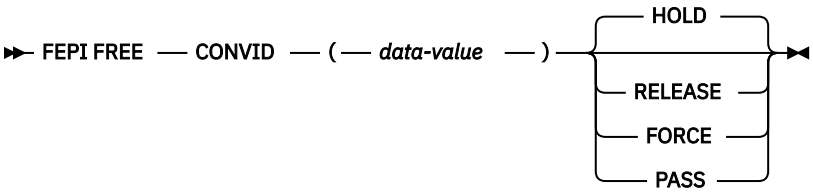
FEPI FREE ends a task's use and ownership of a conversation. The conversation may be ended completely, or may be passed to another task.

The action depends on the processing state of the conversation:

- Begin session handler
- STSN handler
- Access program
- End session handler
- Unsolicited-data handler.

Syntax

FEPI FREE



Notes:

Options

CONVID(8-character data-value)
specifies the ID of the conversation to free. The conversation must be owned by the task issuing the command.

FORCE
tells FEPI what action to take. For all processing states of the conversation, FORCE instructs FEPI to end the conversation unconditionally, and to take the connection that it was using out of service immediately and, if possible, reset it.

HOLD
tells FEPI what action to take.

For the access program and the unsolicited-data handler, HOLD instructs FEPI to end the conversation and to retain the session for use by another conversation. However, this is subject to any end-session processing.

For the begin-session handler and the STSN handler, HOLD tells FEPI that begin-session or STSN processing has ended, and that the conversation is ready for the next processing state.

For the end-session handler, HOLD tells FEPI that end-session processing has ended, and instructs FEPI to end the conversation and to retain the session for use by another conversation. (If CICS shutdown is in progress, HOLD is the same as RELEASE.)

PASS
tells FEPI what action to take. For all the processing states of the conversation, PASS specifies that the task is relinquishing ownership of the conversation so that another task can acquire it. There is no change in the processing state of the conversation. (PASS is not allowed if CICS shutdown is in progress.)

RELEASE
tells FEPI what action to take.

For the access program and the unsolicited-data handler, RELEASE instructs FEPI to end the conversation, and to release and unbind the session that it was using, thereby forcing a new session to be started next time the connection is used. However, this is subject to any end-session processing.

For the begin-session handler and the STSN handler, RELEASE tells FEPI that begin-session or STSN processing has ended, and instructs FEPI to end the conversation without proceeding to the next processing state, and to release and unbind the session that it was using, thereby forcing a new session to be started next time the connection is used. However, this is subject to any end-session processing.

For the end-session handler, RELEASE tells FEPI that end-session processing has ended, and instructs FEPI to end the conversation, and to release and unbind the session that it was using, thereby forcing a new session to be started next time the connection is used.

Note that, under normal circumstances, after a FEPI FREE RELEASE command has been issued the session does not remain in RELEASED state, because FEPI automatically tries to reacquire the session. However, if a FEPI SET CONNECTION ACQSTATUS(RELEASED) command is issued before the FREE RELEASE, the session remains in RELEASED state.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

214
CICS shutting down, conversation should be ended.

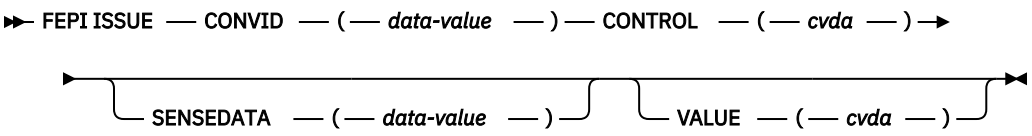
240
Conversation ID not owned by this task.

FEPI ISSUE

FEPI ISSUE sends control data, such as standard responses and sense data, to the target system. The command completes as soon as the corresponding z/OS Communications Server VTAM SEND has been accepted.

Syntax

FEPI ISSUE



Notes:

Options

CONTROL(cvda)
specifies what type of control data to send. The relevant CVDA values depend upon the data type and the mode of the conversation:

`For all modes:`

Value
Meaning

NORMALRESP
Send a normal response, as specified by the VALUE option.

EXCEPTRESP
Send an exception response, as specified by the VALUE option, and with the sense data specified by the SENSEDATA option.

ATTENTION
Send an attention (SNA ‘signal' command X'00010000').

LUSTAT
Send an SNA ‘LUSTAT' command with the sense data specified by the SENSEDATA option.

`For data stream only:`

Value
Meaning

CANCEL
Send an SNA ‘cancel' command.

`For SLU P mode only:`

Value
Meaning

STSN
Send an SNA ‘set and test sequence number' command.

RTR
Send an SNA ‘ready to receive' command.

CONVID(8-character data-value)
specifies the ID of the conversation to use. The conversation must be owned by the task issuing the command.

SENSEDATA(fullword binary data-value)
specifies sense data to send to the target when the CONTROL is LUSTAT or EXCEPTRESP.

VALUE(cvda)
specifies the response type associated with the control data. The relevant CVDA values are determined by what is specified for the CONTROL option:

`For EXCEPTRESP and NORMALRESP:`

Value
Meaning

DEFRESP1OR2
Send definite response 1 or 2 as required.

DEFRESP1
Send definite response 1.

DEFRESP2
Send definite response 2.

DEFRESP3
Send definite response 1 and definite response 2.

For STSN:

Value
Meaning

POSITIVE
Send STSN positive response.

NEGATIVE
Send STSN negative response.

INVALID
Send STSN response not valid (this unbinds the session).

RESET
Send STSN reset response (this unbinds the session).

DEFRESP2
Send definite response 2.

DEFRESP3
Send definite response 1 and definite response 2.

For other controls:

None; the VALUE option is not used with the other controls.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

80
CONTROL value not valid.

81
VALUE value not valid: omitted when required, specified when not required, or unsuitable for the specified CONTROL value.

82
SENSEDATA value omitted when required or specified when not required.

90
Definite response type did not match what was required.

91
Only NORMALRESP or EXCEPTRESP are allowed at this point in the conversation.

92
Response to STSN SET was not positive.

93
Only FEPI ISSUE CONTROL(STSN) allowed at this point in the conversation.

94
Only FEPI ISSUE CONTROL(STSN) or FEPI ISSUE CONTROL(NORMALRESP) allowed at this point in the conversation.

95
CONTROL value not allowed at this point in the conversation.

211
Option not allowed for SLU2 mode.

215
Session lost.

216
Error occurred on previous FEPI SEND.

230
SNA CLEAR command received.

231
SNA CANCEL command received.

232
SNA CHASE command received.

233
Exception response received.

234
Exception request received.

240
Conversation ID not owned by this task.

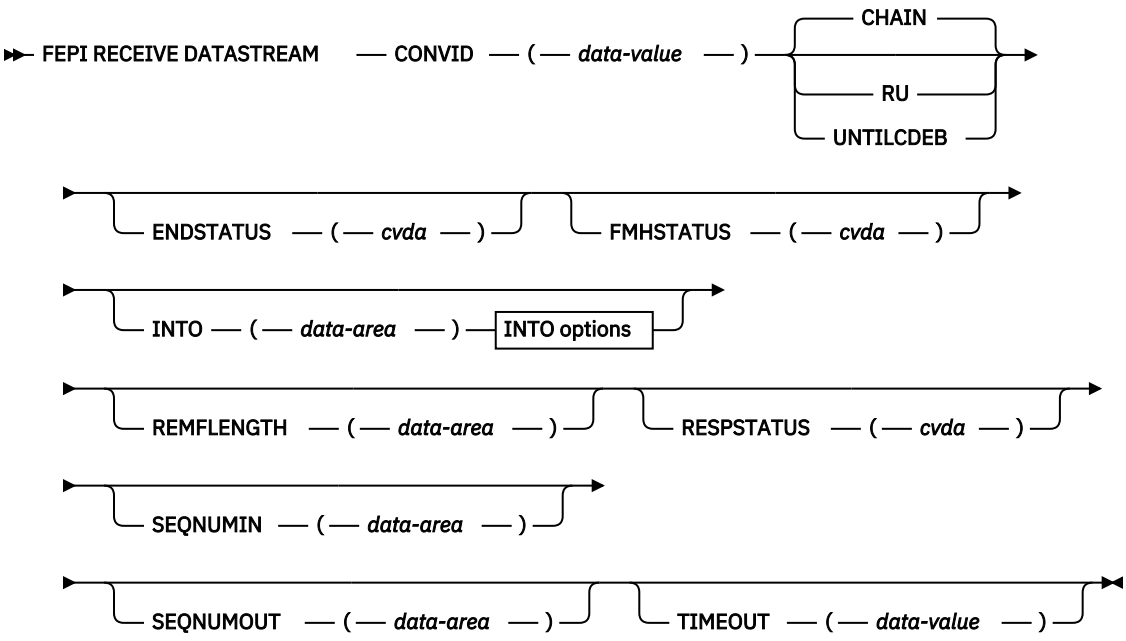
FEPI RECEIVE DATASTREAM

FEPI RECEIVE DATASTREAM receives data from a target and places the received data stream into the application's data area.

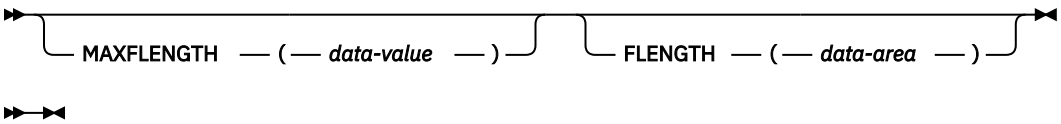
Full details about the data are given in “Data formats” on page 132. By default, **FEPI RECEIVE DATASTREAM** completes when a whole chain of data has been received. A time limit can be set for this command. For more details of ending conditions, see “Ending status” on page 134.

Syntax

FEPI RECEIVE DATASTREAM



INTO options



Notes:

Options

CHAIN
specifies that the command should complete when a whole chain has been received.

CONVID(8-character data-value)
specifies the ID of the conversation to use. The conversation must be owned by the task issuing the command.

ENDSTATUS(cvda)
returns a value that indicates the ending status for the received data. The relevant CVDA values are:

Value
Meaning

CD
"Change direction" received.

EB
"End bracket" received.

LIC
"Last in chain" received.

RU
RU received.

MORE
The data area identified by the INTO option was too small to receive all the requested data.

For more details of ending status and how additional data is handled, see "Ending status" on page 134.

FLENGTH(fullword binary data-area)
returns the actual length of data received in the data area identified by the INTO option.

FMHSTATUS(cvda)
returns a value that indicates whether the received data contains a function management header. The relevant CVDA values are:

- FMH
- NOFMH

INTO(data-area)
specifies the data area in which the received data is to be returned. The length of the area is specified by the MAXFLENGTH option, and the actual length of data written into the area is returned by the FLENGTH option.

MAXFLENGTH(fullword binary data-value)
specifies the maximum amount of data that can be returned; that is, the length of the data area identified by the INTO option. It must not be more than the maximum length allowed for the pool.

REMFLENGTH(fullword binary data-area)
returns the length, if known, of data remaining after filling the data area identified by the INTO option.

RESPSTATUS(cvda)
returns a value that indicates the type of response that is required at the back-end system. The relevant CVDA values are:

Value
Meaning

DEFRESP1
Definite response 1 required.

DEFRESP2
Definite response 2 required.

DEFRESP3
Definite response 1 and definite response 2 required.

NONE
No response required.

RU
specifies that the command should complete when a request unit has been received.

SEQNUMIN(fullword binary data-area)
in SLU P mode, returns the current sequence number for inbound data, as at the completion of the command. (SEQNUMIN has no significance in SLU2 mode.)

SEQNUMOUT(fullword binary data-area)
in SLU P mode, returns the current sequence number for outbound data, as at the completion of the command. (SEQNUMOUT has no significance in SLU2 mode.)

TIMEOUT(fullword binary data-value)
specifies the maximum time in seconds that the command is to wait for the requested data to begin to arrive. If TIMEOUT is not specified or the specified time is zero, the command is not timed out.

UNTILCDEB
specifies that the command should complete when "change direction" or "end bracket" is received.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

60
MAXFLENGTH value negative or more than maximum allowed for the current pool.

71
z/OS Communications Server VTAM RECEIVE failed.

212
Conversation has wrong data format.

215
Session lost.

216
Error occurred on previous FEPI SEND.

221
FEPI RECEIVE not allowed at this point in the conversation.

224
Only FEPI ISSUE or FEPI FREE commands allowed at this point in the conversation.

230
SNA CLEAR command received.

231
SNA CANCEL command received.

232
SNA CHASE command received.

233
Exception response received.

234
Exception request received.

240
Conversation ID not owned by this task.

241
TIMEOUT value negative or not valid.

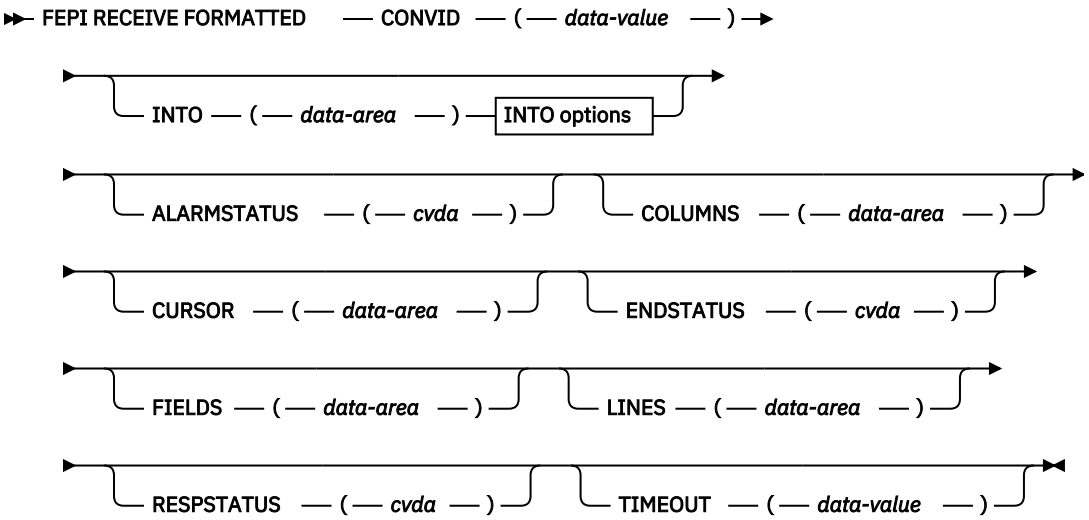
FEPI RECEIVE FORMATTED

This command is for SLU2 mode only. FEPI RECEIVE FORMATTED receives data from a target. The data received into the application's data area is a screen image.

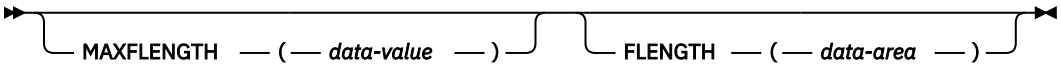
Full details about the data are given in "Data formats" on page 132. FEPI RECEIVE FORMATTED completes after receiving the inbound data with "last in chain", 'end bracket' or "change direction" indicated. A time limit can be set for this command. For more details of ending conditions, see "Ending status" on page 134.

Syntax

FEPI RECEIVE FORMATTED



INTO options



Notes:

Options

ALARMSTATUS(cvda)
returns a value that indicates whether the received data sounded the alarm. The relevant CVDA values are:

- ALARM
- NOALARM

COLUMNS(fullword binary data-area)
returns the number of columns in the screen image.

CONVID(8-character data-value)
specifies the ID of the conversation to use. The conversation must be owned by the task issuing the command.

CURSOR(fullword binary data-area)
returns the position of the cursor in the received screen image, expressed as an offset from the start of the screen image; offset zero is the top left-hand corner of the screen.

ENDSTATUS(cvda)
returns a value that indicates the ending status for the received data. The relevant CVDA values are:

Value
Meaning

CD
"Change direction" received.

EB
"End bracket" received.

LIC
"Last in chain" received.

For more details of ending status and how additional data is handled, see “Ending status” on page 134.

FIELDS(fullword binary data-area)
returns the number of fields in the screen image.

FLENGTH(fullword binary data-area)
returns the actual length of data received in the data area identified by the INTO option.

INTO(data-area)
specifies the data area in which the received data is to be returned. The length of the area is specified by the MAXFLENGTH option, and the actual length of data written into the area is returned by the FLENGTH option.

LINES(fullword binary data-area)
returns the number of lines in the screen image.

MAXFLENGTH(fullword binary data-value)
specifies the maximum amount of data that can be returned; that is, the length of the data area identified by the INTO option. It must not be more than the maximum length allowed for the pool.

RESPSTATUS(cvda)
returns a value that indicates the type of response that is required at the back-end system. The relevant CVDA values are:

Value
Meaning

DEFRESP1
Definite response 1 required.

DEFRESP2
Definite response 2 required.

DEFRESP3
Definite response 1 and definite response 2 required.

NONE
No response required.

TIMEOUT(fullword binary data-value)
specifies the maximum time in seconds that the command is to wait for the requested data to begin to arrive. If TIMEOUT is not specified or the specified time is zero, the command is not timed out.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

60
MAXFLENGTH value negative or more than maximum allowed for the current pool.

71
z/OS Communications Server VTAM RECEIVE failed.

72
RECEIVE FORMATTED processing found invalid, or unexpected data while interpreting the 3270 data stream for a WRITE, ERASE/WRITE, ERASE/WRITE ALTERNATE, or WRITE STRUCTURED FIELD command code.

210
Command not allowed for SLU P mode.

212
Conversation has wrong data format.

213
Command timed out.

215
Session lost.

216
Error occurred on previous FEPI SEND.

221
FEPI RECEIVE not allowed at this point in the conversation.

224
Only FEPI ISSUE or FEPI FREE commands allowed at this point in the conversation.

230
SNA CLEAR command received.

231
SNA CANCEL command received.

232
SNA CHASE command received.

233
Exception response received.

234
Exception request received.

240
Conversation ID not owned by this task.

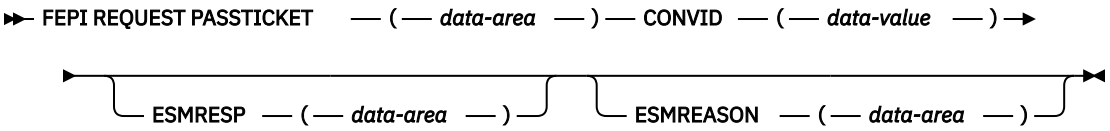
241
TIMEOUT value negative or not valid.

FEPI REQUEST PASSTICKET

Request an external security manager to build a PassTicket for a FEPI application.

Syntax

FEPI REQUEST PASSTICKET



Notes:

Description

The **FEPI REQUEST PASSTICKET** command requests an external security manager (ESM), such as RACF, to build a PassTicket. The PassTicket is a password substitute that your application can use to sign on to the back-end system associated with the conversation.

The PassTicket that the external security manager generates is for the user ID associated with the task that issues the **FEPI REQUEST PASSTICKET** command. Use the **EXEC CICS ASSIGN** command with the USERID option to identify the user ID that is associated with the task.

Note: A request for a PassTicket succeeds if a user ID is revoked, but an attempt to sign on with that user ID and PassTicket fails.

A PassTicket must be used within 10 minutes of being generated. If the PassTicket times out (for example, because of a session failure), your application must generate another PassTicket before it attempts to sign on again. Repeated failed sign-on attempts with PassTickets can result in the user ID being revoked.

Your program uses the generated PassTicket and the user ID to sign on to the back-end system as if it was sending a password and user ID, as in the following example:

```
EXEC CICS FEPI SEND FORMATTED
                   CONVID(convid) FROM(CESN userid PassTicket)
                   FROMLENGTH(length_of_data)
```

Your application must provide the sign-on processing, because CICS cannot know either the type of back-end system, or the back-end program that is used for sign-on processing.

PassTickets are not displayed when the CICS execution diagnostic facility (EDF) is used.

Before you use PassTickets, ensure that the system clocks for the target system and the originating system are synchronized to within the valid time range. You must also define a Secure Signon key for each target system. For information about the requirements for using PassTickets, see Generating and using PassTickets for secure sign-on.

Options

CONVID(8-character data-value)
specifies the ID of the conversation with the back-end system for which a PassTicket is required.

ESMREASON(fullword binary data-area)
returns the reason code from the ESM.

ESMRESP(fullword binary data-area)
returns the response code from the ESM. For an explanation of the response and reason codes that are returned by RACF, see z/OS Security Server RACF Messages and Codes.

PASSTICKET(8-character data-area)
returns the PassTicket generated by the ESM.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

240
Conversation ID not owned by this task.

248
The external security manager does not authorize a request to generate a PassTicket for this region.

249
User not logged on.

250
PassTicket not built successfully.

251
CICS ESM interface not initialized.

252
Unknown return code in ESMRESP from the ESM.

253
Unrecognized response from CICS security modules.

254
Function unavailable.

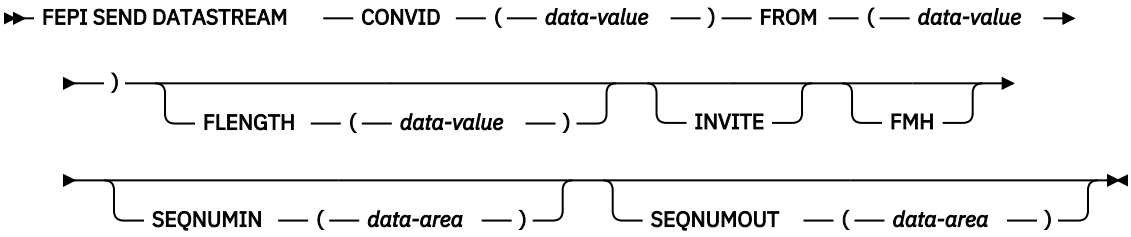
FEPI SEND DATASTREAM

FEPI SEND DATASTREAM sends application data to a target. The data supplied by the application must be currently valid data stream appropriate to the mode of the conversation (SLU2 or SLU P).

Full details about the data are given in "Data formats" on page 132. The command completes as soon as the (first) z/OS Communications Server VTAM SEND has been accepted.

Syntax

FEPI SEND DATASTREAM



Notes:

Options

CONVID(8-character data-value)
specifies the ID of the conversation to use. The conversation must be owned by the task issuing the command.

FMH
indicates that the data to send includes a function management header.

FLENGTH(fullword binary data-value)
specifies the length of the data to send; that is, the length of the data area identified by the FROM option. It must not be zero or more than the maximum length allowed for the pool.

FROM(data-value)
specifies the data to send to the back-end application. Its length is specified by the FLENGTH option.

INVITE
requests FEPI to send "last in chain" and "change direction" at the end of the data. This indicates that the data is complete, and that inbound data is expected next.

SEQNUMIN(fullword binary data-area)
in SLU P mode, returns the current sequence number for inbound data, as at the completion of the command. (SEQNUMIN has no significance in SLU2 mode.)

SEQNUMOUT(fullword binary data-area)
in SLU P mode, returns the current sequence number for outbound data, as at the completion of the command. (SEQNUMOUT has no significance in SLU2 mode.)

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

40
FLENGTH value negative or more than maximum allowed for the current pool.

50
Inbound data with "begin bracket" to be received.

58
z/OS Communications Server VTAM SEND failed.

212
Conversation has wrong data format.

215
Session lost.

216
Error occurred on previous FEPI SEND.

220
FEPI SEND not allowed at this point in the conversation.

224
Only FEPI ISSUE or FEPI FREE commands allowed at this point in the conversation.

230
SNA CLEAR command received.

231
SNA CANCEL command received.

232
SNA CHASE command received.

233
Exception response received.

234
Exception request received.

240
Conversation ID not owned by this task.

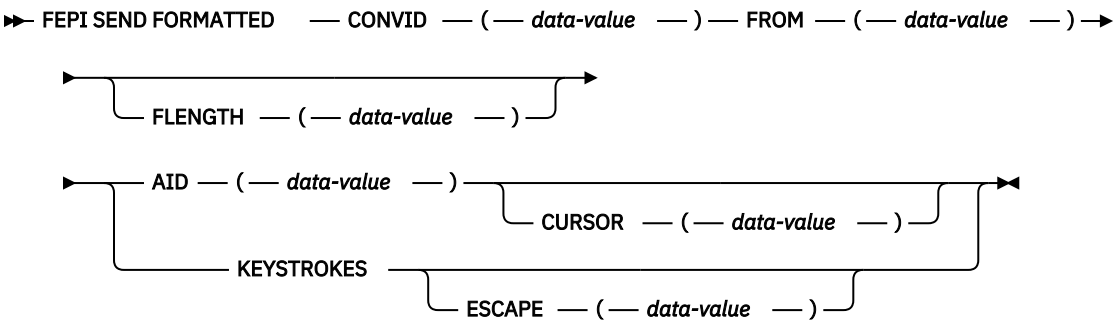
FEPI SEND FORMATTED

This command is for SLU2 mode only. FEPI SEND FORMATTED sends application data to a target. The data supplied by the application must be formatted data, as key strokes or as a screen image.

Full details about the data are given in "Data formats" on page 132. The command completes as soon as the (first) z/OS Communications Server VTAM SEND has been accepted.

Syntax

FEPI SEND FORMATTED



Notes:

Options

AID(1-character data-value)
specifies the attention identifier value to send with the data. Specifying AID also indicates that the data to send is in screen-image format, as described in "Data formats" on page 132. A value of null (`X'00'`) may be specified to indicate that no attention is to be sent, and that a further FEPI SEND is to follow.

Symbolic names for the AID values are available for the supported languages in the language-specific DFHAID copybooks.

CONVID(8-character data-value)
specifies the ID of the conversation to use. The conversation must be owned by the task issuing the command.

CURSOR(fullword binary data-value)
for send data in screen-image format, specifies the position of the cursor, expressed as an offset from the start of the screen image; offset zero is the top left-hand corner of the screen. If CURSOR is not specified, the cursor remains where it was positioned by the last inbound data.

ESCAPE(1-character data-value)
for send data in key stroke format, specifies the escape character used to indicate character combinations representing special keys. You can use any value in the range `X'40'` through `X'FE'`. The default escape character is & (`X'50'`).

FLENGTH(fullword binary data-value)
specifies the length of the data to send; that is, the length of the data area identified by the FROM option. It must not be zero or more than the maximum length allowed for the pool.

FROM(data-value)
specifies the data to send to the back-end application. Its length is specified by the FLENGTH option. For send data in screen-image format, if the length is more than the screen image, the additional data is ignored; if it is less, the data is the first part of the screen image, and the last part of the screen image is not changed.

KEYSTROKES
specifies that the data to send is in key stroke format, a sequence of key strokes, as described in "Data formats" on page 132.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

40
FLENGTH value negative or more than maximum allowed for the current pool.

41
ESCAPE value not valid.

50
Inbound data with "begin bracket" to be received.

51
AID value not valid.

52
Cursor position not valid.

53
Character values in send data not valid.

54
Attribute positions or values in send data not valid.

55
Key stroke escape sequence in send data not valid.

56
Field validation (mandatory fill, mandatory error, trigger) failed.

57
Input inhibited.

58
z/OS Communications Server VTAM SEND failed.

59
DBCS data rules violated.

210
Command not allowed for SLU P mode.

212
Conversation has wrong data format.

215
Session lost.

220
FEPI SEND not allowed at this point in the conversation.

224
Only FEPI ISSUE or FEPI FREE commands allowed at this point in the conversation.

230
SNA CLEAR command received.

231
SNA CANCEL command received.

232
SNA CHASE command received.

233
Exception response received.

234
Exception request received.

240
Conversation ID not owned by this task.

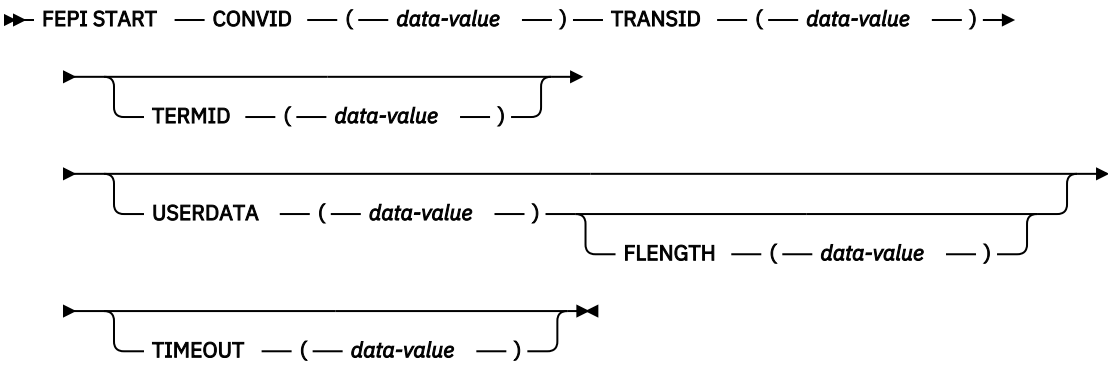
FEPI START

FEPI START is used to relinquish control of a conversation and to specify a new transaction to be started when the next inbound data arrives.

A maximum of 128 characters of user data can be passed to the transaction as part of the start data, as described in “Start data” on page 130 below.

Syntax

FEPI START



Notes:

Options

CONVID(8-character data-value)
specifies the ID of the conversation to suspend. The conversation must be owned by the task issuing the command.

FLENGTH(fullword binary data-value)
specifies the length of the optional user data to pass to the transaction that is started; that is, the length of the data area identified by the USERDATA option. The FLENGTH value must not be greater than 128.

TERMID(4-character data-value)
specifies the name of the terminal, if any, to be associated with the transaction that is started.

TIMEOUT(fullword binary data-value)
specifies the maximum time in seconds that FEPI is to wait for inbound data to begin to arrive before starting the transaction. If TIMEOUT is not specified or the specified time is zero, the command is not timed out.

TRANSID(4-character data-value)
specifies the name of the transaction that is to be started when the next inbound data arrives.

USERDATA(data-value)
specifies optional user data to pass to the transaction that is started, in addition to control information passed by FEPI. Its length is specified by the FLENGTH option.

Conditions

The INVREQ condition can have the following RESP2 values:

RESP2
Meaning

61
FLENGTH value negative or too large.

62
TRANSID name not valid.

63
TERMID name not valid.

214
CICS shutting down, conversation should be ended.

215
Session lost.

216
Error occurred on previous FEPI SEND.

223
FEPI START not allowed at this point in the conversation.

224
Only FEPI ISSUE or FEPI FREE commands allowed at this point in the conversation.

230
SNA CLEAR command received.

231
SNA CANCEL command received.

232
SNA CHASE command received.

233
Exception response received.

234
Exception request received.

240
Conversation ID not owned by this task.

241
TIMEOUT value negative or not valid.

Start data

For various events, FEPI invokes a transaction, as a CICS started task, to handle the event. This might be in response to FEPI START, or to handle STSN, begin-session, end-session, or unsolicited-data.

The transactions have a start code of 'SZ', as can be determined with the **EXEC CICS ASSIGN** command. FEPI provides start data which describes the event, and the conversation which is to be used to handle it. All of this data must be retrieved by the transaction using EXEC CICS RETRIEVE. The transaction can then gain access to the conversation identified in the data by using FEPI ALLOCATE PASSCONVID.

The structure for start data is shown in the table; the copy books DFHSZAPA, DFHSZAPO, DFHSZAPC, and DFHSZAPP (according to your programming language) provide declarations for this structure.

Name	Description
DATATYPE	Fullword binary data-area
EVENTTYPE	CVDA
EVENTVALUE	CVDA
EVENTDATA	8-character data-area
spare	4-character data-area

Name	Description
POOL	8-character data-area
TARGET	8-character data-area
NODE	8-character data-area
CONVID	8-character data-area
DEVICE	CVDA
FORMAT	CVDA
spare	8-character data-area
FLENGTH	Fullword binary data-area
USERDATA	128-character data area.

Fields

CONVID(8-character data-area)
the ID of the conversation for which the event occurred (this is the CONVID that should be used in FEPI ALLOCATE PASSCONVID).

DATATYPE(fullword binary data-area)
Type and structure of data. Value is 1 for FEPI start data.

DEVICE(cvda)
the device type of conversation for which the event occurred, values being as for FEPI EXTRACT CONV.

EVENTDATA(8-character data-area)
always nulls.

EVENTTYPE(cvda)
Indicates why the transaction was started. Values are:

Value	Event
BEGINSESSION	Begin-session to be handled
DATA	Inbound data arrived, following a FEPI START command
FREE	End-session transaction started to handle end of conversation as a result of a FEPI FREE request
SESSIONLOST	Active session lost while waiting for inbound data to arrive following a FEPI START command
STSN	Set and test sequence number (STSN) to be handled
TIMEOUT	Timed out waiting for inbound data to arrive following a FEPI START command
UNSOLDATA	Inbound data arrived outside a conversation.

EVENTVALUE(cvda)
A CVDA giving further information about event types FREE and RELEASE.

Values for FREE:

FORCE	A FEPI FREE FORCE command was issued.
HOLD	A FEPI FREE HOLD command was issued.
RELEASE	A FEPI FREE RELEASE command was issued.

SHUTDOWN	CICS is shutting down.
TASK	Conversation being freed by end-of-task.

The EVENTVALUE value is zero for all other event types.

FLENGTH(fullword binary data-area)
the length of the data in USERDATA.

FORMAT(cvda)
the data format of conversation for which the event occurred, values being as for FEPI EXTRACT CONV.

NODE(8-character data-area)
the name of the node for which the event occurred.

POOL(8-character data-area)
the name of the pool for which the event occurred.

TARGET(8-character data-area)
the name of the target for which the event occurred.

USERDATA(128-character data-area)
user data as specified on the FEPI START command.

spare
nulls.

Data formats

The format of outbound and inbound are described here.

Outbound data

Data stream
The data is a standard outbound data stream, exactly as would be sent from the simulated terminal to z/OS Communications Server.

Screen-image format, SLU2 mode
The data replaces, byte for byte, the data in the character buffer of the simulated terminal. Any data value is allowed. Data that goes into positions within a protected field must be identical to that in the field; data for positions occupied by an attribute byte is ignored. MDTs can be set forcibly for fields by setting the value in the attribute position to `X'01'`. (FEPI will set MDT automatically if data has changed.)

Key stroke format, SLU2 mode
The data can contain any combination of data characters together with manipulative, special, and attention key values. Data characters are represented by their EBCDIC code values in the range `X'40'`–`X'FE'`, or by their DBCS code values of pairs of bytes in the range `X'41'`–`X'FE'`, plus `X'4040'`. Manipulative, special, and attention key values are represented by escape sequences, comprising the escape character specified by the ESCAPE option and a 2-character code. Using "&" for the escape character, the escape sequences are:

`Manipulative keys`

&HO
home

&Ln
cursor left, n times

&Rn
cursor right, n times

&Un
cursor up, n times

&Dn
cursor down, n times

&Tn
tab, n times

&Bn
backtab, n times

&Nn
newline, n times (where n = 1–9)

`Special keys`

&IN
insert

&DL
delete

&RS
reset

&EF
erase to end of field

&EI
erase input

&FM
field mark

&DU
DUP

&ES
escape character

&MS
start secure MSR

&SO
shift out

&SI
shift in

`Attention keys`

&AT
attention

&An
PAn (n = 1–3)

&nn
PFnn (where nn = 01–24, leading 0 must be specified)

&CL
clear

&CS
cursor select (light pen)

&EN
enter

&ME
end secure MSR

Keys not listed and data characters less than X'40' are not supported. Thus, nulls (X'00') are excluded —nulls can be generated by use of the erase or delete keys. Key strokes following an attempt to enter into a protected field are ignored until "reset" is keyed.

For magnetic stripe reader support, the sequence &MS...data...&ME represents passing a secure magnetic stripe card through the reader. Nonsecure cards have to be simulated by using the corresponding key strokes.

Zero, one, or more than one, attention keys may be used. If an attention key is followed by data characters, FEPI does an implicit receive operation for each one until the back-end application unlocks the keyboard and sends "change direction" or "end bracket" (and FEPI responds positively to any definite response requests); then the subsequent key strokes are sent.

Inbound data

Data stream
The data is a standard inbound data stream, exactly as would be sent to the simulated terminal from z/OS Communications Server. Note that the received data is not complete if the command that received the data returned an ENDSTATUS of MORE.

Formatted, SLU2 mode
The data is the contents of the simulated terminal character buffer that FEPI holds. Data characters are represented by their EBCDIC or DBCS code values; positions corresponding to field attributes contain X'FF'.

Ending status

This information describes in detail the conditions under which FEPI CONVERSE and FEPI RECEIVE commands complete and how the completion condition is reported to the application.

The completion conditions for each command are:

FEPI CONVERSE DATASTREAM using a temporary conversation
On the first to occur of:

- INTO data area full
- ‘change direction’ indicated
- ‘end bracket’ indicated.

It does not end at ‘end of chain’ alone; if a definite response request is indicated on a chain, FEPI responds positively and continues receiving data.

FEPI CONVERSE DATASTREAM using a previously allocated conversation
As for FEPI RECEIVE DATASTREAM.

FEPI CONVERSE FORMATTED using a temporary conversation
on the first to occur of:

- ‘change direction’ indicated
- ‘end bracket’ indicated.

It does not end at ‘end of chain’ alone; if a definite response request is indicated on a chain, FEPI responds positively and continues receiving data.

FEPI CONVERSE FORMATTED using a previously allocated conversation
As for FEPI RECEIVE FORMATTED.

FEPI RECEIVE DATASTREAM
This can be specified or defaulted to end in one of the following ways:

RU
on the first to occur of:

- INTO data area full
- end of request unit.

CHAIN
on the first to occur of:

- INTO data area full
- ‘end of chain’.

UNTILCDEB
on the first to occur of:

- INTO data area full
- ‘end of chain’ with definite response request
- ‘change direction’ indicated
- ‘end bracket’ indicated.

FEPI RECEIVE FORMATTED
At end of chain.

In all cases, ENDSTATUS is set to indicate the completion conditions and RESPSTATUS is set to indicate whether a response is required and, if so, the type of response. Where several conditions occur together, ENDSTATUS shows the most significant. The values and their meanings are shown in Table 12 on page 135.

Table 12. ENDSTATUS values and associated meanings

ENDSTATUS	Commands						Conditions					Next command expected (except after CONVERSE with POOL)
	RECEIVE		CONVERSE without POOL		CONVERSE with POOL		End bracket	Change direction	End chain	End RU	INTO area full	
	DS	FM	DS	FM	DS	FM						
EB	X	X	X	X	X	X	Y	-	Y	Y	-	Any
CD	X	X	X	X	X	X	-	Y	Y	Y	-	FEPI SEND or CONVERSE
LIC	X	X	X	X	-	-	-	-	Y	Y	-	FEPI RECEIVE
RU	R	-	R	-	-	-	-	-	-	Y	-	FEPI RECEIVE
MORE	X	-	X	-	X	-	-	-	-	-	Y	FEPI RECEIVE

Note:

- DS=Datastream
- FM=Formatted
- X=Possible with command
- R=Possible with RU option of command
- Y=Condition indicated.

Appendix B. FEPI system programming reference

Use the FEPI system programming commands to configure and operate FEPI.

Application programming commands such as **ALLOCATE**, **CONVERSE**, and **EXTRACT** are described in Appendix A, “FEPI application programming reference,” on page 95.)

For EIB response codes, see Response codes of EXEC CICS commands.

For EIB function codes, see Function codes of EXEC CICS commands.

Overview of the FEPI SPI commands

The FEPI system programming commands are an addition to the system programming group of **EXEC CICS** commands and have the same features and properties.

The notation used to describe the syntax of FEPI commands is the same as that used to describe all system programming commands in CICS. To use these commands, you must be familiar with:

- The format of **EXEC CICS** commands
- Input and output values, and CVDAs
- The use of the RESP, RESP2, and NOHANDLE options
- Security checking
- The use of **INQUIRE** and **SET** commands
- Browsing

Unlike other CICS system programming commands, the FEPI system programming commands do not need the SP translator option. However, you do need to specify the FEPI translator option.

The FEPI **INQUIRE** and **SET** commands work in the same way as other CICS **INQUIRE** and **SET** commands. They allow you to look at named FEPI resource definitions, browse sets of related definitions, and modify some of the defined values.

FEPI commands can be issued in either 24-bit or 31-bit addressing mode, by programs that reside either above or below the 16MB line. No information is passed through the EXEC interface block (EIB) except that, as for all CICS commands, the EIBRESP, EIBRESP2, EIBFN, and EIBRCODE fields are set.

For EIB response codes, see Response codes of EXEC CICS commands.

For EIB function codes, see Function codes of EXEC CICS commands.

Arguments and data types

The text used to identify arguments indicates the type of data represented by the argument and whether it is a value used by the command, or an area in which the command returns data. For example:

- `POOL(8-character data-value)` indicates that the argument is, or identifies, a string of eight characters, and that the string is passed to the command as an input value.
- `ACQNUM(fullword binary data-area)` indicates that the argument is a user-defined fullword data area in which the command can return a binary number as an output value.

Exceptionally, arguments that are lists have to be data areas, even though they are input values.

Command format

The general format of a command is:

```
EXEC CICS FEPI command option(argument)...
```

where:

command
Is the command name (for example, ADD)

option
Is an option name (for example, POOL)

argument
Is the source or destination for data, as required for the specified option, that is passed to or returned from the command.

The way that you terminate the command is determined by the programming language that you use—COBOL, for example, requires an END-EXEC statement.

Errors and exception conditions

All FEPI commands support the RESP and RESP2 options to signal successful completion or an exception condition. Alternatively, you can use HANDLE CONDITION to trap errors.

Most FEPI command errors give the 'INVREQ' exception condition. The particular error in each case is uniquely identified by the RESP2 value.

Both RESP and RESP2 take, as an argument, the name of a user-defined fullword binary data area. Possible values of the RESP2 option are given in the description of each of the commands and a full list is given in "FEPI RESP2 values" on page 181. The following copy books provide declarations for the RESP2 values:

- DFHSZAPA for assembler language
- DFHSZAPO for COBOL
- DFHSZAPP for PL/I
- DFHSZAPC for C.

The following conditions and RESP2 values can occur for any system programming command:

Condition	RESP2	Meaning
INVREQ	10	Command bypassed by user exit.
INVREQ	11	FEPI not installed, or not active.
INVREQ	12	CICS shutting down, command not allowed.
INVREQ	13	FEPI unavailable.
INVREQ	14	FEPI busy or cannot get storage.
INVREQ	15	Unknown command.
INVREQ	16	Internal error.
INVREQ	17	FEPI cannot get storage for user exit.
INVREQ	18	Command failed through operator or system action.
NOTAUTH	100	Not authorized for this command.

If there is an error, the command does nothing, and the output arguments are not changed.

By their nature, some commands (for example, FEPI SET NODE INSERVICE) initiate a function and return before the function has completed. Errors in the execution of the function cannot be reported as an exception condition on the command. Such errors are reported by writing a record to a transient data (TD) queue and a message to the message log CSZL. See "Transient data queue records" on page 172 for details.

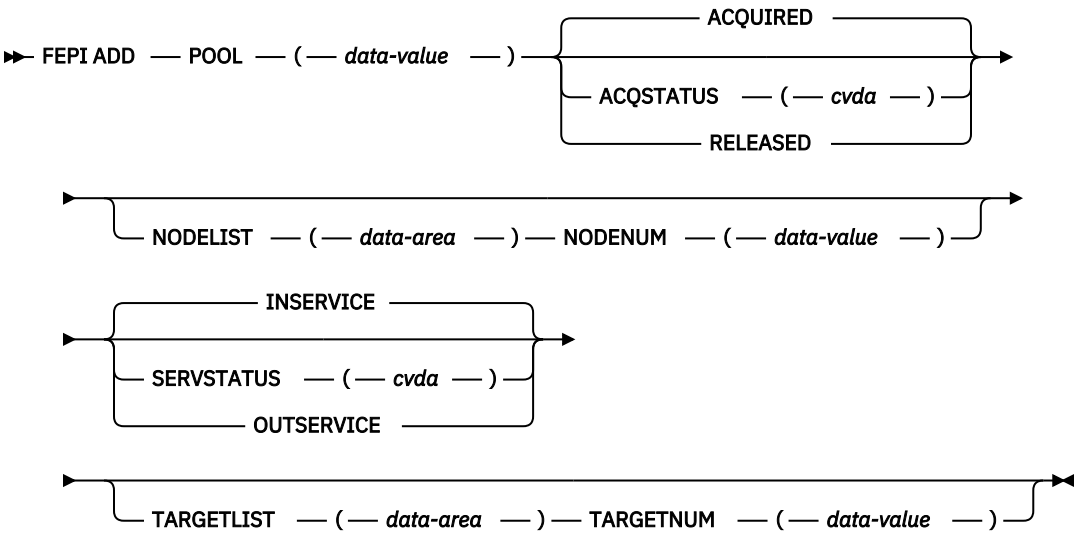
List processing
Commands that operate on a list of resources can fail for some of the resources in the list, but succeed for others. If this happens, a 'list error' is returned on the command. A record is written to a TD queue for each of the resources for which the command failed.

Even if the command fails for all of the resources in the list, it may still be partially successful if other parameters are valid. For example, a **FEPI INSTALL POOL** command installs a valid pool even if the array of node names specified on the **NODELIST** parameter does not exist.

FEPI ADD POOL

Add targets or nodes to an existing pool.

FEPI ADD POOL



Notes:

Description
FEPI ADD POOL adds targets or nodes, or both, to an existing pool, thereby creating new connections in the pool. The targets or nodes must not be in the pool already. You can specify initial service and acquire states for these new connections. The command completes when the resources have been added to the pool but without waiting for the requested states to be achieved.

Options

ACQSTATUS(cvda)
specifies the initial acquire state of the connections being created. All the new connections have the same state. The relevant CVDA values are:

ACQUIRED
The connections are to have sessions established (that is, be 'bound').

RELEASED
The connections are not to have sessions established (that is, remain "unbound").

NODELIST(data-area)
specifies a contiguous array of 8-character node names to be added to the pool. They must already be defined by FEPI INSTALL NODELIST, but can have any service state.

NODENUM(fullword binary data-value)
specifies the number of names in the NODELIST, in the range 0–256.

POOL(8-character data-value)
specifies the name of the pool to which the targets or nodes, or both, are being added.

SERVSTATUS(cvda)
specifies the initial service state of the connections being created. All the new connections have the same state. The relevant CVDA values are:

INSERVICE
The connections are to be in service, and so can be used in a conversation.

OUTSERVICE
The connections are to be out of service and cannot be used for any conversation.

TARGETLIST(data-area)
specifies a contiguous array of 8-character target names to be added to the pool. They must already be defined by FEPI INSTALL TARGETLIST, but can be in any service state.

TARGETNUM(fullword binary data-value)
specifies the number of names in TARGETLIST, in the range 0–256.

Conditions

INVREQ
RESP2 values:

110
SERVSTATUS value not valid.

111
ACQSTATUS value not valid.

115
POOL name unknown.

116
TARGET name unknown.

117
NODE name unknown.

119
The command failed for one or more items in the list.

130
TARGETNUM value is out of range.

131
NODENUM value is out of range.

173
NODE name already exists in the specified pool.

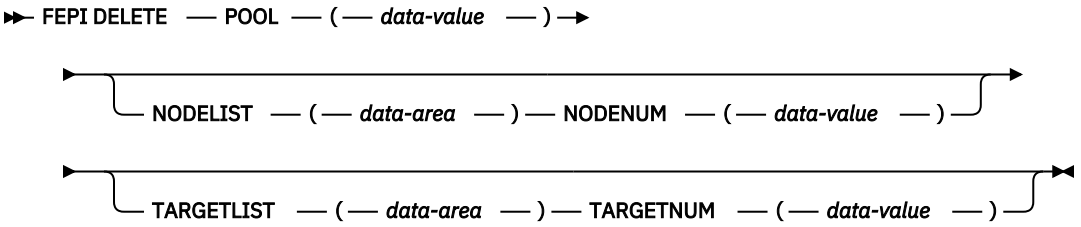
174
TARGET name already exists in the specified pool.

175
Connection already exists.

FEPI DELETE POOL

Remove targets or nodes from a specified FEPI pool.

FEPI DELETE POOL



Notes:

Description

FEPI DELETE POOL removes targets or nodes, or both, from a specified pool, thereby removing connections from the pool. The targets or nodes must be in the pool already. The command completes immediately, without waiting for the necessary deletions to be achieved. When the connections are deleted, they are no longer defined to FEPI.

Options

NODELIST(data-area)
specifies a contiguous array of 8-character node names that are to be deleted from the pool.

NODENUM(fullword binary data-value)
specifies the number of names in the NODELIST, in the range 0–256.

POOL(8-character data-value)
specifies the name of the pool from which targets or nodes are to be removed.

TARGETLIST(data-area)
specifies a contiguous array of 8-character target names that are to be deleted from the pool.

TARGETNUM(fullword binary data-value)
specifies the number of names in TARGETLIST, in the range 0–256.

Conditions

INVREQ
RESP2 values:

115
POOL name unknown.

116
TARGET name unknown.

117
NODE name unknown.

119
The command failed for one or more items in the list.

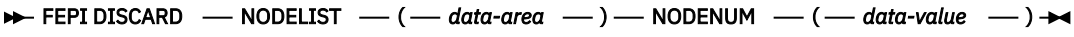
130
TARGETNUM value out of range.

131
NODENUM value out of range.

FEPI DISCARD NODELIST

Remove nodes from FEPI.

FEPI DISCARD NODELIST



Notes:

Description

FEPI DISCARD NODELIST removes nodes completely from FEPI. The state of each node to be discarded is set to OUTSERVICE RELEASED (see "FEPI SET NODE" on page 168). When this state is achieved, the node is deleted from any pool that it is in. The nodes are then discarded so that they are no longer defined to FEPI. The command completes immediately without waiting for the necessary service and acquire states to be achieved.

Options

NODELIST(data-area)
specifies a contiguous array of 8-character node names that are to be discarded.

NODENUM(fullword binary data-value)
specifies the number of names in NODELIST, in the range 1–256.

Conditions

INVREQ
RESP2 values:

117
NODE name unknown.

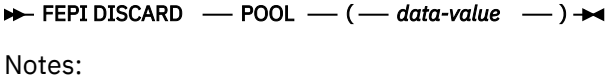
119
The command failed for one or more items in the list.

131
NODENUM value out of range.

FEPI DISCARD POOL

Remove a pool of connections.

FEPI DISCARD POOL



Notes:

Description

FEPI DISCARD POOL removes a pool of connections completely from FEPI. The state of the connections in the pool is set to OUTSERVICE RELEASED (see "FEPI SET CONNECTION" on page 166), and the state of the pool is set to OUTSERVICE (see "FEPI SET POOL" on page 169). When these states have been achieved, the pool and its connections are discarded, so that they are no longer defined to FEPI. The command completes immediately, without waiting for the necessary service and acquire states to be achieved.

Options

POOL(8-character data-value)
specifies the name of the pool to be discarded.

Conditions

INVREQ
RESP2 values:

115
POOL name unknown.

FEPI DISCARD PROPERTYSET

Remove a set of properties.

FEPI DISCARD PROPERTYSET

Notes:

Description

FEPI DISCARD PROPERTYSET removes a set of properties. The properties are discarded immediately so that they are no longer defined to FEPI, but any pool that was installed using the properties is not affected.

Options

PROPERTYSET(8-character data-value)
Specifies the name of the set of properties to be discarded.

Conditions

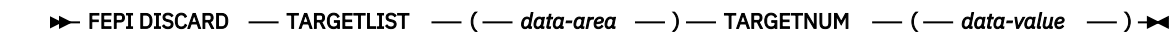
INVREQ
RESP2 values:

171
PROPERTYSET name unknown.

FEPI DISCARD TARGETLIST

Remove targets from FEPI.

FEPI DISCARD TARGETLIST



Notes:

Description

FEPI DISCARD TARGETLIST removes targets completely from FEPI. The state of the targets to be discarded is set to OUTSERVICE (see “FEPI SET TARGET” on page 171). When this state is achieved, the targets are deleted from any pool they are in, and are then discarded, so that they are no longer defined to FEPI. The command completes immediately, without waiting for the necessary service and acquire states to be achieved.

Options

TARGETLIST(data-area)
specifies a contiguous array of eight character target names that are to be discarded.

TARGETNUM(fullword binary data-value)
specifies the number of names in TARGETLIST, in the range 1–256.

Conditions

INVREQ
RESP2 values:

116
TARGET name unknown.

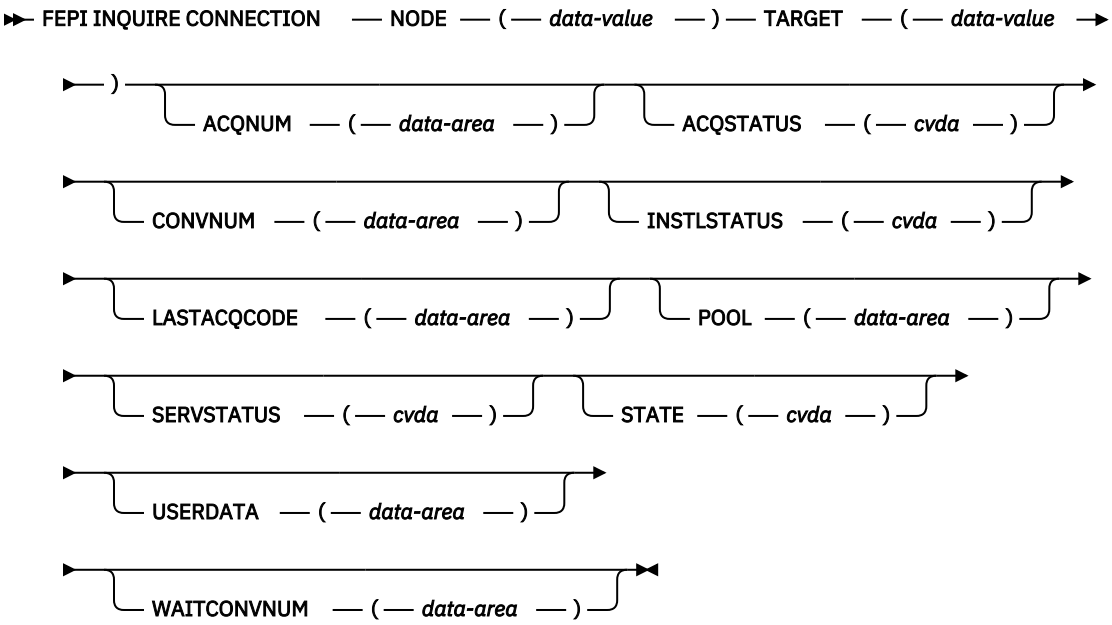
119
The command failed for one or more items in the list.

130
TARGETNUM value out of range.

FEPI INQUIRE CONNECTION

Inquire on a FEPI connection.

FEPI INQUIRE CONNECTION



Notes:

Description

FEPI INQUIRE CONNECTION returns information about a FEPI connection. A connection is identified by specifying its target and node.

The following commands allow you to browse all FEPI connections.

```
FEPI INQUIRE CONNECTION START
FEPI INQUIRE CONNECTION NEXTNODE|NEXTTARGET
NODE(8-character data-area)
TARGET(8-character data-area)
[The options are as for FEPI INQUIRE CONNECTION]
FEPI INQUIRE CONNECTION END
```

Conditions: INVREQ, NOTAUTH

The next connection for which information is returned depends on whether NEXTNODE or NEXTTARGET is specified. If NEXTNODE is specified, the information returned is for the following:

- The next node connected to the current target
- If there are no more nodes connected to the current target, the first node connected to the next target.

If NEXTTARGET is specified, the information returned is for the following:

- The next target connected to the current node
- If there are no more targets connected to the current node, the first target connected to the next node.

Options

ACQNUM(fullword binary data-area)
Returns the number of times that the connection has been acquired.

ACQSTATUS(cvda)
Returns the acquire state; that is, whether a session on the connection is bound or not. The relevant CVDA values are as follows:

ACQUIRED
The session is bound.

ACQUIRING
A state of ACQUIRED has been requested but binding a session has not yet been completed.

RELEASED
No session is bound.

RELEASING
A state of RELEASED has been requested but unbinding the session has not yet been completed.

If ACQUIRING or RELEASING persist, the operator might need to intervene using z/OS Communications Server commands to recover the connection.

CONVNUM(fullword binary data-area)
Returns the number of conversations that have used the connection.

INSTLSTATUS(cvda)
Returns the installation state of the connection. The relevant CVDA values are as follows:

INSTALLED
The connection is in a pool defined by INSTALL and is available for use.

NOTINSTALLED
The connection is in a pool, or involves a node or target that is being discarded but is still in use.

LASTACQCODE(fullword binary data-area)
Returns the result of the last acquire request for the connection; that is, the sense code from the last z/OS Communications Server VTAM REQSESS, zero indicating success.

Note: CLSDST(PASS) X'32020000' can be returned in this field. This is the unbind flow received by CICS during CLSDST(PASS) processing.

For details of z/OS Communications Server sense codes, see z/OS Communications Server: IP and SNA Codes.

NODE(8-character data-value/8-character data-area)
The node that identifies the connection.

POOL(8-character data-area)
Returns the name of the pool that defines the connection.

SERVSTATUS(cvda)
Returns the service state of the connection. The relevant CVDA values are as follows:

INSERVICE
The connection is in service and can be used in a conversation.

OUTSERVICE
The connection is out of service and cannot be used for any new conversation, but a conversation using the connection is unaffected. The service state is GOINGOUT until any such conversation ends.

GOINGOUT
A state of OUTSERVICE has been requested but the connection is still being used by some conversation.

STATE(cvda)
Returns the state of the conversation using the connection. The relevant CVDA values are as follows:

NOCONV
No conversation is active on the connection.

PENDSTSN
An STSN-handling task has been scheduled.

STSN
An STSN-handling task owns the conversation.

PENDBEGIN
A begin-session handling task has been scheduled.

BEGINSESSION
A begin-session handling task owns the conversation.

APPLICATION
A normal application task owns the conversation.

PENDDATA
FEPI is waiting for inbound data, following a FEPI START command.

PENDSTART
Inbound data having arrived, a task specified by FEPI START has been scheduled.

PENDFREE
An end-session handling task has been scheduled, following a FEPI FREE command.

FREE
An end-session handling task owns the conversation, following a FEPI FREE command.

PENDRELEASE
An end-session handling task has been scheduled, following an unbind request.

RELEASE
An end-session handling task owns the conversation, following an unbind request.

PENDUNSOL
An unsolicited-data handling task has been scheduled.

UNSOLDATA
An unsolicited-data handling task owns the conversation.

PENDPASS
The conversation is unowned, following a FEPI FREE PASS command.

The pending states indicate that the conversation is unowned, pending the event or task indicated; the state ceases to be pending when a task issues a FEPI ALLOCATE PASSCONVID command. If a pending state persists, it is likely that the application has failed in some way; consider resetting the connection by issuing FEPI SET CONNECTION RELEASED.

TARGET(8-character data-value/8-character data-area)
The target that identifies the connection.

USERDATA(64-character data-area)
Returns the user data for the connection. If no user data has been set, nulls are returned.

WAITCONVNUM(fullword binary data-area)
Returns the number of conversations that are waiting to start using the connection. If a conversation could use any one of several connections, it is counted as waiting on each one.

Conditions

ILLOGIC
RESP2 values:

1
For START: browse of this resource type is already in progress. For NEXT or INQUIRE: END was not issued.

END
RESP2 values:

2
For NEXT: all resource definitions have been retrieved.

INVREQ
RESP2 values:

116
TARGET name unknown.

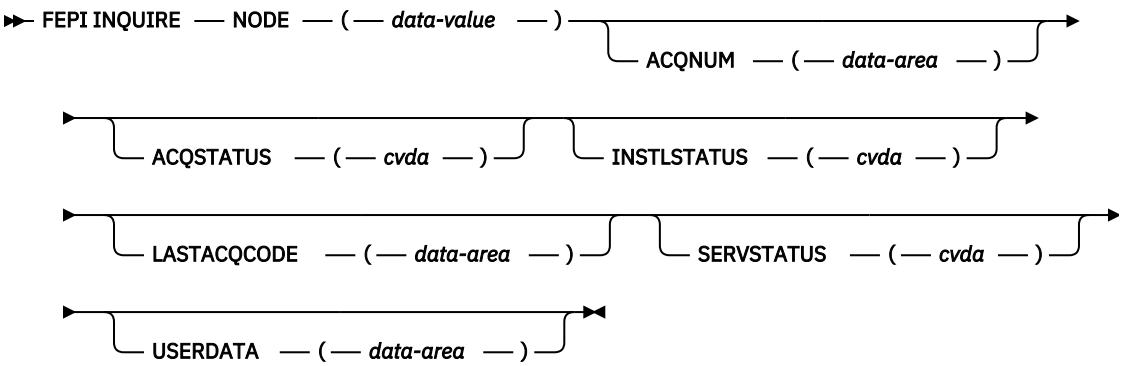
117
NODE name unknown.

118
Connection unknown (TARGET and NODE names known, but not in a common pool).

FEPI INQUIRE NODE

Inquire on a FEPI node.

FEPI INQUIRE NODE



Notes:

Description

FEPI INQUIRE NODE returns information about a FEPI node.

The following commands allow you to browse all FEPI NODE definitions.

```
FEPI INQUIRE NODE START
FEPI INQUIRE NODE(8-character data-area) NEXT
[The options are as for FEPI INQUIRE NODE]
FEPI INQUIRE NODE END
```

Options

ACQNUM(fullword binary data-area)
returns the number of times that the node has been acquired.

ACQSTATUS(cvda)
returns the acquire state—that is, whether the z/OS Communications Server ACB is opened or closed. The relevant CVDA values are:

ACQUIRED
The z/OS Communications Server ACB for the node is open and ‘set logon start' has completed.

ACQUIRING
A state of ACQUIRED has been requested but opening the z/OS Communications Server ACB for the node and issuing ‘set logon start' has not yet been completed.

RELEASED
Sessions on any connections involving the node have been unbound and the z/OS Communications Server ACB has been closed.

RELEASING
A state of RELEASED has been requested but closing the z/OS Communications Server ACB for the node has not yet been completed.

If ACQUIRING or RELEASING persist, the operator might need to intervene using z/OS Communications Server commands to recover the node.

INSTLSTATUS(cvda)
returns the installation state of the node. The relevant CVDA values are:

INSTALLED
The node has been defined by INSTALL and is available for use.

NOTINSTALLED
The node is being discarded, but is still in use.

LASTACQCODE(fullword binary data-area)
returns the result of the last acquire request for the node; that is, the return code from the last z/OS Communications Server OPEN ACB, zero indicating success. For details of z/OS Communications Server return codes, see z/OS Communications Server: IP and SNA Codes.

NODE(8-character data-value/8-character data-area)
is the name of the node.

SERVSTATUS(cvda)
returns the service state of the node. The relevant CVDA values are:

INSERVICE
The node is in service and can be used in a conversation.

OUTSERVICE
The node is out of service and cannot be used for any conversation.

GOINGOUT
A state of OUTSERVICE has been requested but the node is still being used by a conversation.

USERDATA(64-character data-area)
returns the user data for the node. If no user data has been set, nulls are returned.

Conditions

ILLOGIC
RESP2 value:

1
For START: browse of this resource type is already in progress. For NEXT or END: START was not issued.

END
RESP2 value:

2
For NEXT: all resource definitions have been retrieved.

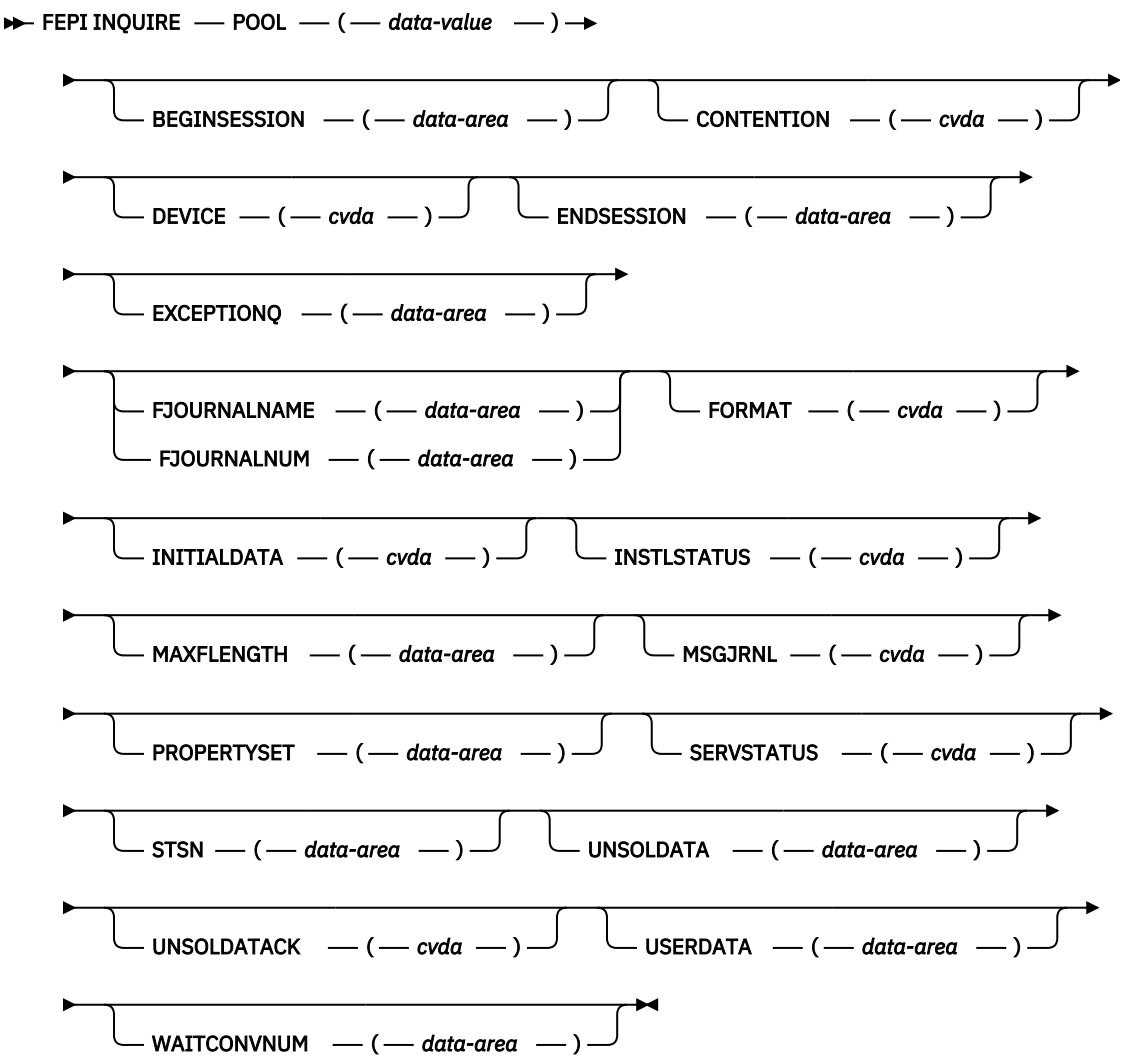
INVREQ
RESP2 value:

117
NODE name unknown.

FEPI INQUIRE POOL

Inquire on a FEPI pool.

FEPI INQUIRE POOL



Notes:

Description

FEPI INQUIRE POOL returns information about a FEPI pool of connections.

The following commands allow you to browse all FEPI POOL definitions.

```
FEPI INQUIRE POOL START
FEPI INQUIRE POOL(8-character data-area) NEXT
[The options are as for FEPI INQUIRE POOL]
FEPI INQUIRE POOL END
```

Options

BEGINSESSION(4-character data-area)
returns the name of the transaction performing begin-session processing, or blanks if no transaction was specified.

CONTENTION(cvda)
returns a value that specifies what happens when a FEPI SEND command is issued and there is inbound data with 'begin bracket'. The relevant CVDA values are:

LOSE
FEPI SEND command fails; a FEPI RECEIVE must be issued to get the inbound data.

WIN
FEPI SEND command succeeds; inbound data is rejected with a negative response.

DEVICE(cvda)
returns a value that identifies the mode of conversation and the type of device. Defined values are:

T3278M2
SLU2 mode, 3278 Model 2

T3278M3
SLU2 mode, 3278 Model 3

T3278M4
SLU2 mode, 3278 Model 4

T3278M5
SLU2 mode, 3278 Model 5

T3279M2
SLU2 mode, 3279 Model 2B

T3279M3
SLU2 mode, 3279 Model 3B

T3279M4
SLU2 mode, 3279 Model 4B

T3279M5
SLU2 mode, 3279 Model 5B

TPS55M2
SLU2 mode, PS/55, 24 lines

TPS55M3
SLU2 mode, PS/55, 32 lines

TPS55M4
SLU2 mode, PS/55, 43 lines

LUP
SLU P mode, all cases.

ENDSESSION(4-character data-area)
returns the name of the transaction performing end-session processing, or blanks if no transaction was specified.

EXCEPTIONQ(4-character data-area)
returns the name of the TD queue to which exceptional events are notified, or blanks if no queue was specified.

FJOURNALNAME(8-character data-area)
returns the 1- to 8-character name of the journal where data is to be logged.

FJOURNALNUM(fullword binary data-area)
returns the number of the journal where data is to be logged.

FORMAT(cvda)
returns a value that identifies the data format. The relevant CVDA values are:

FORMATTED
Formatted operation

DATASTREAM
Data stream operation

NOTAPPLIC
Option is not applicable for the specified pool.

INITIALDATA(cvda)
returns a value indicating whether initial inbound data is expected when a session is started. The relevant CVDA values are:

NOTINBOUND
No inbound data expected

INBOUND
Inbound data expected.

INSTLSTATUS(cvda)
returns the installation state of the pool. The relevant CVDA values are:

INSTALLED
The pool has been defined by INSTALL and is available for use.

NOTINSTALLED
The pool is being discarded, but is still in use.

MAXFLENGTH(fullword binary data-area)
returns the maximum length of data that can be returned on any FEPI RECEIVE, CONVERSE, or EXTRACT FIELD command for a conversation, or that can be sent by any FEPI SEND or CONVERSE command for a conversation.

MSGJRNL(cvda)
returns a value indicating whether journaling is performed for inbound and outbound data. The relevant CVDA values are:

NOMSGJRNL
No journaling is to be performed.

INPUT
Inbound data is journaled.

OUTPUT
Outbound data is journaled.

INOUT
Inbound and outbound data are journaled.

POOL(8-character data-value/8-character data-area)
is the name of the pool.

PROPERTYSET(8-character data-area)
returns the name of the set of properties with which the pool was installed.

SERVSTATUS(cvda)
returns the service state of the pool. The relevant CVDA values are:

INSERVICE
The pool is in service and can be used in a conversation.

OUTSERVICE
The pool is out of service and cannot be used for any conversation.

GOINGOUT
A state of OUTSERVICE has been requested but the pool is still being used by some conversation.

STSN(4-character data-area)
returns the name of the transaction handling STSN data, or blanks if no transaction was specified.

UNSOLDATA(4-character data-area)
returns the name of the transaction handling unsolicited data (data received outside a conversation), or blanks if no transaction was specified.

UNSOLDATACK(cvda)
if there is no unsolicited data processing, this indicates what acknowledgment FEPI gives to a BID. The relevant CVDA values are:

NEGATIVE
Negative response X'0813', BID not accepted

POSITIVE
Positive response, BID accepted and subsequent data is accepted and discarded

NOTAPPLIC
Option is not applicable for the specified pool.

USERDATA(64-character data-area)
returns the user data for the pool. If no user data has been set, nulls are returned.

WAITCONVNUM(fullword binary data-area)
returns the number of conversations that are waiting to start using a connection in the pool.

Conditions

ILLOGIC
RESP2 value:

1
For START: browse of this resource type is already in progress. For NEXT or END: START was not issued.

END
RESP2 value:

2
For NEXT: all resource definitions have been retrieved.

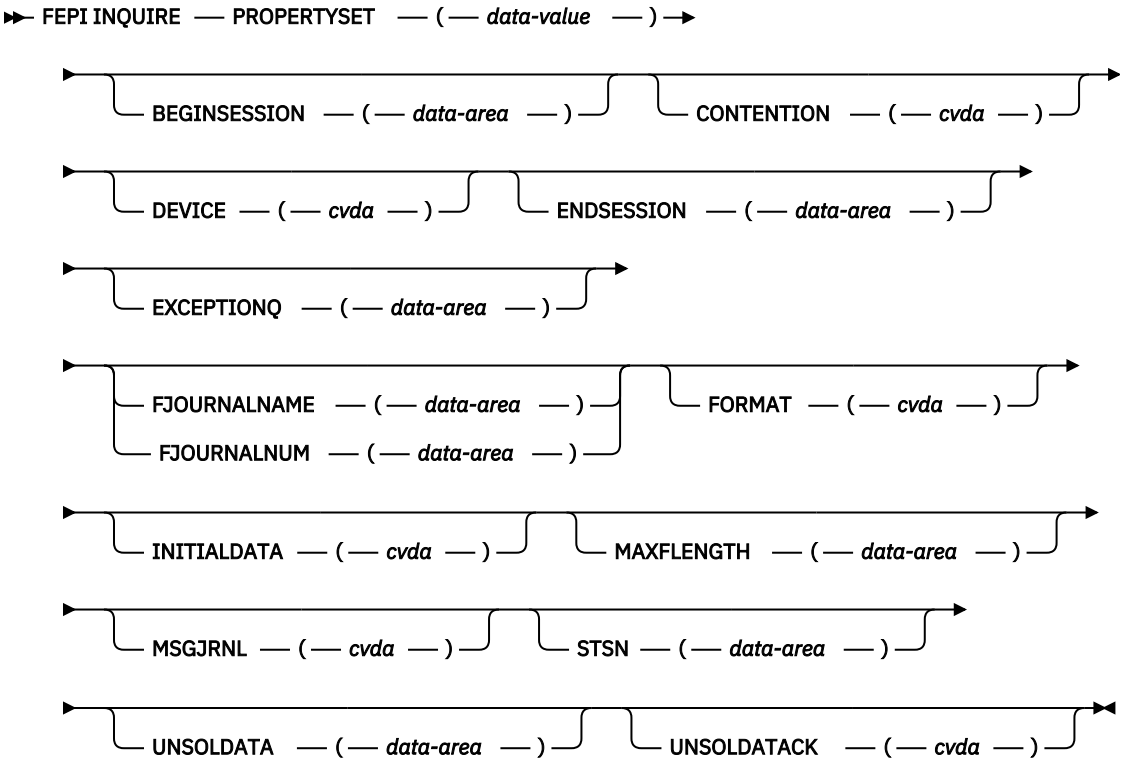
INVREQ
RESP2 value:

115
POOL name unknown.

FEPI INQUIRE PROPERTYSET

Inquire on a FEPI property set.

FEPI INQUIRE PROPERTYSET



Notes:

Description

FEPI INQUIRE PROPERTYSET returns information about a FEPI property set.

The following commands allow you to browse all FEPI PROPERTYSET definitions:

```
FEPI INQUIRE PROPERTYSET START
FEPI INQUIRE PROPERTYSET(8-character data-area) NEXT
[The options are as for FEPI INQUIRE PROPERTYSET]
FEPI INQUIRE PROPERTYSET END
```

Options

BEGINSESSION(4-character data-area)
returns the name of the transaction performing begin-session processing, or blanks if no transaction was specified.

CONTENTION(cvda)
returns a value that specifies what happens when a FEPI SEND command is issued and there is inbound data with ‘begin bracket'. The relevant CVDA values are:

LOSE
FEPI SEND command fails; a FEPI RECEIVE must be issued to get the inbound data.

WIN
FEPI SEND command succeeds; inbound data is rejected with a negative response.

DEVICE(cvda)
returns a value that identifies the mode of conversation and the type of device. Defined values are:

T3278M2
SLU2 mode, 3278 Model 2

T3278M3
SLU2 mode, 3278 Model 3

T3278M4
SLU2 mode, 3278 Model 4

T3278M5
SLU2 mode, 3278 Model 5

T3279M2
SLU2 mode, 3279 Model 2B

T3279M3
SLU2 mode, 3279 Model 3B

T3279M4
SLU2 mode, 3279 Model 4B

T3279M5
SLU2 mode, 3279 Model 5B

TPS55M2
SLU2 mode, PS/55, 24 lines

TPS55M3
SLU2 mode, PS/55, 32 lines

TPS55M4
SLU2 mode, PS/55, 43 lines

LUP
SLU P mode, all cases.

ENDSESSION(4-character data-area)
returns the name of the transaction performing end-session processing, or blanks if no transaction was specified.

EXCEPTIONQ(4-character data-area)
returns the name of the TD queue to which exceptional events are notified, or blanks if no queue was specified.

FJOURNALNAME(8-character data-area)
returns the 1- to 8-character name of the journal where data is to be logged.

FJOURNALNUM(fullword binary data-area)
returns the number of the journal where data is to be logged.

FORMAT(cvda)
returns a value that identifies the data format. The relevant CVDA values are:

FORMATTED
Formatted operation

DATASTREAM
Data stream operation

NOTAPPLIC
Option is not applicable for the specified pool.

INITIALDATA(cvda)
returns a value indicating whether initial inbound data is expected when a session is started. The relevant CVDA values are:

NOTINBOUND
No inbound data expected

INBOUND
Inbound data expected.

MAXFLENGTH(fullword binary data-area)
returns the maximum length of data that can be returned on any FEPI RECEIVE, CONVERSE, or EXTRACT FIELD command for a conversation, or that can be sent by any FEPI SEND or CONVERSE command for a conversation.

MSGJRNL(cvda)
returns a value indicating whether journaling is performed for inbound and outbound data. The relevant CVDA values are:

NOMSGJRNL
No journaling is to be performed.

INPUT
Inbound data is journaled.

OUTPUT
Outbound data is journaled.

INOUT
Inbound and outbound data are journaled.

PROPERTYSET(8-character data-value/8-character data-area)
is the name of the set of properties.

STSN(4-character data-area)
returns the name of the transaction handling STSN data (SLU P mode only), or blanks if no transaction was specified.

UNSOLDATA(4-character data-area)
returns the name of the transaction handling unsolicited data (data received outside a conversation), or blanks if no transaction was specified.

UNSOLDATACK(cvda)
indicates what acknowledgment FEPI gives to a BID, if there is no unsolicited-data processing. The relevant CVDA values are:

NEGATIVE
Negative response X'0813', BID not accepted

POSITIVE
Positive response, BID accepted and subsequent data is accepted and discarded

NOTAPPLIC
Option is not applicable for the specified pool.

Conditions

ILLOGIC
RESP2 value:

1
For START: browse of this resource type is already in progress. For NEXT or END: START was not issued.

END
RESP2 value:

2
For NEXT: all resource definitions have been retrieved.

INVREQ
RESP2 value:

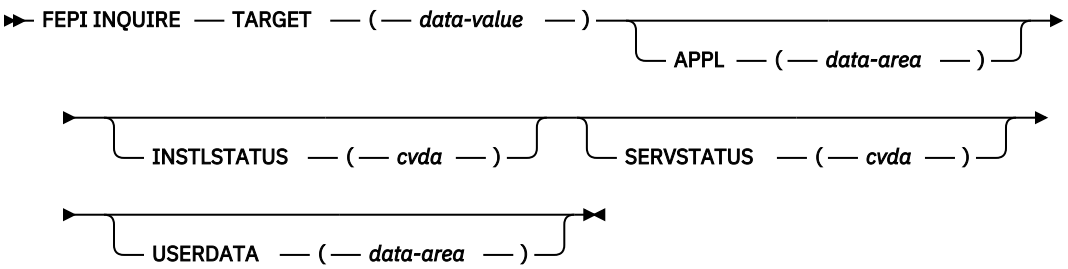
11
FEPI not installed, or not active.

171
PROPERTYSET name unknown.

FEPI INQUIRE TARGET

Inquire on a FEPI target.

FEPI INQUIRE TARGET



Notes:

Description

FEPI INQUIRE TARGET returns information about a FEPI target.

The following commands allow you to browse all FEPI TARGET definitions.

```
FEPI INQUIRE TARGET START
FEPI INQUIRE TARGET(8-character data-area) NEXT
[The options are as for FEPI INQUIRE TARGET]
FEPI INQUIRE TARGET END
```

Options

APPL(8-character data-area)
returns the z/OS Communications Server application name of the back-end system that the target system represents.

INSTLSTATUS(cvda)
returns the installation state of the target. The relevant CVDA values are:

INSTALLED
The target has been defined by INSTALL and is available for use.

NOTINSTALLED
The target is being discarded but is still in use.

SERVSTATUS(cvda)
returns the service state of the target. The relevant CVDA values are:

INSERVICE
The target is in service and can be used in a conversation.

OUTSERVICE
The target is out of service and cannot be used for any conversation.

GOINGOUT
A state of OUTSERVICE has been requested but the target is still being used by some conversation.

TARGET(8-character data-value/8-character data-area)
is the name of the target.

USERDATA(64-character data-area)
returns the user data for the target. If no user data has been set, nulls are returned.

Conditions

ILLOGIC
RESP2 value:

1
For START: browse of this resource type is already in progress. For NEXT or END: START was not issued.

END
RESP2 value:

2
For NEXT: all resource definitions have been retrieved.

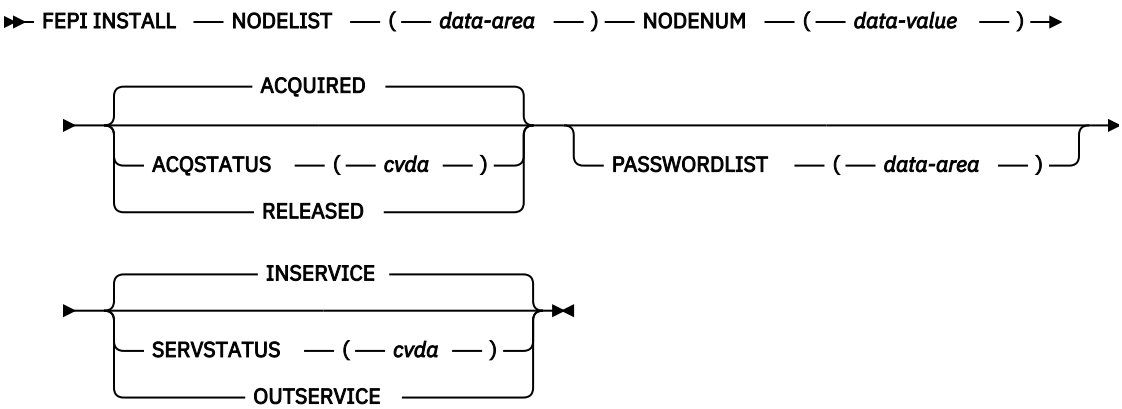
INVREQ
RESP2 value:

116
TARGET name unknown.

FEPI INSTALL NODELIST

Define new nodes to FEPI.

FEPI INSTALL NODELIST



Notes:

Description

FEPI INSTALL NODELIST defines new nodes to FEPI. You may specify initial service and acquire states for these new nodes. A node cannot be used for a conversation until it has been acquired, put in service, and added to a pool so that it is connected to a target. The command completes when the nodes have been defined without waiting for the requested states to be achieved.

Options

ACQSTATUS(cvda)
specifies the initial acquire state of the nodes being defined. All nodes in the list have the same state. The relevant CVDA values are:

ACQUIRED
The z/OS Communications Server ACB for the node is to be opened and ‘set logon start' is to be done.

RELEASED
The z/OS Communications Server ACB for the node is not to be opened.

NODELIST(data-area)
specifies a contiguous array of 8-character node names (that is, z/OS Communications Server application minor node names in the front-end) to be defined. Names must not contain null characters (X'00'), leading blanks, or embedded blanks.

NODENUM(fullword binary data-value)
specifies the number of names in NODELIST, in the range 1–256.

PASSWORDLIST(data-value)
specifies a contiguous array of 8-character passwords. They correspond one-to-one with the node names in NODELIST. The passwords are those that z/OS Communications Server requires to access the application minor nodes. They are not required if passwords are not used. You can use a value of 8 null characters (X'00') to indicate ‘no password'.

SERVSTATUS(cvda)
specifies the initial service state of the nodes being defined. All nodes in the list have the same state. The relevant CVDA values are:

INSERVICE
The nodes are in service and can be used in a conversation.

OUTSERVICE
The nodes are out of service and cannot be used for any conversation.

Conditions

INVREQ
RESP2 values:

11
FEPI not installed or not active.

110
SERVSTATUS value not valid.

111
ACQSTATUS value not valid.

119
The command failed for one or more items in the list.

131
NODENUM value out of range.

163
NODE name not valid.

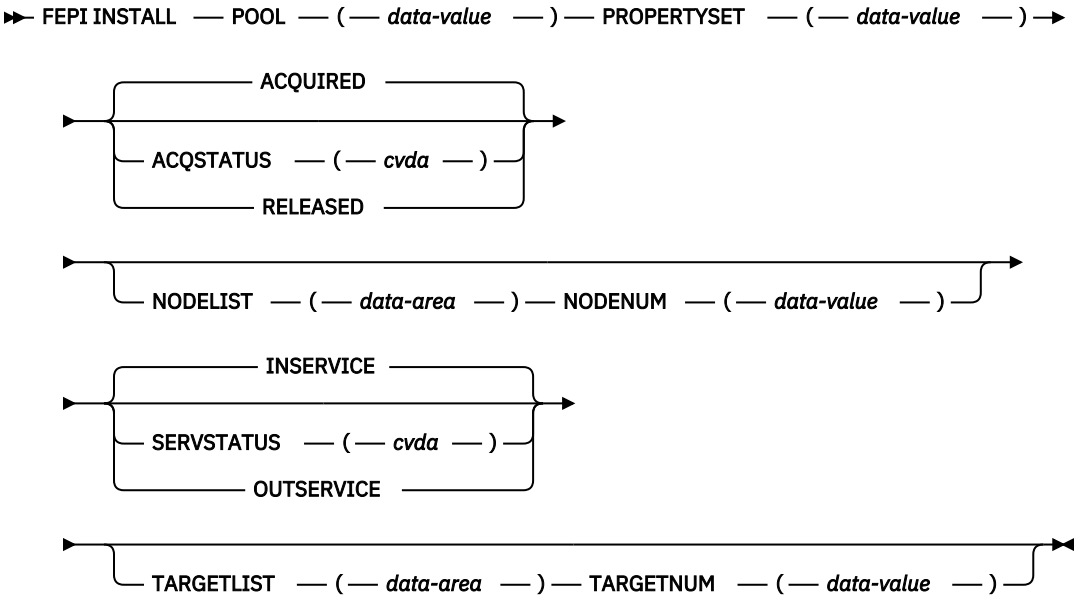
173
NODE name already exists.

176
The z/OS Communications Server OPEN ACB failed.

FEPI INSTALL POOL

Define a new pool of connections.

FEPI INSTALL POOL



Notes:

Description

FEPI INSTALL POOL defines a new pool of connections to FEPI. Any targets and nodes specified in the command are added to it, thereby creating new connections in the pool. You may specify an initial service state for the pool, and initial service and acquire states for any new connections. A pool cannot be used for a conversation until it has been put in service. The command completes when the pool has been created and any resources added; it does not wait for the requested states to be achieved.

Options

ACQSTATUS(cvda)
specifies the initial acquire state of the connections being created. All the new connections have the same state. The relevant CVDA values are:

ACQUIRED
The connections are to have sessions established (that is, "bound").

RELEASED
The connections are not to have sessions established (that is, the remain "unbound").

NODELIST(data-area)
specifies a contiguous array of 8-character node names. They must already be defined by FEPI INSTALL NODELIST.

NODENUM(fullword binary data-value)
specifies the number of names in NODELIST, in the range 0–256.

POOL(8-character data-value)
specifies the name of the pool to be defined. The name must not contain null characters (X'00'), leading blanks, or embedded blanks.

PROPERTYSET(8-character data-value)
specifies the name of the set of properties for the pool, which must have been installed already.

SERVSTATUS(cvda)
specifies the initial service state of the pool being defined and of the connections being created. All the new connections have the same state. The relevant CVDA values are:

INSERVICE
The pool and any connections are in service and can be used in a conversation.

OUTSERVICE
The pool and any connections are out of service and cannot be used for any conversation.

TARGETLIST(data-area)
specifies a contiguous array of 8-character target names. They must already be defined by FEPI INSTALL TARGETLIST.

TARGETNUM(fullword binary data-value)
specifies the number of names in TARGETLIST, in the range 0–256.

Conditions

INVREQ
RESP2 values:

11
FEPI not installed or not active.

110
SERVSTATUS value not valid.

111
ACQSTATUS value not valid.

116
TARGET name unknown.

117
NODE name unknown.

119
The command failed for one or more items in the list.

130
TARGETNUM value out of range.

131
NODENUM value out of range.

162
POOL name not valid.

171
PROPERTYSET name unknown.

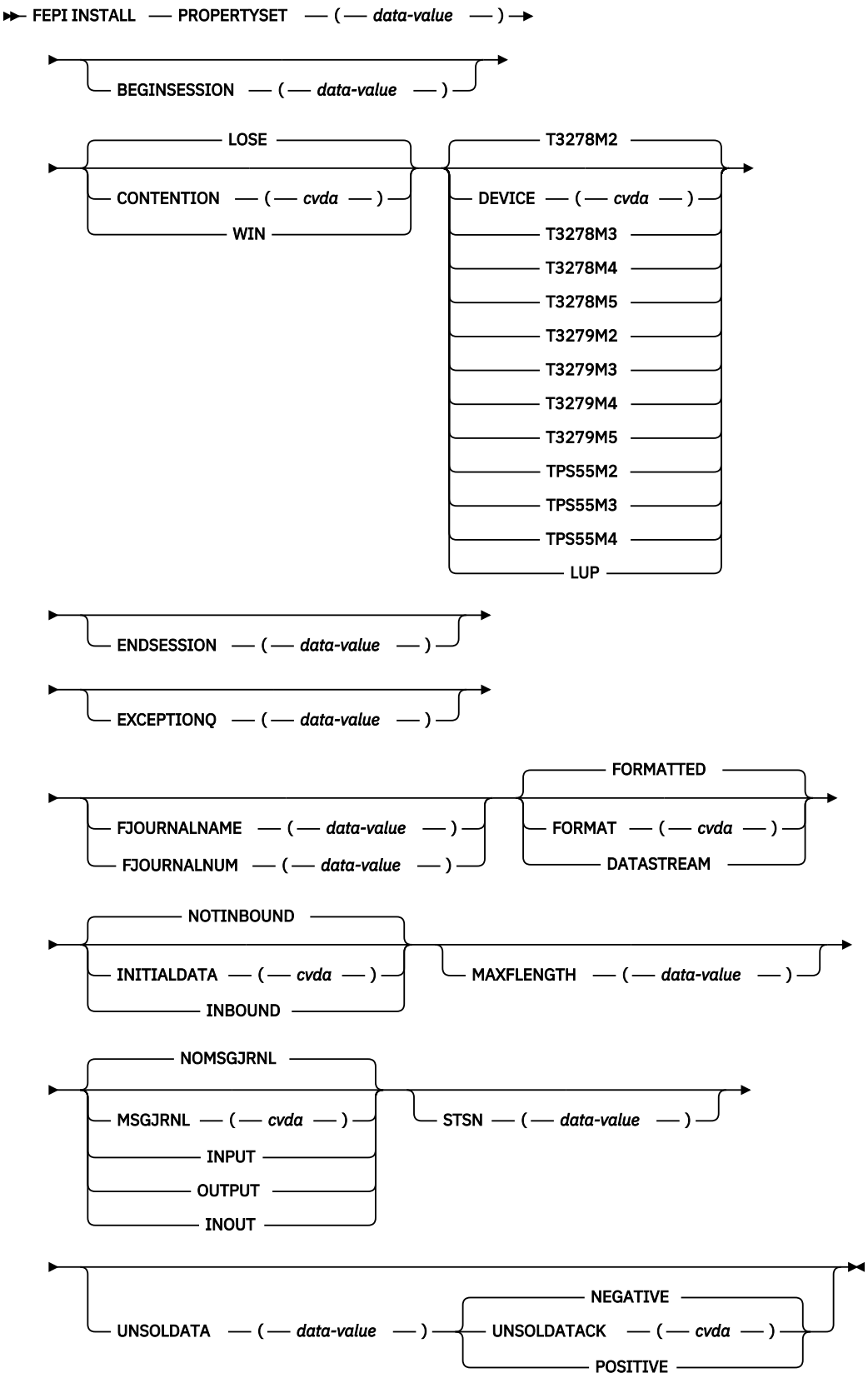
172
POOL name already exists.

175
The connection already exists.

FEPI INSTALL PROPERTYSET

Define a new set of properties.

FEPI INSTALL PROPERTYSET



Notes:

Description

FEPI INSTALL PROPERTYSET defines a new set of properties to FEPI, which can be applied to any subsequently defined pool.

Options

Note: Specifying a blank value for BEGINSESSION, ENDSESSION, EXCEPTIONQ, STSN, or UNSOLDATA has the same effect as omitting the option.

BEGINSESSION(4-character data-value)
specifies the name of the transaction to perform begin-session processing, immediately after a session has been established ("bound"). If omitted, there is to be no user-supplied begin-session processing.

CONTENTION(cvda)
specifies what happens when a FEPI SEND command is issued and there is inbound data with begin-bracket. The relevant CVDA values are:

LOSE
The FEPI SEND command fails; a FEPI RECEIVE must be issued to get the inbound data.

WIN
The FEPI SEND command succeeds; inbound data is rejected with a negative response.

DEVICE(cvda)
specifies the LU mode and device type that is to be simulated. The relevant CVDA values are:

T3278M2
SLU2 mode, 3278 Model 2

T3278M3
SLU2 mode, 3278 Model 3

T3278M4
SLU2 mode, 3278 Model 4

T3278M5
SLU2 mode, 3278 Model 5

T3279M2
SLU2 mode, 3279 Model 2B

T3279M3
SLU2 mode, 3279 Model 3B

T3279M4
SLU2 mode, 3279 Model 4B

T3279M5
SLU2 mode, 3279 Model 5B

TPS55M2
SLU2 mode, PS/55, 24 lines

TPS55M3
SLU2 mode, PS/55, 32 lines

TPS55M4
SLU2 mode, PS/55, 43 lines

LUP
SLU P mode, all cases.

ENDSESSION(4-character data-value)
specifies the name of the transaction to perform end-session processing, when a conversation is ended (by a FEPI FREE command) or when a session is to be ended ("unbound"). If omitted, there is to be no user-supplied end-session processing.

EXCEPTIONQ(4-character data-value)
specifies the name of the TD queue to which pool-specific exceptional events are to be notified. If EXCEPTIONQ is omitted, there is to be no user-supplied exceptional event processing.

FJOURNALNAME(8-character data-value)
specifies the 1- to 8-character name of the journal where data is to be logged. You are not permitted to specify DFHLOG or DFHSHUNT, the primary and secondary system logs. If the value is zero or omitted, no journaling is done.

FJOURNALNUM(fullword binary data-value)
specifies the number of the journal where data is to be logged, in the range 1 through 99. Specifying a value here implies the journal name DFHJ*nn*, where *nn* is the journal number. If the value is zero or omitted, no journaling is done.

FORMAT(cvda)
specifies, for SLU2 mode, the data mode to be used. The relevant CVDA values are:

FORMATTED
Formatted operation. Character attributes are not supported on outbound data and ignored on inbound data.

DATASTREAM
Data stream operation.

This option is not valid for SLU P operation.

INITIALDATA(cvda)
specifies whether initial inbound data is expected when a session is started. The relevant CVDA values are:

NOTINBOUND
No inbound data is expected.

INBOUND
Inbound data is expected.

If the target is a back-end IMS system, you should specify INBOUND. See FEPI programming: Begin-session handler.

MAXFLENGTH(fullword binary data-value)
specifies the maximum length of data that can be returned on any FEPI RECEIVE, CONVERSE, or EXTRACT FIELD command for a conversation, or that can be sent by any FEPI SEND or CONVERSE command for a conversation. This value helps FEPI use storage more efficiently, so should be set no larger than is necessary. It must be in the range 128–1 048 576. If MAXFLENGTH is not specified, 4096 is used.

MSGJRNL(cvda)
specifies the required journaling of data to and from the back-end system. The relevant CVDA values are:

NOMSGJRNL
No journaling

INPUT
Journal inbound data

OUTPUT
Journal outbound data

INOUT
Journal inbound and outbound data.

PROPERTYSET(8-character data-value)
specifies the name of the set of properties to be defined. The name must not contain null characters (X'00'), leading blanks, or embedded blanks.

STSN(4-character data-value)
specifies the name of the transaction to be started to handle "set and test sequence number" (STSN), for SLU P mode only. If omitted, there is to be no user-supplied STSN-handling; FEPI handles STSN automatically.

UNSOLDATA(4-character data-value)
specifies the name of the transaction to handle unsolicited data (data received outside a conversation). If omitted, there is to be no user-supplied unsolicited-data processing; FEPI treats unsolicited data as specified by UNSOLDATACK.

UNSOLDATACK(cvda)
if there is to be no unsolicited-data processing, this specifies what acknowledgment FEPI is to give to a BID. The relevant CVDA values are:

NEGATIVE
Negative response X'0813', BID not accepted

POSITIVE
Positive response, BID accepted and subsequent data is accepted and discarded.

Conditions

INVREQ
RESP2 values:

11
FEPI not installed or not active.

140
DEVICE value not valid.

141
CONTENTION value not valid.

142
INITIALDATA value not valid.

143
UNSOLDATACK value not valid.

144
MSGJRNL value not valid.

150
FORMAT value not valid or is unsuitable for the LU mode and device type specified by the DEVICE value.

153
STSN name not valid or STSN is not allowed for the LU mode and device type specified by the DEVICE value.

154
BEGINSESSION name not valid.

155
UNSOLDATA name not valid.

156
EXCEPTIONQ name not valid.

157
FJOURNALNUM value not valid.

158
MAXFLENGTH value not valid.

159
ENDSESSION name not valid.

160
PROPERTYSET name not valid.

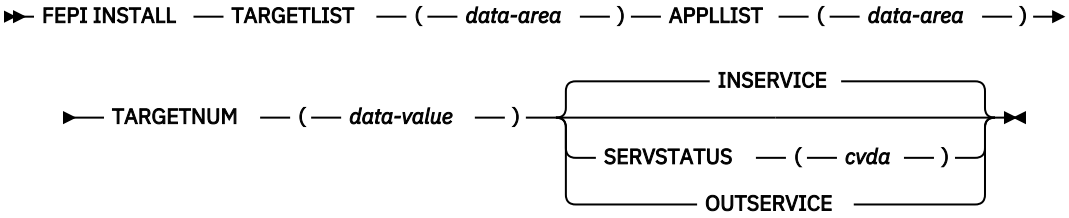
170
PROPERTYSET name already exists.

178
FJOURNALNAME value not valid.

FEPI INSTALL TARGETLIST

Define new targets to FEPI.

FEPI INSTALL TARGETLIST



Notes:

Description

FEPI INSTALL TARGETLIST defines new targets to FEPI. You can specify an initial service state for these new targets. A target cannot be used for a conversation until it has been put in service, and has been added to a pool so that it is connected to a node. The command completes when the targets have been installed without waiting for the requested states to be achieved.

Options

APPLLIST(data-area)
specifies a contiguous array of 8-character primary logical unit (PLU) names. These are the z/OS Communications Server application names (APPLID) of the back-end CICS or IMS systems with which FEPI applications are to communicate; they correspond one-to-one with the target names in TARGETLIST. The names must not contain null characters (X'00'), leading blanks, or embedded blanks. Each name must be unique within the list; duplicate names result in an INVREQ condition being returned.

If a target specified in TARGETLIST is a CICS terminal-owning region that is a member of a z/OS Communications Server generic resource group, you can specify in APPLLIST its generic resource name. This enables you to use the z/OS Communications Server generic resource function to balance sessions across the available TORs. See Workload routing in a sysplex.

SERVSTATUS(cvda)
specifies the initial service state of the targets being defined. All the targets in the list have the same state. The relevant CVDA values are:

INSERVICE
The target is in service and can be used in a conversation.

OUTSERVICE
The target is out of service and cannot be used for any conversation.

TARGETLIST(data-area)
specifies a contiguous array of 8-character target names to be defined. A target name is the logical FEPI front-end name of a back-end system. The names must not contain null characters (X'00'), leading blanks, or embedded blanks. Each name must be unique within the list; duplicate names result in an INVREQ condition being returned.

TARGETNUM(fullword binary data-value)
specifies the number of names in TARGETLIST, in the range 1–256.

Conditions

INVREQ
RESP2 values:

11
FEPI not installed or not active.

110
SERVSTATUS value not valid.

119
The command failed for one or more items in the list.

130
TARGETNUM value out of range.

164
TARGET name not valid.

167
Application name not valid.

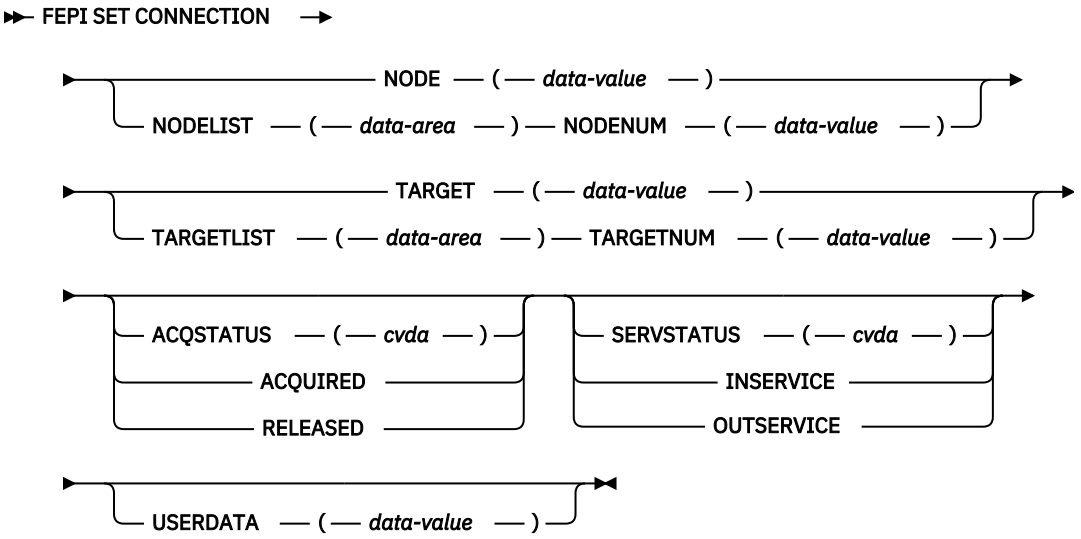
174
TARGET name already exists.

177
Application name already exists.

FEPI SET CONNECTION

Set a FEPI connection.

FEPI SET CONNECTION



Notes:

Description

FEPI SET CONNECTION controls the use of FEPI connections. Lists may be used to set more than one connection at a time; all connections in the list are set to the same state. The command completes immediately, although the requested settings may not be achieved until later.

Options

ACQSTATUS(cvda)
specifies the acquire state of the connection; that is, whether a session should be established ('bound') or not ('unbound'). The relevant CVDA values are:

ACQUIRED
The connection is to have a session established (that is, 'bound'). The state is ACQUIRING until this is completed.

RELEASED
The connection is to have its session ended (that is, 'unbound'), when usage of the connection by all owned conversations ends. (An unowned conversation on the connection is ended immediately. See the STATE option of "FEPI INQUIRE CONNECTION" on page 144.) The state is RELEASING until this is completed.

If this option is not coded, the acquire state is not changed.

NODE(8-character data-value)
specifies the node name that identifies a connection.

NODELIST(data-area)
specifies a contiguous array of 8-character node names identifying connections.

NODENUM(fullword binary data-value)
specifies the number of node names in NODELIST, in the range 1–256.

SERVSTATUS(cvda)
specifies the service state of the connection; that is, whether the connection can be used for a conversation or not. The relevant CVDA values are:

INSERVICE
Allows usage of the connection in a conversation.

OUTSERVICE
Stops usage of a connection for any new conversation, although existing conversations are unaffected. The service state is GOINGOUT until these conversations end.

If this option is not coded, the service state is not changed.

TARGET(8-character data-value)
Specifies the target name that identifies a connection.

TARGETLIST(data-area)
specifies a contiguous array of 8-character target names identifying a connection or connections.

TARGETNUM(fullword binary data-value)
specifies the number of target names in TARGETLIST, in the 1–256.

USERDATA(64-character data-value)
Specifies optional user data relating to the connections; it is not used by FEPI. It replaces any previous user data that was set.

Conditions

INVREQ
RESP2 values:

110
SERVSTATUS value not valid.

111
ACQSTATUS value not valid.

116
TARGET name unknown.

117
NODE name unknown.

118
Unknown connection (TARGET and NODE names are known but not connected in any pool).

119
The command failed for one or more items in the list.

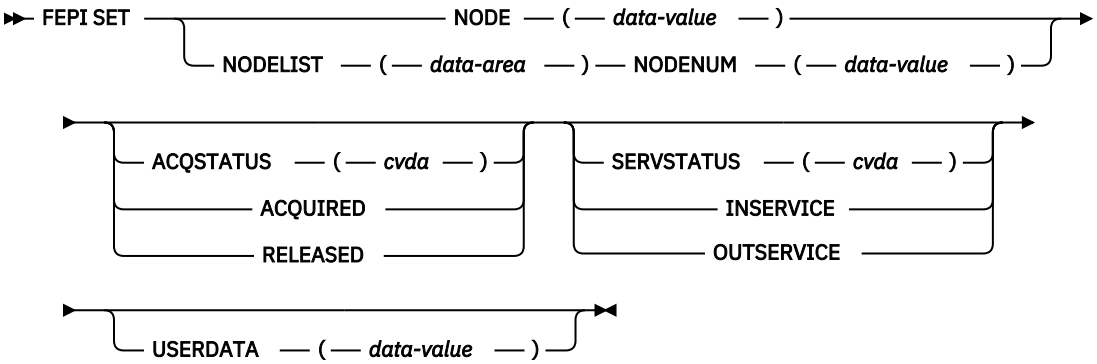
130
TARGETNUM value out of range.

131
NODENUM value out of range.

FEPI SET NODE

Control the use of FEPI nodes.

FEPI SET NODE



Notes:

Description

FEPI SET NODE controls the use of FEPI nodes. Lists may be used to set more than one node at a time; all nodes in the list are set to the same state. The function completes immediately, although the requested settings may not be achieved until later.

Options

ACQSTATUS(cvda)
specifies the acquire state of the node; that is, whether its z/OS Communications Server ACB should be opened or closed. The relevant CVDA values are:

ACQUIRED
The z/OS Communications Server ACB for the node is to be opened and ‘set logon start' is to be done. The state is ACQUIRING until this is completed.

RELEASED
The z/OS Communications Server ACB for the node is to be closed when usage of the node by any conversation ends. The state is RELEASING until this is completed.

If this option is not coded, the acquire state is not changed.

NODE(8-character data-value)
specifies the node to be set.

NODELIST(data-area)
specifies a contiguous array of 8-character node names to be set.

NODENUM(fullword binary data-value)
specifies the number of node names in NODELIST, in the range 1–256.

SERVSTATUS(cvda)
specifies the service state of the node; that is, whether the node can be used for a conversation or not. The relevant CVDA values are:

INSERVICE
Allows usage of the node in a conversation.

OUTSERVICE
Stops usage of a node for any new conversation, although existing conversations are unaffected. The service state is GOINGOUT until these conversations end.

If this option is not coded, the service state is not changed.

USERDATA(64-character data-value)
Specifies optional user data relating to the nodes; it is not used by FEPI. It replaces any previous user data that was set.

Conditions

INVREQ
RESP2 values:

110
SERVSTATUS value not valid.

111
ACQSTATUS value not valid.

117
NODE name unknown.

119
The command failed for one or more items in the list.

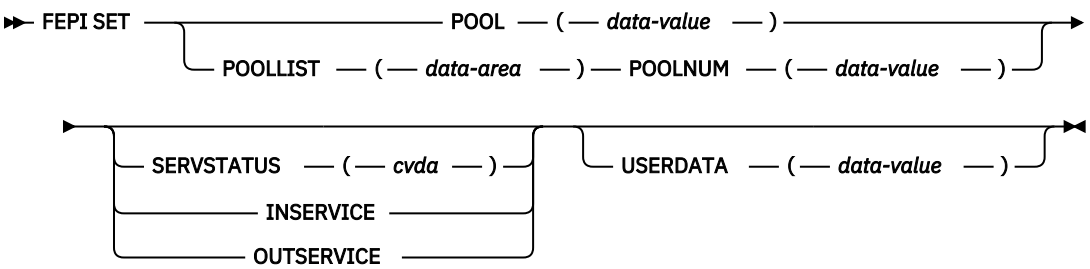
131
NODENUM value is out of range.

174
The z/OS Communications Server OPEN ACB failed.

FEPI SET POOL

Set the use of FEPI pools.

FEPI SET POOL



Notes:

Description

FEPI SET POOL controls the use of FEPI pools. Lists may be used to set more than one pool at a time; all pools in the list are set to the same state. The function completes immediately, although the requested settings may not be achieved until later.

Options

POOL(8-character data-value)
specifies the pool to be set.

POOLLIST(data-area)
specifies a contiguous array of 8-character pool names to be set.

POOLNUM(fullword binary data value)
specifies the number of pool names in POOLLIST, in the range 1–256.

SERVSTATUS(cvda)
specifies the service state of the pool; that is, whether the pool can be used for a conversation or not. The relevant CVDA values are:

INSERVICE
Allows usage of the pool in a conversation.

OUTSERVICE
Stops usage of a pool for any new conversation, although existing conversations are unaffected. The service state is GOINGOUT until these conversations end.

If this option is not coded, the service state is not changed.

USERDATA(64-character data-value)
Specifies optional user data relating to the pools; it is not used by FEPI. It replaces any previous user data that was set.

Conditions

INVREQ
RESP2 values:

110
SERVSTATUS value not valid.

115
POOL name unknown.

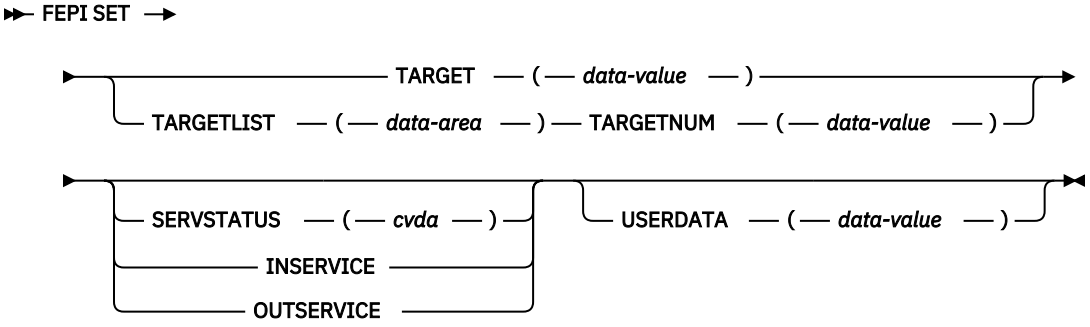
119
The command failed for one or more items in the list.

132
POOLNUM value is out of range.

FEPI SET TARGET

Set the use of FEPI targets.

FEPI SET TARGET



Notes:

Description

FEPI SET TARGET controls the use of FEPI targets. Lists may be used to set more than one target at a time; all targets in the list are set to the same state. The function completes immediately, although the requested settings may not be achieved until later.

Options

SERVSTATUS(cvda)
specifies the service state of the target; that is, whether the target can be used for a conversation or not. The relevant CVDA values are:

INSERVICE
Allows usage of the target in a conversation.

OUTSERVICE
Stops usage of a target for any new conversation, although existing conversations are unaffected. The service state is GOINGOUT until these conversations end.

If this option is not coded, the service state is not changed.

TARGET(8-character data-value)
specifies the name of the target to be set.

TARGETLIST(data-area)
specifies a contiguous array of 8-character target names to be set.

TARGETNUM(fullword binary data-value)
specifies the number of target names in TARGETLIST, in the 1–256.

USERDATA(64-character data-value)
Specifies optional user data relating to the targets; it is not used by FEPI. It replaces any previous user data that was set.

Conditions

INVREQ
RESP2 values:

110
SERVSTATUS value not valid.

116
TARGET name unknown.

119
The command failed for one or more items in the list.

130
TARGETNUM value is out of range.

FEPI SP NOOP

FEPI SP NOOP

Notes:

Description

FEPI SP NOOP has no effect.

Options

None.

Conditions

None specific to this command.

Transient data queue records

In response to various unexpected events, FEPI writes a record, describing the event and its circumstances, to a transient data (TD) queue.

Such events include:

- Errors in functions initiated by a system programming command
- Errors for items in a list on a system programming command
- Events unrelated to any command.

If the event relates to a specific pool, the record is written to the queue specified by EXCEPTIONQ for that pool; if EXCEPTIONQ was not specified, no record is written. If the event does not relate to a specific pool, the record is written to queue CSZX. In all cases, if the appropriate TD queue does not exist or if it is not defined as non-recoverable, the record is lost.

The format of the record is as follows. The copy books DFHSZAPA, DFHSZAPO, DFHSZAPC, and DFHSZAPP (according to your programming language) provide declarations for this record structure.

Type	Description
DATATYPE	Fullword binary data-area
EVENTTYPE	CVDA
EVENTVALUE	Fullword binary data-area
EVENTDATA	8-character data-area
Reserved	4-character data-area
POOL	8-character data-area
TARGET	8-character data-area
NODE	8-character data-area

Type	Description
CONVID	8-character data-area
DEVICE	CVDA
FORMAT	CVDA
Reserved	8-character data-area.

Fields

CONVID(8-character data-area)
the conversation ID for which the event occurred; null if not applicable.

DATATYPE(fullword binary data-area)
identifies the type and structure of the data. A value of 2 indicates FEPI TD queue data.

DEVICE(cvda)
the device type of the conversation for which the event occurred (the values are as for FEPI INQUIRE POOL); zero if not applicable.

EVENTDATA(8-character data-area)
contains data about the event:

Event	Data
ACQFAIL	2 fullword binary numbers: • z/OS Communications Server reason code • Count
SESSIONFAIL	2 fullword binary numbers: • z/OS Communications Server reason code • Count
SESSIONLOST	2 fullword binary numbers: • z/OS Communications Server reason code • Count
Others	Nulls

If the count is nonzero, it indicates the number of times the node acquire or session start has failed; it will be tried again. A zero count indicates that several failures have occurred and that there will be no further attempts to acquire the node or start the session.

EVENTTYPE(cvda)
indicates what the event was.

Exceptional events queued to common TD queue CSZX:

ACQFAIL	A node could not be acquired (its z/OS Communications Server ACB could not be opened).
DISCARDFAIL	A resource in a list could not be discarded by FEPI DISCARD.
INSTALLFAIL	A resource in a list could not be installed by FEPI INSTALL.
SESSION	An unsolicited bind was received.
SETFAIL	A connection or resource in a list could not be set by FEPI SET or FEPI INSTALL.

Exceptional events queued to pool-specific TD queue:

ADDFAIL	A connection in a list could not be added to the pool by FEPI ADD.
DELETEFAIL	A connection in a list could not be deleted from the pool by FEPI DELETE.
SESSIONFAIL	Session could not be started.
SESSIONLOST	Active session was lost.

EVENTVALUE(fullword binary data area)
provides further information about the event. Values are:

Event	Value
ACQFAIL	0
ADDFAIL	The RESP2 value describing the failure, as given in the description of the FEPI ADD command
DELETEFAIL	The RESP2 value describing the failure, as given in the description of the FEPI DELETE command
DISCARDFAIL	The RESP2 value describing the failure, as given in the description of the FEPI DISCARD command
INSTALLFAIL	The RESP2 value describing the failure, as given in the description of the FEPI INSTALL command
SESSION	0
SESSIONFAIL	The RESP2 value describing the communication failure; it can be any of the RESP2 values in the range 182–199.
SESSIONLOST	The RESP2 value describing the communication failure; it can be any of the RESP2 values in the range 182–199.
SETFAIL	The RESP2 value describing the failure, as given in the description of the FEPI SET command

FORMAT(cvda)
the data format of the conversation for which the event occurred (the values being as for FEPI INQUIRE POOL); zero if not applicable.

NODE(8-character data-area)
the name of the node for which the event occurred; nulls if not applicable.

POOL(8-character data-area)
the name of the pool for which the event occurred; nulls if not applicable.

TARGET(8-character data-area)
the name of the target for which the event occurred; nulls if not applicable. For the SESSION event, it is the z/OS Communications Server application name of the back-end system, rather than the FEPI target name.

Reserved
nulls.

CVDA and RESP2 values for FEPI commands

This reference information lists the CVDA and RESP2 values returned by FEPI commands.

It contains:

- “FEPI CVDAs and numeric values in numerical sequence” on page 178
- “FEPI RESP2 values” on page 181.

FEPI CVDAs and numeric values in alphabetical sequence

The CVDA values used or returned by the FEPI commands are listed in alphabetical sequence.

Table 13. CVDA values in alphabetical sequence

CVDA	**value**
ACQFAIL	515
ACQUIRED	69
ACQUIRING	71
ADDFAIL	519
ALARM	501
APPLICATION	559
ATTENTION	524
BEGINSESSION	510
CANCEL	526
CD	491
DATA	508
DATASTREAM	543
DEFRESP1	497
DEFRESP1OR2	528
DEFRESP2	498
DEFRESP3	499
DELETEFAIL	520
DISCARDFAIL	513
EB	490
EXCEPTRESP	523
FMH	502
FORCE	342
FORMATTED	542
FREE	85
GOINGOUT	172
HOLD	163
INBOUND	547
INOUT	532
INPUT	226
INSERVICE	73

Table 13. CVDA values in alphabetical sequence (continued)	
CVDA	**value**
INSTALLED	550
INSTALLFAIL	512
INVALID	359
LIC	493
LOSE	544
LUP	541
LUSTAT	525
MDT	506
MORE	492
NEGATIVE	530
NEWSESSION	485
NOALARM	500
NOCONVERT	734
NOCONV	556
NOFMH	503
NOMDT	507
NOMSGJRNL	531
NONE	496
NORMALRESP	522
NOSTSN	487
NOTINBOUND	546
NOTINSTALLED	551
OLDSESSION	486
OUTPUT	227
OUTSERVICE	74
PENDBEGIN	558
PENDDATA	560
PENDFREE	86
PENDPASS	565
PENDRELEASE	562
PENDSTART	561
PENDSTSN	557
PENDUNSOL	564
POSITIVE	529

Table 13. CVDA values in alphabetical sequence (continued)	
CVDA	**value**
PROTECTED	504
RELEASE	563
RELEASED	70
RELEASING	549
RESET	290
RTR	527
RU	494
SESSION	372
SESSIONFAIL	517
SESSIONLOST	516
SETFAIL	514
SHUTDOWN	288
STSN	509
STSNSET	488
STSNTEST	489
TASK	233
TIMEOUT	511
TPS55M2	552
TPS55M3	553
TPS55M4	554
T3278M2	533
T3278M3	534
T3278M4	535
T3278M5	536
T3279M2	537
T3279M3	538
T3279M4	539
T3279M5	540
UNPROTECTED	505
UNSOLDATA	521
WIN	545

Related reference

FEPI CVDAs and numeric values in numerical sequence
The CVDA values used or returned by the FEPI commands are listed in numerical sequence.

FEPI RESP2 values

RESP2 values are used in the EVENTVALUE area of FEPI transient data queue records and returned by the RESP2 option of FEPI commands.

FEPI CVDAs and numeric values in numerical sequence

The CVDA values used or returned by the FEPI commands are listed in numerical sequence.

Table 14. CVDA values in numerical sequence

value	CVDA
69	ACQUIRED
70	RELEASED
71	ACQUIRING
73	INSERVICE
74	OUTSERVICE
85	FREE
86	PENDFREE
163	HOLD
172	GOINGOUT
226	INPUT
227	OUTPUT
233	TASK
288	SHUTDOWN
290	RESET
342	FORCE
359	INVALID
372	SESSION
485	NEWSESSION
486	OLDSESSION
487	NOSTSN
488	STSNSET
489	STSNTEST
490	EB
491	CD
492	MORE
493	LIC
494	RU
496	NONE
497	DEFRESP1
498	DEFRESP2

Table 14. CVDA values in numerical sequence (continued)	
value	**CVDA**
499	DEFRESP3
500	NOALARM
501	ALARM
502	FMH
503	NOFMH
504	PROTECTED
505	UNPROTECTED
506	MDT
507	NOMDT
508	DATA
509	STSN
510	BEGINSESSION
511	TIMEOUT
512	INSTALLFAIL
513	DISCARDFAIL
514	SETFAIL
515	ACQFAIL
516	SESSIONLOST
517	SESSIONFAIL
519	ADDFAIL
520	DELETEFAIL
521	UNSOLDATA
522	NORMALRESP
523	EXCEPTRESP
524	ATTENTION
525	LUSTAT
526	CANCEL
527	RTR
528	DEFRESP1OR2
529	POSITIVE
530	NEGATIVE
531	NOMSGJRNL
532	INOUT
533	T3278M2

Table 14. CVDA values in numerical sequence (continued)	
value	**CVDA**
534	T3278M3
535	T3278M4
536	T3278M5
537	T3279M2
538	T3279M3
539	T3279M4
540	T3279M5
541	LUP
542	FORMATTED
543	DATASTREAM
544	LOSE
545	WIN
546	NOTINBOUND
547	INBOUND
549	RELEASING
550	INSTALLED
551	NOTINSTALLED
552	TPS55M2
553	TPS55M3
554	TPS55M4
556	NOCONV
557	PENDSTSN
558	PENDBEGIN
559	APPLICATION
560	PENDDATA
561	PENDSTART
562	PENDRELEASE
563	RELEASE
564	PENDUNSOL
565	PENDPASS
734	NOCONVERT

Related reference

FEPI CVDAs and numeric values in alphabetical sequence
The CVDA values used or returned by the FEPI commands are listed in alphabetical sequence.

FEPI RESP2 values

RESP2 values are used in the EVENTVALUE area of FEPI transient data queue records and returned by the RESP2 option of FEPI commands.

FEPI RESP2 values

RESP2 values are used in the EVENTVALUE area of FEPI transient data queue records and returned by the RESP2 option of FEPI commands.

For details of the error conditions and related RESP2 values for each FEPI command, see the command definitions in Appendix B, “FEPI system programming reference,” on page 137 and Appendix A, “FEPI application programming reference,” on page 95.

Declarations for the RESP2 values are provided in the following copy books:

- DFHSZAPA for Assembler language
- DFHSZAPO for COBOL
- DFHSZAPP for PL/I
- DFHSZAPC for C.

Table 15. RESP2 values

Numerical order	Data
1	INQUIRE START, NEXT, or END command not valid here: **START** Browse of this resource type already in progress **NEXT** INQUIRE START not issued **END** INQUIRE START not issued.
2	All resource definitions have been retrieved.
10	Command bypassed by user exit.
11	FEPI not installed or not active.
12	CICS shutting down, command not allowed.
13	FEPI not available.
14	FEPI busy or cannot get storage.
15	Unknown command.
16	Internal problem.
17	FEPI cannot get storage for user exit parameters.
18	Command failed because of operator or system action.
30	POOL name not known.
31	POOL name out of service.
32	TARGET name not known.
33	TARGET name out of service.
34	TARGET name required but not specified.
35	Pool name is unsuitable for temporary conversations. It has CONTENTION(LOSE) or INITIALDATA(INBOUND) but no begin-session handler.
36	No suitable session available and in service.

Table 15. RESP2 values (continued)

Numerical order	Data
40	[FROM]FLENGTH value is negative, zero, or more than MAXFLENGTH value for pool.
41	ESCAPE value not valid.
50	Inbound data with ‘begin bracket' to be received.
51	Attention identifier (AID) not valid.
52	Cursor position not valid.
53	Code points in formatted data not valid.
54	Attribute positions or values in send data not valid.
55	Key stroke escape sequence in send data not valid.
56	Field validation (mandatory fill, mandatory enter, trigger) failed.
57	Input is inhibited.
58	z/OS Communications Server VTAM SEND failed.
59	DBCS data rules violated.
60	MAXFLENGTH value negative, or greater than MAXFLENGTH value for pool.
61	FLENGTH value negative or greater than 128.
62	TRANSID name not valid.
63	TERMID name not valid.
70	FIELDLOC or FIELDNUM value negative or not valid.
71	z/OS Communications Server VTAM RECEIVE failed.
72	RECEIVE FORMATTED processing found invalid, or unexpected data while interpreting the 3270 data steam for a WRITE, ERASE/WRITE, ERASE/WRITE ALTERNATE, or WRITE STRUCTURED FIELD command code.
80	CONTROL value not valid.
81	VALUE not valid: omitted when required; included when not required; or unsuitable for specified CONTROL.
82	SENSEDATA option omitted when required, or specified when not required.
90	Definite response type did not match what was required.
91	Only NORMALRESP or EXCEPTRESP allowed at this point in conversation.
92	Response to STSN SET was not positive.
93	Only STSN allowed at this point in conversation.
94	Only STSN or NORMALRESP allowed at this point in conversation.
95	CONTROL value not allowed at this point in conversation.
100	Not authorized to issue command.
110	SERVSTATUS value not valid.
111	ACQSTATUS value not valid.
115	POOL name not known.
116	TARGET name not known.

Table 15. RESP2 values (continued)

Numerical order	Data
117	NODE name not known.
118	Unknown connection (TARGET and NODE names known, but not in a common POOL).
119	Request failed for one or more items in list. Detailed errors reported to TD queue for monitor to handle.
130	TARGETNUM value negative, zero, or not valid.
131	NODENUM value negative, zero, or not valid.
132	POOLNUM value negative, zero, or not valid.
140	DEVICE value not valid.
141	CONTENTION value not valid.
142	INITIALDATA value not valid.
143	UNSOLDATACK value not valid.
144	MSGJRNL value not valid.
150	FORMAT value not valid or unsuitable for specified device.
153	STSN name not valid or STSN unsuitable for specified device.
154	BEGINSESSION value not valid.
155	UNSOLDATA value not valid.
156	EXCEPTIONQ value not valid.
157	FJOURNALNUM value not valid.
158	MAXFLENGTH value not valid.
159	ENDSESSION name not valid.
160	PROPERTYSET name not valid.
162	POOL name not valid.
163	NODE name not valid.
164	TARGET name not valid.
167	APPL name not valid.
170	PROPERTYSET name already exists.
171	PROPERTYSET name not known.
172	POOL name already exists.
173	NODE name already exists.
174	TARGET name already exists.
175	Connection already exists.
176	z/OS Communications Server VTAM OPEN NODE failed.
177	z/OS Communications Server APPLID already known.
178	FJOURNALNAME value not valid.
182	Session unbound, unrecoverable.
183	Session unbound, recoverable.

Table 15. RESP2 values (continued)

Numerical order	Data
184	Session unbound, error.
185	Session unbound, bind coming.
186	Session unbound.
187	Lost terminal.
188	CLEANUP, abnormal.
189	CLEANUP.
190	UNBIND error.
191	SETUP error.
192	SSCP error.
193	SLU error.
194	PLU error.
195	BIND error.
196	CINIT error.
197	REQSESS error.
198	REQSESS inhibited.
199	REQSESS not available.
210	Option not valid for SLU P.
211	Option not valid for SLU2.
212	Wrong data format for conversation.
213	Command has timed out.
214	CICS shutting down, conversation should be ended.
215	Session lost.
216	Error occurred on previous SEND command.
220	SEND or CONVERSE command not allowed at this point in conversation.
221	RECEIVE command not allowed at this point in conversation.
223	START command not allowed at this point in conversation.
224	Only ISSUE or FREE allowed at this point in conversation.
230	SNA CLEAR command received.
231	SNA CANCEL command received.
232	SNA CHASE command received.
233	Exception response received.
234	Exception request received.
240	Conversation ID unknown or not owned by task.
241	TIMEOUT value negative or not valid.
250	Passticket not built successfully.

Table 15. RESP2 values (continued)

Numerical order	Data
251	CICS ESM interface not initialized.
252	Unknown return code in ESMRESP from the ESM.
253	Unrecognized response from CICS security modules.
254	Function unavailable.
259	No signed-on user.

Related reference

FEPI CVDAs and numeric values in alphabetical sequence
The CVDA values used or returned by the FEPI commands are listed in alphabetical sequence.

FEPI CVDAs and numeric values in numerical sequence
The CVDA values used or returned by the FEPI commands are listed in numerical sequence.

Appendix C. FEPI samples

The FEPI samples show you how to set up and use FEPI. Although the samples are copyrighted, you can use and copy them freely for educational purposes to help you write FEPI applications.

The samples form an integrated set. The setup program provides the FEPI resource definitions that the other samples use. The monitor and the various handlers support and complement the access programs, to form a complete FEPI communication package, just as you need to provide.

The two back-end programs, one for CICS and one for IMS, provide applications for the front-end programs to access. The back-end CICS program is for access by the front-end SLU2 mode programs, and the back-end IMS program is for access by the front-end SLU P mode programs; no SLU2 mode access to IMS is provided. Although the back-end programs are supplied in source form, it is not necessary for you to understand the internal logic - only the external operations, as is the case for a “real” existing back-end application.

The FEPI sample front-end and back-end transactions assume that the datastream sent from the back-end application is received unaltered by the front-end application. For example, FEPI samples may perform unexpectedly if the datastreams are compressed after having been sent from the back-end application.

Note: These are samples designed for illustration purposes. For any particular circumstance, you must consider exactly what your requirements are.

About the FEPI samples

Each FEPI sample program performs a specific function and is associated with a CICS transaction. A subset of the sample programs is available in each of the supported programming languages.

The programs and their names are given in Table 16 on page 187.

Table 16. Sample programs and their names

Description	Transaction name	COBOL	Assembler	PL/I	C
Programs:					
Setup	CZXS	DFH0VZXS	DFH0AZXS		DFH0CZXS
Monitor and unsolicited data handler	CZUX	DFH0VZUX			
Begin-session handler	CZUC	DFH0VZUC			
3270 data stream pass-through	CZTD	DFH0VZTD	DFH0AZTD		
Key stroke CONVERSE	CZTK	DFH0VZTK		DFH0PZTK	DFH0CZTK
Screen image SEND and START	CZTS	DFH0VZTS			
Screen image RECEIVE and EXTRACT	CZTR	DFH0VZTR			
End-session handler	CZUU	DFH0VZUU			
SLU P, one-out, one-in	CZPS	DFH0VZPS	DFH0AZPS		
SLU P, pseudoconversational	CZPA	DFH0VZPA	DFH0AZPA		

Table 16. Sample programs and their names (continued)

Description	Transaction name	COBOL	Assembler	PL/I	C
STSN handler	CZQS	DFH0VZQS	DFH0AZQS		
Back-end CICS	CZBC		DFH0AZBC		
Back-end IMS	CZBI		DFH0AZBI		

VS COBOL II sample restrictions: The following COBOL samples can only be compiled with Release 3 and later versions of the VS COBOL II compiler:

- DFH0VZUC
- DFH0VZUX
- DFH0VZPS
- DFH0VZPA

Copy books:

Customization data	DFH0BZCO	DFH0BZCA	DFH0BZCP	DFH0BZCC
Messages and other text	DFH0BZMO	DFH0BZMA	DFH0BZMP	DFH0BZMC
Key stroke map	DFH0BZ1O		DFH0BZ7P	DFH0BZ6C
Send/receive map	DFH0BZ2O			
Back-end CICS map		DFH0BZ3A		
SLU P, one-out, one-in map	DFH0BZ4O	DFH0BZ8A		
SLU P, pseudoconversational map	DFH0BZ5O	DFH0BZ9A		

Maps:

Key stroke	DFH0MZ1		DFH0MZ7	DFH0MZ6
Send/receive	DFH0MZ2			
SLU P, one-out, one-in	DFH0MZ4	DFH0MZ8		
SLU P, pseudoconversational	DFH0MZ5	DFH0MZ9		
Back-end CICS		DFH0MZ3		

There are also some sample resource definitions. Sample definitions for front-end and back-end CICS regions are in the RDO groups DFH$0AZ, DFH$0BZ, DFH$0CZ, DFH$0BZ, DFH$0PZ, and DFH$0VZ. A sample definition for a back-end IMS region is in DFH0IZRI. A sample definition of a CICS TD queue, DFH0IZRQ, is in the DFHDCTG RDO group.

Table 17 on page 188 shows you which samples illustrate which functions.

Table 17. Functional cross-reference for sample programs

Functions	Samples (Last two letters of sample program name. See notes.)										
—	TD	TK	TS	TR	PA	PS	QS	UC	UU	UX	XS
SLU2	X	X	X	X	X	X	X				
SLU P	X	X	X	X	X	X					

Table 17. Functional cross-reference for sample programs (continued)

Functions	**Samples (Last two letters of sample program name. See notes.)**										
—	**TD**	**TK**	**TS**	**TR**	**PA**	**PS**	**QS**	**UC**	**UU**	**UX**	**XS**
Data stream	X	X	X	X	X	X					
Screen-image	X	X	X	X							
Key stroke	X	X									
ALLOCATE	X	X	X	X							
ALLOCATE with PASSCONVID	X	X	X	X	X	X	X				
EXTRACT STSN	X										
EXTRACT FIELD	X	X									
SEND	X	X									
START	X	X									
RECEIVE	X	X	X	X							
CONVERSE	X	X	X								
CONVERSE with POOL	X										
ISSUE	X										
FREE	X	X	X	X	X	X	X	X			
FREE with PASS	X	X									
INSTALL	X										
ADD	X										
Start data	X	X	X	X	X						
TD queue data	X										
One-out one-in	X										
Conversational	X	X									
Pseudo- conversational	X	X	X								
Assembler language	X	X	X	X	X						
COBOL	X	X	X	X	X	X	X	X	X	X	X
C	X	X									
PL/I	X										

Table 17. Functional cross-reference for sample programs (continued)

Functions	Samples (Last two letters of sample program name. See notes.)										
—	TD	TK	TS	TR	PA	PS	QS	UC	UU	UX	XS

Notes:

TD
Data stream

TK
Key stroke

TS
Screen image send/start

TR
Screen image receive

PA
SLU P pseudoconversational

PS
SLU P one-out, one-in

QS
STSN

UC
Begin session

UU
End session

UX
Monitor, unsolicited data

XS
Setup

FEPI EXTRACT CONV, SET/INQUIRE/browse, and DELETE/DISCARD commands are not illustrated in the sample programs.

The back-end CICS program

This program is the CICS back-end application used by the FEPI sample programs.

Module name	DFH0AZBC
Transaction name	CZBC
Abend code	USZA
Map name	DFH0MZ3

On the first invocation of the transaction, a map is sent to the terminal.

When there is input from the terminal, CICS invokes the transaction again. The customer data for the customer number from the input is found and sent to the terminal, and further input is awaited. PF3 or CLEAR ends the transaction.

Certain customer numbers cause special processing such as abends and delays, to show how a front-end application could manage such events. The valid customer numbers are:

0001-0005
Normal

0006
Delayed response

0007
Abend before send

0008
Abend after send.

```
CZBC                  Customer Inquiry
Please type a customer number in the range 1 to 9999, then Enter.
Customer Number . . . . .
         Name . . . . . :
         Balance. . . . :
         Address. . . . :
Last Transaction Date . :
         F3=EXIT to CICS
```

Figure 10. CZBC transaction: customer inquiry

Program logic

```
  Main procedure:
     Set up exception condition handling:
         Map error - SEND_NEW_MAP
         CLEAR/PF3 - END_PROG
     Test COMMAREA
     If transaction not previously invoked
         Call SEND_NEW_MAP
     RECEIVE map
     If customer number not valid
         SEND message
         RETURN
     If customer type is 'ABEND before MAP'
         ABEND
     Build map with customer data
     If customer type is 'LONG DELAY'
         DELAY
     SEND map
     If customer type is 'ABEND after MAP'
         ABEND
     RETURN
  SEND_NEW_MAP routine:
     SEND new map
     RETURN
  END_PROG routine:
     Clear terminal
     RETURN
```

The back-end IMS program

This program is the IMS back-end application used by the FEPI sample programs.

Module name	DFH0AZBI
Transaction name	CZBI

This is a simple IMS back-end response mode program that is driven by input from a front-end FEPI application. It modifies the time stamp in the input message and returns the message to the front-end application.

IMS schedules this transaction when an input message is queued for it. It addresses the I/O PCB, DLI call function, and I/O area to build the parameter list for the GU call to retrieve the queued input message.

The time field of the input message is updated and the program then issues an ISRT call to place the message on the output queue. IMS then sends the output message to the front-end FEPI application.

Output messages from this program are all prefixed with a 5-byte function management header.

If any errors occur, the program ends with a nonzero return code.

Program logic

```
        GETMAIN storage areas for reentrancy
        Address PCB
        Issue GU call to get input message
        Use TIME to obtain system time
        Update I/O area
        Issue ISRT call to send output message
        RETURN
```

Setup program

This program installs the resources—property sets, nodes, targets, and pools—that are used by the FEPI sample programs.

Module names	DFH0VZXS, DFH0AZXS, DFH0CZXS
Transaction name	CZXS

The definitions of each of these resources are organized so that they can easily be changed. They are kept separate from the processing that does the installation, and there is no hard-coding of values in the CICS commands. There are four main tables, holding details of each resource type. This enables the resources to be changed by repeating sets of definitions which are in an easy-to-understand form. If required, you can change the program to obtain the resource definitions from a file.

The resources defined are:

```
   Pool     Property set   Nodes                  Targets
   POOL1    PROPSET1       NODE1  NODE2  NODE3    TARGET1
                           NODE4  NODE5
   POOL2    PROPSET2       NODE6  NODE7  NODE8    TARGET1
                           NODE9  NODE10
   POOL3    PROPSET3       NODE1  NODE2  NODE3    TARGET2
                           NODE4  NODE5
```

You must customize these definitions to match the requirements of your system. If you do, you may also need to change the definitions in the sample customization constants copy book DFH0BZCx. You do not need to change any other samples—you need recompile them.

Each table is processed in turn. Nodes and targets are organized into lists for reasons of efficiency. Details of resource installation are written to the CICS log automatically by FEPI.

On completion, a message is sent. The setup program would typically be started by a PLT program, in which case the message goes to the CICS log. It can, however, be invoked from a terminal and, in this case, the message is sent to the terminal.

For clarity, error checking is minimal. In particular, the FEPI INSTALL commands do not check errors at all, because FEPI reports any errors that occur to the FEPI transient data queue, and they are then recorded by the sample monitor program.

Program logic

```
For each property set in table
   FEPI INSTALL PROPERTYSET
For each node in table
   Add node to list
FEPI INSTALL NODELIST
For each target in table
   Add target to list
FEPI INSTALL TARGETLIST
For each pool in table
   Start new lists of nodes and targets
   For each entry within pool definition
       If node, add details to node list
       If target, add details to target list
   FEPI INSTALL POOL with NODELIST and TARGETLIST
Send completion message
RETURN
```

Monitor and unsolicited data-handler

This program monitors unexpected events and handles unsolicited data for the FEPI sample programs.

Module name	DFH0VZUX
Transaction name	CZUX
TS queue name	MONITOR

This transaction handles:

- Unexpected events that are reported by FEPI to a TD queue, which triggers this transaction
- Unsolicited data from a back-end system, for which FEPI starts this transaction.

Because the event descriptions provided by FEPI and the processing required is basically the same for both cases, this common program is used.

ASSIGN STARTCODE is used to determine how the transaction was started, and ASSIGN QNAME to determine what TD queue triggered it. Details of the event are in the start data or the TD queue record as the case may be.

For illustrative purposes, all events are handled similarly by reporting their details to a TS queue named MONITOR, which can be browsed using CEBR. In practice, for any of the events you can do whatever extra or different processing you require, or (except for unsolicited data) you can ignore the event.

For unsolicited data, the conversation started by FEPI must be accessed so that FEPI knows that the data is being handled. The data itself should be received, or else FEPI ends and restarts the session. For illustration purposes, this program discards the data; in practice, you will probably want to process the data in some way.

However, if you did want to discard such data, you should specify no unsolicited-data handling and use the UNSOLDATACK property to tell FEPI what action to take, as is done for SLU P mode by these samples.

The general format of the TS queue records is:

```
date time CZUX description
          Event type..ACQFAIL      Pool........POOLNAME
          Target......TGTNAME      Node........NODENAME
          Device......T3278M2      Event data..X'00000000'
          Format......0            Event value.176
```

The actual details for each event vary. Events with similar details are grouped together for processing. The groups are:

- Unknown event—an event that is not recognized
- Unsolicited data
- Session lost
- Standard events—all other events.

The groups also determine any additional processing needed. Only unsolicited data needs any processing.

If any errors occur, they are reported to the TS queue.

Program logic

```
  Main procedure:
    Determine how transaction was started using ASSIGN
    If started with data by FEPI
        RETRIEVE start data
    If triggered by TD queue
        READ the queue record
    Otherwise
        Report start code
        RETURN
    TD-LOOP:
```

```
    Locate event type
    Locate device type
    Build description of event:  event type, device type,
        format, event value, date/time, transaction
    Call UNKNOWN-EVENT, UNSOLDATA, STANDARD-EVENT, or
        SESSION-LOST according to event group
    If triggered by TD queue
        READ the next queue record
        If one exists, loop to TD-LOOP
    RETURN
  UNKNOWN-EVENT routine:
    Write event details to TS queue:  description and
        event value
  UNSOLDATA routine:
    Write event details to TS queue:  description, event
        type, pool, target, and node
    Access conversation using FEPI ALLOCATE with PASSCONVID
    FEPI RECEIVE unsolicited data
    Free conversation
    Handle data as required
```

Begin session

This program prepares sessions for use by the FEPI sample application programs.

Module name	DFH0VZUC
Transaction name	CZUC
TS queue name	SESSION

The CZUC transaction is started by FEPI when it begins a new session.

The conversation started by FEPI must be accessed so that FEPI knows that the event is being handled. The processing required depends on the data mode and type that the session uses (this is obtained from the start data), and whether the back-end system is IMS or CICS.

For SLU P mode (necessarily IMS), processing depends entirely on local requirements, and is typically used for handling security applications. For illustration purposes, this program gets and discards the initial data. Note that the setup for these samples does not specify a begin-session transaction for SLU P mode.

For SLU2 mode with CICS using formatted data, there is a CICS “good morning” message waiting. The message is received, and the back-end screen is cleared and ready for a transaction ID to be entered.

For SLU2 mode with CICS using data stream, there may be a “read partition” request waiting which requires a reply—for example, if your pool has device T3279Mx or TPS55Mx specified, or if the logon mode table being used has “extended data stream” specified). Then there is a CICS “good morning” message to be received. A reply is sent to any “read partition” query request, the “good morning” message is received, and the back-end screen is cleared and ready for a transaction ID to be entered.

For SLU2 mode with IMS, no processing is illustrated.

After the processing, the conversation is freed with the HOLD option, which leaves it ready for use by applications. A report is written to a TS queue named SESSION, which can be browsed using CEBR. The format of the TS queue records is:

```
date time CZUC Begin session completed
          RESP........0            RESP2.......0
          Target......TGTNAME      Node........NODENAME
          Pool........POOLNAME
```

If any errors occur, a report is written to the TS queue, and the conversation is freed with the RELEASE option, so that the session is ended.

Program logic

```
  Main procedure:
    RETRIEVE start data
```

```
    Access conversation using FEPI ALLOCATE with PASSCONVID
    Call PROCESS-LUP, PROCESS-FORMATTED, or
        PROCESS-DATASTREAM according to data mode and type
    Free conversation, keeping session
    Write event details to TS queue
    RETURN
  PROCESS-LUP routine:
    FEPI RECEIVE initial data
    Handle data as required
  PROCESS-FORMATTED routine:
    FEPI RECEIVE initial data
    Clear back-end screen and make ready for transaction ID
      to be entered, using FEPI CONVERSE
  PROCESS-DATASTREAM routine:
    FEPI RECEIVE
    If 'read partition' query
        FEPI CONVERSE query reply and get acknowledgment
        FEPI RECEIVE initial data
    Clear back-end screen and make ready for transaction ID
      to be entered, using FEPI CONVERSE
```

Key stroke CONVERSE

This sample program demonstrates using FEPI to obtain information from a back-end transaction using the key stroke data format.

Module names	DFH0VZTK, DFH0PZTK, DFH0CZTK
Transaction name	CZTK
Map names	DFH0MZ1, DFH0MZ6, DFH0MZ7

On the first invocation of the transaction, a map is sent to the front-end terminal.

When there is input from the front-end terminal, CICS invokes the transaction again. The customer number from the input is built into a key stroke sequence which runs a transaction at the back-end. The key strokes are sent and the results received using a FEPI ALLOCATE-CONVERSE-FREE command sequence. Information is extracted from the results and sent to the front-end terminal. Further input is then awaited.

When PF3 or CLEAR is received from the front-end terminal, the transaction ends. If there is an error, the front-end map is reset. These situations are detected using HANDLE CONDITION.

If the back-end sends a CICS message, it is sent on to the front-end terminal, and the transaction ends.

For clarity, error checking is minimal except for the FEPI commands. Note that the key stroke sequence used involves several attention keys, so that if the intermediate responses are not what is expected, the effects are unpredictable. According to your requirements, it may be advisable to send each attention sequence individually and to check each time that the results are as expected.

Screen

```
CZTK                  Customer Name and Address Inquiry
Please type a customer number in the range 1 through 9999, then Enter.
Customer Number . . . . .
         Name . . . . . :
         Address. . . . :
         F3=EXIT to CICS
```

Figure 11. CZTK transaction: customer name and address inquiry

Program logic

```
  MAIN procedure:
    Test COMMAREA
    If transaction not previously invoked
        Call SEND-NEW-MAP
    Set up exception condition handling:
        Map error - SEND-NEW-MAP
```

```
        CLEAR/PF3 - END-PROG
    RECEIVE MAP from front-end terminal
    Build key stroke sequence to:
        clear back-end screen
        type  transaction ID
        ENTER
        type the customer number
        ENTER
    FEPI ALLOCATE conversation with back-end
    FEPI CONVERSE to send key strokes to back-end and get
        the resulting screen image
    FEPI FREE     conversation with back-end
    If CICS message received from back-end
        SEND message to front-end terminal
        RETURN
    Get customer information from back-end screen image
    Build data for front-end terminal map
    SEND map data to front-end terminal
    RETURN TRANSID(CZTK) with COMMAREA
  SEND-NEW-MAP routine:
    SEND new map to front-end terminal
    RETURN TRANSID(CZTK) with COMMAREA
  END-PROG routine:
    Clear front-end terminal
    RETURN
```

Screen image SEND and START

This sample program demonstrates using FEPI to send formatted data to a back-end transaction, and requesting a transaction to be started when the reply to the data arrives.

Module name	DFH0VZTS
Transaction name	CZTS
Map name	DFH0MZ2

This program is the SEND part of a SEND-RECEIVE pair of programs, the RECEIVE part being DFH0VZTR.

On the first invocation of this send transaction, a map is sent to the front-end terminal.

When there is input from the front-end terminal, CICS invokes this send transaction again. The customer number is extracted from the input. Using FEPI ALLOCATE a conversation is started with the back-end system. Then FEPI SEND with screen image data is used to start a back-end transaction. FEPI START is issued to specify that the receive transaction is to be started when the back-end system replies.

In due course, the receive transaction is started and XCTLs to this send transaction. The customer number can now be sent to the back-end using FEPI SEND with screen image data. FEPI START is again issued.

The receive transaction gets the results from the back-end transaction and sends them on to the front-end terminal.

When there is more input from the front-end terminal, CICS invokes this transaction again. FEPI ALLOCATE with PASSCONVID is issued to gain ownership of the conversation and the customer number is sent to the back-end as before. The cycle continues until PF3 or CLEAR is received. These are passed on to the receive transaction (using the FEPI START user data) and to the back-end transaction to indicate that it is to end.

```
CZTS                  Customer Name and Balance Inquiry
Please type a customer number in the range 1 through 9999, then Enter.
Customer number . . . . .
         Name . . . . . :
         Balance. . . . :
         F3=EXIT to CICS
```

Figure 12. CZTS transaction: customer name and balance inquiry

Program logic

```
MAIN procedure:
  Test COMMAREA
  If transaction not previously invoked
      Call SEND-MAP
  If first customer number to process
      Call CONTINUE-CONVERSATION
  Set up exception condition handling:
      Map error - SEND-MAP
      PF3/CLEAR - CONTINUE-CONVERSATION
  RECEIVE MAP from front-end terminal
  If conversation not started
      Call INITIATE-CONVERSATION
  Else
      Call CONTINUE-CONVERSATION
SEND-MAP routine:
  SEND new map to front-end terminal
  RETURN TRANSID(CZTS) with COMMAREA
INITIATE-CONVERSATION routine:
  FEPI ALLOCATE conversation with back-end
  Build screen image to invoke back-end transaction
  FEPI SEND screen image to back-end
  FEPI START the receive transaction
  RETURN
CONTINUE-CONVERSATION routine:
  Unless first customer number
      Reaccess conversation with FEPI ALLOCATE PASSCONVID
  Build screen image to send customer number
  FEPI SEND screen image to back-end
  FEPI START the receive transaction
  RETURN
```

Screen image RECEIVE and EXTRACT FIELD

This sample program demonstrates using FEPI to get formatted data from a back-end transaction.

Module name	DFH0VZTR
Transaction name	CZTR
Map name	DFH0MZ2

This program is the RECEIVE part of a SEND-RECEIVE pair of programs, the SEND part being DFH0VZTS.

This transaction is started by CICS either when data is received from the back-end transaction or if no data is received in the time set in the send transaction, as is determined from the start data obtained with RETRIEVE. The user data in the start data indicates whether the conversation is starting, continuing, or finishing.

A FEPI RECEIVE obtains the screen image from the back-end transaction and FEPI EXTRACT FIELD is used to obtain specific fields.

If the conversation is starting, control is passed to the send transaction using XCTL to allow an inquiry to be sent to the back-end transaction.

If the conversation is continuing, the results from the back-end are sent on to the front-end terminal. Access to the conversation is relinquished, and control is returned to CICS specifying that the send transaction is to be invoked when there is next user input.

If the conversation has finished, a message to that effect is sent to the front-end terminal. The conversation is freed and the transaction ends.

Program logic

```
MAIN procedure:
  RETRIEVE start data
  Reaccess conversation with FEPI ALLOCATE PASSCONVID
  If time out
      Call REPORT-PROBLEM
  FEPI RECEIVE back-end screen image
  If conversation ending (PF3 or CLEAR indicated)
```

```
        Call REPORT-PROBLEM
    If back-end problem
    (CICS message or back-end transaction message)
        Call REPORT-PROBLEM
    If conversation starting (user data has customer number)
        XCTL to program DFH0VZTS
    If conversation continuing
        Get interesting fields from back-end data using
            FEPI EXTRACT FIELD
        Build and send map to front-end terminal
        Release conversation using FEPI FREE PASS
        RETURN TRANSID(CZTS) with COMMAREA
  REPORT-PROBLEM routine:
    SEND message to front-end terminal
    FEPI FREE conversation
    RETURN
```

3270 data stream passthrough

This sample program demonstrates using FEPI to passthrough 3270 data stream between a back-end application and a front-end terminal.

Module names	DFH0VZTD, DFH0AZTD
Transaction name	CZTD

On the first invocation of the transaction, a request is sent to the back-end system to start a transaction there. The response is sent on to the front-end terminal.

When there is input from the front-end terminal, CICS reinvokes the transaction. This input is sent on to the back-end system, using the FEPI CONVERSE command, and the resulting response is returned to the front-end terminal.

If there is an error, or the back-end system sends a CICS message, or PF3 is received from the front-end terminal, the transaction ends.

Program logic

```
Test COMMAREA
If transaction not previously invoked
    Build data stream request to start back-end transaction
    FEPI ALLOCATE conversation with back-end system
    FEPI CONVERSE data stream to and from back-end system
    SEND returned data stream to the front-end terminal
Else
    RECEIVE data stream from the front-end terminal
    Prepare data stream to send on to back-end system
    Reaccess conversation with FEPI ALLOCATE PASSCONVID
    FEPI CONVERSE data stream to and from back-end system
    SEND data stream to the front-end terminal
If error during processing
    SEND explanatory message
If continuing
    Release conversation using FEPI FREE PASS
    RETURN TRANSID(CZTD) with COMMAREA
Else (error, CICS message, or PF3)
    FEPI FREE conversation
    RETURN
```

End-session handler

This program cleans up sessions after use by FEPI sample application programs.

Module name	DFH0VZUU
Transaction name	CZUU
TS queue name	SESSION

This transaction is started by FEPI when an application ends a conversation or when a session is released.

The conversation passed by FEPI must be accessed so that FEPI knows that the event is being handled. The processing required depends entirely on local requirements. For illustration purposes, this program keeps the session for use by another conversation or lets it end, depending on the event type.

The CONVID picked up from the START data and passed on the FEPI ALLOCATE PASSCONVID is not the same as the CONVID for the conversation that has been freed. Nevertheless, the end-session handler can use it to access the same FEPI terminal.

For end of conversation (EVENTTYPE=FREE in start data), processing could typically involve setting the session back to a known state (such as a clear back-end screen ready to accept a new transaction name), or handling security, or overriding the type of FREE used. Such processing would depend on the data mode and type that the session uses (which is obtained from the start data), whether the back-end system is CICS or IMS, and the type of FREE used (also obtained from the start data).

For end of session (EVENTTYPE=FREE and EVENTVALUE=RELEASE in start data), processing could typically involve handling security.

For both cases, there could be an indication (in EVENTVALUE in the start data) that CICS is shutting down, which might require alternative special processing. This transaction would have to be in the XLT to allow it to be started during shutdown.

After the processing, a report is written to a TS queue named SESSION, which can be browsed using CEBR. The format of the TS queue records is:

```
date time CZUU End-session handling completed
          RESP........0            RESP2.......0
          Target......TGTNAME      Node........NODENAME
          Pool........POOLNAME
```

Program logic

```
  Main procedure:
    RETRIEVE start data
    Access conversation using FEPI ALLOCATE with PASSCONVID
    Call PROCESS-RELEASE or PROCESS-FREE as appropriate
    Write event details to TS queue
    RETURN
  PROCESS-RELEASE routine:
    Handle as required
    Free conversation, ending session
  PROCESS-FREE routine:
    Handle as required
    Free conversation, keeping session
```

SLU P one-out one-in

This sample program demonstrates using FEPI to obtain information from a back-end IMS system, using SLU P mode and the FEPI CONVERSE command with the POOL option.

Module names	DFH0VZPS, DFH0AZPS
Transaction name	CZPS
Map names	DFH0MZ4, DFH0MZ8

On the first invocation of the program, a map is sent to the front-end terminal.

When there is input from the front-end terminal, CICS reinvokes the program. A simple inquiry is made to the back-end system—for illustration purposes, it asks the time—and the answer is displayed on the front-end terminal. Because the inquiry requires only a one-out one-in exchange with the back-end system, a temporary conversation can be used, so the FEPI CONVERSE command with the POOL option is used.

When PF3 or CLEAR is received from the front-end terminal, the transaction ends. If there is an error, the front-end map is reset. These situations are detected using HANDLE CONDITION.

If the back-end system sends an IMS message, it is sent on to the front-end terminal and the transaction ends.

For clarity, error checking is minimal except for the FEPI commands.

```
CZPS             SLU P Sample Program.
IMS SLU P conversational sample program
This transaction will process a FEPI CONVERSE command to obtain time
         and date from a back-end IMS system.
         DATE   :  02/04/92
         TIME   :  10:57:10
         STATE  :  Not started
F3=EXIT to CICS  ENTER=obtain time and date stamp from IMS
```

Figure 13. CZPS transaction: SLU P sample program

Program logic

```
  MAIN procedure:
    Test COMMAREA
    If transaction not previously invoked
        Call SEND-NEW-MAP
    Set up exception condition handling:
        Map error - SEND-NEW-MAP
        CLEAR/PF3 - END-PROG
    RECEIVE MAP from front-end terminal
    Build SLU P data stream to request time from back-end IMS
        system
    FEPI CONVERSE to send data stream to the back-end and get
        the message containing the time
    If IMS message received from back-end system
        SEND message to front-end terminal
        RETURN
    Build data for front-end terminal map
    SEND map data to front-end terminal
    RETURN TRANSID(CZPS) with COMMAREA
  SEND-NEW-MAP routine:
    SEND new map
    RETURN TRANSID(CZPS) with COMMAREA
  END-PROG routine:
    Clear front-end terminal
    RETURN
```

SLU P pseudoconversational

This sample program demonstrates using FEPI to obtain data from an IMS back-end transaction. It is in pseudoconversational style, using the FEPI START command to schedule itself when the results arrive.

Module names	DFH0VZPA, DFH0AZPA
Transaction name	CZPA
Map names	DFH0MZ5, DFH0MZ9

On the first invocation of the program, a map is sent to the front-end terminal.

When there is input from the front-end terminal, CICS invokes the program again. After establishing a conversation, an inquiry is sent to the back-end system. FEPI START is issued to start this program again when the results arrive. Meanwhile it returns to CICS, so releasing resources.

When the results arrive, FEPI starts the program again. The results are obtained using FEPI RECEIVE, and sent on to the front-end terminal. The conversation is freed and the program returns to CICS to await more input. If the back-end system sends an IMS message, it is sent on to the front-end terminal and the transaction ends.

When PF3 or CLEAR is received from the front-end terminal, the transaction ends. If there is an error, the front-end map is reset. These situations are detected using HANDLE CONDITION.

For clarity, error checking is minimal except for the FEPI commands.

Screen

```
CZPA             SLUP Sample Program.
IMS SLUP Pseudoconversational sample program
This transaction will process SEND/START/RECEIVE requests with MFS
         specified, to a back-end IMS system.
         DATE   :  02/04/92
         TIME   :  10:58:50
         STATE  :  Not Started
F3=EXIT to CICS  ENTER=obtain time and date stamp from IMS
```

Figure 14. CZPA transaction: SLU P pseudoconversational sample program

Program logic

```
  MAIN procedure:
    If started from terminal
        Test COMMAREA
        If transaction not previously invoked
            Call SEND-NEW-MAP
        Set up exception condition handling:
            Map error - SEND-NEW-MAP
            CLEAR/PF3 - END-PROG
        RECEIVE map from front-end terminal
        FEPI ALLOCATE conversation with back-end system
        Build SLU P data stream to request time
        FEPI SEND data stream to back-end system
        FEPI START transaction
        RETURN
    If started by FEPI
        RETRIEVE start data
        Reaccess conversation using FEPI ALLOCATE PASSCONVID
        If EVENTTYPE = data received
            FEPI RECEIVE data stream from back-end system
            FEPI FREE conversation
            If IMS message received
                SEND message to front-end terminal
                RETURN
            Build data for front-end terminal map
            SEND map to front-end terminal
            RETURN TRANSID(CZPA) with COMMAREA
        Otherwise (timeout or session loss)
            SEND map with message to front-end terminal
            RETURN (freeing conversation implicitly)
  SEND-NEW-MAP routine:
    SEND new map
    RETURN TRANSID(CZPA) with COMMAREA
  END-PROG routine:
    Clear front-end terminal
    RETURN
```

STSN handler

This program handles STSN processing for the FEPI sample application programs.

Module name	DFH0AZQS
Transaction name	CZQS
TS queue name	SESSION

The CZQS transaction is started by FEPI when a request for message resynchronization (set and test sequence number, STSN) or a "start data traffic indication" is received from a back-end IMS system.

The conversation passed by FEPI must be accessed so that FEPI knows that the event is being handled. The processing required depends on the STSN status, which is obtained using FEPI EXTRACT STSN.

For STSNSTATUS=NOSTSN, the transaction was started because "start data traffic" arrived. A DR1 normal response must be sent.

For STSNSTATUS=STSNSET, a positive STSN response must be sent.

For STSNSTATUS=STSNTEST, processing would typically involve comparing saved sequence numbers with those received from the back-end IMS system to determine what response to send. Communications and connections in IMS product documentation gives advice on the appropriate action.

After the processing, the response is sent using FEPI ISSUE. A report is written to a TS queue named SESSION, which can be browsed using CEBR. The general format of the TS queue records is:

```
date time CZQS STSN processing completed
          Target......TGTNAME      Node........NODENAME
          Seqnumin....nnnn          Seqnumout...nnnn
          STSN status.XXXXXXX       Response....XXXXXXXX
```

Program logic

```
   Main procedure:
     RETRIEVE start data
     Access conversation using FEPI ALLOCATE with PASSCONVID
     Get STSN status using FEPI EXTRACT STSN
     Call NOSTSN, STSNSET, or STSNTEST
         according to STSN status
     Send response using FEPI ISSUE CONTROL
     Write event details to TS queue
     Free conversation, keeping session
     RETURN
   NOSTSN routine:
     Build DR1 normal response
   STSNSET routine:
     Build STSN positive response
   STSNTEST routine:
     Handle as required
     Build required response
```

Setting up the FEPI samples

To get the FEPI samples running, you need to customize them for your system.

Procedure

1. You need to change the following samples:
   - The customization data copy book, DFH0BZCx
   - The setup program, DFH0xZXS
   - The resource definitions, DFH0IZRx.
2. Compile or assemble and link-edit all the samples (and their maps) that you want, as you would for any CICS application program.
3. Define the samples to your front-end system, using the sample resource definitions listed in Appendix C, “FEPI samples,” on page 187.

   The resource definitions are in the form required as input to the DFHCSDUP utility. Note that there is a separate resource group for each language because the transaction names used are the same for each programming language. You should have defined the necessary transient data (TD) queues when you installed FEPI itself. Sample definitions are provided in group, DFHDCTG.
4. Assemble, link-edit, define, and install the appropriate back-end program and maps on your back-end system.

   If you want to use the IMS back-end samples, complete these steps:

   a) Use the sample resource definitions in DFH0IZRI.

   b) Link-edit the back-end program with the IMS version of ASMTDLI (or the appropriate language module), and specify RMODE and AMODE as 24.

      If you use the CICS version of ASMTDLI, the program will abend when executed in the IMS environment.

Running the FEPI samples

Appendix D. Front End Programming Interface exits XSZARQ and XSZBRQ

If you have installed the Front End Programming Interface (FEPI), you can use global user exits XSZARQ and XSZBRQ before and after FEPI commands.

XSZBRQ
Invoked before a FEPI command is executed (but after the syntax of the command has been validated, and therefore after EDF processing).

XSZARQ
Invoked immediately after a FEPI command has completed (before EDF processing).

Note that both the FEPI application programming and system programming commands cause XSZBRQ and XSZARQ to be invoked, but the latter do not provide the exit programs with any meaningful information.

You cannot use exit programming interface (XPI) calls or EXEC CICS commands in programs invoked from these exits. The exits allow you to monitor the FEPI commands and data being processed; you can inhibit commands, and modify specific command options. You could use them for:

- Monitoring the issue of FEPI commands
- Workload routing
- External security on application programming commands.

XSZBRQ

XSZBRQ is invoked before a FEPI command is executed; the input parameters for the command are passed to the exit program.

The majority of the information passed is read-only, but you can write a program to update specific parameters. FEPI does not check the validity of the new values for the updated parameters. In addition, your exit program can decide whether the request is to be processed or bypassed. You could use XSZBRQ, for example, to log commands, to bypass commands that violate the conventions of your installation, or to reroute commands by changing their specified targets or pools.

Together, UEPSZALP and UEPSZALT contain the information necessary to initiate a conversation.

When invoked
Invoked by FEPI before a FEPI command runs, but after syntax and semantic checking.

Exit-specific parameters

UEPSZACT
A 2-byte field that identifies the command. The values are given in Table 18 on page 207.

UEPSZCNV
An 8-character field containing the conversation ID (CONVID) for the command. Applicable on FEPI ALLOCATE, SEND, RECEIVE, CONVERSE, EXTRACT, ISSUE, START, and FREE commands.

UEPSZALP
An 8-character field containing the name of the pool (POOL). Modifiable and applicable on FEPI ALLOCATE and CONVERSE commands.

UEPSZALT
An 8-character field containing the name of the target (TARGET). Modifiable and applicable on FEPI ALLOCATE and CONVERSE commands.

UEPSZTIM
Fullword binary field containing the timeout value (TIMEOUT). Modifiable and applicable on FEPI ALLOCATE, RECEIVE, CONVERSE, and START commands.

UEPSZSND
Address of the 'send' data-area (FROM). Applicable on FEPI CONVERSE and SEND commands.

UEPSZSNL
Fullword binary field containing the length of the 'send' data (FROMFLENGTH, FLENGTH). Applicable on FEPI CONVERSE and SEND commands.

UEPSZSTT
A 4-character field containing the transaction ID (TRANSID). Modifiable and applicable on FEPI START commands.

UEPSZSTM
A 4-character field containing the terminal ID (TERMID). Modifiable and applicable on FEPI START commands.

UEPSZSNK
A 1-bit flag field indicating whether data is in key stroke format (KEYSTROKE). Applicable on FEPI CONVERSE FORMATTED and SEND FORMATTED commands. It can contain the following values:

UEPSZSNK_OFF
Not key stroke format.

UEPSZSNK_ON
Key stroke format.

UEPSZSNE
A 1-character field containing the key stroke escape character (ESCAPE). Applicable on FEPI CONVERSE FORMATTED and SEND FORMATTED commands.

Return codes

UERCNORM
Continue processing.

UERCBYP
Do not process the request; return INVREQ to the application.

Note: Your exit program cannot bypass events (like CICS shutdown or end-of-task).

XPI calls
Do not use XPI calls.

XSZARQ

XSZARQ is invoked immediately after a FEPI command has been executed; the exit program is passed the parameters that are output from the command. All of the information passed is read-only.

When invoked
Invoked by FEPI immediately after a FEPI command has been processed.

Exit-specific parameters

UEPSZACN
A 2-byte field that identifies the command. The values are given in Table 18 on page 207.

UEPSZCON
An 8-character field containing the conversation ID (CONVID) for the command. Applicable on FEPI ALLOCATE, SEND, RECEIVE, CONVERSE, EXTRACT, ISSUE, START, and FREE commands.

UEPSZRP2
Fullword containing the response code for the command (RESP2).

UEPSZRVD
Address of the 'receive' data-area (INTO). Applicable on FEPI RECEIVE, CONVERSE, and EXTRACT FIELD commands.

UEPSZRVL
Fullword binary data field containing the length of the receive data (FLENGTH, TOFLENGTH). Applicable on FEPI RECEIVE, CONVERSE, and EXTRACT FIELD commands.

Return code

UERCNORM
Continue processing.

XPI calls
Do not use any XPI calls.

The UEPSZACT and UEPSZACN exit-specific parameters

Both XSZBRQ and XSZARQ are passed a parameter (**UEPSZACT** for XSZBRQ, and **UEPSZACN** for XSZARQ) indicating the command or event being processed.

Table 18 on page 207. relates the hexadecimal values passed in UEPSZACT and UEPSZACN to the FEPI commands they represent.

Table 18. Settings of UEPSZACT for exit XSZBRQ and UEPSZACN for exit XSZARQ

Name	Setting (hex)	FEPI command or event
UEPSZNOA	820E	AP NOOP
UEPSZOAL	8210	ALLOCATE
UEPSZOCF	8212	CONVERSE FORMATTED
UEPSZOCD	8214	CONVERSE DATASTREAM
UEPSZOXC	8216	EXTRACT CONV
UEPSZOXF	8218	EXTRACT FIELD
UEPSZOXS	821A	EXTRACT STSN
UEPSZOFR	821C	FREE
UEPSZOSU	821E	ISSUE
UEPSZORF	8220	RECEIVE FORMATTED
UEPSZORD	8222	RECEIVE DATASTREAM
UEPSZOSF	8224	SEND FORMATTED
UEPSZOSD	8226	SEND DATASTREAM
UEPSZOST	8228	START
UEPSZSDN	8402	CICS normal shutdown 1
UEPSZSDI	8404	CICS immediate shutdown 1
UEPSZSDF	8406	CICS forced shutdown 1
UEPSZEOT	8408	CICS end-of-task 1
UEPSZNOS	840E	SP NOOP
UEPSZOQY	8422	INQUIRE PROPERTYSET
UEPSZOIY	8428	INSTALL PROPERTYSET
UEPSZODY	8430	DISCARD PROPERTYSET
UEPSZOQN	8442	INQUIRE NODE
UEPSZOTN	8444	SET NODE

Table 18. Settings of UEPSZACT for exit XSZBRQ and UEPSZACN for exit XSZARQ (continued)		
Name	**Setting (hex)**	**FEPI command or event**
UEPSZOIN	8448	INSTALL NODELIST
UEPSZOAD	844A	ADD POOL
UEPSZODE	844C	DELETE POOL
UEPSZODN	8450	DISCARD NODELIST
UEPSZOQP	8462	INQUIRE POOL
UEPSZOTP	8464	SET POOL
UEPSZOIP	8468	INSTALL POOL
UEPSZODP	8470	DISCARD POOL
UEPSZOQT	8482	INQUIRE TARGET
UEPSZOTT	8484	SET TARGET
UEPSZOIT	8488	INSTALL TARGETLIST
UEPSZODT	8490	DISCARD TARGETLIST
UEPSZOQC	84A2	INQUIRE CONNECTION
UEPSZOTC	84A4	SET CONNECTION

Note:

- 1 These events are generated internally by CICS; you cannot bypass them.

Notices

This information was developed for products and services offered in the U.S.A. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming interface information

CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 4 are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- Securing overview
- Developing for external interfaces
- Reference: application developmenth
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 5 Release 4, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- Reference: diagnostics

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 4 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services
- Customization Guide

- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- Supplied Transactions
- CICSPlex SM Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Application Programming Guide and Application Programming Reference
- Java Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 5 Release 4, but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability
These terms and conditions are in addition to any terms of use for the IBM website.

Personal use
You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM online privacy statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below:

For the CICSPlex® SM Web User Interface (main interface):

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface (data interface):

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface ("hello world" page):

Depending upon the configurations deployed, this Software Offering may use session cookies that collect no personally identifiable information. These cookies cannot be disabled.

For CICS Explorer®:

Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user's user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you, as customer, the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM Privacy Policy and IBM Online Privacy Statement, the section entitled "Cookies, Web Beacons and Other Technologies" and the IBM Software Products and Software-as-a-Service Privacy Statement.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z