CICS Transaction Server for z/OS
5.4

Business Transaction Services

Note

Before using this information and the product it supports, read the information in "Notices" on page 245.

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 to use Business Transaction Services (BTS) in CICS.

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

Date of this PDF

This PDF was created on 2023-10-09 (Year-Month-Date).

Chapter 1. CICS and Business Transaction Services (BTS)

Read this introductory information to learn about CICS business transaction services (BTS).

To operate BTS, there are no additional requirements beyond the requirements for CICS itself.

Table 1. Concepts road map

If you want to...	Refer to...
Understand the problems that BTS is designed to solve	"Why do I need CICS business transaction services?" on page 1
Get a high-level view of BTS	"What are CICS business transaction services?" on page 2

CICS-supplied sample BTS application

CICS supplies a sample BTS application and fragments of example code. You can set up the Sale example application to see the workings of a business transaction. The business transaction has four basic actions; order entry, delivery, invoice, and payment. It also includes some more advanced BTS features. For details, see The Sale example application.

Why do I need CICS business transaction services?

In a complex environment, CICS business transaction services (BTS) bring sophistication to the CICS application programming interface (API), making it better able to model complex business transactions.

CICS provides a robust transaction processing environment. For example, you can create transactions with the ACID properties of atomicity, consistency, isolation, and durability. CICS provides the environment for transactions to continue to run under various conditions.

Much emphasis is placed on the continuous operation and high availability of CICS. Use of sophisticated technologies, such as the Parallel Sysplex®, with resource managers sharing data across the sysplex, has led to improved system availability through the elimination of single points-of-failure. CICS business transaction services complement these technologies.

Business transactions and CICS transactions

Business transactions were typically modeled by CICS transactions in ways that now have some shortcomings. CICS business transaction services can overcome these shortcomings.

Business transactions

A *business transaction* is a self-contained business deal. Some business transactions, for example, buying a newspaper, are simple and short-lived. However, many business transactions involve multiple actions that take place over an extended period.

As an example, selling a vacation might involve the travel agent in actions such as:

- Recording customer details
- Booking seats on an aircraft
- Booking a hotel
- Booking a rental car
- Invoicing the customer
- Checking for receipt of payment

- Processing the payment
- Arranging foreign currency.

Both the customer and the travel agent regard the purchase of the vacation as a single business transaction, as indeed it is, because each action only makes sense in the context of the whole. The example illustrates some typical properties of complex business transactions:

- They tend to be made up of a series of logical actions.
- Some actions might be taken days, weeks, or even months after the transaction was started - arranging foreign currency, in this example.
- Some of the actions might be optional - not everyone wants to rent a car, for example.
- At any point, an action could fail. For example, a communications failure could mean that it is not possible to book a hotel. In this case, the action must be tried again. Or the customer might fail to meet their final payment; this failure would require a reminder to be sent. If the reminder produces no response, the vacation must be canceled - that is, the actions that have already been taken must be undone.
- Data - for example, a customer account number - must be passed between the individual actions that make up the business transaction.
- Some control logic is required, to “glue” the actions together. For example, there must be logic to deal with the conditional invocation of actions, and with failures.

CICS transactions

The basic building blocks used by CICS applications are the CICS transaction, which in turn is made up of one or more units of work (UOW). Both the transaction and the UOW are typically short-lived.

A UOW is short-lived because it should not hold locks for long periods, thus causing other UOWs to wait on resources and possibly abend. A CICS transaction provides the environment in which its associated UOWs run; for example, the transaction ID, program name, and user ID. A CICS transaction is typically short-lived, because the aim is for it to use CICS resources only while it is doing work; it should not spend long periods waiting for input, for example.

Before CICS Transaction Server for OS/390® Release 3, the largest transaction processing unit that CICS understood was the terminal-related pseudoconversation. A pseudoconversational application appears to a terminal user as a continuous conversation, but consists internally of multiple transactions.

What are CICS business transaction services?

CICS business transaction services consist of an application programming interface and support services that simplify the development of business transactions. Business transactions are often made up of multiple actions, which might be spread over hours, days, or months.

Terminology: These terms are explained in context, as they occur.

You can use CICS business transaction services to control the execution of complex business transactions. Using BTS, each action that makes up the business transaction is implemented as one or more CICS transactions, using the traditional approach. However, a top-level program is used to control the overall progress of the business transaction. The top-level program manages the inter-relationship, ordering, parallel execution, commit scope, recovery, and restart of the actions that make up the business transaction. This program management brings a number of benefits:

- Management and control are at a business transaction level, as well as at an action level.
- Control logic is separated from business logic. The individual CICS transactions that make up the business transaction no longer need to be concerned with “before and after” actions. This logic separation simplifies the development of such transactions and makes it easier to reuse them.

What is a BTS application?

A BTS application is an application written using the CICS business transaction services API to take advantage of CICS business transaction services.

The components of an application written using the CICS business transaction services API are illustrated, in simplified form, in Figure 1 on page 3. (For brevity, in the rest of this book we refer to an application that uses the CICS business transaction services API as “a BTS application”.)

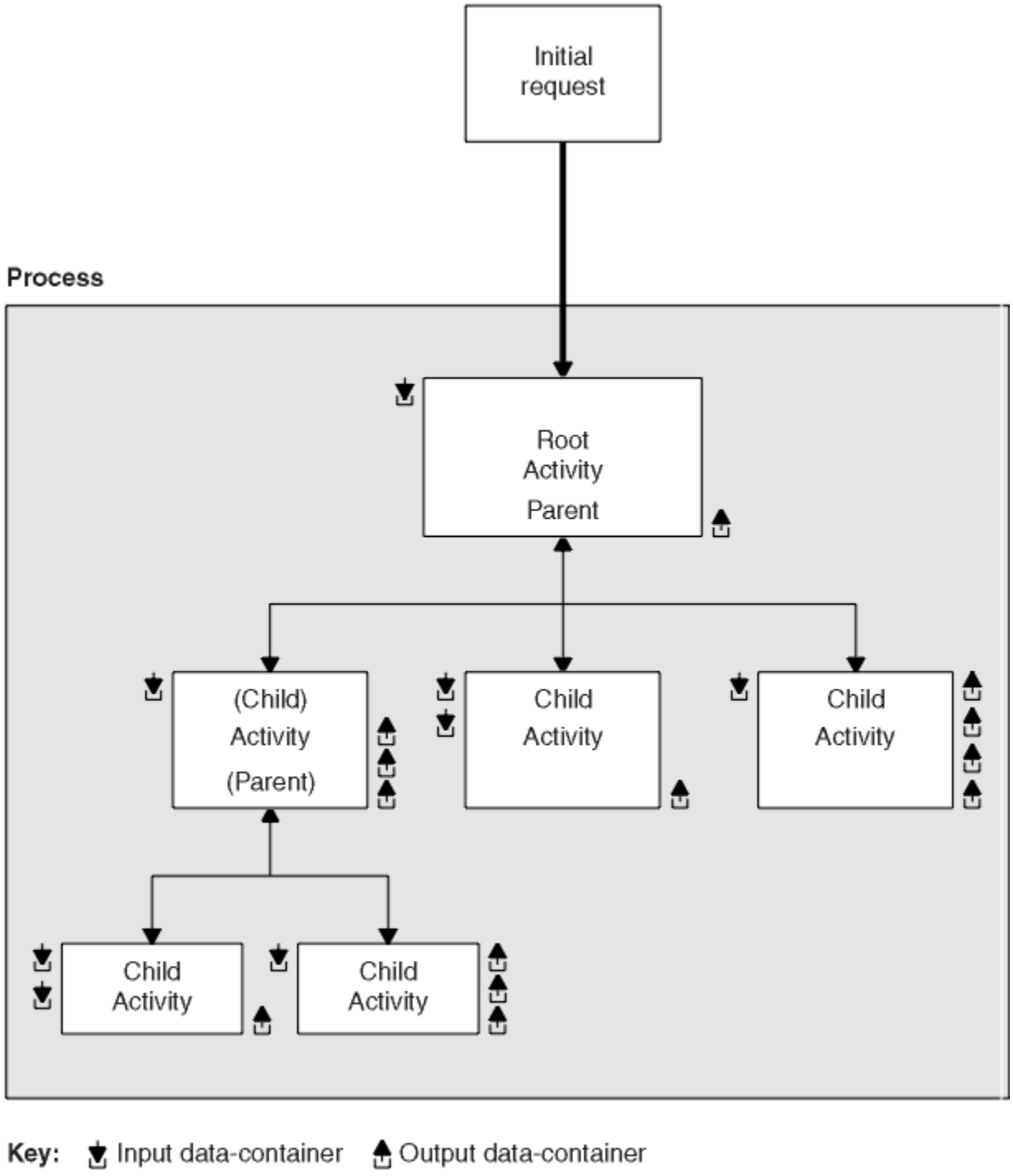


Figure 1. Components of a BTS application

The roles of the components are as follows:

Initial Request
A CICS transaction that starts a CICS business transaction services **process**.

Process
A collection of one or more BTS **activities**. It has a unique name by which it can be referenced and invoked. Typically, a process is an instance of a business transaction.

In the vacation example, an instance of the business transaction might be started to sell Jane Doe a vacation in Florida. To identify this particular transaction as relating to Jane Doe, the process could be given the name of Jane Doe's account number.

Activity
The basic unit of BTS execution. Typically, it represents one of the actions of a business transaction—in the vacation example, renting a car, for instance.

A program that implements an activity differs from a traditional CICS application program only in its being designed to respond to BTS **events**. It can be written in any of the languages supported by CICS.

Activities can be hierarchically organized, in a tree structure. An activity that starts another activity is known as a **parent activity**. An activity that is started by another is known as a **child activity**.

Root activity
The activity at the top of the activity tree - it has no parent activity. A process always contains a root activity. When a process is started, the program that implements its root activity receives control. Typically, a root activity is a parent activity that:

- Creates and controls a set of child activities - that is, it manages their ordering, concurrent execution, and conditional execution
- Controls synchronization, parameter passing and saving of state data.

Data-container
A named area of storage, associated with a particular process or activity, and maintained by BTS. Each process or activity can have any number of data-containers. They are used to hold state data, and inputs and outputs for the activity.

Event (not shown in Figure 1 on page 3)
A BTS event is a means by which CICS business transaction services signal progress in a process. It informs an activity that an action is required or has completed. “Event” is used in its ordinary sense of “something that happens”. To define an event recognizable by CICS business transaction services, such a happening is given a name.

Timer (not shown in Figure 1 on page 3)
A BTS object that expires when the system time becomes greater than a specified date and time, or after a specified period has elapsed. Each timer has an event associated with it. The event occurs ("fires") when the timer expires.

You can use a timer to, for example, cause an activity to be invoked at a particular time in the future.

The preceding components are managed by CICS, which:

- Manages many business transactions (processes).
- Records the status of each business transaction.
- Ensures that each activity is invoked at the appropriate times.

The components of a BTS application, and how they relate to each other, are described in more detail in Developing with the BTS API.

Control flow

Follow the high-level control flow of a typical BTS business transaction.

The high-level control flow of a typical BTS business transaction is as follows:

1. A CICS transaction makes an initial request to start a process.
2. CICS initiates the appropriate root activity.

3. The root activity program, using the BTS API, creates a child activity—or several child activities. It provides the child activity with some input data (by placing the data in a data-container associated with the child), and requests CICS to start the child activity.

   If, as is often the case, the child activity is to run asynchronously with the root activity, the root activity program returns and becomes dormant.
4. The root activity is invoked again when one of its child activities completes. It determines which event caused it to be invoked again - that is, the completion of the activity that it started earlier. It retrieves, from the output data-containers of the completed activity, any return data that the completed activity has placed there.
5. Steps “3” on page 5 and “4” on page 5 are repeated until all the child activities that make up the business transaction have completed.
6. CICS terminates the root activity.

   If the business transaction has completed normally, the process is no longer known to CICS.

Recovery and restart in BTS

CICS maintains state data for BTS processes in a recoverable VSAM KSDS. This file can be RLS-enabled. On an emergency restart, CICS automatically restarts any BTS activities that were in flight at the time it failed.

Client/server support in BTS

CICS business transaction services support *client/server* processing. A server process is one that is typically waiting for work. When work arrives, BTS restarts the process, which retrieves any state data that it has previously saved.

Web Interface support in BTS

With the CICS Web Interface, you can run CICS transactions from a web browser. CICS business transaction services extend CICS support for the Internet.

In a typical current scenario, a web-based business transaction might be implemented as a pseudoconversational CICS application. The initial request from the web browser invokes a CICS transaction that does some setup work, returns a page of HTML to the web browser, and ends. Subsequent requests are handled by other CICS transactions (or by further invocations of the same transaction). The CICS application is responsible for maintaining state data between requests.

Using BTS, a web-based business transaction could be implemented as a BTS process. A major advantage of this approach is that state data is now maintained by BTS. This is useful if the business transaction is long-lived.

Support for existing code in BTS

BTS supports the 3270 bridge function. Therefore, BTS applications can be integrated with, and use, existing 3270–based applications.

The 3270 bridge function is described in Introduction to the 3270 bridge.

Even though BTS activities are not terminal-related, they are never started directly from a terminal; a BTS activity can use a 3270-based program.

Sysplex support in BTS

You can operate BTS in a single CICS region. However, BTS processes are sysplex-enabled, and in a sysplex you can create one or more *BTS-sets*.

A BTS-set is a set of CICS regions across which related BTS processes and activities can run. For example, the activities that constitute a single process might run on several regions.

Dynamic routing of BTS activities

In a BTS set, you can control the dynamic routing of the CICS transactions that implement your BTS activities across the participating regions.

When an event is signaled, an activity is activated in the most appropriate region in the BTS set, based on one or more of the following criteria:

- Any workload separation specified by the system programmer
- Any affinities the associated transaction of the activity has with a particular region
- The availability of regions
- The relative workload of regions

You can control the dynamic routing of your BTS activities using one of the following methods:

- Use the CICSPlex® System Manager (CICSPlex SM) component of CICS Transaction Server for z/OS for the following functions:
  - Specify workload separation for your BTS processes.
  - Manage affinities.
  - Control workload routing of the transactions that implement BTS activities.
- Write a CICS distributed routing program.

Dynamic routing of BTS activities is described in Administering BTS.

Audit trails

You can create an audit trail for the BTS processes and activities that run in your CICS regions. By doing so, you can, for example, track the progress of a complex business transaction across the sysplex.

The CICS code contains BTS audit points in much the same way as it contains trace points. However, because in a sysplex environment different parts of a process might execute on different regions, each audit record contains system, date, and time information. By sharing log streams across regions, you can gather audit information from different regions in the same log.

Monitoring in BTS

CICS maintains monitoring information for both processes and activities. You can request information about use of resources of a business transaction without knowing the identifiers of all its constituent CICS transactions. Information is now available at the business transaction level, as well as at the CICS transaction level.

BTS monitoring is described in Improving the performance of a CICS system.

Chapter 2. Configuring for BTS

You can use system definitions to define your BTS data sets and to name your routing program.

Defining BTS data sets

You can use the IDCAMS program to define your BTS data sets to MVS™.

About this task

You must define two types of BTS data set:

- Repository data sets.
- A local request queue data set.

Repository data sets

When a process is not running under the control of the CICS business transaction services domain, its state and the states of its constituent activities are preserved by being written to a VSAM data set known as a *repository*.

The states of all processes of a particular process-type (and of their activity instances) are stored on the same repository data set. Records for multiple process-types can be written to the same repository. You specify the repository on which processes of a particular process-type are stored when you define the process-type - see “CEDA DEFINE PROCESSTYPE” on page 12.

You must define at least one BTS repository data set to MVS. You might decide to define more than one, assigning a different set of process-types to each. One reason for defining more than one BTS repository might be storage efficiency - perhaps some of your process-types tend to produce longer records than others. To enable you to distinguish between process-types during a browse, you do not need to assign each process-type to a separate repository.

If you operate BTS in a sysplex, several CICS regions might share access to one or more repository data sets. This enables requests for the processes and activities stored on the data sets to be routed across the participating regions - see Administering BTS.

You must define the repository file as recoverable. Specify the following parameters to IDCAMS:

INDEXED
BTS repository data sets must be in KSDS format.

KEYS(50 0)
The file key. The file key is 50 bytes in length and is located at offset X'0' in the record.

LOG(UNDO|ALL)
The recovery options for the data set:

UNDO
The data set is recoverable.

ALL
Forward recovery is required. If you specify LOG(ALL), you must also specify a log stream on the **LOGSTREAMID** parameter.

LOGSTREAMID(*log_stream_ID*)
The identifier of the log stream to which forward recovery records are to be written. This parameter is required only if you specify LOG(ALL).

RECORDSIZE(*average maximum*)
The average and maximum size of records on the data set, in bytes.

Specify *maximum* as 16384 bytes. CICS automatically splits any records that are larger than 16 KB.

It is difficult to predict the average size of repository records. (A notional record of 20000 bytes, for example, is split into one record of 16384 bytes and one of 3616 bytes.) Initially, specify *average* as 8 KB. If you find in practice that the average size of records is markedly different from this average, you can specify a different value.

SPANNED
A single record can span control intervals. Specify this parameter if the record size is larger than the CI.

Figure 2 on page 8 shows example JCL for defining a BTS repository data set. This JCL is for illustration only.

```
//SMITHGOT JOB (WINVMC,SMITH),CLASS=E,USER=username
//IJMRBTS EXEC PGM=IDCAMS,REGION=6144K
//SYSPRINT DD SYSOUT=A
//AMSDUMP  DD SYSOUT=A
//SYSIN    DD *
  DELETE ('CICSTS54.CICS.BTS') PURGE CLUSTER
  DEFINE CLUSTER (                          -
                  NAME( CICSTS54.CICS.BTS )  -
                  LOG(UNDO)                  -
                  CYL(2,1)                   -
                  CISZ(4096)                 -
                  SPANNED                    -
                  VOLUMES (P2DA62)           -
                  KEYS( 50 0 )               -
                  INDEXED                    -
                  RECORDSIZE(8192  16384 )   -
                  FREESPACE( 5 5 )           -
                  SHAREOPTIONS( 2 3 )        -
                  )                          -
  INDEX           (                          -
                  NAME( CICSTS54.CICS.BTS.INDEX )   -
                  )                          -
  DATA            (                          -
                  NAME( CICSTS54.CICS.BTS.DATA )    -
                  )
/*
//
```

Figure 2. Example JCL for defining a BTS repository data set

To ensure that your repositories are continuously available, you are recommended to define them to use the backup while open (BWO) facility provided by DFSMSdss and DFSMShsm. For details of BWO, and how to define VSAM data sets to use it, see Defining backup while open (BWO) for VSAM files.

Local request queue data set

The local request queue data set stores pending BTS requests; for example, timer requests or requests to run activities. It is recoverable and is used to ensure that, if CICS fails, no pending requests are lost.

Requests that CICS can run immediately, for example, requests to run activities, are stored on the data set only briefly. Requests that CICS cannot run immediately, for example, timer or unserviceable requests, might be stored for longer periods. When CICS has processed a request, the request is deleted from the data set.

The local request queue data set differs from repository data sets in the following ways:

- It is a mandatory CICS data set. You must define one, even if you do not use BTS.

  Note: Procedure DFHDEFDS in library SDFHINST contains a definition of the LRQ. For information about how to use DFHDEFDS, see DFHDEFDS job for CICS region data sets.
- You must define one, and only one, to each CICS region.
- It is never shared. The local request queue data set relates solely to requests generated on the local region.

Specify the following parameters to IDCAMS:

INDEXED
BTS local request queue data sets must be in KSDS format.

KEYS(40 0)
The file key. The file key is 40 bytes in length and is located at offset X'0' in the record.

LOG(UNDO|ALL)
The recovery options for the data set:

UNDO
The data set is recoverable.

ALL
Forward recovery is required. If you specify LOG(ALL), you must also specify a log stream on the LOGSTREAMID parameter.

LOGSTREAMID(*log_stream_ID*)
The identifier of the log stream to which forward recovery records are to be written. This parameter is required only if you specify LOG(ALL).

RECORDSIZE(*average maximum*)
The average and maximum size of records on the data set, in bytes.

Specify *average* as 2232 bytes and *maximum* as 2400 bytes.

Figure 3 on page 9 shows example JCL for defining a BTS local request queue data set.

```
//SMITHGOT JOB (WINVMC,SMITH),CLASS=E,USER=username
//IJMRLRQ  EXEC PGM=IDCAMS,REGION=6144K
//SYSPRINT DD SYSOUT=A
//AMSDUMP  DD SYSOUT=A
//SYSIN    DD *
  DEFINE CLUSTER (                        -
                NAME( CICSTS54.CICS.LRQ )  -
                LOG(UNDO)                  -
                CYL(2,1)                   -
                VOLUME (SYSDAV)            -
                KEYS( 40 0 )               -
                INDEXED                    -
                RECORDSIZE( 2232 2400 )    -
                FREESPACE( 0 10 )          -
                SHAREOPTIONS( 2 3 )        -
                )                          -
  DATA          (                          -
                NAME( CICSTS54.CICS.LRQ.DATA )    -
                CISZ(2560) -
                ) -
  INDEX         ( -
                NAME( CICSTS54.CICS.LRQ.INDEX )   -
                )                          -
/*
//
```

Figure 3. Example JCL for defining a BTS local request queue data set

The LRQ data set is critical to the operation of BTS. Because its loss could severely impact the progression of BTS activities, you should consider defining it to use backup while open (BWO) and forward recovery.

Local request queue data set space allocation

The space that you allocate for the LRQ data set depends on how often you plan to trim the file and the amount of BTS work likely to be undertaken in the periods between maintenance.

Each **RUN ASYNCHRONOUS** or **DEFINE TIMER** request causes a record to be written to the LRQ file, each record being 2 KB in length.

You must allow plenty of space for contingencies. Remember that records representing timers that are defined to expire far in the future will remain on the data set until their timers expire. Even after the records have expired, VSAM still maintains control interval key ranges for their key values, even though the records for these keys have been deleted by CICS. These key ranges are never reused because they

represent time in the past, and their control interval space can only be recovered by periodic maintenance of the data set.

For more information about maintaining your local request queue data set, see "Local request queue data set maintenance" on page 10.

Local request queue data set maintenance

You must regularly perform maintenance on the local request queue (LRQ) data set to prevent the data set from becoming too large.

The key of the LRQ data set includes a time stamp. This means that BTS generally adds requests to the end of the file. Although BTS deletes requests when they have completed, VSAM does not physically delete the records; so the space is not reused. Therefore, you must perform regular maintenance on the local request queue (LRQ) data set to prevent the data set from becoming too large.

You can use an AMS command such as REPRO to shrink the data set. For example, to remove the records that have been logically deleted by VSAM. Before using the REPRO command, you must quiesce CICS. Disabling the LRQ file definition is not sufficient - if you only disable the LRQ file definition, activations might fail with ASP7 abends.

Before undertaking maintenance, quiesce the CICS region. If you use CICSPlex SM for routing BTS activities, you could route work away from the region by altering a CICSPlex SM workload definition (WLMDEF). If you use a CICS distributed routing program, you could alter your routing program. Although the region has been temporarily quiesced, BTS operations in the sysplex are uninterrupted.

When reorganizing the data set, consider the percentage of control areas specified on the FREESPACE parameter on the cluster definition. If there are many records representing timers far in the future, and new timers are to be added for events that will expire earlier than these existing records, a large FREESPACE percentage will decrease the number of likely control area splits that might occur. If most new timer events are added for times later than existing events, the FREESPACE percentage for control areas can be set to 0.

Naming the routing program

If you are using BTS in a sysplex, you must name the routing program that will be used to dynamically route BTS activities around the BTS-set. Use the DSRTPGM system initialization parameter to name the routing program that you will use.

Table 1. The DSRTPGM system initialization parameter

DFHSIT	[TYPE={CSECT|DSECT}]
	⋮
	[,DSRTPGM={NONE|DFHDSRP|program-name|EYU9XLOP}]
	⋮
END	DFHSITBA

DSRTPGM={NONE|DFHDSRP|program-name|EYU9XLOP}
specifies the name of the distributed routing program to be used for dynamically routing:

- Eligible CICS business transaction services (BTS) processes and activities.

  For information about which BTS processes and activities are eligible for dynamic routing, see Which BTS activities can be dynamically routed?.
- Eligible non-terminal-related EXEC CICS START requests.

  For information about which non-terminal-related START requests are eligible for dynamic routing, see Non-terminal-related START commands.

DFHDSRP
The CICS sample distributed routing program.

EYU9XLOP
The CICSPlex SM routing program.

NONE
For eligible BTS processes and activities, no routing program is invoked. BTS processes and activities cannot be dynamically routed.

For eligible non-terminal-related START requests, the CICS sample distributed routing program, DFHDSRP, is invoked.

program-name
The name of a user-written program.

Note: See also the DTRPGM parameter, used to name the dynamic routing program.

Resource definition for BTS

You can use BTS API commands to define most BTS resources such as processes, activities, events, and containers at run time. However, you must define some BTS resources on the CICS system definition file (CSD).

The only BTS resources that must be defined on the CICS system definition file (CSD) are as follows:

Process-types
See "CEDA DEFINE PROCESSTYPE" on page 12.

Note: As an alternative to using RDO CEDA DEFINE PROCESSTYPE commands to define your process-types, you can use the CICSPlex SM Business Application Services (BAS) PROCDEF object. You might want to use the PROCDEF object if you are using BTS in a sysplex, with routing of processes and activities controlled by CICSPlex SM. For information about BAS, see Administering BAS.

The BTS data set files
The CICS files that relate to the physical VSAM data sets used by BTS must be defined to CICS file control in the standard way, as described in FILE attributes.

BTS uses two kinds of data set:

- Local request queue data set. See "Local request queue data set" on page 8.
- Repository data sets, on which process and activity records are stored. See "Repository data sets" on page 7.

  For information about how to define repository files to the CSD, see "Defining repository files to the CSD" on page 11.

Audit logs
The journals used for auditing purposes must be defined to the CICS log manager in the standard way, as described in JOURNALMODEL resources.

Defining repository files to the CSD

You can define repository files to the CSD by specifying permitted operations, STRINGS values, VSAM record-level sharing (RLS) access, and any remote systems that share repository data.

About this task

On the FILE definition that defines the repository file to CICS:

- Specify that ADD, BROWSE, DELETE, READ, and UPDATE operations are all permitted.
- Specify the value of the STRINGS option to reflect the likely number of concurrent activations of processes that use the repository. The default is STRINGS(1), which is unlikely to be high enough.
- If you are using VSAM record-level sharing (RLS) to share the repository data between the regions of a BTS-set, specify RLSACCESS(YES).

If you are using function-shipping to a file-owning region (FOR) to share the repository data between the regions of a BTS-set, specify REMOTESYSTEM(name_of_FOR).

For information about BTS-sets, see Administering BTS.

CEDA DEFINE PROCESSTYPE

You can use the CICS business transaction services (BTS) CEDA DEFINE command to define and execute complex business applications called *processes*.

A process is represented in memory as a block of storage containing information relevant to its execution. It also has associated with it at least one additional block of information called an activity instance. When not executing under the control of the CICS business transaction services domain, a process and its activity instances are written to a data set known as a *repository*.

You can categorize your BTS processes by assigning them to different process-types. This categorization is useful, for example, for browsing purposes. The activities that constitute a process are of the same process-type as the process itself. A PROCESSTYPE definition defines a BTS process-type. The definition names the CICS file which relates to the physical VSAM data set (repository) on which details of all processes of this type (and their activity instances) are to be stored.

You might want to record the progress of BTS processes and activities for audit purposes, and to help diagnose errors in BTS applications. If so, you can name the CICS journal to which audit records are to be written, and the level of auditing that is required, for processes of the specified type.

Figure 4 on page 13 shows the relationship between PROCESSTYPE definitions, FILE definitions, and BTS data sets. Notice that multiple PROCESSTYPE definitions can reference the same FILE definition; and that multiple FILE definitions can reference the same BTS data set.

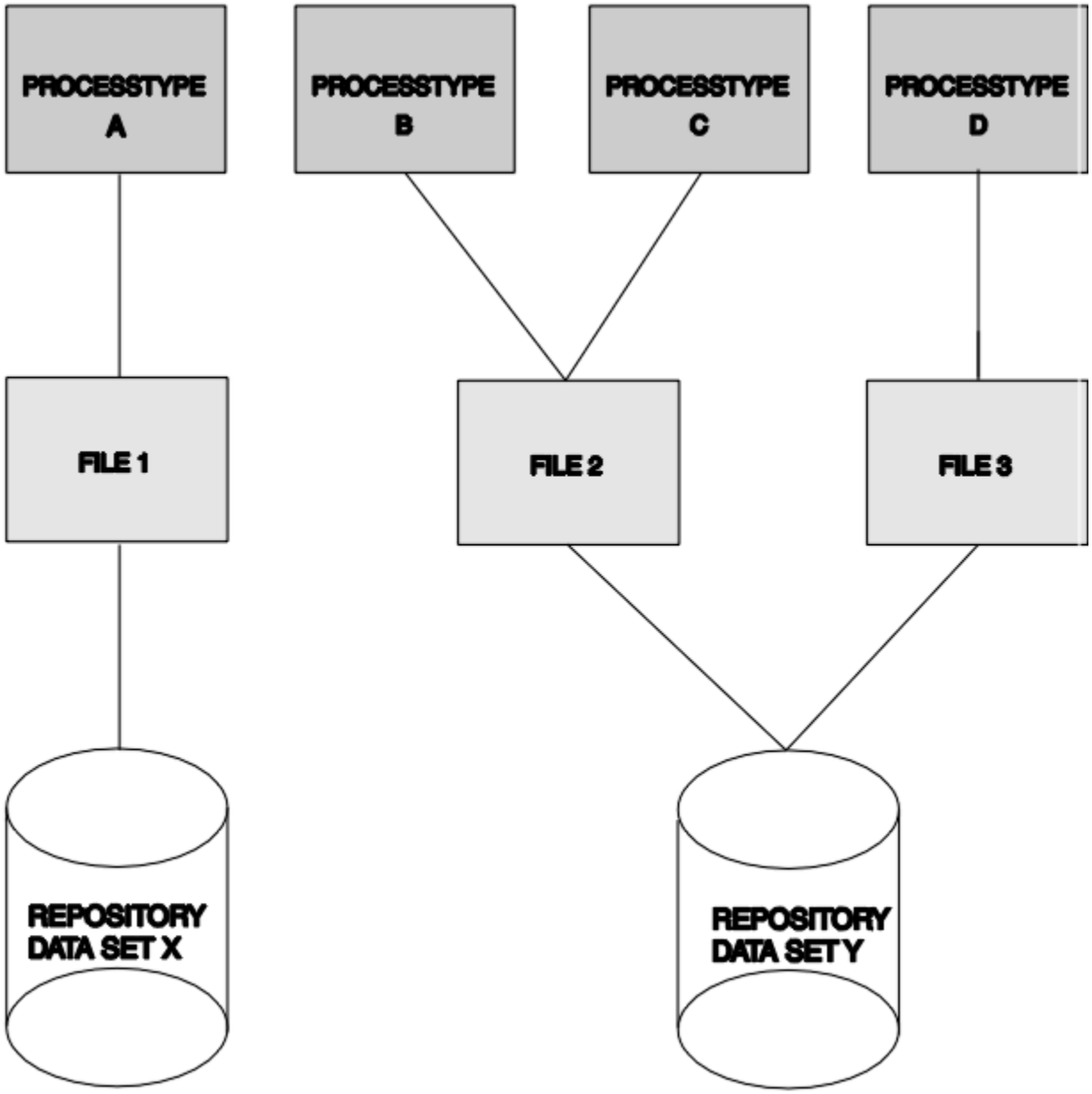


Figure 4. PROCESSTYPE definitions, FILE definitions, and repository data sets

DEFINE panel

```
 Processtype  ==>
  Group        ==>
  DEScription  ==>

INITIAL STATUS
 STAtus       ==> Enabled           Enabled | Disabled
DATA SET PARAMETERS
 File         ==>
AUDIT TRAIL
 Auditlog     ==>
 Auditlevel   ==> Off                Off | Process ? Activity ? Full
```

Figure 5. The DEFINE panel for PROCESSTYPE

Options

AUDITLEVEL({OFF|*PROCESS*?*ACTIVITY*?*FULL*})
specifies the initial level of audit logging for processes of this type. If you specify any value other than OFF, you must also specify the AUDITLOG option.

ACTIVITY
Activity-level auditing. Audit records are written from:

1. The process audit points
2. The activity primary audit points.

FULL
Full auditing. Audit records are written from:

1. The process audit points
2. The activity primary *and* secondary audit points.

OFF
No audit trail records are written.

PROCESS
Process-level auditing. Audit records are written from the process audit points only.

For details of the records that are written from the process, activity primary, and activity secondary audit points, see Specifying the level of audit logging

.

AUDITLOG(*name*)
specifies the name of a CICS journal to which audit trail records are written, for processes of this type and their constituent activities. The name can be up to eight characters long. If you do not specify an audit log, no audit records are kept for processes of this type.

DESCRIPTION(*text*)
You can provide a description of the resource you are defining in this field. The DESCRIPTION text can be up to 58 characters in length. There are no restrictions on the characters that you can use. However, if you use parentheses, ensure that for each left parenthesis there is a matching right one. For each single apostrophe in the text, code 2 apostrophes.

FILE(*name*)
specifies the name of the CICS file definition that is used to write the process and activity records of this process-type to its associated repository data set. The name can be up to eight characters long. The acceptable characters are A-Z 0-9 $ @ and #. Lowercase characters are converted to uppercase.

You must specify the FILE option.

PROCESSTYPE(*name*)
specifies the name of this PROCESSTYPE definition. The name can be up to eight characters in length. The acceptable characters are A-Z a-z 0-9 $ @ # . / -_ % & ? ! : | " = ¬ , ; < >. Leading and embedded blank characters are not permitted. If the name supplied is less than eight characters, it is padded with trailing blanks up to eight characters.

STATUS({ENABLED|*DISABLED*})
specifies the initial status of the process-type following a CICS initialization with START=COLD or START=INITIAL. After initialization, you can use the CEMT SET PROCESSTYPE command to change the status of the process-type. The status of the process-type following a restart is recovered to its status at the previous shutdown.

DISABLED
Processes of this type cannot be created. An EXEC CICS DEFINE PROCESS request that tries to create a process of this type results in the INVREQ condition being returned to the application program.

ENABLED
Processes of this type can be created.

Chapter 3. Developing with the BTS API

This section provides a detailed description of the BTS application components, and explains how you can use them.

BTS activities and processes

An *activity* is the BTS unit of execution. It holds the environment for an instance of the BTS equivalent of program execution. A process consists of a collection of one or more activities, and is the biggest entity recognized by BTS.

The state of a BTS activity is stored on disk and reinstantiated in memory as required. Typically, it represents one of the actions of a business transaction.

Activities can be hierarchically organized, in a tree structure that may be several layers deep. The activity at the top of the hierarchy is called the **root activity** . An activity that starts another activity is known as a *parent activity* . An activity that is started by another is known as a *child activity* . For example, if activity A starts activity B, B is a child of A; A is the parent of B. Notice that—with the exception of the root activity, which has no parent—an activity can be both a parent and a child.

A *process* is the biggest entity recognized by BTS. It consists of a collection of one or more activities. It always contains a root activity. When a process is run, the program that implements its root activity receives control. Typically, a process is an instance of a business transaction.

Processes can be categorized, using the PROCESSTYPE option of the DEFINE PROCESS command. All the activities in a process inherit the same PROCESSTYPE attribute. Categorizing processes makes it easier to find a particular process—the BTS browsing commands allow filtering by process-type.

Names and identifiers

You can use a program to define a process. The program gives the process a name, known as its *process name* , which is used to reference the process from outside the BTS system.

This user-assigned name, which can be up to 36 characters long, must be unique within the process-type to which the process belongs.

Similarly, when an activity program defines a child activity, it gives the child a name (its *activity name*), which it uses to reference the child. This user-assigned name, which can be up to 16 characters long, only needs to be unique within the set of child activities defined by the parent. For example, it is perfectly valid for several activities within the same process to each define a child called *Invoice*.

Note: A root activity always has the CICS -assigned name DFHROOT.

Besides its name, each activity has a CICS-assigned **activity identifier** . An activity identifier, which is 52 characters long, is a means of uniquely referring to an activity-instance. It is guaranteed to be unique across the sysplex, and its lifetime is the same as the activity it refers to. Activity identifiers are frequently used as arguments on inquiry and browsing commands. Only its parent can refer to a child activity by name; other programs can access the activity with its identifier.

Activation sequences

To complete all its work, an activity might need to run as a sequence of separate processing steps, or *activations* . For example, a parent activity typically needs to run for a while, stop temporarily, and then continue execution when one of its children has completed.

Each activation is triggered by a BTS *event* , and consists of a single transaction. First activation of the activity is triggered by the system event DFHINITIAL, supplied by BTS after the first RUN or LINK command is issued against the activity. (In the case of a root activity, DFHINITIAL occurs after the first **RUN** or **LINK** command is issued against the process. It is possible to issue multiple **RUN** or **LINK** commands against a process - see “Using client/server processing” on page 41.) When the last

activation ends, the *activity completion event* is fired, which might, in turn, trigger another activation of the activity. See “BTS events” on page 20.

Figure 6 on page 16 shows a BTS activity being reattached in a series of activations.

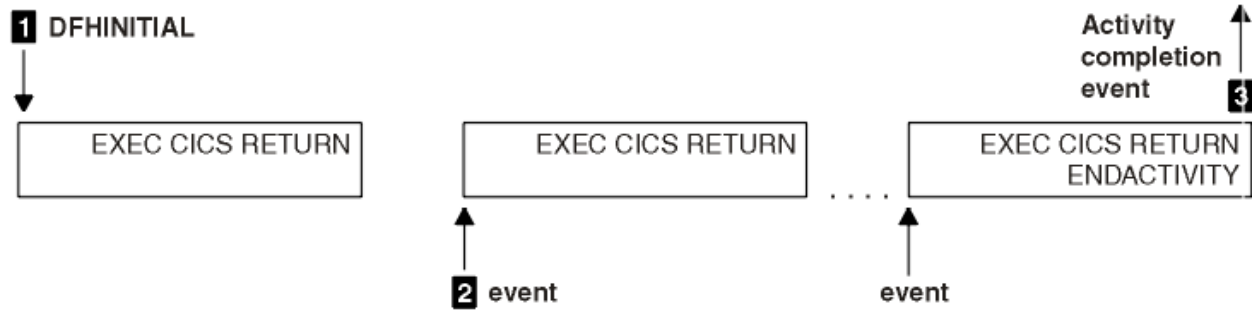


Figure 6. A sequence of activations

1
The first event that “wakes up” the activity is DFHINITIAL. The activity determines that the event which caused it to be activated was DFHINITIAL and therefore performs its first processing step. Typically, this involves defining further events for which it might be activated. The activity program issues an **EXEC CICS RETURN** command to relinquish control. The activity “sleeps”.

2
The next event occurs and “wakes up” the activity. The activity program determines which event caused it to be activated and performs the processing step appropriate for that event. It issues an **EXEC CICS RETURN** command to relinquish control.

3
Eventually, no more processing steps are necessary. To confirm that its current activation is the last, and that it is not to be reactivated for any future events, the activity program issues an **EXEC CICS RETURN ENDACTIVITY** command. The activity completion event is fired.

Note: Root activities do not have completion events.

Figure 7 on page 17 is a comparison between a terminal-related pseudoconversation and a BTS activity that is activated multiple times.

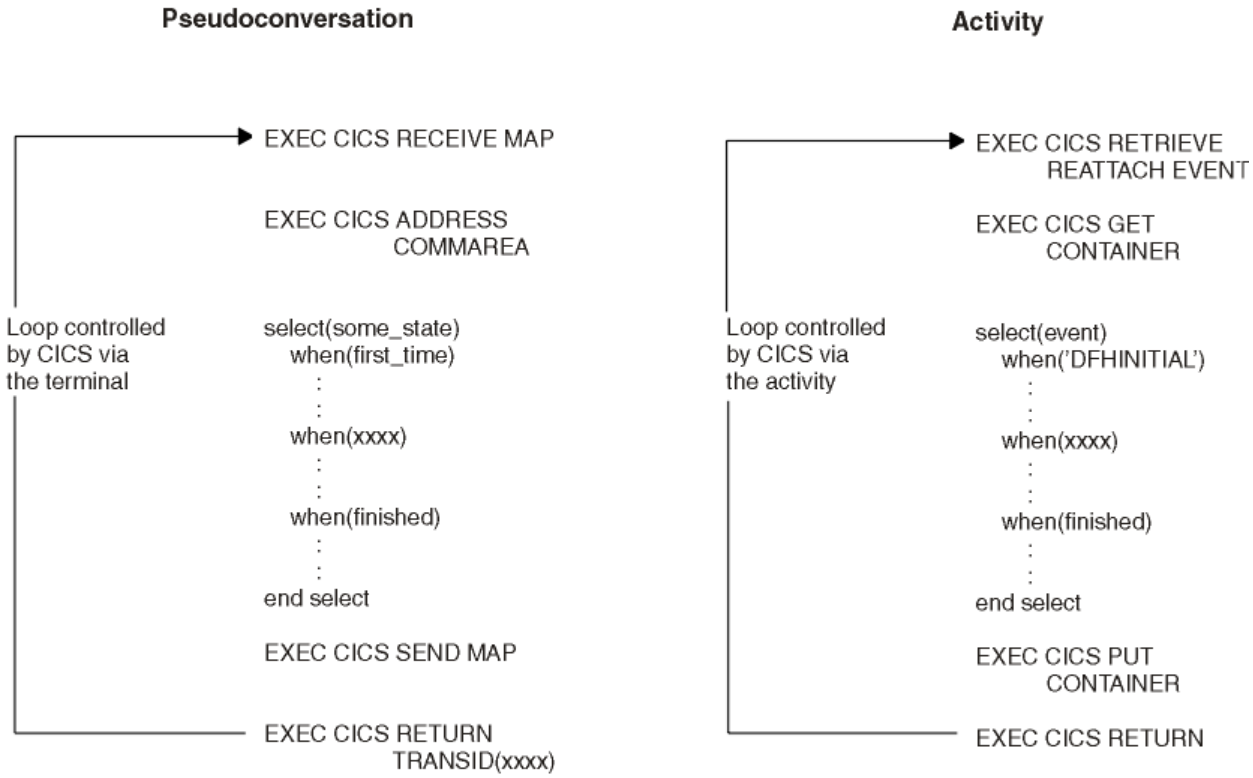


Figure 7. Comparison between a terminal-related pseudoconversation and a BTS activity that is activated multiple times

Note: The **RETRIEVE REATTACH EVENT** command issued by the activity retrieves the name of an event that caused the activity to be reactivated. The **GET** and **PUT CONTAINER** commands retrieve and store input and output data.

Synchronous and asynchronous activations

You can cause an activity or process to be activated in one of two ways, synchronously or asynchronously.

Synchronously
The activity or process is executed synchronously with the requestor. Exactly how it is run varies, depending on which command is used to activate it:

LINK
The activity is included as part of the current unit of work; all locks and resources are shared with the requestor. The activity runs with the transaction attributes of the requestor; any transaction attributes (transaction ID or user ID) specified on its resource definition are ignored. In other words, there is no **context-switch**.

RUN SYNCHRONOUS
The activity is run in a separate unit of work from the unit of work of the requestor, and with the transaction attributes (transaction ID or user ID) specified on its resource definition . In other words, a **context-switch** takes place.

The two units of work are linked; if the requestor backs out, the activity is backed out also.

Asynchronously
The activity or process is executed asynchronously with the requestor, following a RUN ASYNCHRONOUS command.

The activity is run in a separate unit of work from the unit of work of the requestor, and with the transaction attributes (transaction ID or user ID) specified on its resource definition - that is, a context-switch takes place.

Checking the response from a child activity

After a parent has requested a child activity to be run, it must check the response from the child by issuing a **CHECK ACTIVITY** command to find out whether the child activity succeeded.

The response to the request to run the activity does not contain any information about the success or failure of the child activity itself, only about the success or failure of the request to run it.

Typically, in the case of a synchronous child activity, the **CHECK ACTIVITY** command is issued immediately after the **RUN** command. For an asynchronous child activity, it could be issued:

- When the parent is reattached due to the completion event of the child firing. See "Reattachment events and activity activation" on page 25.
- When the parent is reattached due to the expiry of a timer.

If the child activity needs more than one processing step (transaction) to complete its work, on return from its first activation it is not complete. The **CHECK ACTIVITY** command returns the current completion status.

Following the execution of a **CHECK ACTIVITY** command issued by its parent, if the child activity has completed, its completion event, and its name, is deleted by CICS. The event cannot be deleted in any other way, because it *is* the completion of the activity.

For further information about the uses of the **CHECK ACTIVITY** command, see "Dealing with BTS errors and response codes" on page 29.

Lifetime of activities

A child activity is created when its parent issues a **DEFINE ACTIVITY** command. It is deleted automatically by CICS, either when its parent completes or if the parent issues a **DELETE ACTIVITY** command against it.

Note: It is not typically necessary to delete an activity explicitly.

Processing modes

Processing states or modes for activities can be active, canceling, complete, dormant, or initial. These modes describe the current state of an activity.

An activity is always in one of the following processing states or **modes** :

ACTIVE
An activation of the activity is running.

CANCELLING
CICS is waiting to cancel the activity. A **CANCEL ACTIVITY** command has been issued, but CICS cannot cancel the activity immediately because one or more of the descendants of the activity are inaccessible. For example, if one of the children of the activity holds a retained lock.

COMPLETE
The activity has completed, either successfully or unsuccessfully. The value returned on the COMPSTATUS option of a **CHECK ACTIVITY** command tells you how it completed.

DORMANT
The activity is waiting for an event to fire its next activation.

INITIAL
No RUN or LINK command has yet been issued against the activity; or the activity has been reset to its initial state with a **RESET ACTIVITY** command.

Figure 8 on page 19 is a (slightly simplified) view of how the processing modes relate to each other. The BTS commands that cause an activity to move from one mode to another are shown in uppercase.

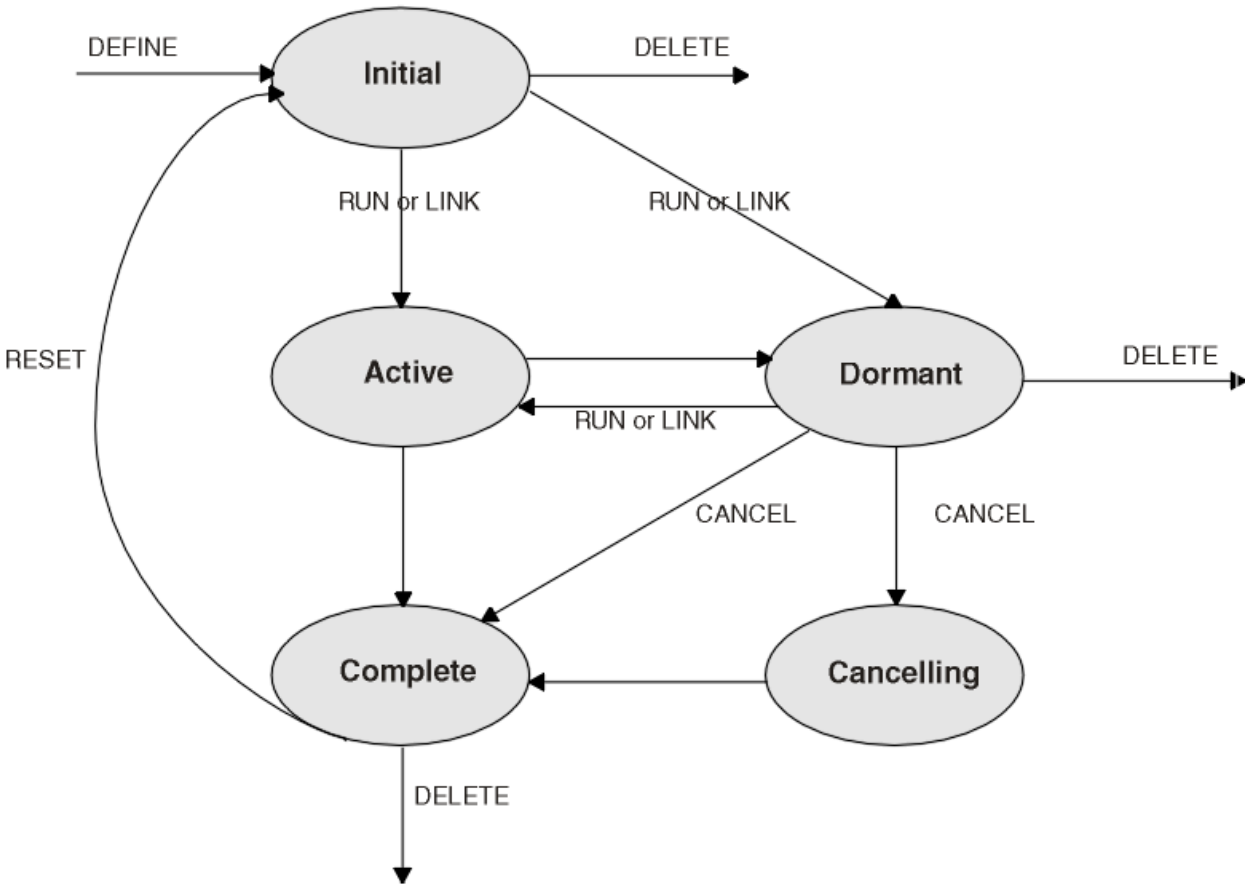


Figure 8. Activity mode transitions

Use the **CHECK ACTIVITY** or **INQUIRE ACTIVITYID** command to determine the current mode of an activity.

User sync points

These programs might issue user sync points: a *top-level* transaction that defines and runs a BTS process, a program that runs as the activation of a child activity, or a program executing outside the BTS environment that acquires a process or activity with an **ACQUIRE** command

A program that is running as the activation of a BTS *process* cannot issue user sync points (**EXEC CICS SYNCPOINT** commands).

For more information about acquiring a process or activity with an **ACQUIRE** command, see “Acquiring processes and activities” on page 40.

BTS data-containers

A *data-container* is a named area of storage, maintained by BTS. Because data-containers are preserved across multiple activations of the activity, they can be used to hold state data or inputs and outputs for the activity. They are recoverable resources, written to disk as necessary, and restored at system restart.

Each data-container is associated with an activity or process. It is identified by its name and by the activity for which it is a container. An activity can have any number of containers, as long as they all have different names within the scope of the activity. For example, several activities can each have containers named “Input”, “Output”, and “State”.

Data-containers of an activity serve as its working storage. They can be read and updated by the activity itself, by the parent of the activity, or by a program that has “acquired” the activity.

Just like an activity, a process might have a set of data-containers associated with it. These are called *process containers* : every activity in the process can access them, but only the root activity, or a program that has “acquired” the process, can update them.

Remember: A process's containers are not the same as its root activity's containers.

Before running a process, the program that creates it can:

- Create and set the process containers
- Create and set the containers of the root activity.

Alternatively, the root activity can create and set the process containers.

Lifetime of data-containers

Data-containers of the activity have the same lifetime as the activity itself. They are only destroyed when the activity itself is destroyed. While a child activity exists, its data-containers are always accessible to its parent, whatever processing mode the child is in (including complete).

If you issue a DELETE ACTIVITY command against an activity, bear in mind that you destroy the containers of the activity. It is typically best to allow activities to be deleted automatically by CICS. For child activities, this happens when the parent of the activity completes. At this stage, the parent no longer needs access to its children's containers. If the parent is reset and run again, it recreates its child activities.

BTS timers

A *timer* is a BTS object that expires when the system time becomes greater than a specified date and time, or after a specified period has elapsed. You manage timers with TIMER commands; for example, to cause an activity to be activated at a particular time.

Note: A timer that specifies a date and time that has already passed expires immediately. Similarly, if the requested interval is zero, the timer expires immediately.

To define a timer, use the DEFINE TIMER command. When you define a timer, a *timer event* is automatically associated with it. For more information, see “Atomic events” on page 21.

To force a timer to expire before its specified time, use the FORCE TIMER command.

To check whether a timer has expired and, if it has, whether it expired normally or following a FORCE TIMER command, use the CHECK TIMER command.

Timer management tips

- If a piece of processing (for example, *At midnight on 31st December, prepare an annual customer statement*) could result in many timers being set to expire at the same time, put the timers in groups and stagger the expiry times. Staggering the expiry times spreads the load on CICS and improves performance.
- If you shut down CICS at regular times, and know beforehand that at certain times it is unavailable, try not to set many timers to expire at these times. The timer events all fire when CICS is restarted, which could affect CICS startup performance.

BTS events

CICS business transaction services uses BTS events to signal progress in a process. An *event* informs an activity that an action is required or has completed.

“Event” is used in its ordinary sense of “something that happens”. To define an event recognizable by CICS business transaction services, such a happening is given a name. An activity program uses such commands as **DEFINE INPUT EVENT** , **DEFINE TIMER** , and the EVENT option of **DEFINE ACTIVITY** to name events about which it wants to be informed.

Named events have Boolean values - FIRED or NOTFIRED. When first defined, an event has the NOTFIRED value. When an event occurs it is said to *fire* (that is, to make the transition from NOTFIRED to FIRED). An activity can, for example:

- Discover the event (or events) whose firing caused it to be reattached (**RETRIEVE REATTACH EVENT**)
- Test whether an event has fired (**TEST EVENT**).

BTS events can be *atomic* or *composite*.

Atomic events

An *atomic event* is a single, low-level occurrence, which might happen under the control of BTS or outside the control of BTS.

There are four types of atomic event:

- Input events
- Activity completion events
- Timer events
- System events.

Atomic events are the basic components out of which composite events can be constructed. For more information, see “Composite events” on page 22.

Input events

Input events inform activities why they are being run. A RUN or LINK ACTIVITY command delivers an input event to an activity, and thus activates the activity. The INPUTEVENT option on the command *names* the input event and thus defines it to the requestor.

The first time an activity is run, CICS always sends it the DFHINITIAL **system event** . DFHINITIAL tells the activity to perform its initial housekeeping. Typically, this involves defining further events for which it might be activated.

An activity must use the RETRIEVE REATTACH EVENT command to discover the event or events that caused it to be activated. On any activation (but typically on its first, when it is started with DFHINITIAL), it might use the DEFINE INPUT EVENT command to define some input events for which it can be activated after.

Note: The RUN command can also be used to activate a *process* multiple times, delivering a different input event on each activation. This is not discussed here - see “Using client/server processing” on page 41.

Activity completion events

The completion of a child activity, but not a root activity, causes the *activity completion event* to fire. The EVENT option on the DEFINE ACTIVITY command names the activity completion event and thus defines it. If EVENT is not specified, the completion event is given the same name as the activity itself.

Timer events

When you define a timer, a *timer event* is automatically associated with it. When the timer expires, its associated event starts.

Note: If you do not specify the EVENT option of the DEFINE TIMER command, the timer event is given the same name as the timer itself.

System events

BTS *system events* are a special input event defined by BTS, unlike *user-defined events* , which are defined by the BTS application programmer.

All the other types of event described in this chapter , including composite events, are referred to as *user-defined events* , because they are defined by the BTS application programmer, using commands such as DEFINE INPUT EVENT, DEFINE TIMER, DEFINE COMPOSITE EVENT, and the EVENT option of DEFINE ACTIVITY.

There is only one type of BTS system event - DFHINITIAL. For more information, see “BTS system events” on page 81.

System events cannot be included in composite events.

Composite events

A *composite event* is a high-level event, formed from zero or more user-defined (that is, nonsystem) atomic events. When included in a composite event, an atomic event is known as a *sub-event* .

The **DEFINE COMPOSITE EVENT** command defines a *predicate* , which is a logical expression typically involving sub-events. At all times, the composite event's fire status reflects the value of the predicate. When the predicate becomes true, the composite event fires; when it becomes false, the composite's fire status reverts to NOTFIRED.

The logical operator that is applied to the composite event's predicate is one of the Boolean operators AND or OR.

When first defined, a composite event contains between zero and eight sub-events. (A composite event that contains zero sub-events is said to be “empty”.) The ADD SUBEVENT command can be used to add further sub-events to the composite event.

- A composite event that uses the OR Boolean operator fires when any of its sub-events fires.
- A composite event that uses the AND operator fires when all of its sub-events have fired, or when it is empty.

Figure 9 on page 22 shows four composite events, C1 through C4. Each composite event contains two sub-events. C1 and C2 use the OR Boolean operator. C3 and C4 use the AND operator. The shaded circles indicate the events that have fired.

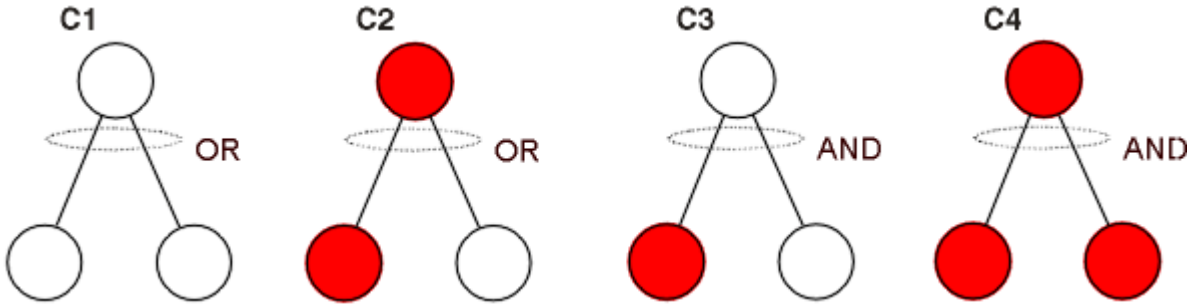


Figure 9. Composite events

Note:

1. An empty composite event that uses the AND operator is always true (FIRED). An empty composite event that uses the OR operator is always false (NOTFIRED).
2. The following *cannot* be added as sub-events to a composite event:
   - Composite events
   - Sub-events of other composite events
   - System events
   - Input events, if the composite uses the AND operator.

The subevent queue

The names of subevents that fire are placed on the subevent queue of the composite event, from where they can be retrieved by issuing one or more **RETRIEVE SUBEVENT** commands.

Each composite event has a subevent queue associated with it. The subevent queue might be empty or contain only the names of those subevents that have fired and not been retrieved.

Figure 10 on page 23 shows all the events that are recognized by a particular activity. Among them are two composite events, C1 and C2. The sub-event queue for C1 contains the name T1. The sub-event queue for C2 contains the names S1 and S3.

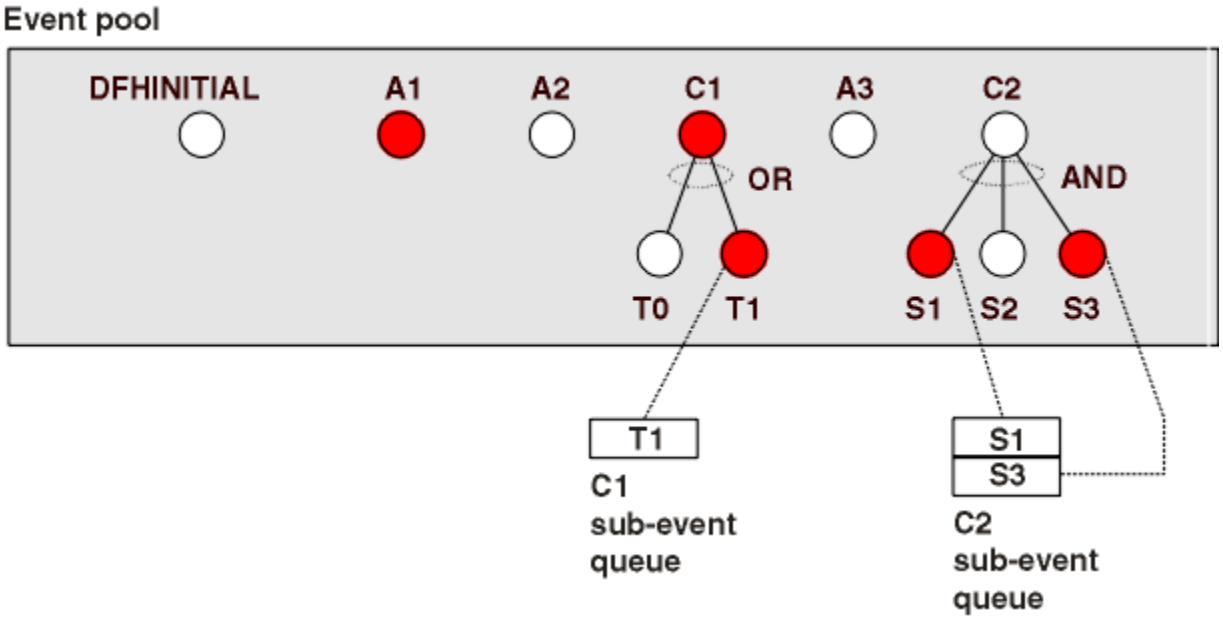


Figure 10. Subevent queues

Event pools

Events are defined in *event pools* . Each activity has an event pool, which contains the set of events that it recognizes.

The events that an activity recognizes are:

1. Events that have been defined to it with:
   - DEFINE COMPOSITE EVENT
   - DEFINE INPUT EVENT
   - DEFINE TIMER
   - The EVENT option of the DEFINE ACTIVITY command.
2. System events.

An event pool of an activity is initialized when the activity is created, and deleted when the activity is deleted. All the event-related commands described in “Event-related commands” on page 75 , except FORCE TIMER, operate on the event pool associated with the *current* activity.

Figure 11 on page 24 shows an event pool of an activity.

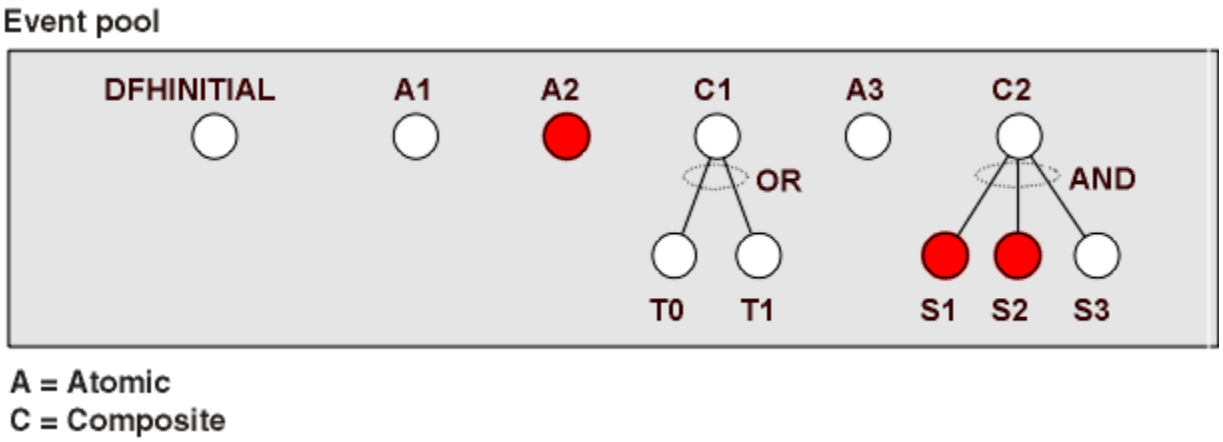


Figure 11. An event pool

Deleting events

You can delete an event by discarding both the event and its name. If the event is a subevent, the value of the composite event is that of its predicate, after the subevent is removed from the Boolean expression of the predicate.

About this task

The command you use to delete an event depends on the type of event to be deleted:

- To delete an input event explicitly, use the **DELETE EVENT** command.
- To delete a composite event explicitly, use the **DELETE EVENT** command. Deleting a composite event does not delete the subevents of the composite event.
- An activity completion event is implicitly deleted when a response from the completed activity has been acknowledged by a **CHECK ACTIVITY** command issued by the parent of the activity; or when a **DELETE ACTIVITY** command is issued.
- A timer event is implicitly deleted if its associated timer has expired and a **CHECK TIMER** command is issued by the activity that owns it; or when a **DELETE TIMER** command is issued.
- You cannot delete system events.
- If an activity program issues a **RETURN ENDACTIVITY** command, CICS automatically deletes all user events other than activity completion events, which must always be deleted with **CHECK ACTIVITY** or **DELETE ACTIVITY** commands in the event pool of the activity. See "Using the ENDACTIVITY option of the RETURN command" on page 29.

Table 2 on page 24 summarizes the commands that can be used to delete each type of event.

Table 2. Commands used to delete events

Event type	Deletion commands
Activity completion	1. **CHECK ACTIVITY** (if the activity has completed) 2. **DELETE ACTIVITY**
Composite	1. **DELETE EVENT** 2. **RETURN ENDACTIVITY**
Input	1. **DELETE EVENT** 2. **RETURN ENDACTIVITY**

Table 2. Commands used to delete events (continued)	
Event type	**Deletion commands**
System	Cannot be deleted
Timer	1. **`CHECK TIMER`** (if the timer has expired) 2. **`DELETE TIMER`** 3. **`RETURN ENDACTIVITY`**

Before an activity can complete normally, it must have deleted all the activity completion events in its event pool. This means that it must have dealt with all its child activities, see "Activity completion" on page 28.

Reattachment events and activity activation

An activity is reattached (reactivated) on the firing of any event, other than a subevent, that is in its event pool.

In other words, an activity is reattached when either of the following types of event occurs:

- A user-event that has been defined to the activity and not included in a composite event. The user-event might be:
  - An input event
  - The completion event for a child activity
  - A timer event
  - A composite event.
- A system event.

An event that causes an activity to be reactivated is known as a **reattachment event**.

Note: The firing of a subevent never directly causes an activity to be reattached—it is the firing of the associated composite event that does so. Therefore, a subevent can never be a reattachment event.

Handling reattachment events

When an activity is reattached, it uses the RETRIEVE REATTACH EVENT command to discover the event that caused reattachment. The names of reattachment events are placed on the reattachment queue, and from there the activity deals with them.

About this task

If the event that caused it to be reattached is composite, the activity might also need to issue one or more RETRIEVE SUBEVENT commands to discover the subevent or subevents that fired.

At times reattachment might occur because of the firing of more than one event. When reattachment events occur, their names are placed on a queue—the **reattachment queue** —from where they can be retrieved with RETRIEVE REATTACH EVENT commands. Each activity has a reattachment queue, which:

- May be empty
- Contains only the names of those reattachment events that have fired and not been retrieved.

Often, when an activity is reattached there is only one event on the reattachment queue, because activities are reactivated as each reattachment event occurs. However, it is possible for the reattachment queue to contain more than one event—if, for example, the activity has previously been suspended, and reattachment events occurred while it was suspended; or if two or more timer events fire simultaneously.

Figure 12 on page 26 shows the event pool and reattachment queue for a particular activity. The reattachment queue contains the names A1 and C1.

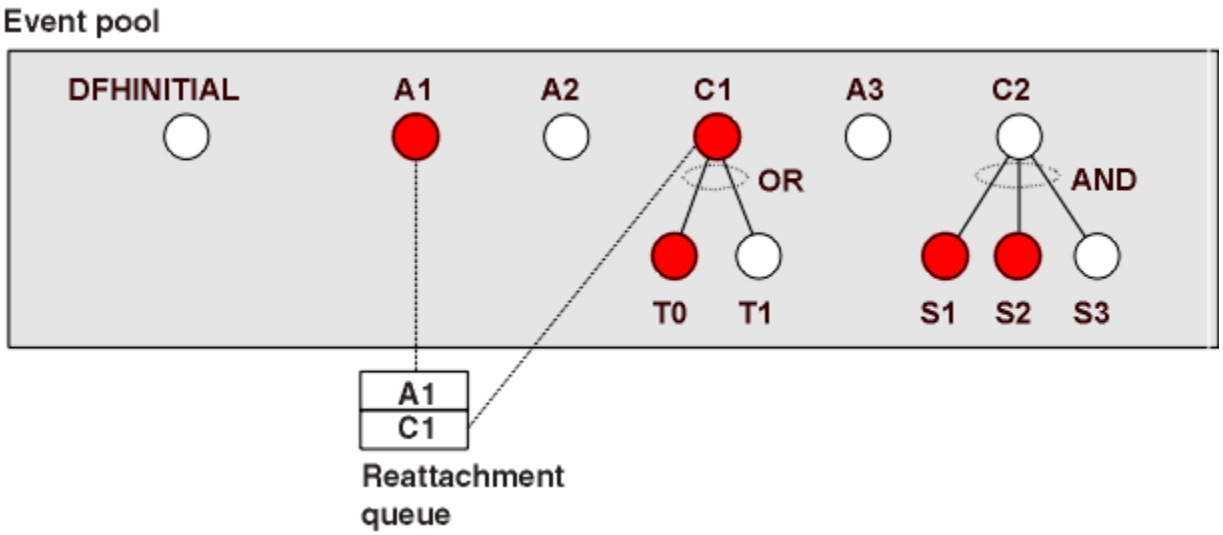


Figure 12. A reattachment queue

Important: With one exception, each time it is activated an activity must deal with at least one reattachment event. That is, it must issue at least one RETRIEVE REATTACH EVENT command, and (if this is not done automatically by CICS) reset the fire status of the retrieved event to NOTFIRED—see “Resetting and deleting reattachment events” on page 26 . Failure to do so results in the activity abending, because it has not progressed, it has not reset any reattachment events and is therefore in danger of getting into an unintentional loop.

The one exception to this general rule is if the activity program issues a RETURN ENDACTIVITY command —in which case, it is not required to have issued a RETRIEVE REATTACH EVENT command in the current activation.

If there are multiple events on its reattachment queue, an activity can, by issuing multiple RETRIEVE REATTACH EVENT commands, deal with several or all of them in a single activation. Alternatively, it can deal with them singly, by issuing only one RETRIEVE REATTACH EVENT command per activation and returning; it is then reactivated to deal with the next event on its reattachment queue. Which approach you choose is a matter of program design. Bear in mind, if you deal with several reattachment events in the same activation, that a sync point does not occur until the activation returns.

Resetting and deleting reattachment events

Retrieving an atomic event (but not a composite event) from the reattachment queue automatically causes the event's fire status to be reset to NOTFIRED. You can follow this sequence that an activity might use to handle reattachment events.

About this task

Retrieving a composite event from the reattachment queue does *not* reset the event's fire status to NOTFIRED, because a composite event is only reset when its predicate becomes false. Thus, if an activity program retrieves a composite event, it should reset the fire status of the sub-event or sub-events that have fired. (One way of doing this is to issue one or more RETRIEVE SUBEVENT commands.) This in turn causes the fire status of the composite event to be re-evaluated.

If the activity was reattached because of the completion of one of its children, it should issue a CHECK ACTIVITY command to check whether the child activity completed normally. On return from the CHECK ACTIVITY command, CICS deletes the activity completion event from the parent's event pool.

If the activity was reattached because of the expiry of a timer, it can issue a CHECK TIMER command to check whether the timer expired normally. On return from the CHECK TIMER command, CICS deletes the timer event from the activity's event pool.

If the activity wants to delete input and composite events from its event pool, it can issue DELETE EVENT commands. Alternatively, it can rely on a RETURN ENDACTIVITY command, issued on its final activation, to delete them.

Figure 13 on page 27 shows a typical sequence that an activity might use to handle reattachment events. The "Handle atomic event" box is expanded in Figure 14 on page 28.

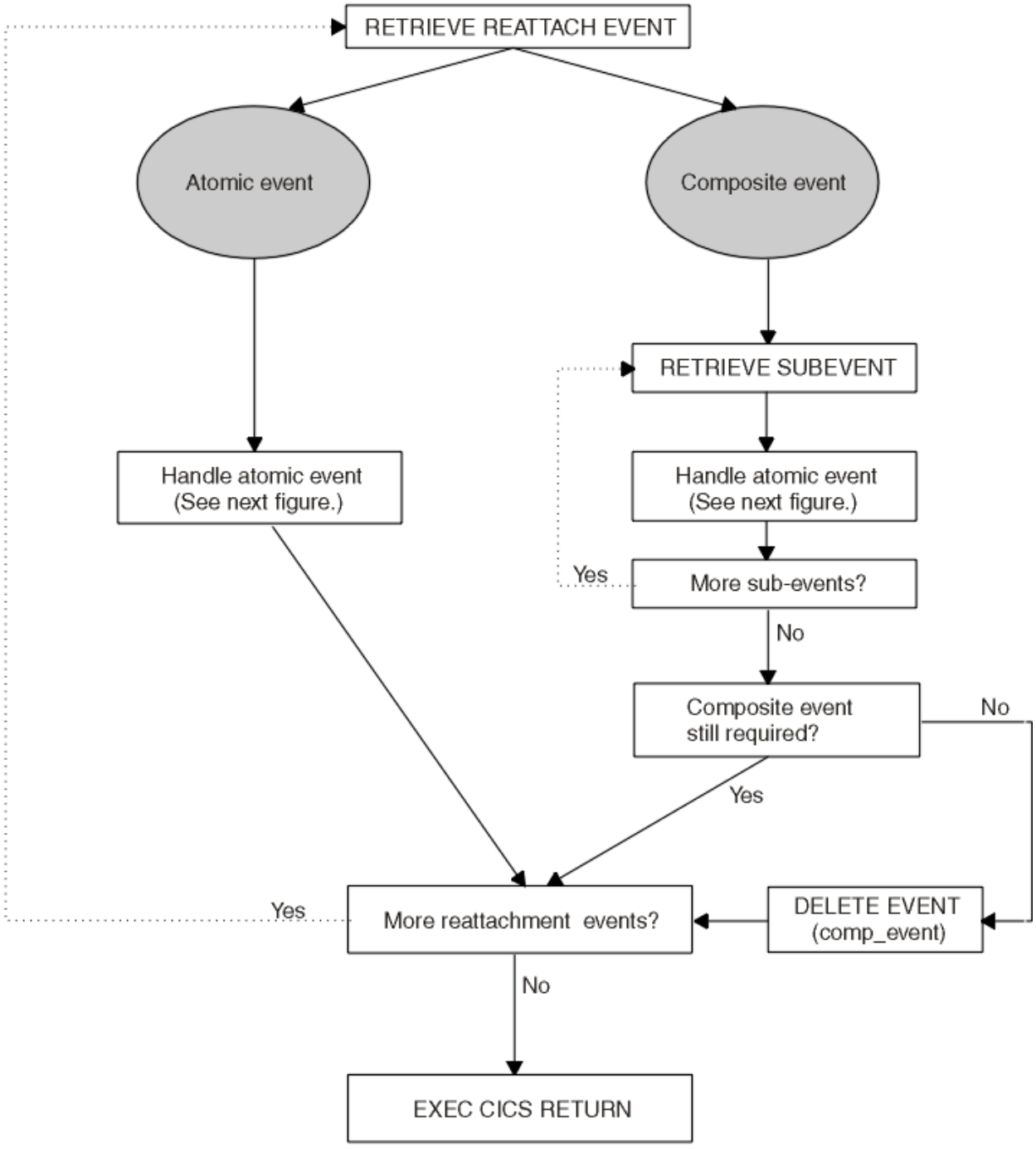


Figure 13. Handling reattachment events

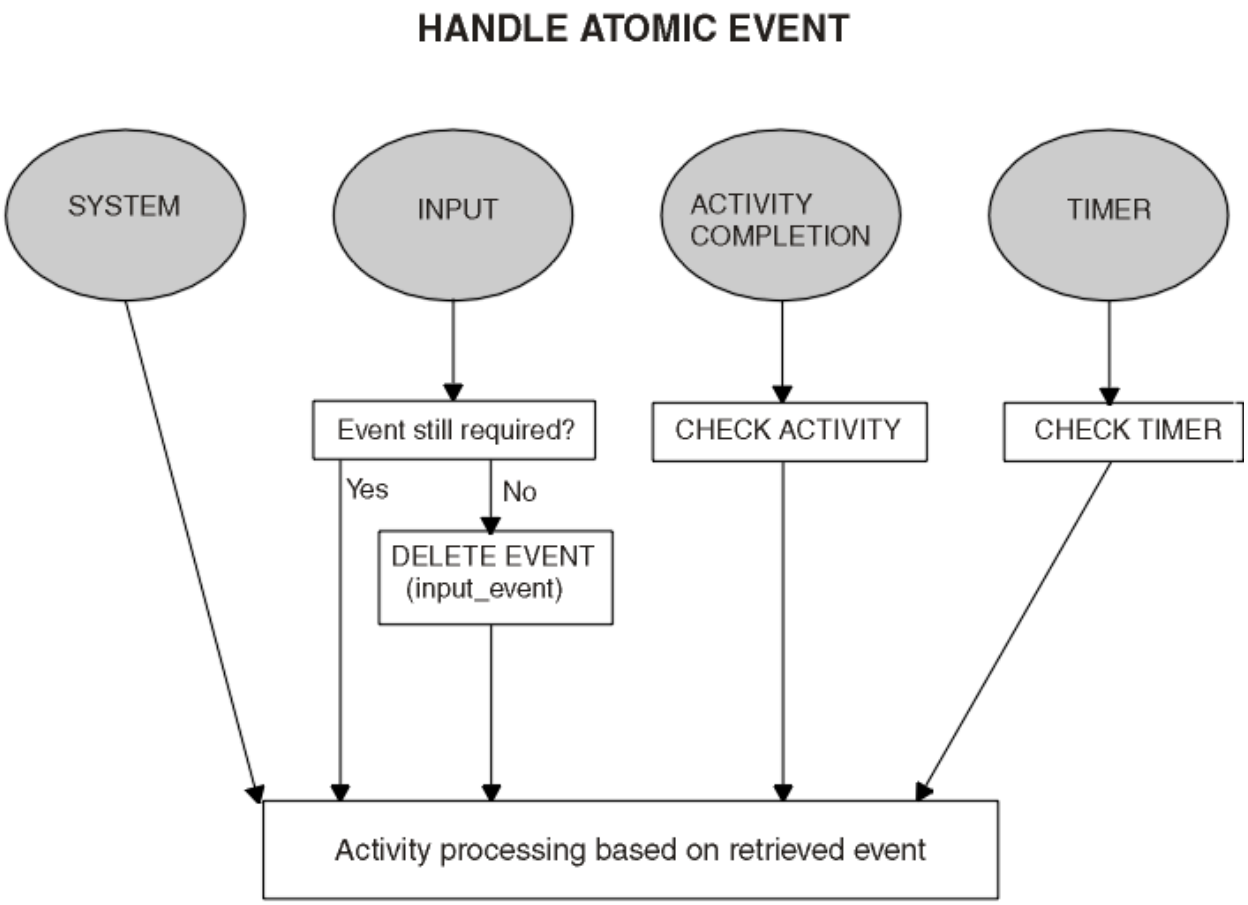


Figure 14. Handling atomic events

Note:

1. Figure 13 on page 27 shows multiple reattachment events being handled in a single activation. This may not always be appropriate. You may want always to retrieve only one reattachment event per activation, even if there is more than one event on the reattachment queue. This could be the case if, for example, you want a syncpoint to be taken between each processing step. (Note especially that a child activity that is run asynchronously is not started until a syncpoint occurs when its parent returns. Dealing with many reattachment events in the same activation could delay the start of the child.)
2. The figures show input and composite events being explicitly deleted by means of DELETE EVENT commands. This is not always strictly necessary—see "Using the ENDACTIVITY option of the RETURN command" on page 29 . Similarly, it may not always be necessary to issue CHECK TIMER commands. If you don't, timer events can be deleted by means of a RETURN ENDACTIVITY command issued on the activity's final activation.

Activity completion

An activity completes normally when it returns with no user events in its event pool.

When an activity issues an EXEC CICS RETURN command (without the ENDACTIVITY option):

1. If the activity has correctly dealt with at least one reattachment event during its current activation (see "Handling reattachment events" on page 25):

   If there are events on the reattachment queue
   The activity is immediately reactivated to deal with the fired events.

   If there are no events on the reattachment queue

   If there are user events in the event pool
   The activity becomes dormant until a reattachment event occurs.

If there are no user events in the event pool
The activity completes normally.

2. If the activity has not correctly dealt with at least one reattachment event during its current activation, it abends.

Using the ENDACTIVITY option of the RETURN command

You can use the ENDACTIVITY option of the **EXEC CICS RETURN** command to signal that an activity program has completed all of its processing steps and is not to be reactivated.

About this task

Optionally, an activity program can use the ENDACTIVITY option of the **EXEC CICS RETURN** command to signal that it has completed all its processing steps and should not be reactivated. One advantage of using ENDACTIVITY is that the activity program does not have to bother about deleting user events—other than activity completion events—from its event pool before completing; the events are deleted automatically by CICS.

When an activity issues an **EXEC CICS RETURN ENDACTIVITY** command:

If there are no user events in the event pool of the activity
The activity completes normally.

If there are user events, fired or unfired, in the event pool of the activity

- If one or more of the events are activity completion events, the activity abends. Trying to force an activity to complete before it has dealt with one or more of its child activities is a program logic error.
- If none of the events are activity completion events, the events are deleted and the activity completes normally.

It is recommended that you issue a **RETURN ENDACTIVITY** command at the end of the final activation of an activity, as a way of ensuring that the activity completes. For example, if, through a program logic error, an activity returns from what it believes to be its final activation with an unfired event in its event pool, it is possible that the activity could go dormant forever, and never complete. Coding **RETURN ENDACTIVITY** deletes the event and forces the activity to complete.

Dealing with BTS errors and response codes

Each time one of your applications issues a CICS command, CICS automatically raises a condition to tell it how the command worked. This condition (which is usually NORMAL) is returned by the CICS EXEC interface in the RESP and RESP2 options of the command.

About this task

If something out of the ordinary happens, the application receives an *exceptional condition* , which means a condition other than NORMAL. By testing this condition, it can tell what happened, and possibly why.

The tasks that you can perform include:

- “Checking the response from a synchronous activity” on page 30
- “Checking the response from an asynchronous activity” on page 31
- “Getting details of activity ABENDs” on page 31
- “Trying failed activities again” on page 32

Checking the response from a synchronous activity

The Order activity of the Sale application is created and run synchronously with SAL002.

About this task

Figure 15 on page 30 shows the Order activity of the Sale application being created and run synchronously with SAL002.

```
Order-Activity..
EXEC CICS DEFINE ACTIVITY('Order')
TRANSID('SORD')
PROGRAM('ORD001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Sale-Container)
ACTIVITY('Order') FROM(Process-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS LINK ACTIVITY('Order')
RESP(data-area) RESP2(data-area) END-EXEC.
```

Figure 15. Requests to create and activate an activity

The RESP and RESP2 options on a RUN ACTIVITY or LINK ACTIVITY command return any exceptional condition that is raised during the processing of the command. However, what is processed is a request for BTS to run the activity—that is, for BTS to accept and schedule the activity. Therefore, the RESP and RESP2 options do not return any exceptional condition that might result from processing the activity itself.

To check the response from the actual processing of any activity other than a root activity, you must issue one of the following commands:

CHECK ACTIVITY(child_name)
Used to check a child of the current activity.

CHECK ACQACTIVITY
Used to check the activity that the current unit of work has acquired with an ACQUIRE ACTIVITYID command.

For information about acquiring activities, see “Acquiring processes and activities” on page 40.

Root activities are a special case. They are activated automatically by BTS after a RUN ACQPROCESS or LINK ACQPROCESS command is issued; also, they do not have completion events. To check the processing of a process, and therefore of a root activity, use the CHECK ACQPROCESS command.

```
 EXEC CICS CHECK ACTIVITY('Order') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If.
.
```

Figure 16. The Sale root activity. SAL002,

Because Order is one of its child activities, SAL002 uses the CHECK ACTIVITY(child_name) form of the command.

The RESP and RESP2 options on the CHECK ACTIVITY command return a condition that tells you whether the CHECK command is understood by CICS —for example, ACTIVITYERR occurs if an activity named *Order* has not been defined to SAL002.

The COMPSTATUS option returns a CVDA value indicating the completion status of the activity:

- NORMAL is returned if the activity has completed all its processing steps.
- FORCED is returned if the activity was forced to complete with a CANCEL ACTIVITY command.
- INCOMPLETE is returned if the activity needs to be reactivated in order to complete all its processing steps.

- ABEND is returned if the program that implements the activity abended.

If a child activity completes, either successfully or unsuccessfully, and its parent issues a CHECK ACTIVITY command, the execution of the command causes CICS to delete the activity-completion event. Before a parent activity completes, it should ensure that the completion events of all its child activities have been deleted.

Note: If an activity completes and a CHECK ACQACTIVITY command is issued by a program other than its parent, the activity-completion event is not deleted. For example, a program executing outside a BTS process might issue an ACQUIRE ACTIVITYID command to acquire control of an activity within the process. It might then run the activity, and issue a CHECK ACQACTIVITY command to check the outcome. If the activity has completed, its completion event is not deleted.

The firing of the completion event causes the parent of the activity to be activated. Only if the parent issues a CHECK ACTIVITY command does CICS delete the completion event.

For an explanation of why a program executing outside a process might want to acquire an activity within the process, see “Interacting with BTS processes and activities” on page 39 . For an example of the use of the ACQUIRE ACTIVITYID and CHECK ACTIVITYID commands, see “Activity processing” on page 45.

Checking the response from an asynchronous activity

Asynchronous activities are treated almost identically to synchronous activities, the only difference being in the point at which the CHECK ACTIVITY command is issued.

About this task

Typically, for a synchronous activity, the CHECK ACTIVITY command is issued immediately after the RUN or LINK command. For an asynchronous activity, it might, for example, be issued:

- When the parent is reattached due to the firing of the activity's completion event.
- When the requestor is reattached due to the expiry of a timer. This could occur if the requestor expects the activity to return without completing; the requestor may then reactivate the activity by sending it an input event.

Getting details of activity ABENDs

If a CHECK ACTIVITY command returns a completion status (COMPSTATUS) of ABEND, you can use the INQUIRE ACTIVITYID command to obtain further information about how the activity abended.

Example

This example returns the name of the program in which the abend occurred, together with the corresponding CICS abend code.

```
 If status = DFHVALUE(ABEND).
To get the activity-identifier of the failed child,
start a browse of child activities
EXEC CICS STARTBROWSE ACTIVITY
BROWSETOKEN(root-token)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GETNEXT ACTIVITY(child-name)
BROWSETOKEN(root-token)
ACTIVITYID(child-id)
RESP(data-area) RESP2(data-area) END-EXEC.
loop until the failed child is found by name
EXEC CICS GETNEXT ACTIVITY(child-name)
BROWSETOKEN(root-token)
ACTIVITYID(child-id)
RESP(data-area) RESP2(data-area) END-EXEC.
end child activity browse loop
Inquire on the failed child, using its activity-identifier
EXEC CICS INQUIRE ACTIVITYID(child-id)
ABCODE(data-area)
ABPROGRAM(data-area)
RESP(data-area) RESP2(data-area) END-EXEC
```

Tip: A simpler way of obtaining the activity-identifier of the failed child activity (used on the EXEC CICS INQUIRE ACTIVITYID command) would be to code the ACTIVITYID option of the DEFINE ACTIVITY command used to define the child, and to store the returned value.

Trying failed activities again

If a child activity fails, you can try it again. The parent issues a CHECK ACTIVITY command, if it has not already done so, to check the current completion status of the child activity.

About this task

Procedure

1. Issue a RESET ACTIVITY command.

   The child activity is reset to its initial state: its completion event is added to the event pool of the parent, with the status set to NOTFIRED; any children of the child activity are deleted. The data-containers of the child activity are not disturbed.
2. Issue a RUN ACTIVITY command.

   The child activity is invoked with a DFHINITIAL event.

Example of running parallel BTS activities

Many business transactions include activities that can run in parallel with one another. To illustrate parallel activities, the following example extends the Sale business transaction to support multiple delivery activities.

The logic of the Sale business transaction is changed so that an order can include multiple items, each potentially requiring delivery to a separate location. Each delivery request (activity) can run in parallel, but the customer is not invoiced until all the items have been delivered.

Data flow

Follow this example to see how data flows in the Sale example application when parallel activities are included. The root activity is not shown. Changes from the basic Sale example described in The Sale example application are shown in bold.

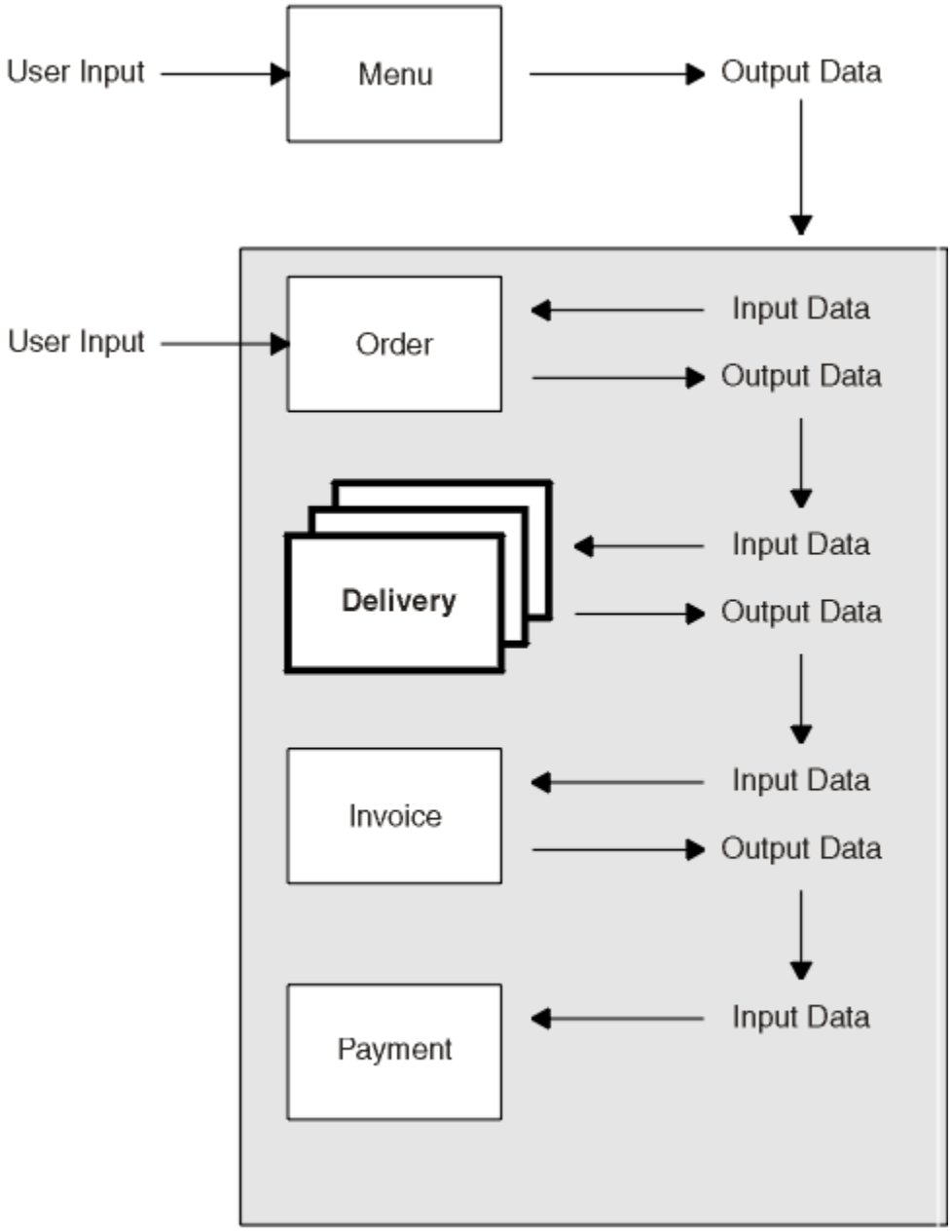


Figure 17. Data flow for parallel activities

- User data (an account number) collected after the user selects the `Sale` menu option is used as input to the Order activity.
- User data collected by the Order activity is used as input to multiple Delivery activities.
- The output data produced by the Delivery activities is used as input to the Invoice activity.
- The output produced by the Invoice activity is used as input to the Payment activity.

Root activity

Figure 18 on page 34 shows, in COBOL pseudocode, the Sale root activity with modifications for parallel activities. **`CHECK ACTIVITY`** commands have also been added, to check the response from each child activity and to delete its completion event. The changes are in bold text.

```
Identification Division.
Program-id. SAL002.
Environment Division.
Data Division.
Working-Storage Section.
01 Switches.
05 No-More-Events pic x value space.
88 No-More-Events value 'y'.
01 Switch-Off Pic x value 'n'.
01 RC pic s9(8) comp.
01 Process-Name pic x(36).
01 Event-Name pic x(16).
88 DFH-Initial value 'DFHINITIAL'.
88 Delivery-Complete value 'Delivry-Complete'.
88 Invoice-Complete value 'Invoice-Complete'.
88 Payment-Complete value 'Payment-Complete'.
01 Sale-Container pic x(16) value 'Sale'.
01 Order-Container pic x(16) value 'Order'.
01 Order-Buffer.
05 Order-Count Pic 9(2).
05 Order-Item occurs 1 to 20 times
Depending on Order-Count Pic X(10).
01 Delivery-Container pic x(16) value 'Delivery'.
01 Delivery-Buffer.
05 Delivery-Count pic 9(2).
05 Delivery-Item occurs 1 to 20 times
Depending on Delivery-Count pic x(30).
01 Invoice-Container pic x(16) value 'Invoice'.
01 Invoice-Buffer Pic x(..).
01 Work-Activity.
05 Work-Name Pic x(8) value 'Delivery'.
05 Filler pic x(6) value '-Item-'.
05 Work-Count pic 9(2) value zero.
01 Work-Event.
05 Event-Name pic x(8) value 'Del-Comp'.
05 Filler pic x(6) value '-Item-'.
05 Event-Count pic x(2) value zero.
Linkage Section.
01 DFHEIBLK..
```

Figure 18. The SAL002 root activity program, with modifications for parallel activities highlighted (Part 1)

```
Procedure Division.
Begin-Process..
EXEC CICS RETRIEVE REATTACH EVENT(Event-Name)
RESP(RC) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
Evaluate True
When DFH-Initial
Perform Initial-Activity
Perform Order-Activity
 Perform Order-Response
Perform Delivery-Activity
When Delivery-Complete
 Perform Delivery-Response
Perform Invoice-Activity
When Invoice-Complete
 Perform Invoice-Response
Perform Payment-Activity
When Payment-Complete
 Perform Payment-Response
Perform End-Process
When Other.
End Evaluate.

EXEC CICS RETURN END-EXEC.
Initial-Activity..
EXEC CICS ASSIGN PROCESS(Process-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
Order-Activity..
EXEC CICS DEFINE ACTIVITY('Order')
TRANSID('SORD')
PROGRAM('ORD001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Sale-Container)
ACTIVITY('Order') FROM(Process-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS LINK ACTIVITY('Order')
RESP(data-area) RESP2(data-area) END-EXEC.
```

Figure 19. The SAL002 root activity program, with modifications for parallel activities highlighted (Part 2)

```
Order-Response..
EXEC CICS CHECK ACTIVITY('Order') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If.
.
Delivery-Activity..
EXEC CICS GET CONTAINER(Order-Container)
ACTIVITY('Order') INTO(Order-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.

EXEC CICS DEFINE COMPOSITE EVENT('Delivry-Complete') AND
RESP(data-area) RESP2(data-area) END-EXEC.
Perform Delivery-Work varying Work-Count from 1 by 1
until Work-Count greater than Order-Count..
Delivery-Work..
Move Work-Count to Event-Count.
EXEC CICS DEFINE ACTIVITY(Work-Activity)
TRANSID('SDEL')
PROGRAM('DEL001')
EVENT(Work-Event)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS ADD SUBEVENT(Work-Event) EVENT('Delivry-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Order-Container)
ACTIVITY(Work-Activity) FROM(Order-Item(Work-Count))
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY(Work-Activity)
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Delivery-Response..
Move zeros to Delivery-Count
Move Switch-Off to No-More-Events.
Perform until No-More-Events
EXEC CICS RETRIEVE SUBEVENT(Work-Event) EVENT('Delivry-Complete')
RESP(data-area) RESP2(data-area) END-EXEC
```

Figure 20. The SAL002 root activity program, with modifications for parallel activities highlighted (Part 3)

```
 If RC NOT = DFHRESP(NORMAL).
If RC = DFHRESP(END)
Set No-More-Events to TRUE
EXEC CICS DELETE EVENT('Delivry-Complete')
Else.
End-If
Else
Move Event-Count to Work-Count
Add 1 to Delivery-Count.
EXEC CICS CHECK ACTIVITY(Work-Activity) COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If..
EXEC CICS GET CONTAINER(Delivery-Container)
ACTIVITY(Work-Activity)
INTO(Delivery-Item(Work-Count))
RESP(data-area) RESP2(data-area) END-EXEC.
End-If
End-Perform
.
Invoice-Activity..
EXEC CICS DEFINE ACTIVITY('Invoice')
TRANSID('SINV')
EVENT('Invoice-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Delivery-Container)
ACTIVITY('Invoice') FROM(Delivery-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Invoice')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Invoice-Response..
EXEC CICS CHECK ACTIVITY('Invoice') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If.
.
```

Figure 21. The SAL002 root activity program, with modifications for parallel activities highlighted (Part 4)

```
Payment-Activity..
EXEC CICS DEFINE ACTIVITY('Payment')
TRANSID('SPAY')
EVENT('Payment-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GET CONTAINER(Invoice-Container)
ACTIVITY('Invoice') INTO(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Invoice-Container)
ACTIVITY('Payment') FROM(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Payment')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Payment-Response..
EXEC CICS CHECK ACTIVITY('Payment') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If.
.
End-Process..
EXEC CICS RETURN ENDACTIVITY
RESP(data-area) RESP2(data-area) END-EXEC
End Program.
```

Figure 22. The SAL002 root activity program, with modifications for parallel activities highlighted (Part 5)

The output from the Order activity (retrieved into the variable *Order-Buffer*) is now an array of order items. There can be 1 - 20 items in an order. Having first defined a composite event (*Delivry-Complete*), SAL002 requests a delivery activity to be run for each item ordered:

```
 EXEC CICS DEFINE COMPOSITE EVENT('Delivry-Complete') AND
RESP(data-area) RESP2(data-area) END-EXEC.
Perform Delivery-Work varying Work-Count from 1 by 1
until Work-Count greater than Order-Count.
```

All the delivery activities run in parallel. The following set of requests are made for each order item:

```
Delivery-Work..
Move Work-Count to Event-Count.
EXEC CICS DEFINE ACTIVITY(Work-Activity)
TRANSID('SDEL')
PROGRAM('DEL001')
EVENT(Work-Event)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS ADD SUBEVENT(Work-Event) EVENT('Delivry-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Order-Container)
ACTIVITY(Work-Activity) FROM(Order-Item(Work-Count))
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY(Work-Activity)
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC
```

Note that:

- The delivery activity for each order item is given a unique name (*Delivery-Item-n* —the value of *Work-Activity* —where *n* is the 1-20 item number).
- Each delivery activity is provided with an input data-container named *Order* , which contains one of the order items from the *Order-Buffer* array.
- The completion event for each delivery activity is given a unique name (*Del-Comp-Item-n* , the value of *Work-Event*). The **ADD SUBEVENT** command is used to add the completion event for each delivery activity to the composite event *Delivry-Complete*.

  The completion of an individual delivery activity does *not* cause SAL002 to be reattached—because the completion events of the delivery activities have been specified as subevents of the composite event *Delivry-Complete* . Instead, SAL002 is reattached when *Delivry-Complete* fires. Because *Delivry-Complete* uses the AND Boolean operator, it fires when *all* the completion events of the individual delivery activities have fired.

Before the Invoice activity is run, the output from each of the delivery activities is accumulated into a *Delivery-Item* array:

```
Delivery-Response..
Move zeros to Delivery-Count
Move Switch-Off to No-More-Events.
Perform until No-More-Events
EXEC CICS RETRIEVE SUBEVENT(Work-Event) EVENT('Delivry-Complete')
RESP(data-area) RESP2(data-area) END-EXEC

If RC NOT = DFHRESP(NORMAL).
If RC = DFHRESP(END)
Set No-More-Events to TRUE
 EXEC CICS DELETE EVENT('Delivry-Complete')
Else
.
End-If
Else
Move Event-Count to Work-Count
Add 1 to Delivery-Count.
 EXEC CICS CHECK ACTIVITY(Work-Activity) COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If.
.
EXEC CICS GET CONTAINER(Delivery-Container)
```

```
ACTIVITY(Work-Activity)
INTO(Delivery-Item(Work-Count))
RESP(data-area) RESP2(data-area) END-EXEC.
End-If
End-Perform
```

The contents of the *Delivery-Item* array are placed in the input data-container of the Invoice activity.

Note that:

- When SAL002 is reattached due to the firing of the *Delivry-Complete* composite event, it uses a succession of **EXEC CICS RETRIEVE SUBEVENT** commands to retrieve, in turn, each subevent on the subevent queue of the composite event - that is, each subevent whose firing was instrumental in the firing of the composite event. These subevents are the completion events for each of the delivery activities. The number of each subevent (contained in the *Event-Count* field of *Work-Event*) is used to identify the particular delivery activity for which the subevent is the completion event.
- When all the subevents have been retrieved, SAL002 deletes the composite event *Delivry-Complete* . This deletion is not strictly necessary, because user-defined events, other than activity completion events, are automatically deleted by CICS when a **RETURN ENDACTIVITY** command is issued.

  Deleting a composite event does *not* delete any associated subevents. In this example, the subevents are the completion events for child activities. The completion event for a child activity is deleted automatically when, as here, an **EXEC CICS CHECK ACTIVITY** command is issued by the parent after the child has completed. The **CHECK ACTIVITY** command is described in “Dealing with BTS errors and response codes” on page 29.

  It is an error for an activity to issue an **EXEC CICS RETURN ENDACTIVITY** command while there are still activity completion events in its event pool.

Interacting with BTS processes and activities

You can use BTS processes and activities to interact with the world outside the BTS environment. A program running outside a process can use BTS to acquire access to an activity in the process.

In the examples we have looked at so far, after the initial order details are collected from a user terminal the Sale business transaction proceeds without further interaction with the outside world. Each activity is started automatically by CICS business transaction services, following the completion of its predecessor.

In practice, many business transactions require some external interaction. For example, most business transactions include activities that require human involvement. These activities are known as *user-related activities* . User-related activities cannot be started automatically by BTS, because they rely on the user being ready to process the work. Other examples of external interactions are dependencies on input from the World Wide Web or from IBM® MQ queues.

For example, the CICS transactions can:

Use BTS processes as servers
A client transaction outside a process can “acquire” the root activity of the process. This enables it to pass business data to the process in the process or root activity's containers. The transaction does not become part of the process - rather, it is able to activate the process and use it as a server.

Acquire BTS activities
A transaction outside a process can acquire a descendant activity within the process. Acquiring the activity gives the transaction access to the activity's containers, and allows it to activate the activity.

Both these examples use input events to signify that a process or activity requires some external interaction to take place before it can complete.

It contains:

- “Using client/server processing” on page 41
- “Activity processing” on page 45
- “Transferring data to asynchronous activations” on page 55.

Acquiring processes and activities

Before a program that runs outside a process can activate an activity in the process, it uses the **ACQUIRE** command to acquire access to the activity.

About this task

Acquiring an activity enables the program to:

- Read and write to the containers for the activity.
- Issue various commands, including RUN and LINK, against the activity. If the acquired activity is a root activity, the program can issue the commands against the process.

To gain access to an activity from outside the process that contains it, you use the ACQUIRE command. An activity that a program accesses with an ACQUIRE command is known as an *acquired activity*.

There are two forms of the ACQUIRE command:

ACQUIRE ACTIVITYID
Acquires the specified descendant (non-root) activity.

ACQUIRE PROCESS
Acquires the root activity of the specified process.

Note: When a program defines a process, it is automatically given access to root activity for the process. This enables the defining program to access the process containers and root activity containers before running the process. When a program gains access to a root activity with *either* a DEFINE PROCESS or an ACQUIRE PROCESS command, the process is known as the *acquired process*.

For definitive information about the ACQUIRE command, see ACQUIRE .

Process and Activity rules

These rules apply when acquiring or accessing an activity or a process.

1. A program can acquire only one activity within the same unit of work. The activity remains acquired until the next sync point. This means, for example, that a program:
   - Cannot issue both a DEFINE PROCESS and an ACQUIRE PROCESS command within the same unit of work.
   - Cannot issue both an ACQUIRE PROCESS and an ACQUIRE ACTIVITYID command within the same unit of work. That is, it can acquire *either* a descendant activity or a root activity, not one of each.
2. If a program is executing as an activation of an activity, it cannot:
   - Acquire an activity in the same process as itself. It cannot, for example, issue ACQUIRE PROCESS for the current process.
   - Use a LINK command to activate the activity that it has acquired.
3. A process of an acquired activity is accessible in the same way as the activity itself can access it. Thus, if the acquired activity is a descendant activity:
   - The containers of the process might be read but not updated.
   - The process might not be the subject of any command - such as RUN, LINK, SUSPEND, RESUME, or RESET - that directly manipulates the process or its root activity.

   Conversely, if the acquired activity is a root activity:
   - The containers of the process might be both read and updated.
   - The process might be the subject of commands such as RUN, LINK, SUSPEND, RESUME, or RESET. The ACQPROCESS keyword on the command identifies the subject process as the one the program that issues the command has acquired in the current unit of work.

Using client/server processing

CICS business transaction services support client/server processing. These examples show you how BTS client/server processing works.

A server process is one that is typically waiting for work. When work arrives, BTS restarts the process, which retrieves any state data that it has previously saved. Typically, the client invokes the server with a named input event, and sends it some input data in a data-container. From these inputs, the server determines what actions it needs to take. It returns any output for the client in a data-container.

When the client has dealt with any output returned by the server, it releases the server process. Releasing the server means that its in-memory instance is freed. The server process is maintained only by BTS.

Client/server examples

The client/server examples in this section show:

1. A client program initiating a server process and calling it with some work to do.
2. The server defining some input events for which it might be invoked again; then performing some work and returning output to the client.
3. After dealing with the output returned by the server, the client releasing the in-memory instance of the server.
4. The client reacquiring the server process and requesting it to run again.
5. The server process determining the input event that caused it to be invoked again, and retrieving some state data that it saved when it last ran; then performing some work and returning output to the client.
6. Eventually, the client telling the server to shut down, and the server responding to this event by indicating that it must not be invoked again.

The client program

Step through the client program PRG001, in COBOL pseudocode, as it creates a server process and requests that it is run.

```
Identification Division.
Program-id. PRG001.
Environment Division.
Data Division.
Working-Storage Section.
01 RC pic s9(8) comp.
01 Unique-Reference pic x(36) value low-values..
01 Process-Type pic x(8) value 'Servers'.
.
01 Event-Name pic x(16) value low-values..
01 Work-Buffer..
01 Work-request Pic x.
88 Work-New value 'N'.
88 Work-Continue value 'C'.
88 Work-End value 'E'.

Linkage Section.
01 DFHEIBLK..
01 DFHCOMMAREA...
```

Figure 23. Example client program, PRG001 (Part 1)

```
Procedure Division using DFHEIBLK DFHCOMMAREA.
In-The-Beginning..
EXEC CICS SEND ...
RESP(data-area) END-EXEC.
EXEC CICS RECEIVE ...
RESP(data-area) END-EXEC.
Move ..unique.. TO Unique-Reference
Move ..request.. TO Work-Request.
Evaluate True
When Work-New
Perform New-Process
When Work-Continue
Move 'SRV-WORK' TO Event-Name
Perform Existing-Process
When Work-End
Move 'SRV-SHUTDOWN' TO Event-Name
Perform Existing-Process
When Other.
End Evaluate..
EXEC CICS GET CONTAINER('Server-Out')
 ACQPROCESS INTO(Work-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS SEND ...
RESP(data-area) END-EXEC.
EXEC CICS RETURN END-EXEC.
New-Process..
 EXEC CICS DEFINE PROCESS(Unique-Reference)
PROCESSTYPE(Process-Type)
TRANSID('SERV')
PROGRAM('SRV001')
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS PUT CONTAINER('Server-In')
ACQPROCESS FROM(Work-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS RUN ACQPROCESS
SYNCHRONOUS
RESP(RC) RESP2(data-area) END-EXEC.
```

Figure 24. Example client program, PRG001 (Part 2)

```
Existing-Process..
 EXEC CICS ACQUIRE PROCESS(Unique-Reference)
PROCESSTYPE(Process-Type)
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS PUT CONTAINER('Server-In')
ACQPROCESS FROM(Work-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS RUN ACQPROCESS
SYNCHRONOUS
INPUTEVENT(Event-Name)
RESP(RC) RESP2(data-area) END-EXEC.
End Program.
```

Figure 25. Example client program, PRG001 (Part 3)

First, PRG001 determines if this is the first time the server is to be called. If it is, it establishes a unique name for this instance of the server process. Then it creates the server process by issuing a DEFINE PROCESS command with that unique name. PRG001 provides some input data for the server in a data-container named *Server-In* :

```
 EXEC CICS PUT CONTAINER('Server-In')
ACQPROCESS FROM(Work-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC
```

The ACQPROCESS option associates the *Server-In* container with the process that PRG001 has “acquired”. A program “acquires” access to a process in one of two ways: either, as here, by defining it; or, if the process has already been defined, by issuing an ACQUIRE PROCESS command.

Having created the server process, PRG001 issues a request to run it synchronously. The RUN ACQPROCESS command causes the currently acquired process to be activated. Because RUN ACQPROCESS rather than LINK ACQPROCESS is used, the server process is run in a separate unit of

work from that of the client. PRG001 waits for the server to run, and then retrieves any data returned from a data-container named *Server-Out*.

PRG001 has now temporarily finished using the server process; the implicit sync point at RETURN causes it to be released.

To use this instance of the server again, PRG001 must first acquire access to the correct process. It does this by issuing an ACQUIRE PROCESS command which specifies the unique combination of the name and process-type of the process:

```
 EXEC CICS ACQUIRE PROCESS(Unique-Reference)
PROCESSTYPE(Process-Type)
RESP(data-area) RESP2(data-area) END-EXEC
```

Once again, PRG001 provides input data for the server in a data-container named *Server-In* , and requests the process to be run:

```
 EXEC CICS RUN ACQPROCESS
SYNCHRONOUS
INPUTEVENT(Event-Name)
RESP(RC) RESP2(data-area) END-EXEC
```

PRG001 uses the INPUTEVENT option of the RUN command to tell the server why it has been invoked—in this case, it is for *SRV-WORK*. (The server must have defined an input event of that name.)

Again, PRG001 waits for the process to complete, retrieves any returned data, and releases the process.

Eventually, PRG001 tells the server to shut down by invoking it with an event of *SRV-SHUTDOWN*.

The server program

Step through the server program SRV001, in COBOL pseudocode, as it determines why it was called, carries out housekeeping, and performs its work.

```
Identification Division.
Program-id. SRV001.
Environment Division.
Data Division.
Working-Storage Section.
01 Event-Name pic x(16).
88 DFH-Initial value 'DFHINITIAL'.
88 SRV-Request value 'SRV-REQUEST'.
01 Sub-Event-Name pic x(16).
88 SRV-Work value 'SRV-WORK'.
88 SRV-Shutdown value 'SRV-SHUTDOWN'.
01 Input-Buffer..
01 Output-Buffer..
01 State-Buffer..
Linkage Section.
01 DFHEIBLK..
Procedure Division.
Begin-Process..
EXEC CICS RETRIEVE REATTACH EVENT(Event-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
Evaluate True
When DFH-Initial
Perform Initial-Request
Perform Server-work
When SRV-Request
Perform Server-Event
When Other.
End Evaluate..
EXEC CICS RETURN END-EXEC
.
```

Figure 26. Example server program, SRV001 (Part 1)

```
Server-Event..
EXEC CICS RETRIEVE SUBEVENT(Sub-Event-Name) EVENT(Event-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
Evaluate True
When SRV-Work
Perform Server-Work
When SRV-Shutdown
Perform Server-Shutdown
When Other.
End Evaluate..
Initial-Request..
EXEC CICS DEFINE INPUT EVENT('SRV-WORK')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DEFINE INPUT EVENT('SRV-SHUTDOWN')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DEFINE COMPOSITE EVENT('SRV-REQUEST') OR
SUBEVENT1('SRV-WORK')
SUBEVENT2('SRV-SHUTDOWN')
RESP(data-area) RESP2(data-area) END-EXEC.
Server-Work..
EXEC CICS GET CONTAINER('Server-In') INTO(Input-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
If DFH-Initial
EXEC CICS DEFINE ACTIVITY('Work')
TRANSID('SWRK')
PROGRAM('PRG002')
RESP(data-area) RESP2(data-area) END-EXEC.
Else
EXEC CICS GET CONTAINER('Previous-State') INTO(State-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
End-If..
EXEC CICS PUT CONTAINER('Work-Input')
ACTIVITY('Work') FROM(Input-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Work')
SYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC
```

Figure 27. Example server program, SRV001 (Part 2)

```
 EXEC CICS CHECK ACTIVITY('Work') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If..
EXEC CICS GET CONTAINER('Work-Output')
ACTIVITY('Work') INTO(Output-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER('Previous-State') FROM(State-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER('Server-Output') FROM(Output-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
Server-Shutdown.
EXEC CICS DELETE EVENT('SRV-WORK')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DELETE EVENT('SRV-SHUTDOWN')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DELETE EVENT('SRV-REQUEST')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RETURN ENDACTIVITY
RESP(data-area) RESP2(data-area) END-EXEC
End Program.
```

Figure 28. Example server program, SRV001 (Part 3)

The server program, SRV001, first issues a RETRIEVE REATTACH EVENT command to determine the reason for its invocation. On its first invocation, the event returned is DFHINITIAL, which tells SRV001 to perform any initial housekeeping. The housekeeping of the SRV001 program includes defining two input events for which it could later be invoked again:

```
 EXEC CICS DEFINE INPUT EVENT('SRV-WORK')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DEFINE INPUT EVENT('SRV-SHUTDOWN')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DEFINE COMPOSITE EVENT('SRV-REQUEST') OR
```

```
SUBEVENT1('SRV-WORK')
SUBEVENT2('SRV-SHUTDOWN')
RESP(data-area) RESP2(data-area) END-EXEC.
```

The DEFINE COMPOSITE EVENT command defines a third, composite, event (*SRV-REQUEST*), and adds the two input events to it. Because the composite event uses the OR Boolean operator, it fires when *either* of the two input events fires; SRV001 is reattached.

SRV001 obtains its input data from a data-container named *Server-In*. It then performs the work activity *Work*.

When the work activity has completed, SRV001 saves some state data for the next time it is run, and returns the output data produced by the work activity to the client program in a data-container named *Server-Output*.

On subsequent invocations, SRV001 determines that it has been invoked to perform work. (The RETRIEVE REATTACH EVENT command returns the composite event *SRV-REQUEST* , and a RETRIEVE SUBEVENT command with an event-name of *SRV-REQUEST* returns the subevent *SRV-WORK* .)

Eventually, the RETRIEVE SUBEVENT command returns the subevent *SRV-SHUTDOWN* , and SRV001 responds by ending the server process. First it deletes the user events that it has defined, then issues an EXEC CICS RETURN ENDACTIVITY command to indicate that it has completed all its processing.

Activity processing

Activity processing is organized in two activations. The first activation sets up the environment. The second activation starts when a defined external interaction occurs.

About this task

To set up the environment to enable the second activation to take place, the first activation must:

1. Define an input event that depicts the external interaction. The activity cannot now complete until this input event has been dealt with.
2. Obtain an **activity identifier** that uniquely identifies this activity-instance. To do this, it issues an ASSIGN command.

   The transaction that starts the second activation must use this identifier to gain access to the activity.
3. Save details of the activity identifier and input event to a suitable medium. For example, a VSAM file or IBM MQ queue, to which the transaction that starts the second activation has access.
4. Return without completing. (That is, issue an EXEC CICS RETURN command on which the ENDACTIVITY option is omitted. Because of the user event in its event pool—the input event that it has defined—the activity does not complete but becomes dormant.)

When the external interaction occurs—for example, a clerk enters some data at a terminal—the transaction that starts the second activation of the activity is invoked. This transaction must:

1. Retrieve the activity identifier and input event
2. Gain access to the activity by issuing an ACQUIRE ACTIVITYID command that specifies the activity identifier.
3. Reactivate the activity, and specify why it is being activated by issuing a RUN ACQACTIVITY command that specifies the input event.

Figure 29 on page 46 shows an activity that interacts with the outside world. The first activation sets up the environment, saves details of the activity identifier and input event to a VSAM file, and returns without completing. Some time later, a user starts the SPAR transaction from a terminal. The SPAR transaction retrieves the activity identifier and input event, issues an ACQUIRE ACTIVITYID command to gain access to the activity, supplies the activity with some input data, and reactivates it.

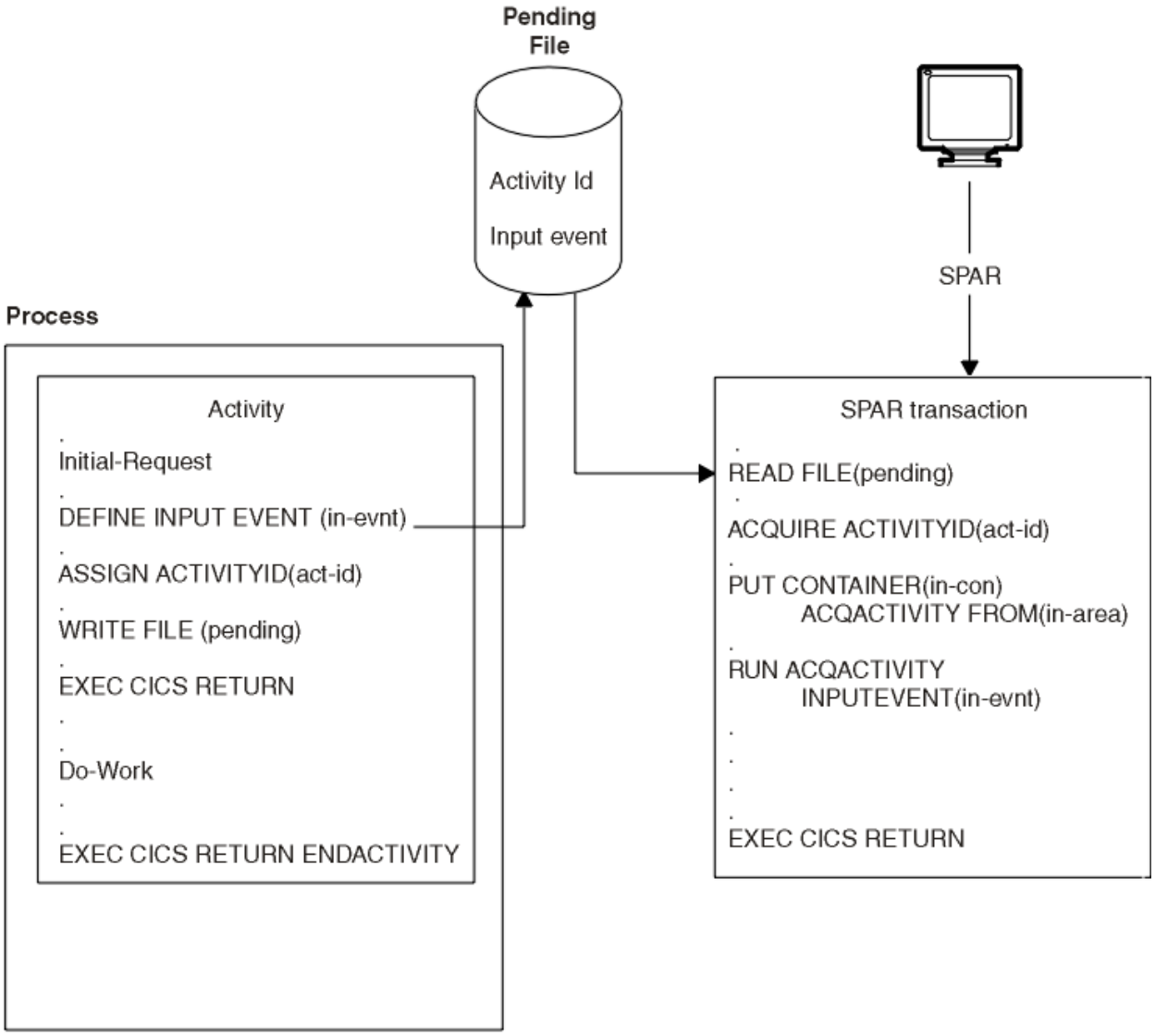


Figure 29. Acquiring an activity

A user-related example

You can use the user-related example to demonstrate user-related activities by changing the logic and process flow of the Sale business transaction.

The Sale example application assumes that none of its later activities require human involvement. The only child activity to require human involvement is the first (Order), and this is included as part of the initial terminal request to start the new business transaction.

To demonstrate user-related activities, this section changes the logic and process flow of the Sale business transaction. Now, instead of the Invoice activity being started automatically after the Delivery activity has completed, it is not started until a user notifies the Sale transaction that the delivery has taken place. In addition, the Payment activity requires the user to enter data.

Data flow

Follow the data flows in the Sale example application when user-related activities are included.

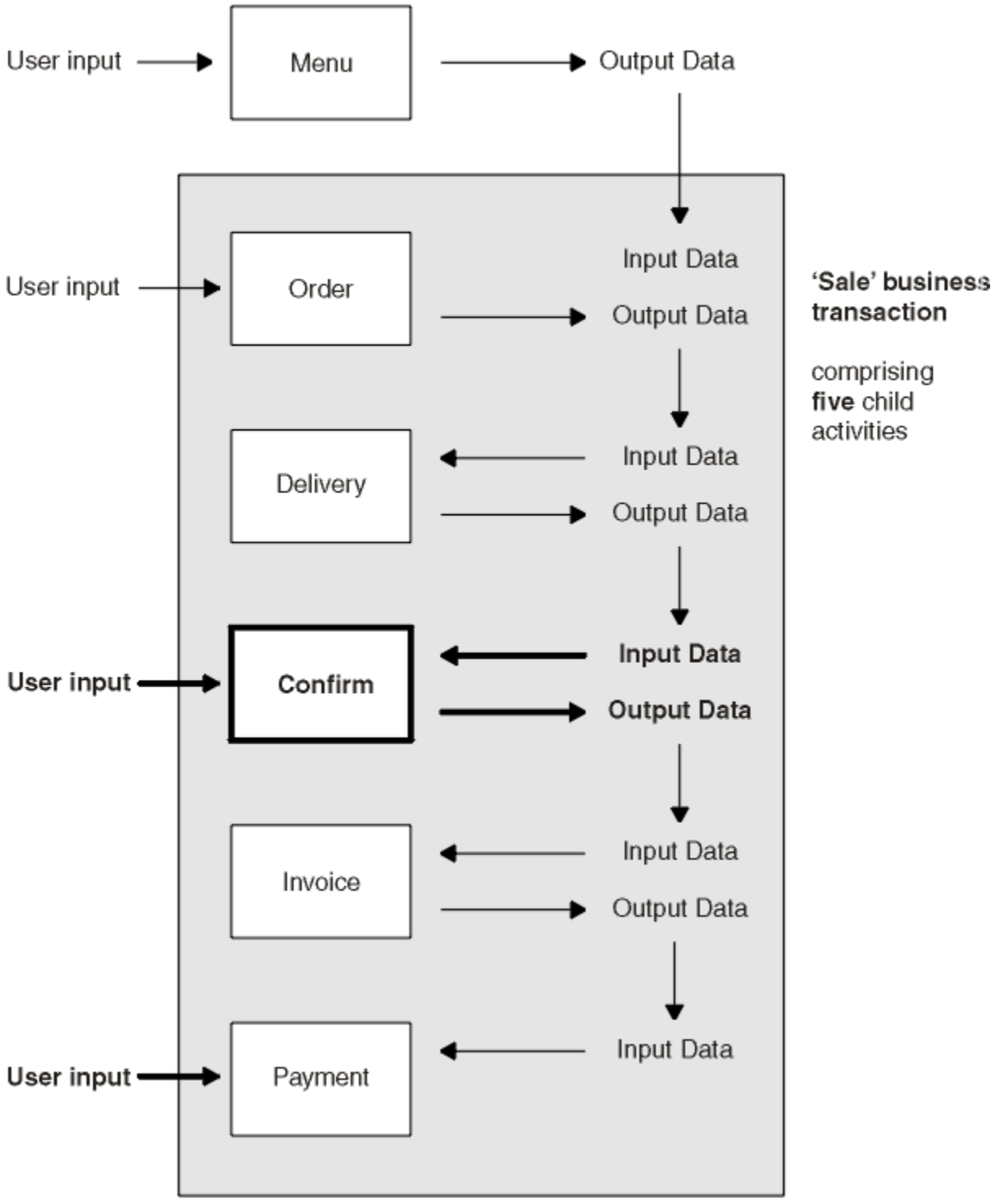


Figure 30. Data flow in the Sale example application, showing user-related activities

1. User data collected after the user selects the `Sale` menu option is used as input to the Order activity.
2. The user data collected by the Order activity is used as input to the Delivery activity.
3. The output data produced by the Delivery activity is used as input to the Confirm activity.
4. The output produced by the Confirm activity (which requires user input) is used as input to the Invoice activity.

5. The output produced by the Invoice activity is used as input to the Payment activity.

The root activity

Step through the Sale root activity, in COBOL pseudocode, with modifications for user-related activities. The main change is to introduce a new Confirm activity.

The modifications for user-related activities are highlighted in the examples here using bold text.

```
Identification Division.
Program-id. SAL002.
Environment Division.
Data Division.
Working-Storage Section.
01 RC pic s9(8) comp.
01 Process-Name pic x(36).
01 Event-Name pic x(16).
88 DFH-Initial value 'DFHINITIAL'
88 Delivery-Complete value 'Delivry-Complete'.
 88 Delivery-Confirmed value 'Delivry-Confirmd'.
88 Invoice-Complete value 'Invoice-Complete'.
88 Payment-Complete value 'Payment-Complete'.
01 Sale-Container pic x(16) value 'Sale'.
01 Order-Container pic x(16) value 'Order'.
01 Order-Buffer pic x(..).
01 Delivery-Container pic x(16) value 'Delivery'.
01 Delivery-Buffer pic x(..).
01 Confirm-Container pic x(16) value 'Confirm'.
01 Confirm-Buffer pic x(..).
01 Invoice-Container pic x(16) value 'Invoice'.
01 Invoice-Buffer pic x(..).
Linkage Section.
01 DFHEIBLK..
Procedure Division.
Begin-Process..
EXEC CICS RETRIEVE REATTACH EVENT(Event-Name)
RESP(RC) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
Evaluate True
When DFH-Initial
Perform Initial-Activity
Perform Order-Activity
Perform Order-Response
Perform Delivery-Activity
When Delivery-Complete
Perform Delivery-Response
 Perform Delivery-Confirmation
When Delivery-Confirmed
Perform Confirm-Response
Perform Invoice-Activity
When Invoice-Complete
Perform Invoice-Response
Perform Payment-Activity
```

Figure 31. The SAL002 root activity program, with user-related modifications highlighted (Part 1)

```
When Payment-Complete
Perform Payment-Response
Perform End-Process
When Other.
End Evaluate..
EXEC CICS RETURN END-EXEC.
Initial-Activity..
EXEC CICS ASSIGN PROCESS(Process-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
Order-Activity..
EXEC CICS DEFINE ACTIVITY('Order')
TRANSID('SORD')
PROGRAM('ORD001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Sale-Container)
ACTIVITY('Order') FROM(Process-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS LINK ACTIVITY('Order')
RESP(data-area) RESP2(data-area) END-EXEC.
Order-Response..
EXEC CICS CHECK ACTIVITY('Order') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If...
Delivery-Activity..
EXEC CICS DEFINE ACTIVITY('Delivery')
TRANSID('SDEL')
EVENT('Delivry-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GET CONTAINER(Order-Container)
ACTIVITY('Order') INTO(Order-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Order-Container)
ACTIVITY('Delivery') FROM(Order-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
```

Figure 32. The SAL002 root activity program, with user-related modifications highlighted (Part 2)

```
EXEC CICS RUN ACTIVITY('Delivery')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Delivery-Response..
EXEC CICS CHECK ACTIVITY('Delivery') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If...
Delivery-Confirmation..
EXEC CICS DEFINE ACTIVITY('Confirm')
TRANSID('SCON')
PROGRAM('CON001')
EVENT('Delivry-Confirmd')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GET CONTAINER(Delivery-Container)
ACTIVITY('Delivery') INTO(Delivery-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Delivery-Container)
ACTIVITY('Confirm') FROM(Delivery-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Confirm')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Confirm-Response..
EXEC CICS CHECK ACTIVITY('Confirm') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If..
.
Invoice-Activity..
EXEC CICS DEFINE ACTIVITY('Invoice')
TRANSID('SINV')
EVENT('Invoice-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
```

Figure 33. The SAL002 root activity program, with user-related modifications highlighted (Part 3)

```
 EXEC CICS GET CONTAINER(Confirm-Container)
ACTIVITY('Confirm') INTO(Confirm-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Confirm-Container)
ACTIVITY('Invoice') FROM(Confirm-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS RUN ACTIVITY('Invoice')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Invoice-Response..
EXEC CICS CHECK ACTIVITY('Invoice') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If...
Payment-Activity..
EXEC CICS DEFINE ACTIVITY('Payment')
TRANSID('SPAY')
EVENT('Payment-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GET CONTAINER(Invoice-Container)
ACTIVITY('Invoice') INTO(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Invoice-Container)
ACTIVITY('Payment') FROM(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Payment')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Payment-Response..
EXEC CICS CHECK ACTIVITY('Payment') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If...
```

Figure 34. The SAL002 root activity program, with user-related modifications highlighted (Part 4)

```
End-Process..
EXEC CICS RETURN ENDACTIVITY
RESP(data-area) RESP2(data-area) END-EXEC
End Program.
```

Figure 35. The SAL002 root activity program, with user-related modifications highlighted (Part 5)

The main change to SAL002 is to introduce a new Confirm activity. The purpose of the Confirm activity is to confirm that delivery has taken place, before the Invoice activity is started. Confirmation requires the user to enter some data. The following pseudocode creates the Confirm activity:

```
Delivery-Confirmation..
EXEC CICS DEFINE ACTIVITY('Confirm')
TRANSID('SCON')
EVENT('Delivry-Confirmd')
RESP(data-area) RESP2(data-area) END-EXEC
.
```

Because the Confirm activity is executed asynchronously with the root activity, the EVENT option of DEFINE ACTIVITY is used to name the completion event of the activity as *Delivry-Confirmd* . CICS reattaches SAL002 when the Confirm activity event fires - that is, when the Confirm activity completes.

SAL002 places the data entered by the user for the Confirm activity into a data-container named *Delivery* , and issues the RUN command:

```
 EXEC CICS GET CONTAINER(Delivery-Container)
ACTIVITY('Delivery') INTO(Delivery-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Delivery-Container)
ACTIVITY('Confirm') FROM(Delivery-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Confirm')
```

```
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC
```

Now SAL002 terminates, returning control to CICS. BTS reattaches the root activity only when the Confirm activity has completed.

Implementation of a user-related activity

You can use the Confirm activity to notify the Sale business transaction that actual delivery has taken place. In the COBOL pseudocode for program CON001, you can see how it implements the Confirm user-related activity.

```
Identification Division.
Program-id. CON001
Environment Division.
Data Division.
Working-Storage Section.
01 RC pic s9(8) comp.
01 Event-Name pic x(16).
88 DFH-Initial value 'DFHINITIAL'
88 User-Ready value 'User-Ready'.
01 Data-Record.
03 User-Reference pic x(60).
03 Act-Id pic x(52).
03 Usr-Event pic x(16).
01 Data-Record-Len pic s9(8) comp..
01 Delivery-Container pic x(16) value 'Delivery'.
01 User-Container pic x(16) value 'User'.
01 Confirm-Container pic x(16) value 'Confirm'.
01 Delivery-Details.
03 Deliv-Details ..
03 User-Details ...
Linkage Section.
01 DFHEIBLK..
Procedure Division.
In-The-Beginning..
EXEC CICS RETRIEVE REATTACH EVENT(Event-Name)
RESP(RC) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
Evaluate True
When DFH-Initial
Perform Initialization
When User-Ready
Perform Do-Work
When Other.
End Evaluate..
EXEC CICS RETURN END-EXEC.
Initialization..
EXEC CICS DEFINE INPUT EVENT(User-Ready)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS ASSIGN ACTIVITYID(Act-Id)
RESP(data-area) RESP2(data-area) END-EXEC.
MOVE User-Ready TO Usr-Event
MOVE LENGTH OF Data-Record TO Data-Record-Len
.
```

Figure 36. Pseudocode for the CON001 program, that implements the Confirm activity (Part 1)

```
 EXEC CICS WRITE FILE('PENDING')
FROM(Data-Record) LENGTH(Data-Record-Len)
RIDFLD(User-Reference)
RESP(data-area) RESP2(data-area) END-EXEC.
Do-Work..
Merge contents of two input data-containers into Delivery-Details.
EXEC CICS GET CONTAINER(Delivery-Container)
INTO(Deliv-Details)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GET CONTAINER(User-Container)
INTO(User-Details)
RESP(data-area) RESP2(data-area) END-EXEC.
Set up the output data-container.
EXEC CICS PUT CONTAINER(Confirm-Container)
FROM(Delivery-Details)
RESP(data-area) RESP2(data-area) END-EXEC.
Clean up.
EXEC CICS DELETE FILE('PENDING') RIDFLD(User-Reference)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DELETE EVENT(User-Ready)
RESP(data-area) RESP2(data-area) END-EXEC.
End the activity.
EXEC CICS RETURN ENDACTIVITY
RESP(data-area) END-EXEC.
End Program.
```

Figure 37. Pseudocode for the CON001 program, that implements the Confirm activity (Part 2)

The initial activation of the Confirm activity
The Confirm activity is activated for the first time after SAL002 issues the RUN ACTIVITY command.

On this initial activation, CON001:

1. Defines an input event for which the activity might then be activated.
2. Obtains the activity identifier which uniquely identifies this activity-instance.
3. Saves the name of the input event and the activity identifier in a pending file. The record in the pending file is given a key—which could, for instance, be the customer reference number which has been used throughout to identify this instance of the Sale business transaction.
4. Returns without completing.

```
Initialization..
EXEC CICS DEFINE INPUT EVENT(User-Ready)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS ASSIGN ACTIVITYID(Act-Id)
RESP(data-area) RESP2(data-area) END-EXEC.
MOVE User-Ready TO Usr-Event
MOVE LENGTH OF Data-Record TO Data-Record-Len.
EXEC CICS WRITE FILE('PENDING')
FROM(Data-Record) LENGTH(Data-Record-Len)
RIDFLD(User-Reference)
RESP(data-area) RESP2(data-area) END-EXEC
.
```

The USRX user transaction
When ready to confirm delivery, the user starts the USRX user-written transaction, which starts the USRCON program. You can step through the USRCON pseudocode to see how conformation takes place.

USRCON executes outside the BTS environment - it is not part of the SAL001 process that contains the Confirm activity. Figure 38 on page 54 shows, in COBOL pseudocode, the USRCON program.

```
Identification Division.
Program-id. USRCON.
Environment Division.
Data Division.
Working-Storage Section.
01 Pending-Record.
03 User-Reference pic x(60).
03 Act-Id pic x(52).
03 Usr-Event pic x(16)..
01 User-Container pic x(16) value 'User'.
01 Confirmation-Details.
03 ...
Linkage Section.
01 DFHEIBLK..
01 DFHCOMMAREA.
.
```

```
Procedure Division using DFHEIBLK DFHCOMMAREA.
In-The-Beginning..
EXEC CICS SEND MAP('......') MAPSET('......') ....
EXEC CICS RECEIVE MAP('......') MAPSET('......) ...
Move ..unique.. to User-Reference..
EXEC CICS READ FILE('PENDING')
INTO(Pending-Record) RIDFLD(User-Reference)
RESP(data-area) RESP2(data-area) END-EXEC.
. Acquire access to the Confirm activity of the SAL001 process.
EXEC CICS ACQUIRE ACTIVITYID(Act-Id)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(User-Container)
ACQACTIVITY
FROM(Confirmation-Details)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACQACTIVITY
INPUTEVENT(Usr-Event)
SYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS CHECK ACQACTIVITY COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If..
EXEC CICS RETURN
RESP(data-area) END-EXEC.
End Program.
```

Figure 38. Pseudocode for the USRCON program, that implements the USRX transaction

First, USRCON sends a map to the screen of the user and requests a unique reference. This reference must be the same as the key used by the CON001 program. This reference might be the customer reference or account number that has been used throughout to identify this instance of the Sale business transaction. However, it might need to be more specific than this. This would be the case if, for example:

- The Sale business transaction has more than one user-related activity.
- The user-related activity has defined more than one input event.

Using the unique reference, USRCON selects the appropriate record from the pending file. It then uses the value of *Act-Id* to acquire access to the Confirm activity for the SAL001 instance of the Sale business transaction:

```
 EXEC CICS ACQUIRE ACTIVITYID(Act-Id)
RESP(data-area) RESP2(data-area) END-EXEC
```

If the ACQUIRE command is successful, USRCON has access to the containers of the Confirm activity. USRCON creates a data-container for the Confirm activity (*User*), and puts some confirmation details into it:

```
 EXEC CICS PUT CONTAINER(User-Container)
ACQACTIVITY
FROM(Confirmation-Details)
RESP(data-area) RESP2(data-area) END-EXEC
```

The ACQACTIVITY option associates the new *User* container with the activity that USERCON has acquired.

Finally, USERCON activates the Confirm activity again, checks whether it completes successfully, and ends:

```
 .
EXEC CICS RUN ACQACTIVITY
INPUTEVENT(Usr-Event)
SYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS CHECK ACQACTIVITY COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
EXEC CICS RETURN
RESP(data-area) END-EXEC
.
```

The value of the INPUTEVENT option of the RUN command is the name of the input event previously defined by the Confirm activity. Although USRCON can check whether the activity it has acquired completes successfully, the execution of the CHECK ACQACTIVITY command does *not* cause CICS to delete the completion event of the Confirm activity. CICS deletes a completion event of the completed activity only after the execution of a CHECK ACTIVITY command issued by the parent of the activity.

The second activation of the Confirm activity
The Confirm activity is activated for a second, and final, time by the RUN ACQACTIVITY command issued by USRCON.

On its second activation, CON001:

1. Establishes why it has been invoked.
2. Merges the contents of the two input data-containers, *Delivery* and *User* , supplied by SAL002 and USRCON.
3. Stores the updated delivery details into the output data-container of the Confirm activity (*Confirm*).

See Figure 36 on page 52. Finally, CON001 does some clean-up work. It:

1. Deletes the entry from the pending file.
2. Deletes the input event defined on its previous invocation. This is not strictly necessary, because the event would be deleted automatically by CICS on the execution of the RETURN ENDACTIVITY command that follows.
3. Issues an EXEC CICS RETURN ENDACTIVITY command to indicate that its processing is complete; the completion event of the Confirm activity (*Delivry-Confirmed*) is fired.

CICS notes completion of the Confirm activity and reattaches the root activity, because of the firing of the *Delivry-Confirmd* completion event defined by SAL002. After the execution of the CHECK ACTIVITY command issued by SAL002, CICS deletes the completion event of the Confirm activity.

Transferring data to asynchronous activations

You can transfer data to asynchronous activations by using a data container. When a server process or an acquired activity runs asynchronously, note these design considerations.

About this task

There are a number of ways in which your applications can handle the transfer of data to and from activities that are run asynchronously with the requestor. In the simplest case, a single data-container can be used for both input and output data. If the activity is activated only once, this activation presents no problems. Separate containers are used, one for input and one for output data. Again, if the activity is activated only once, this presents no problems. However, if the activity might be activated, asynchronously, multiple times, you must take care that the contents of containers are not overwritten inadvertently. You must take particular care when designing client/server applications, and applications which involve activities being acquired and run multiple times by transactions external to their parent process.

If an application chooses to run a server process or an acquired activity asynchronously, it needs to be aware of the state of the activity being activated. In most cases, the activity is dormant - awaiting the activation and ready to perform its function. The activation occurs almost immediately, the activity program runs, and places any results in a container. In a client/server application, the activity might then be left dormant, ready for the next request. If the activation is triggered by an external interaction, it is likely that the activity will complete; the firing of its completion event causes its parent to be activated again.

However, you must consider that, when the RUN ASYNCHRONOUS command is executed, the target activity might not be dormant, waiting for work - it might be in any of the other possible processing modes, or it could be suspended. If, for example, the target activity has been suspended, the asynchronous activation does not happen immediately. Thus, in a client/server application, it is possible for the client program to issue a request to the server before a previous request has been serviced. You should be aware of these possibilities when designing your applications. If, for example, the protocol between a client program and its server activity relies on a single container for passing data, the client needs to check that the container is not occupied by a previous request before issuing subsequent requests. Another solution is for the client to use multiple containers to form a queue of requests for the server activity; the containers could be named sequentially.

Compensation in BTS

You can use compensation in business transaction services (BTS) to reverse or modify actions taken by preceding activities or to stop business transactions.

If a single CICS transaction fails, any uncommitted changes that it has made to recoverable resources are automatically backed out by the CICS recovery manager. However, it is typically not practicable for a business transaction to be implemented as a single CICS transaction, due to the high rate of transaction abends and performance degradation that would result from holding locks for long periods.

Instead, using CICS business transaction services, each part of a business transaction is implemented as a separate BTS activity, consisting of one, or more CICS transactions. If an activity fails, the actions taken by preceding activities might need to be reversed, or possibly modified. Similarly, if application logic determines that the business transaction must be terminated, changes made by activities that have already completed might need to be reversed.

Modifying the actions of completed activities is called *compensation*.

Implementing compensation

Compensation is the act of modifying, or compensating for, the effects of a completed activity. The designer of the business transaction decides how compensation is implemented.

About this task

Often, compensating for an activity means undoing the actions that it took - for example, compensation for accepting an order might be to cancel the order.

Compensation of an activity is always controlled and instigated by the parent of the activity. It is therefore convenient to talk of compensation as an act that a parent performs on a child - as in “compensating an activity”. Strictly speaking, however, it is the parent that is compensated (it “receives compensation” for some previous action taken by the child. The previous action of the child is compensated for—it is reversed or modified.

Here are two ways in which you could implement compensation of a completed child activity.

1. **Run the activity again**.

   To run the activity again, you must first issue a RESET ACTIVITY command, to reset the activity to its initial state. You must then tell the activity that it is being invoked to perform compensation; you could do this by placing a flag in an input data-container. You cannot use the INPUTEVENT option of the RUN command to tell the activity why it is being invoked; specifying INPUTEVENT is invalid when an activity is in its initial state.

In this method, the program used for compensation, the **compensation program** , is the same program used for normal (forward) execution of the activity.

2. **Define and run a new, compensation, activity**.

   This method is more straightforward. You could use a PUT CONTAINER command to provide the compensation activity with the same input data that was passed to the activity for which it compensates.

   In this method, the program used for compensation is likely to be different from the program used for the execution of the activity that is compensated.

   The compensation example in this section uses this method.

A compensation example

You can use this example to change the logic of your Sale business transaction to issue reminders and to initiate the compensation process when certain criteria are met.

In this chapter , the logic of the Sale business transaction is changed so that :

- When payment has not been received within one week of the invoice being dispatched, a reminder is sent.
- If payment has still not been received two weeks after the reminder was sent, compensation is instigated.

  Compensation means that:

  1. The outstanding payment request is canceled.
  2. A request is sent for the goods to be returned.
  3. Confirmation of the goods being returned is required.
  4. The original order is canceled.

Process flow

Follow the Sale example application process flow with compensation actions included.

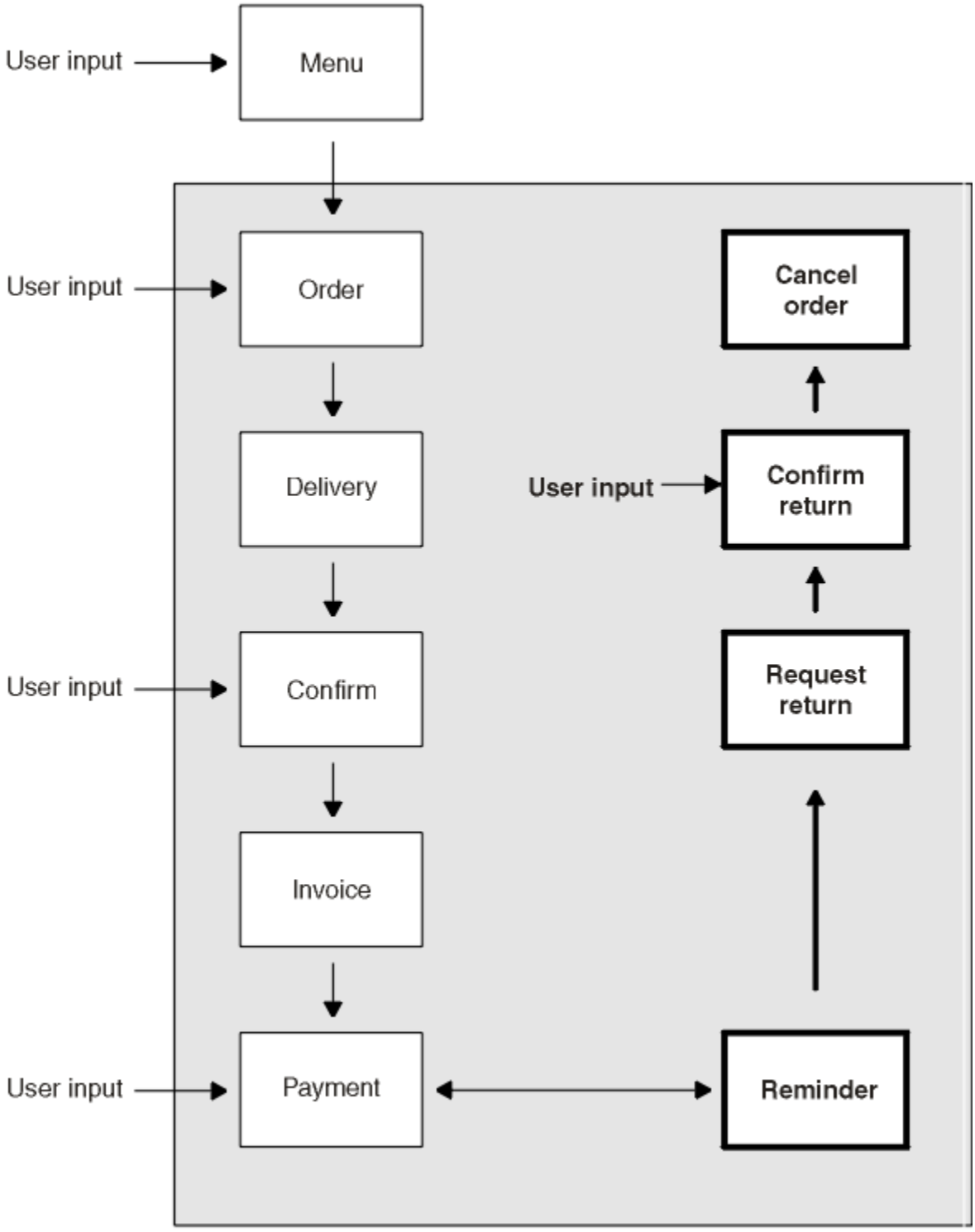


Figure 39. Process flow with compensation actions included

- The terminal user enters the customer's order, which is accepted.
- The goods are delivered to the customer.
- The terminal user confirms that the goods have been delivered.
- An invoice is sent to the customer.

- A reminder is sent if payment has not been received within one week of the invoice being sent.
- If payment has still not been received two weeks after the reminder was sent, compensation is triggered. Compensation causes the following:
  - The outstanding payment request is canceled.
  - A letter is sent, requesting the goods to be returned.
  - Confirmation that the goods have been returned is requested.
  - The order is canceled.

The root activity

Step through the Sale root activity, in COBOL pseudocode, to see the modifications to include compensation actions.

The changes are shown here using bold text.

```
Identification Division.
Program-id. SAL002.
Environment Division.
Data Division.
Working-Storage Section.
01 RC pic s9(8) comp.
01 Process-Name pic x(36).
01 Event-Name pic x(16).
88 DFH-Initial value 'DFHINITIAL'
88 Delivery-Complete value 'Delivry-Complete'.
88 Delivery-Confirmed value 'Delivry-Confirmd'.
88 Invoice-Complete value 'Invoice-Complete'.
 88 Payment-Due value 'Payment-Due'.
88 Payment-Complete value 'Payment-Complete'.
 88 Reminder-Expired value 'Remindr-Expired'.
88 Reminder-Complete value 'Remindr-Complete'.
01 Sale-Container pic x(16) value 'Sale'.
01 Order-Container pic x(16) value 'Order'.
01 Order-Buffer pic x(..).
01 Delivery-Container pic x(16) value 'Delivery'.
01 Delivery-Buffer pic x(..).
01 Confirm-Container pic x(16) value 'Confirm'.
01 Confirm-Buffer pic x(..).
01 Invoice-Container pic x(16) value 'Invoice'.
01 Invoice-Buffer pic x(..).
01 Reminder-Container pic x(16) value 'Reminder'.
01 Status pic x(16).
Linkage Section.
01 DFHEIBLK..
```

Figure 40. The SAL002 root activity program, including compensation actions (Part 1)

```
Procedure Division.
Begin-Process..
EXEC CICS RETRIEVE REATTACH EVENT(Event-Name)
RESP(RC) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
Evaluate True
When DFH-Initial
Perform Initial-Activity
Perform Order-Activity
Perform Order-Response
Perform Delivery-Activity
When Delivery-Complete
Perform Delivery-Response
Perform Delivery-Confirmation
When Delivery-Confirmed
Perform Confirm-Response
Perform Invoice-Activity
When Invoice-Complete
Perform Invoice-Response
Perform Payment-Activity
 When Payment-Due
Perform Payment-Due-Response
When Payment-Complete
Perform Payment-Response
 When Reminder-Expired
Perform Reminder-Expired-Response
When Reminder-Complete
Perform Reminder-Response
When Other.
End Evaluate..
EXEC CICS RETURN END-EXEC.
Initial-Activity..
EXEC CICS ASSIGN PROCESS(Process-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
Order-Activity..
EXEC CICS DEFINE ACTIVITY('Order')
TRANSID('SORD')
PROGRAM('ORD001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Sale-Container)
ACTIVITY('Order') FROM(Process-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS LINK ACTIVITY('Order')
RESP(data-area) RESP2(data-area) END-EXEC.
```

Figure 41. The SAL002 root activity program, including compensation actions (Part 2)

```
Order-Response..
EXEC CICS CHECK ACTIVITY('Order') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If...
Delivery-Activity..
EXEC CICS DEFINE ACTIVITY('Delivery')
TRANSID('SDEL')
PROGRAM('DEL001')
EVENT('Delivry-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GET CONTAINER(Order-Container)
ACTIVITY(Order-Container) INTO(Order-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Order-Container)
ACTIVITY('Delivery') FROM(Order-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Delivery')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Delivery-Response..
EXEC CICS CHECK ACTIVITY('Delivery') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If..
Delivery-Confirmation..
EXEC CICS DEFINE ACTIVITY('Confirm')
TRANSID('FCON')
PROGRAM('CON001')
EVENT('Delivry-Confirmd')
RESP(data-area) RESP2(data-area) END-EXEC
.
```

Figure 42. The SAL002 root activity program, including compensation actions (Part 3)

```
 EXEC CICS GET CONTAINER(Deliver-Container)
ACTIVITY('Delivery') INTO(Delivery-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Deliver-Container)
ACTIVITY('Confirm') FROM(Delivery-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Confirm')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
Confirm-Response..
EXEC CICS CHECK ACTIVITY('Confirm') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If..
Invoice-Activity..
EXEC CICS DEFINE ACTIVITY('Invoice')
TRANSID('SINV')
EVENT('Invoice-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GET CONTAINER(Confirm-Container)
ACTIVITY('Confirm') INTO(Confirm-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Confirm-Container)
ACTIVITY('Invoice') FROM(Confirm-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Invoice')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC
.
```

Figure 43. The SAL002 root activity program, including compensation actions (Part 4)

```
Invoice-Response..
EXEC CICS CHECK ACTIVITY('Invoice') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC NOT = DFHRESP(NORMAL).
End-If..
If status NOT = DFHVALUE(NORMAL).
End-If..
Payment-Activity..
EXEC CICS DEFINE ACTIVITY('Payment')
TRANSID('SPAY')
EVENT('Payment-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
 EXEC CICS DEFINE TIMER('Payment-Due')
AFTER DAYS(7)
RESP(data-area) RESP2(data-area) END-EXEC.

EXEC CICS GET CONTAINER(Invoice-Container)
ACTIVITY('Invoice') INTO(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Invoice-Container)
ACTIVITY('Payment') FROM(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Payment')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
```

Figure 44. The SAL002 root activity program, including compensation actions (Part 5)

```
Payment-Due-Response..
EXEC CICS DELETE TIMER('Payment-Due')
RESP(RC) RESP2(data-area) END-EXEC.
Perform Reminder-Activity
Payment-Response..
EXEC CICS CHECK ACTIVITY('Payment') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC = DFHRESP(NORMAL)
If status = DFHVALUE(NORMAL)
EXEC CICS DELETE TIMER('Payment-Due')
RESP(RC) RESP2(data-area) END-EXEC.
Perform End-process
Else.
End-If
Else.
End-If.
Reminder-Activity..
EXEC CICS DEFINE ACTIVITY('Reminder')
TRANSID('PAYR')
EVENT('Remindr-Complete')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DEFINE TIMER('Remindr-Expired')
AFTER DAYS(14)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS GET CONTAINER(Invoice-Container)
ACTIVITY('Invoice') INTO(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Invoice-Container)
ACTIVITY('Reminder') FROM(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Reminder')
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC
.
```

Figure 45. The SAL002 root activity program, including compensation actions (Part 6)

```
Reminder-Expired-Response..
EXEC CICS DELETE TIMER('Remindr-Expired')
RESP(RC) RESP2(data-area) END-EXEC.
Perform Compensation
Reminder-Response..
EXEC CICS CHECK ACTIVITY('Reminder') COMPSTATUS(status)
RESP(RC) RESP2(data-area) END-EXEC.
If RC = DFHRESP(NORMAL)
If status = DFHVALUE(NORMAL)
EXEC CICS DELETE TIMER('Remindr-Expired')
RESP(RC) RESP2(data-area) END-EXEC.
Perform End-process
Else.
End-If
Else.
End-If.
Compensation..
EXEC CICS DEFINE ACTIVITY('Payment-Compen')
TRANSID('PAYC')
PROGRAM('PEX001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Invoice-Container)
ACTIVITY('Payment-Compen') FROM(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Payment-Compen')
SYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DEFINE ACTIVITY('Confirm-Compen')
TRANSID('CONC')
PROGRAM('REQ001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Deliver-Container)
ACTIVITY('Confirm-Compen') FROM(Delivery-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Confirm-Compen')
SYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC
.
```

Figure 46. The SAL002 root activity program, including compensation actions (Part 7)

```
 EXEC CICS DEFINE ACTIVITY('Delivery-Compen')
TRANSID('DELC')
PROGRAM('RTN001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Order-Container)
ACTIVITY('Delivery-Compen') FROM(Order-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Delivery-Compen')
SYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS DEFINE ACTIVITY('Order-Compen')
TRANSID('ORDC')
PROGRAM('CAN001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Sale-Container)
ACTIVITY('Order-Compen') FROM(Process-Name)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Order-Compen')
SYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC
.
End-Process..
EXEC CICS RETURN ENDACTIVITY
RESP(data-area) RESP2(data-area) END-EXEC
End Program.
```

Figure 47. The SAL002 root activity program, including compensation actions (Part 8)

Note the following:

- A reminder is set for the Payment activity:

```
.
EXEC CICS DEFINE TIMER('Payment-Due')
```

```
 AFTER DAYS(7)
RESP(data-area) RESP2(data-area) END-EXEC.
```

The DEFINE TIMER command defines a timer which will expire in one week. Because the EVENT option is not specified, the event associated with the timer—the timer event—is given the same name as the timer itself (*Payment-Due*). Now, SAL002 will be reattached when either of the following happens:

1. The Payment activity completes. Because Payment is a user-related activity, it will complete only if a terminal user confirms that payment has been received.
2. The timer expires.

- If SAL002 is invoked because the timer expires, it requests the Reminder activity to run. The Reminder activity too is user-related—the request to run it drives the first part of the activity, which sends a reminder letter to the customer, records the activity's details on a pending file, and waits to be reactivated by user input.

  As for the Payment activity, a timer is set for the Reminder activity. Now, SAL002 will be reattached when either of the following happens:

  1. The Reminder activity completes. Because Reminder is a user-related activity, it will complete only if a terminal user confirms that payment has been received.
  2. The timer expires.
- If SAL002 is next invoked because the timer expires, it compensates its completed child activities. For each child activity to be compensated, SAL002 defines a new (compensation) activity, provides the compensation activity with some input data, and runs it:

```
 EXEC CICS DEFINE ACTIVITY('Payment-Compen')
TRANSID('PAYC')
PROGRAM('PEX001')
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER(Invoice-Container)
ACTIVITY('Payment-Compen') FROM(Invoice-Buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY('Payment-Compen')
SYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC
```

Notice that the program used to execute the Payment-Compen compensation activity is different from that used for the Payment activity that is compensated. The PUT CONTAINER command provides the Payment-Compen activity with the same input data that was passed to the Payment activity.

Table 3 on page 64 shows which activities are compensated, and the actions taken by the compensation activity in each case.

Table 3. Compensation activities

Completed child activity	**Compensation activity**	**Actions taken by compensation activity**
Payment	Payment-Compen	Cancels the outstanding payment request
Confirm	Confirm-Compen	Sends a letter requesting return of goods
Delivery	Delivery-Compen	Requests confirmation that the goods have been returned
Order	Order-Compen	Cancels the original order request

- The user-defined timers (*Payment-Due* and *Remindr-Expired*) are deleted as soon as they are no longer required. This has the side effect of automatically deleting the timer events associated with them.
- The CHECK ACTIVITY command is used to check the response from each child activity. This has the side effect of automatically deleting the activity completion event, if the child has completed. (An activity must delete the completion events for all its child activities before it completes.)

  Note: In a real application, it would be necessary to issue CHECK ACTIVITY commands for the compensation activities. For the sake of brevity, these have been omitted from the example.

Dealing with application locking

When an activity completes, any updates that it has made to data are committed and the database manager releases its locks on the data. The updated data is then available to other activities, including activities that are part of other business applications.

About this task

These other activities may make decisions based on the state of the data. If you later compensate the completed activity and return the data to its previous state, some activities might have executed based on data which is no longer valid. If these activities are part of the same process as the compensated activity, you can code your application to compensate them too. However, to cope with the possibility that activities in other applications might take decisions based on data that is later changed by compensation, your application must be coded differently.

If your applications include compensation activities which reverse previously committed data updates, they might need to include logic to provide logical record locking. The “application lock” does not need to be a hard lock preventing access to the data, but a flag which indicates that the data is part of an incomplete business process which might be reversed. All activities working with the “in-process” data could be coded to check this flag and then follow appropriate logic. To support this logic, when you design your database you need to include a “locked” field in your data records.

For example, you might have a “Welcome letter” application which scans the customer database for new customers who have placed their first order, and sends each a welcoming letter thanking them for their order and asking them to complete a customer satisfaction questionnaire. Perhaps your company considers it inappropriate to send such a letter if the order is not yet complete and payment received, because the welcome letter might be received along with less friendly letters demanding payment! Therefore, the Order activity of the Sale business application could set an *order-in-progress* flag on the order record, which would exclude the order from consideration by a “Welcome letter” process. Later, the Payment activity of the Sale application could unset the *order-in-progress* flag.

Reusing existing 3270 applications in BTS

You can use BTS support for the 3270 bridge to integrate existing transactions, including more complex ones, into BTS applications. Sample programs show you how the integration works.

The 3270 bridge is described in Introduction to the 3270 bridge.

Running a 3270 transaction from BTS

BTS supports the 3270 bridge function. Therefore, BTS applications can be integrated with, and make use of, existing 3270-based applications. Follow this basic mechanism for running a 3270 transaction from a BTS application.

About this task

Even though BTS activities are not terminal-related (they are never started directly from a terminal), a BTS activity can be implemented by a 3270-based transaction. The bridge exit program is used to put a “BTS wrapper” around the original 3270 transaction.

Figure 48 on page 66 shows the basic mechanism for running a 3270 transaction from a BTS application.

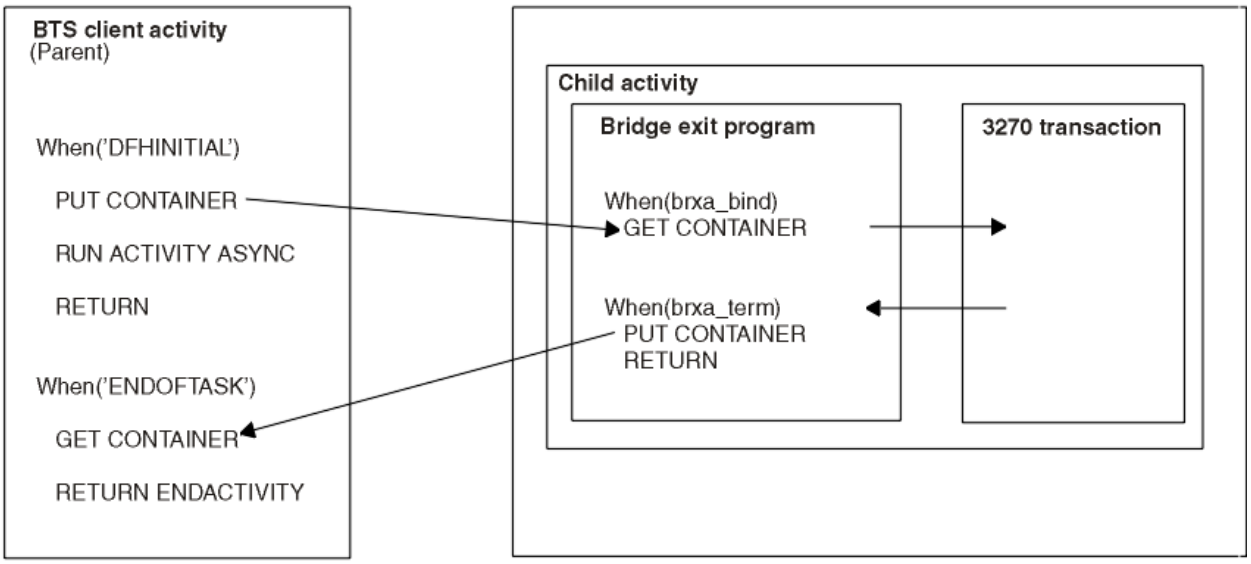


Figure 48. Running a 3270-based transaction as a BTS activity

1. A BTS activity, known in bridge terminology as the **client** , creates data to run a 3270 transaction. It puts the data in a container associated with a child activity.
2. The client runs the child activity—which is implemented by the 3270 transaction—asynchronously.
3. The BTS XM client identifies that the transaction should use the 3270 bridge and calls the bridge XM client.
4. On the 'bind' call to the bridge exit, the bridge exit program issues a GET CONTAINER command to retrieve the data to run the 3270 transaction.

   Note: In a bridge environment, the bridge exit program becomes part of the 3270 transaction. Thus, the exit program does not need to acquire the child activity before issuing the GET CONTAINER command—it is itself *part* of the child activity.
5. The 3270 transaction is run using the retrieved data. Any output data it produces is saved in an output message.
6. When the bridge exit program is invoked for termination of the 3270 transaction, it issues:
   a. A PUT CONTAINER command, to put the output message into a named data-container
   b. A RETURN command, which causes the child activity to complete.
7. The firing of the completion event of the child activity causes the parent (client) activity to be reactivated.
8. The client issues a GET CONTAINER command to retrieve the output from the 3270 transaction.

The following table contains example pseudocode for running a 3270-based transaction as a BTS activity.

Table 4. Pseudocode for running a 3270-based transaction as a BTS activity

Client activity	Bridge exit program
When DFH-Initial encode msg-in-buffer EXEC CICS DEFINE ACTIVITY ('3270-act') TRANSID('T327') EVENT('3270-Complete') RESP(data-area) RESP2(data-area) END-EXEC. EXEC CICS PUT CONTAINER('Message') ACTIVITY('3270-act') FROM(msg-in-buffer) RESP(data-area) RESP2(data-area) END-EXEC. EXEC CICS RUN ACTIVITY('3270-act') ASYNCHRONOUS RESP(data-area) RESP2(data-area) END-EXEC. EXEC CICS RETURN END-EXEC.. When 3270-Complete EXEC CICS GET CONTAINER('Message') ACTIVITY('3270-act') INTO(msg-out-buffer) RESP(data-area) RESP2(data-area) END-EXEC. decode msg-out-buffer EXEC CICS RETURN ENDACTIVITY	Init. pass userdata from the brdata to BRXA.. Bind. EXEC CICS GET CONTAINER('Message') INTO(3270-msg-in-buffer) RESP(data-area) RESP2(data-area) END-EXEC.. Term. EXEC CICS PUT CONTAINER('Message') FROM(3270-msg-out-buffer) RESP(data-area) RESP2(data-area) END-EXEC. EXEC CICS RETURN END-EXEC

The child activity is implemented by the 3270 transaction and the bridge exit program. All the required BTS commands are issued by the exit program.

Resource definition

To enable BTS 3270 bridge support, you must specify the name of a bridge exit program on the BREXIT option of the TRANSACTION definition for the 3270 transaction that you want to run.

If two or more bridge transport mechanisms require the BREXIT parameter to be specified on the transaction definition, you can use an alias transaction definition. For information about how other bridge transport mechanisms support specification of the BREXIT parameter, see Introduction to the 3270 bridge.

Running more complex transactions

You can run more complex 3270 transactions that produce intermediate messages, are conversational in design, or are pseudoconversational.

The basic mechanism described in “Running a 3270 transaction from BTS” on page 65 assumes a straightforward, “one shot” transaction, where the 3270 transaction does an **EXEC CICS RECEIVE MAP** , followed by one or more **EXEC CICS SEND MAP** requests, and ends with an **EXEC CICS RETURN** . In practice, things are not always so simple.

The topics in this section describe how to run various complex transactions:

Intermediate output messages

For a non-conversational 3270 transaction, the bridge exit program might be called to write an intermediate message when either the 3270 transaction has specified WAIT on the EXEC CICS SEND command or the output message buffer is full.

Under some bridge transport mechanisms, it makes sense for the bridge exit program to write an intermediate message containing the data so far. However, under BTS there is no point in trying to send an intermediate message back to the user.

If the exit program is called because of the WAIT option, it can do nothing and return.

If the exit program is called because the message buffer is full, it should :

1. Obtain a new, larger output buffer (by issuing a GETMAIN command).
2. Copy the contents of the original buffer into the new buffer.
3. Release the original buffer (by issuing a FREEMAIN command).

Using this approach, all output from the 3270 transaction is sent to the client at transaction end.

The sample bridge exit program, DFH0CBAE (see “Sample programs” on page 72) obtains all the storage it requires - including the storage for its output buffer - at the same time. It saves the address of the output buffer in field BRXA-OUTPUT-MESSAGE-PTR of the bridge exit area (BRXA) user area. We recommend that your exit programs do the same.

Note: When the exit program is called because the output buffer is full, field BRXA-FMT-RESPONSE of the BRXA is set to BRXA-FMT-OUTPUT-BUFFER-FULL . The current size of the storage is in field BRXA-OUTPUT-MESSAGE-LEN.

Conversational transactions

Follow this pseudocode to see how to run a conversational 3270 transaction. Potential problems with solutions are illustrated.

A potential problem is that, at one or more stages, the 3270 transaction requires further data to continue. The bridge exit program cannot obtain this data from the client. That is, it cannot end its current activation, to be reactivated with the required data—because the 3270 transaction has not completed, issuing an EXEC CICS RETURN command would merely return control to the latter. Nor can the exit program get information back to the client by issuing an EXEC CICS SYNCPOINT command, because this would modify the 3270 transaction.

One solution is for the bridge exit program itself to obtain, or compute, the required data. Perhaps a better solution is for the exit program to create a subtask to obtain the data. It could, for example, create a separate child activity (a grandchild of the client) to deal with each request for data - each intermediate map - sent by the 3270 transaction. For convenience, we shall refer to such child activities as “conversational activities”. Figure 49 on page 68 illustrates this approach.

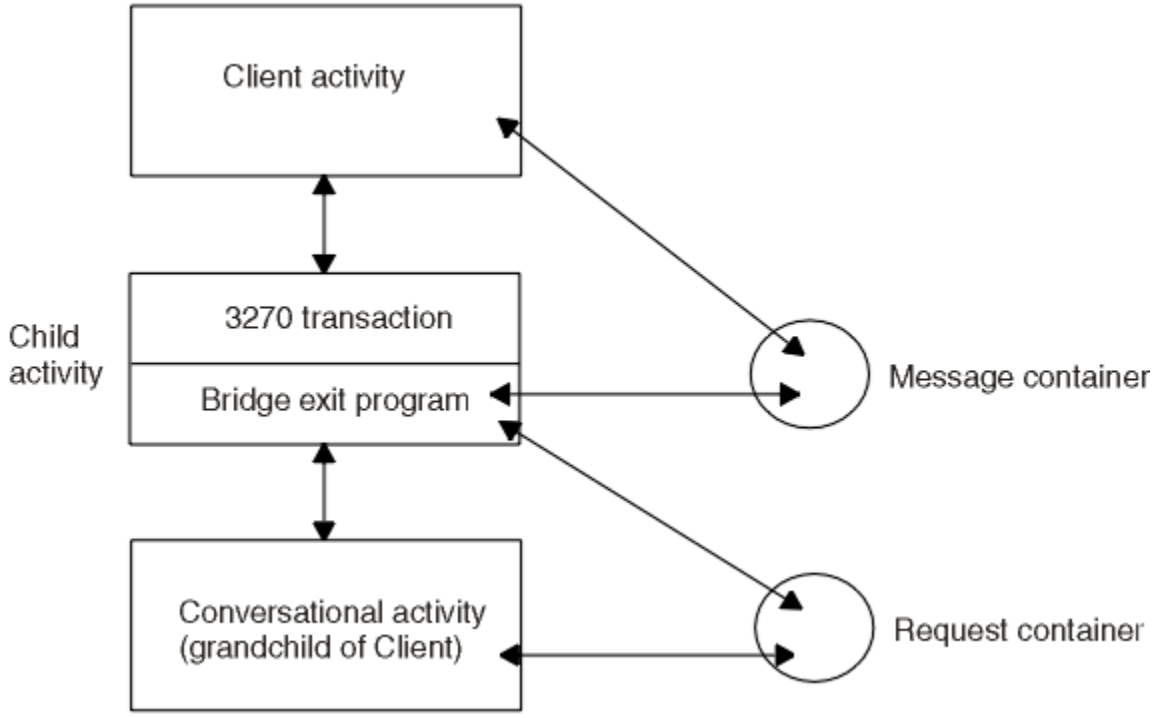


Figure 49. Running a 3270 conversational transaction as a BTS activity

One possible problem of creating a separate activity to deal with each intermediate map is that the output message sent to the client by the exit program at transaction end contains only the final 3270 map. If it is important that intermediate messages are preserved, the conversational activities could put them in other containers associated with the client.

The following table contains example pseudocode for running a 3270 conversational transaction.

Table 5. Pseudocode for running a 3270 conversational transaction as a BTS activity. The bridge exit program creates a child activity to deal with each map sent by the 3270 transaction.

Bridge exit program

```
Read_Message.
encode conv-in-buffer from 3270-msg-out-buffer
EXEC CICS DEFINE ACTIVITY (next-conv-act-name)
TRANSID(conv-transaction-id)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER('Request')
ACTIVITY(next-conv-act-name)
FROM(conv-in-buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS LINK ACTIVITY(next-conv-act-name)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS CHECK ACTIVITY(next-conv-act-name)
COMPSTATUS(status) ABCODE(a)
RESP(data-area) RESP2(data-area) END-EXEC
If status NOT = DFHVALUE(NORMAL)
EXEC CICS ABEND ABCODE(a)
NODUMP
RESP(data-area) RESP2(data-area) END-EXEC
End-If..
EXEC CICS GET CONTAINER('Request')
ACTIVITY(next-conv-act-name)
INTO(3270-msg-in-buffer)
RESP(data-area) RESP2(data-area) END-EXEC..
Write_Message.
Intermediate writes cannot be sent
to the client.
EXEC CICS NOOP
RESP(data-area) RESP2(data-area) END-EXEC.
```

“Conversational” activity

```
WHEN DFH-Initial
EXEC CICS GET CONTAINER('Request')
INTO(msg-in-buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
decode msg-in-buffer
encode msg-out-buffer.
EXEC CICS PUT CONTAINER('Request')
FROM(msg-out-buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RETURN END-EXEC
```

The exit program issues a LINK ACTIVITY, rather than a RUN ACTIVITY SYNCHRONOUS, command to activate the “conversational” child activity. This is necessary to ensure that the child executes in the same unit of work as the exit program.

Pseudoconversational transactions

Follow this pseudocode to see how to run a pseudoconversational 3270 transaction.

A pseudoconversation is indicated by the fact that the output data returned to the client by the exit program contains a bridge facility token (and possibly a next-transaction ID). It is the responsibility of the client to check the appropriate field in the output message and to start the next transaction.

The example shown here contains pseudocode for running a 3270 pseudoconversational transaction.

Table 6. Pseudocode for running a 3270 pseudoconversational transaction as a BTS activity

Client activity

```
When DFH-Initial
encode msg-in-buffer
EXEC CICS DEFINE ACTIVITY (3270-act-name)
TRANSID(transaction-id) EVENT(3270-Complete)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER('Message')
ACTIVITY(3270-act-name) FROM(msg-in-buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY(3270-act-name)
ASYNCHRONOUS
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RETURN END-EXEC..
When 3270-Complete
EXEC CICS CHECK ACTIVITY(3270-act-name)
COMPSTATUS(status) ABCODE(a)
RESP(data-area) RESP2(data-area) END-EXEC
If status NOT = DFHVALUE(NORMAL)
EXEC CICS ABEND ABCODE(a)
NODUMP
RESP(data-area) RESP2(data-area) END-EXEC
End-If..
EXEC CICS GET CONTAINER('Message')
ACTIVITY(3270-act-name) INTO(msg-out-buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
decode msg-out-buffer
If mqcih-facility = blank
EXEC CICS RETURN ENDACTIVITY END-EXEC
Else
encode msg-in-buffer
EXEC CICS DEFINE ACTIVITY (3270-act-name)
TRANSID(next-transaction-id)
EVENT(3270-Complete)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS PUT CONTAINER('Message')
ACTIVITY(3270-act-name)
FROM(msg-in-buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RUN ACTIVITY(3270-act-name)
ASYNCHRONOUS
FACILITYTOKN(8-byte token)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RETURN END-EXEC
End-If.
```

Bridge exit program

```
Init.
pass userdata from the brdata to BRXA..
Bind.
EXEC CICS GET CONTAINER('Message')
INTO(3270-msg-in-buffer)
RESP(data-area) RESP2(data-area) END-EXEC..
Term.
EXEC CICS PUT CONTAINER('Message')
FROM(3270-msg-out-buffer)
RESP(data-area) RESP2(data-area) END-EXEC.
EXEC CICS RETURN END-EXEC
```

Note that:

- The client starts each transaction in the pseudoconversation by defining and running a new child activity, rather than by reactivating the same child activity with a different input event. This is necessary, in case the next-transaction IDs returned by the 3270 application are different—that is, in case each step of the pseudoconversation is implemented by a differently named transaction. (The variable *next-transaction-id* is used to name the transaction that implements each new child activity.)
- In this example, the variable *3270-act-name* is used to name each child activity differently. An alternative approach might be to delete the completed child activity before redefining it with a different TRANSID.
- In this example, the variable *3270-Complete* is used to name each activity completion event differently. This is not strictly necessary, because if the previous child activity completed normally its completion event is deleted from the event pool of the client following the CHECK ACTIVITY command.
- The output message returned by the bridge exit program should contain an 8-byte token representing the bridge facility. So that the bridge facility is reused for the next transaction in the pseudoconversation, the client uses the FACILITYTOKN option of the RUN ACTIVITY command to pass the token to the next child activity.

Transaction routing of pseudoconversations

The 3270 bridge does not support transaction routing of pseudoconversations. If a 3270 transaction is pseudoconversational, and is started from BTS, all of its constituent transactions must run in the same CICS region. Understand the options available in this situation.

If one of the transactions is routed to a different region, an ABRH abend occurs. One way to ensure that all the transactions run in the same region is for the client to run the child activities synchronously. Activities that are run synchronously always run in the local region - they are never routed.

However, although all the transactions in a pseudoconversation have to run in the same region, they do not have to run in the same region as the client; nor do they have to run in a specific region. If you use CICSPlex SM for routing purposes, you can define all the 3270 transactions in a pseudoconversation as part of the same transaction group.

Defining all the 3270 transactions in a pseudoconversation as part of the same transaction group gives you two options:

1. You can define the transaction group to run on a specific named region.
2. You can define the transaction group to run on whichever region the first transaction within a BTS process runs on. This option is preferred.

Using timers

You can use timers to avoid indefinite waits for a 3270 transaction to reply.

To avoid indefinite waits for a 3270 transaction to reply, the client could set a timer. If the timer expires, the client is reactivated and assumes that an error has occurred. The client can cancel the 3270 transaction, by issuing a CANCEL ACTIVITY command if the activity has not started, or by using a SET TASK PURGE command if it has.

Abend processing

You can use this pseudocode example to process your 3270 transaction abends.

If the 3270 transaction ends abnormally, an abend call is made to the bridge exit. This call occurs at the end of the transaction - it cannot be used to implement an abend handler.

If it is necessary for the exit program to reply to the client, it cannot do so by issuing a PUT CONTAINER command. Because BTS activities are always recoverable, the command would be backed out. One solution is for the exit program to write a message to an unrecoverable transient data or temporary storage queue. It could, for example, delegate this task to a child activity.

The following example contains pseudocode for dealing with an abend of the 3270 transaction. The *Requestor* activity is a child of the bridge exit; it handles the abend.

Table 7. Pseudocode for dealing with an abend of the 3270 transaction

Bridge exit program	Requestor activity

`Abend.`
`encode abend-in-buffer from 3270-msg-out-buffer`
`EXEC CICS DEFINE ACTIVITY ('Requestor')`
`TRANSID('ABE1')`
`RESP(data-area) RESP2(data-area) END-EXEC.`
`EXEC CICS PUT CONTAINER('Abend')`
`ACTIVITY('Requestor')`
`FROM(abend-in-buffer)`
`RESP(data-area) RESP2(data-area) END-EXEC.`
`EXEC CICS LINK ACTIVITY('Requestor')`
`RESP(data-area) RESP2(data-area) END-EXEC.`
`EXEC CICS CHECK ACTIVITY('Requestor')`
`COMPSTATUS(status) ABCODE(a)`
`RESP(data-area) RESP2(data-area) END-EXEC`
`If status NOT = DFHVALUE(NORMAL)`
`EXEC CICS ABEND ABCODE(a)`
`NODUMP`
`RESP(data-area) RESP2(data-area) END-EXEC.`
`End-If..`
`EXEC CICS RETURN END-EXEC`
	`WHEN DFH-Initial`
	`EXEC CICS GET CONTAINER('Abend')`
	`INTO(msg-in-buffer)`
	`RESP(data-area) RESP2(data-area) END-EXEC.`
	`decode msg-in-buffer`
	`output a message to a non-recoverable`
	`TD or TS queue.`
	`EXEC CICS RETURN END-EXEC`

The exit program issues a LINK ACTIVITY, rather than a RUN ACTIVITY SYNCHRONOUS, command to activate the *Requestor* activity. This is necessary because the child must execute in the same unit of work as the exit program.

Transaction restart

The 3270 bridge does not support transaction restart. If a client activity is restarted and tries to reuse a bridge facility token, an ABRH abend occurs.

Sample programs

CICS supplies sample programs that demonstrate how to integrate 3270-based transactions into BTS applications.

The samples are:

- DFH0CBAC, a client activity program
- DFH0CBAE, a bridge exit program
- DFH0CBAI, creates an input message
- DFH0CBAO, processes an output message.

The samples are supplied, in COBOL source code, in the SDFHSAMP library. They contain explanatory comments. Like the pseudocode examples in this chapter , the samples use containers named `Message` , `Request` , and Abend.

Note: To use the samples, you will also need to compile the 3270 bridge formatter program, DFH0CBRF.

Sample resource definitions are in RDO group DFH$BABR.

The sample programs are compatible with the 3270 bridge support pack, CA1E. The BTS passthrough transaction is BRCB.

Overview of BTS API commands

Use the CICS business transaction services application programming interface (API) commands to work on processes and activities, data-containers, events, and objects.

It contains:

- “Process- and activity-related commands” on page 73
- “Container commands” on page 74

- “Event-related commands” on page 75
- “Browsing and inquiry commands” on page 77
- “BTS system events” on page 81.

This section groups the API commands by function, giving a brief overview of what each can be used for. For an alphabetical listing of the commands, or for detailed programming information, see BTS application programming commands.

Process- and activity-related commands

For processes and activities, use these CICS business transaction services commands to create, activate, and stop processes and activities, to retrieve status information, and to give a unit of work access to an activity.

Creating, activating, and terminating processes and activities

You can use these commands to create, activate, return to initial state, control the progress of, end, or delete processes and activities.

Use these commands to create processes and activities:

DEFINE PROCESS
Creates a new process.

DEFINE ACTIVITY
Creates a new child activity.

Use these commands to activate a process or activity:

RUN
Invokes a program that implements a process or activity. Runs it synchronously or asynchronously with the requestor, in a separate unit of work, and with the transaction attributes specified on the DEFINE PROCESS or DEFINE ACTIVITY command.

LINK ACTIVITY
Invokes a program that implements an activity. Runs it synchronously with the requestor, in the same unit of work, and with the same transaction attributes as the requestor.

LINK ACQPROCESS
Invokes the program that implements the process that is currently acquired by the requestor. Runs the program synchronously with the requestor, in the same unit of work, and with the same transaction attributes as the requestor.

Use these commands to return a process or activity to its initial state:

RESET ACQPROCESS
Resets the currently acquired process to its initial state - used before trying the process again.

RESET ACTIVITY
Resets an activity to its initial state - used before trying an activity again.

Use these commands to control the progress of a process or activity:

SUSPEND (BTS)
Prevents a process or activity being reattached if events in its event pool fire.

RESUME
Allows a suspended process or activity to be reattached if events in its event pool fire.

Use these commands to terminate an activity:

RETURN ENDACTIVITY
Indicates that a process or activity is complete.

CANCEL (BTS)
Forces a process or activity to complete.

Use this command to destroy an activity:

DELETE ACTIVITY
Removes a child activity from the BTS repository data set where it is defined.

Retrieving information about activities

You can use the **ASSIGN** , **CHECK ACQPROCESS** , or **CHECK ACTIVITY** commands to retrieve status information for your activities.

Use this command to discover the activity the current unit of work is acting for:

ASSIGN
Returns information about the activity the current unit of work is acting for.

Use these commands to check the response from a process or activity:

CHECK ACQPROCESS
Returns the completion status of the process that is currently acquired by the requestor.

CHECK ACTIVITY
Returns the completion status of an activity.

See also "Browsing and inquiry commands" on page 77.

Relating UOWs and activities

You can use the **ACQUIRE** command to give a unit of work access to an activity.

ACQUIRE
Allows a unit of work executing outside a BTS process to gain access to an activity within the process.

Container commands

You can use CICS business transaction services commands to perform actions on data-containers. For example, **PUT CONTAINER (BTS)** , **GET CONTAINER (BTS)** , **MOVE CONTAINER (BTS)** , and **DELETE CONTAINER (BTS)**.

The CICS business transaction services commands that perform actions on data-containers are:

PUT CONTAINER (BTS)
Use this command to save data in a data-container associated with a specified BTS activity or process. If the named container does not exist, it is created. If the named container exists, its previous contents are overwritten.

GET CONTAINER (BTS)
Use this command to read data from a data-container associated with a specified BTS activity or process.

MOVE CONTAINER (BTS)
Use this command, instead of GET CONTAINER (BTS) and PUT CONTAINER (BTS), as a more efficient way of transferring data between activities. Using GET CONTAINER and PUT CONTAINER, you must:

1. Issue a GET CONTAINER NODATA command to retrieve the length of the data in the source container.
2. Allocate an area of working storage sufficient to hold the data.
3. Issue a GET CONTAINER command to retrieve the data into working storage.
4. Issue a PUT CONTAINER command to store the data in the target container.

Using MOVE CONTAINER, only one command is required and no working storage needs to be allocated. No data is moved; only CICS internal references are changed.

Use MOVE CONTAINER, rather than GET CONTAINER and PUT CONTAINER, if you have no need to keep the source container.

DELETE CONTAINER (BTS)
Use this command to delete a BTS data-container and discard any data that it contains.

Event-related commands

Use the terms that describe BTS events and the event-related commands to perform actions on your event pool.

Terminology

These terms are used to describe CICS business transaction services (BTS) events.

For a more detailed introduction to BTS events, see “BTS events” on page 20.

Event states

An event can be in one of two states: FIRED (true) or NOTFIRED (false). Which state it is in is known as the event's *fire status* .

Atomic events

Atomic events are simple, “low-level” events.

The BTS atomic events are:

Activity completion events
Events that fire on completion of an activity.

Input events
Events delivered to an activity when it is activated, conveying the reason for its attachment.

Timer events
Events that fire when a timer expires.

System events
Input events defined by the BTS system.

Note: Activity, input, timer, and composite events are referred to as *user-defined events* , because they are defined by the programmer. System events are defined by BTS.

Composite events

A composite event is a method of grouping user-defined (that is, non-system) atomic events in a logical expression, which is named.

An atomic event that is included in a composite event is known as a *subevent* . Subevents that fire are placed on the *subevent queue* of the composite event. Each composite event has a subevent queue associated with it. The subevent queue:

- May be empty
- Contains only those subevents that have fired and not been retrieved.

Reattachment events

An event that fires, and causes an activity to be reattached is known as a *reattachment event* .

All user-defined events except subevents cause the activity to which they are defined to be reattached when they fire. Thus, all user-defined events (both atomic and composite, but excluding subevents) are potentially reattachment events. All system events are reattachment events.

At times, reattachment might occur because of the firing of more than one event. Reattachment events are placed on the *reattachment queue* of the activity, from which they can be retrieved. Each activity has a reattachment queue, which:

- May be empty
- Contains only those reattachment events that have fired and not been retrieved.

Timers

A timer is a BTS object that expires when the system time becomes greater than a specified time or after a specified period has elapsed. When you define a timer, a timer event is automatically associated with it. When the timer expires, its associated event fires.

Event pools

Events are defined within *event pools* .

Each activity has an event pool, which contains the set of events that it recognizes (that is, events that have been defined to it, and system events). An activity's event pool is initialized when the activity is created, and deleted when the activity is deleted. All the event-related commands except FORCE TIMER operate on the event pool associated with the *current* activity.

The event-related commands

You can use event-related commands to perform actions on your event pool of the current activity. You can define input and composite events, control timers and timer events, manipulate events, and check whether an event has fired.

Use these commands to define user events other than activity completion and timer events:

DEFINE INPUT EVENT
Defines an input event.

DEFINE COMPOSITE EVENT
Defines a composite event.

Use these commands to control timers and timer events:

DEFINE TIMER
Defines a timer, and associates an event with it.

FORCE TIMER
Forces early expiry of a timer, and causes the associated event of the timer to fire.

CHECK TIMER
Returns the status of a timer and, if the timer has expired, deletes its associated event.

DELETE TIMER
Deletes a timer and its associated event (if any).

Use these commands to manipulate events:

ADD SUBEVENT
Adds a subevent to a composite event.

REMOVE SUBEVENT
Removes a subevent from a composite event.

DELETE EVENT
Removes an input or composite event from the event pool of the current activity.

Note: DELETE EVENT cannot be used to delete activity completion events (which are implicitly deleted when a response from the completed activity has been acknowledged by a CHECK ACTIVITY command, or when a DELETE ACTIVITY command is issued), timer events, or system events.

RETRIEVE REATTACH EVENT
Retrieves the name of an event that caused the current activity to be reattached and, if the event is atomic, resets its fire status to NOTFIRED.

RETRIEVE SUBEVENT
Retrieves the name of the next subevent in a subevent queue of the composite event, and resets the retrieved fire status of the sub-event to NOTFIRED.

Use this command to check whether an event has fired:

TEST EVENT
Tests whether an event has fired.

Browsing and inquiry commands

Important: The API commands described here are different in kind from the commands described previously in this chapter.

The commands described previously in this chapter are the basic commands used by application programmers to create BTS applications.

The commands described here have more specialized uses. They might be used, for example, in a utility program written to investigate a stuck process. A typical BTS business application does not need to inquire on or browse the objects it creates, and therefore does not use these commands.

You can use CICS business transaction services commands in your programs to search for and examine BTS objects. These commands can be summarized as browsing commands and inquiry commands.

The object-identifiers and names retrieved from the browsing or inquiry commands can be specified on a subset of the other BTS API commands. By using this method you can start actions against specified activities, processes, and data-containers.

Browsing commands

You can use the browsing commands to locate BTS objects and examine their relationships to each other.

The objects that you can browse include:

- Activities
- Data-containers
- Events
- Processes
- Process-types.

Process-types are a special case. They are browsed using the START, NEXT, and END options of the INQUIRE PROCESSTYPE command, as described in INQUIRE PROCESSTYPE.

Each browse has three commands associated with it:

STARTBROWSE
STARTBROWSE:

1. Tells CICS to begin a browse of a specified type of BTS object.
2. Defines the scope of the browse. Except for browses of processes and process-types, an absence of additional arguments indicates that the browse is to have the scope of the current activity.
3. Returns a browse token which must be included on the remaining commands within the browse.

GETNEXT
Locates the next object within the scope of the browse, or returns the END condition if there are no more objects found.

GETNEXT always returns sufficient information to allow additional actions to be taken. For example, in a browse of the children of a specified parent activity, the GETNEXT ACTIVITY command returns both the name and the identifier (ACTIVITYID) of the next child activity that it finds. The name could be used to decide where the current browse should be paused. The identifier could be used to start a new browse - which might be containers of the child activity, for instance.

ENDBROWSE
Ends the browse.

Inquiry commands

You can use a program to get details of a specific BTS object by issuing an **INQUIRE** command instead of browsing for the object. Note some limitations about using the **INQUIRE** command.

The objects that you can inquire about include:

- Activities
- Data-containers
- Events
- Processes
- Process-types
- Timers.

The object inquired upon might have been located by browsing. For example, a program can use a browse to locate an activity, then issue an INQUIRE ACTIVITYID command to find out the name of the program associated with the activity, the userid under whose authority it runs, or its current completion status.

Note: All INQUIRE commands try to locate a record on a BTS repository data set, and to read information from it if found. This operation does not obtain an exclusive control lock for the record; therefore the data in the record might change while the operation is taking place.

Be careful when issuing INQUIRE commands from within programs that execute as part of an activity, if the commands refer to records which might be modified by the same program. The INQUIRE command always goes to the repository for the record it needs, and might not see changes made by the program. This can lead to unexpected results. For example, a program might define a new activity and then issue a command to inquire upon it, only to be told that the activity does not exist (because the activity-record has not yet been committed to the repository).

Tokens and identifiers

A *browse token* uniquely identifies a browse within a CICS region. An *activity identifier* is a means of uniquely referring to an instance of an activity that has been retrieved from a BTS repository data set.

The browse token that is returned on a **STARTBROWSE** command must be supplied on the corresponding **GETNEXT** and **ENDBROWSE** commands. CICS discards it after the **ENDBROWSE** command.

The lifetime of a browse token is from a **STARTBROWSE** command to an **ENDBROWSE** command or sync point, whichever comes first. Therefore, your applications:

- Should not attempt to use a token after the browse has ended
- Should not attempt to use a token if a sync point is encountered before the browse has completed

When an activity identifier or a process name is known, it can be used as a scoping argument to a new browse. It can also be specified on certain API commands which cause actions to be taken against existing activities or processes, or their containers and events - see “Commands that take identifiers returned by browse operations” on page 78 . The lifetime of an activity identifier is the same as the lifetime of the activity it refers to. Thus, it can be used after an ENDBROWSE and after a sync point.

A data-container or an event cannot be identified in the same way as an activity or a process, because it forms part of a record on a BTS repository data set. Instead, it must be referenced through the activity or process to which it belongs.

Commands that take identifiers returned by browse operations

You can specify the activity identifier or the process name returned by your browse operations on various commands; for example, **GETNEXT ACTIVITY** , **GETNEXT PROCESS** , **INQUIRE PROCESS** , or **INQUIRE ACTIVITYID** . System programmers must be able to modify a business transaction after it has started, especially if your transaction gets into a state in which it cannot complete.

A user-written utility program could, for example:

1. Use a series of browses to locate a particular process or activity
2. When the process or activity is found, inquire about its state
3. Gain control of the process or activity by issuing an ACQUIRE command
4. Correct a processing problem by issuing a further command or commands.

You can specify the activity identifier returned by a GETNEXT ACTIVITY, GETNEXT PROCESS, or INQUIRE PROCESS command on any of the following commands:

- ACQUIRE
- INQUIRE ACTIVITYID
- INQUIRE CONTAINER
- INQUIRE EVENT
- INQUIRE TIMER
- STARTBROWSE ACTIVITY
- STARTBROWSE CONTAINER
- STARTBROWSE EVENT

You can specify the process name returned by a GETNEXT PROCESS (or INQUIRE ACTIVITYID) command on any of the following commands:

- ACQUIRE
- INQUIRE CONTAINER
- INQUIRE PROCESS
- STARTBROWSE ACTIVITY
- STARTBROWSE CONTAINER

After you have acquired a process or activity, you could, for example, issue one or more of the following commands against it:

- CANCEL (BTS)
- CHECK
- DELETE ACTIVITY
- DELETE CONTAINER (BTS)
- FORCE TIMER
- GET CONTAINER (BTS)
- LINK
- MOVE CONTAINER (BTS)
- PUT CONTAINER (BTS)
- RESET ACQPROCESS
- RESUME
- RUN
- SUSPEND (BTS)

Browsing examples

Follow these examples of applications to learn how to use the browsing and inquiry commands.

Example 1

An application, which has not issued any requests to BTS, needs to know whether a particular container belongs to a child of the root activity of a particular process, the name and type of which are known.

```
 EXEC CICS INQUIRE PROCESS(pname)
PROCESSTYPE(ptype)
ACTIVITYID(root_id)

if process found then browse the children of its root activity
EXEC CICS STARTBROWSE ACTIVITY
ACTIVITYID(root_id)
BROWSETOKEN(root_token)
EXEC CICS GETNEXT ACTIVITY(child_name)
BROWSETOKEN(root_token)
ACTIVITYID(child_id)
loop while the child is not found and there are more activities
EXEC CICS GETNEXT ACTIVITY(child_name)
BROWSETOKEN(root_token)
ACTIVITYID(child_id)
end child activity browse loop

if the child we are looking for is found then browse its containers
EXEC CICS STARTBROWSE CONTAINER
ACTIVITYID(child_id)
BROWSETOKEN(c_token)
EXEC CICS GETNEXT CONTAINER(c_name)
BROWSETOKEN(c_token)
loop while container not found and there are more containers
EXEC CICS GETNEXT CONTAINER(c_name)
BROWSETOKEN(c_token)
end container browse loop

EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(c_token)
EXEC CICS ENDBROWSE ACTIVITY BROWSETOKEN(root_token)
```

Figure 50. Browsing example 1

Example 2

An application, which has not issued any requests to BTS, needs to know whether a particular data-container is one of the global containers associated with a particular process.

If it is not, the program needs to know whether the container is owned by the root activity of that process.

```
 EXEC CICS INQUIRE PROCESS(pname) PROCESSTYPE(ptype)
ACTIVITYID(root_id)

if process found then browse its containers
EXEC CICS STARTBROWSE CONTAINER PROCESS(pname) PROCESSTYPE(ptype)
BROWSETOKEN(c_token_1)
EXEC CICS GET NEXT CONTAINER(c_name)
BROWSETOKEN(c_token_1)
loop while container not found and there are more containers
EXEC CICS GET NEXT CONTAINER(c_name)
BROWSETOKEN(c_token_1)
end process container browse loop

if container not found browse the root activity's containers
EXEC CICS STARTBROWSE CONTAINER ACTIVITYID(root_id)
BROWSETOKEN(c_token_2)
EXEC CICS GETNEXT CONTAINER(c_name)
BROWSETOKEN(c_token_2)
loop while container not found and there are more containers
EXEC CICS GETNEXT CONTAINER(c_name)
BROWSETOKEN(c_token_2)
end root activity's container browse loop

EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(c_token_2)
EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(c_token_1)
```

Figure 51. Browsing example 2

Example 3

A program running as an activation of an activity needs to find whether a named event has been defined to any of its children; that is, whether the event exists in any of the children of the event pool. If the event exists, the program needs to retrieve its fire status.

Because the program starts an activity browse on which no activity identifier or process name is specified, BTS browses the current activity. The program retrieves the identifier of each child activity, and uses the identifier to browse the events child.

Activity identifiers remain valid after the browse that obtained them has ended, they are valid for the life of the activity itself. To illustrate this , the program uses the identifier of the activity whose event pool contains the named event, on an INQUIRE EVENT command, after it has ended the browse .

```
 EXEC CICS STARTBROWSE ACTIVITY BROWSETOKEN(parent_token)
loop until the event is found or there are no more child activities
EXEC CICS GETNEXT ACTIVITY(child_activity_name)
BROWSETOKEN(parent_token)
ACTIVITYID(child_activity_id)
EXEC CICS STARTBROWSE EVENT ACTIVITYID(child_activity_id)
BROWSETOKEN(event_token)
loop until event found or there are no more events
EXEC CICS GETNEXT EVENT(event_name)
BROWSETOKEN(event_token)
end event browse loop

EXEC CICS ENDBROWSE EVENT BROWSETOKEN(event_token)

end child activity browse loop
EXEC CICS ENDBROWSE ACTIVITY BROWSETOKEN(parent_token)
EXEC CICS INQUIRE EVENT(event_name)
ACTIVITYID(child_activity_id)
FIRESTATUS(fstatus)
```

Figure 52. Browsing example 3

BTS system events

BTS produces the DFHINITIAL system events as a result of its own processing. DFHINITIAL is the only BTS system event

DFHINITIAL
: The activity is being attached for the first time in this process, *or* it is trying again after being reset with a RESET ACTIVITY command. An activity must be coded to cope with this event, which specifies that it must perform any initial housekeeping.

Chapter 4. Administering BTS

Terminology: This chapter assumes that you are familiar with the terminology and concepts of CICS dynamic routing. For introductory information about dynamic routing, see Introduction to CICS dynamic routing.

You can operate BTS in a single CICS region. However, CICS business transaction services are sysplex-enabled. For example, you can use workload separation to ensure that processes of the same process-type are handled by a particular set of regions, and you can use workload routing to route activity requests across a set of regions.

When you use workload routing the activities that constitute the process might execute on several regions, and different activations of the same activity might execute on different regions

The scope of a BTS-set

You can use the scope of a *BTS-set* to define the set of CICS regions across which related BTS processes and activities can run.

All the regions in a BTS-set:

- Must be interconnected, to support routing of activities between the regions. This does not mean that the regions must be in the same CICSplex, but that each region in the BTS-set must be connected to every other region. For more information, see “Understanding distributed routing” on page 84.
- Must have access to the BTS repository data set (or data sets) on which details of the relevant processes are stored.

  There are two methods of sharing the repository data set:

  1. Using VSAM record-level sharing (RLS). To use this method, all the regions of the BTS-set must be within the same MVS Parallel Sysplex. (A "Parallel Sysplex" is a sysplex in which the MVS images are linked through a coupling facility). VSAM RLS requires a coupling facility. Each region in the BTS-set must open the shared repository file or files in RLS mode.
  2. Using function-shipping to a file-owning region (FOR).

Within an MVS sysplex, it is possible to have multiple BTS-sets. For more information, see Figure 55 on page 88. Imagine, for example, that within your sysplex you operate two CICSplexes. You could decide to divide your BTS processes by process-type, between the two CICSplexes. Alternatively, you could decide to set up two BTS-sets within the same CICSplex.

Using separate BTS-sets is a high-level form of workload separation. By definition, routing of activities *between* BTS-sets is not possible.

A note about audit logs

When you create a BTS-set, the activities that constitute a single process might run on several regions. Therefore, to collate the audit data for each process, your audit logs must use MVS shared log streams.

If all the CICS regions in your BTS-set are in the same MVS image, you can define the log streams to use either coupling facility structures or DASD-only logging. However, if the CICS regions are on *different* MVS images, the log streams must use coupling facility structures rather than DASD-only logging. This is because CICS regions on different MVS images cannot access the same DASD-only log stream at the same time.

If your BTS-set spans multiple MVS images and you use DASD-only log streams for your BTS logs, you are unable to use shared log streams. In this case, the audit records for a particular process could be split between several log streams; you have to collate the data yourself.

For further information about audit logs, see Creating a BTS audit trail.

Dynamic routing of BTS activities

You can use a BTS-set to dynamically route your BTS processes and activities across the participating regions.

BTS routing is at the activation level; for example, in the same process, different activations of the same activity might execute on different regions. When an event is signaled, the relevant activity is activated in the most appropriate region in the BTS-set, based on one or more of these criteria:

- Any workload separation specified by the system programmer
- Any affinities its associated transaction has with a particular region
- The availability of regions
- The relative workload of regions.

Which BTS activities can be dynamically routed?

Not all activations of BTS processes and activities can be routed. Processes and activities that are activated asynchronously with the requestor, with a RUN ASYNCHRONOUS command, can be routed either dynamically or statically.

Processes and activities that are activated synchronously with the requestor, with a RUN SYNCHRONOUS or LINK command, are always run locally. They cannot be routed *dynamically or statically*. A RUN SYNCHRONOUS or LINK command issued against an activity whose associated transaction is defined as DYNAMIC(YES), or as remote, results in the activity being run locally. When an activity is activated by a LINK command, *all* the attributes of its associated transaction are ignored, because the activity runs under the TRANSID of the parent - there is no **context-switch**.

Thus, to be eligible for dynamic routing:

- A process or activity must be run asynchronously with the requestor, with a RUN ASYNCHRONOUS command.
- The TRANSACTION definition for the CICS transaction associated with the process or activity must specify DYNAMIC(YES).

"Daisy-chaining" is not supported. Once a BTS process or activity has been routed to a target region it cannot be rerouted from the target to a third region, even though its associated transaction is defined as DYNAMIC(YES) in the target.

Understanding distributed routing

CICS has two dynamic routing models, the *hub routing model* and the *distributed routing model*. For each model a user-replaceable sample routing program is supplied. The CICSPlex SM routing program is suitable for use with both models.

The two user-replaceable sample routing programs are the dynamic routing program, DFHDYP, which implements the hub routing model, and the distributed routing program, DFHDSRP, which implements the distributed routing model. Both models and their associated routing programs are described in detail in Two routing models.

The CICSPlex SM routing program, EYU9XLOP, can be used with either routing model - that is, it can function as either a dynamic routing program, a distributed routing program, or both.

BTS routing uses the distributed routing model. It is important to understand how the *distributed routing model* differs from the traditional *hub routing model*.

The hub model

The *hub* is the model that has traditionally been used with CICS dynamic transaction routing. The model has advantages and disadvantages.

A dynamic routing program running in a terminal-owning region (TOR) routes transactions between several application-owning regions (AORs). Usually, the AORs (unless they are AOR/TORs) do not perform dynamic routing. Figure 53 on page 85 shows a hub routing model.

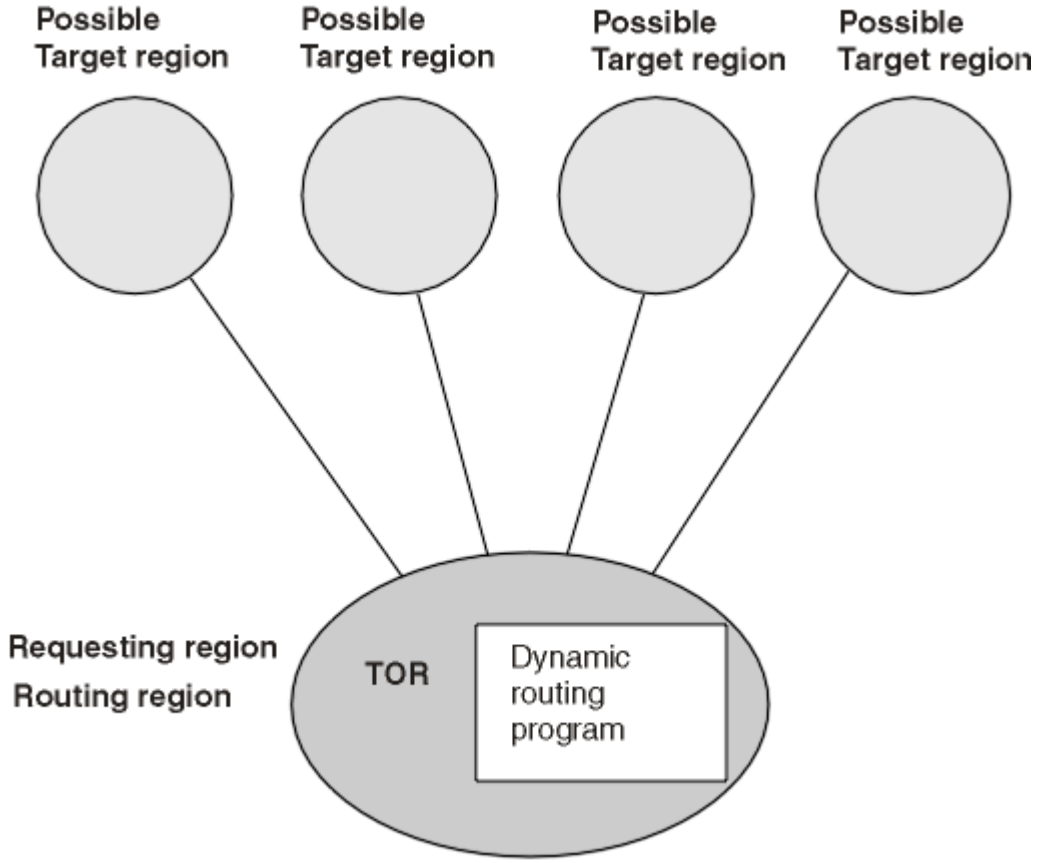


Figure 53. Dynamic routing using a hub routing model

The hub model applies to the routing of:

- Transactions started from terminals
- Transactions started by terminal-related EXEC CICS START commands
- Program-link requests received from outside CICS. The receiving region acts as a hub or TOR because it routes the requests among a set of back-end server regions.

The hub model is a hierarchical system—routing is controlled by one region (the TOR); normally, a routing program runs only in the TOR.

Advantage of the hub model

It is a relatively simple model to implement. For example, compared to the distributed model, there are few inter-region connections to maintain.

Disadvantages of the hub model

- If you use only one hub to route transactions and program-link requests across your AORs, the hub TOR is a single point-of-failure.

- If you use more than one hub to route transactions and program-link requests across the same set of AORs, you may have problems with distributed data. For example, if the routing program keeps a count of routed transactions for load-balancing purposes, each hub TOR will need access to this data.

The distributed model

In the distributed model used for BTS routing, each participating CICS region might be both a routing region and a target region. A distributed routing program runs in each region. The distributed model has advantages and disadvantages.

Figure 54 on page 86 shows a distributed routing model.

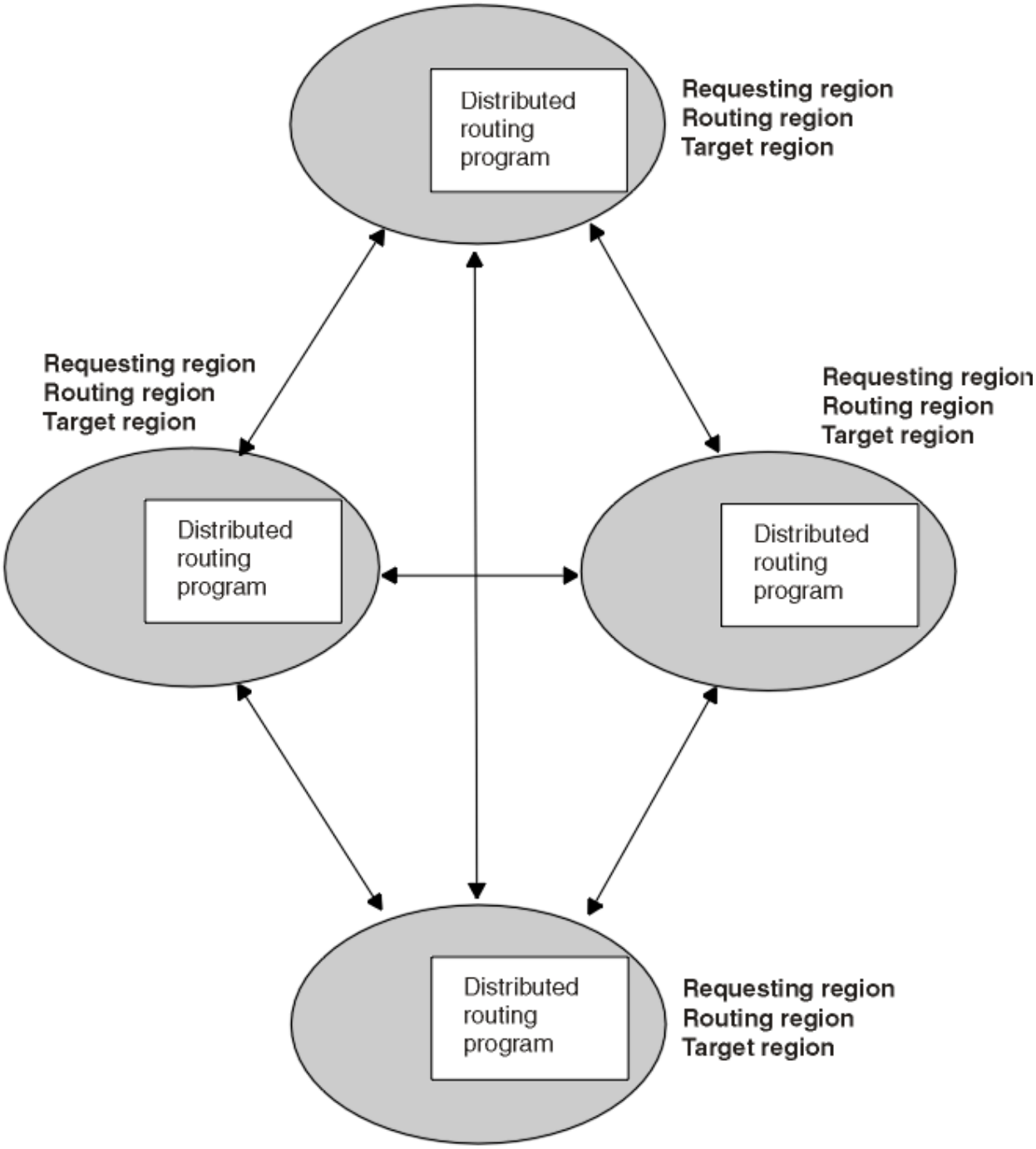


Figure 54. Dynamic routing using a distributed routing model

The distributed model applies to the routing of:

- BTS processes and activities
- Non-terminal-related **EXEC CICS START** requests.

The distributed model is a peer-to-peer system—each participating CICS region may be both a routing region and a target region. A distributed routing program runs in each region.

Advantage of the distributed model

There is no single point-of-failure.

Disadvantages of the distributed model

- Compared to the hub model, there are a great many inter-region connections to maintain.
- You may have problems with distributed data. For example, any data used to make routing decisions must be available to all the regions. CICSPlex SM solves this problem by using dataspaces.

Controlling BTS dynamic routing

You can use the CICSPlex System Manager product or a distributed routing program to control the dynamic routing of your BTS activities.

About this task

You can control the dynamic routing of your BTS activities by either of the following means:

1. Writing your own CICS distributed routing program - see “Using a CICS distributed routing program” on page 89.
2. Using the CICSPlex System Manager (CICSPlex SM) product to:
   - Specify workload separation for your BTS processes
   - Manage affinities
   - Control workload routing of the transactions associated with BTS activities.

   See “Using CICSPlex SM with BTS” on page 91.

Important: The distributed routing program must be specified in the **DSRTPGM** system initialization parameter in the routing region and in all potential target regions.

Creating a BTS-set

When using BTS in a sysplex, you must have several sets of BTS regions (BTS-sets). Each set deals with one or more process-types (types of business transaction). You can create these sets by cloning individual regions.

About this task

Figure 55 on page 88 shows a sysplex that contains two BTS-sets. BTS-set 1 handles all processes of type PAYROLL. All the regions in BTS-set 1 are interconnected and have access to the BTS repository that contains details of PAYROLL-type processes. BTS-set 2 handles all processes of types TRAVEL and MISC. All the regions in BTS-set 2 are interconnected and have access to the BTS repository that contains details of TRAVEL and MISC-type processes.

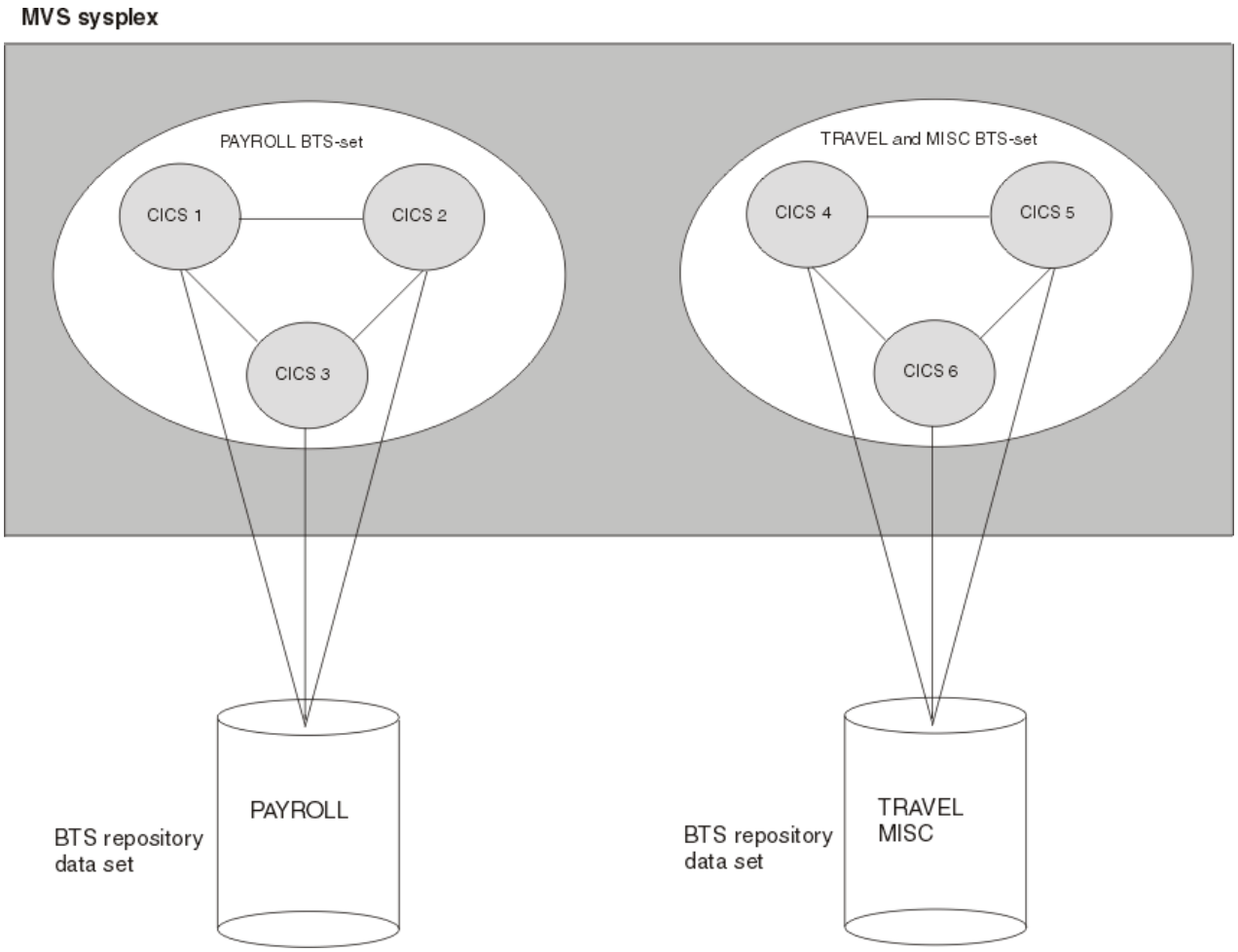


Figure 55. A sysplex containing two BTS-sets

The number of regions in a BTS-set is related to:

- The number of process-types handled by the BTS-set
- The workload associated with each process-type.

To create each BTS-set, perform the following steps on each of the regions in the set.

Procedure

1. Define a connection to every other region in the BTS-set.

   For performance reasons, you are recommended to use MRO or MRO/XCF rather than APPC connections.
2. Give the region access to the BTS repository that contains details of the process-types it is servicing.

   The name of the repository file is specified on the PROCESSTYPE definition or definitions.

   If you are using VSAM RLS to share the repository file, on the FILE definition that defines the repository file to CICS, specify RLSACCESS(YES).

   If you are using function-shipping to share the repository file, on the FILE definition that defines the repository file to CICS, specify REMOTESYSTEM(*name_of_file-owning_region*).
3. On the TRANSACTION definition for each transaction associated with a BTS activity, specify DYNAMIC(YES).

   Do not specify the REMOTESYSTEM option.

For general information about defining transactions for transaction routing, and specific information about defining transactions associated with BTS activities, see Defining transactions for transaction routing.

4. Enable the distributed routing program.

Naming the routing program

You can specify, discover, and change the distributed routing program.

To specify the distributed routing program, use the **DSRTPGM** system initialization parameter. The name you specify might be that of the CICSPlex SM routing program, EYU9XLOP, or of your own user-written program. For information about **DSRTPGM**, see Naming the routing program.

After CICS has initialized, you can discover which distributed routing program, if any, is in use by issuing an **INQUIRE SYSTEM** command. The DSRTPROGRAM option returns the program name.

After CICS has initialized, you can change the distributed routing program currently in use by issuing a **SET SYSTEM** command. The DSRTPROGRAM option specifies the program name.

Using a CICS distributed routing program

To write your own distributed routing program, you can use the CICS-supplied default distributed routing program, DFHDSRP, as a model.

Important: The distributed routing program must be specified in the **DSRTPGM** system initialization parameter in the routing region and in all potential target regions.

How the distributed routing program relates to the dynamic routing program

The supplied user-replaceable routing programs, DFHDYP and DFHDSRP complement each other. Use DFHDYP for dynamic routing and DFHDSRP for distributed routing.

The dynamic routing program, DFHDYP
Can be used to route:

- Transactions started from terminals
- Transactions started by terminal-related START commands
- Program-link requests.

The distributed routing program, DFHDSRP
Can be used to route:

- BTS processes and activities
- Non-terminal-related START requests.

The two routing programs:

1. Are specified on separate system initialization parameters.
2. Are passed the same communications area. Certain fields that are meaningful to one program are not meaningful to the other.
3. Are invoked at similar points - for example, for route selection, route selection error, and (optionally) at termination of the routed transaction or program-link request.

Together, these three factors give you a good deal of flexibility. You could, for example, do any of the following:

1. Use different user-written programs for dynamic and distributed routing.
2. Use the same user-written program for both dynamic and distributed routing.
3. Use a user-written program for dynamic routing and the CICSPlex SM routing program for distributed routing, or vice versa.

The distributed routing program differs from the dynamic routing program in several important ways:

1. The dynamic routing program is invoked only if the resource (the transaction or program) is defined as DYNAMIC(YES). The distributed routing program, however, is invoked (for eligible non-terminal-related START requests and BTS activities) even if the associated transaction is defined as DYNAMIC(NO) - though it cannot route the request. This means that the distributed routing program is better able to monitor the effect of statically routed requests on the relative workloads of the target regions.
2. Because the dynamic routing program uses the hierarchical "hub" routing model - one routing program controls access to resources on several target regions - *the routing program that is invoked at termination of a routed request is the same that was invoked for route selection.*

   The distributed routing program, however, uses the distributed model, which is a peer-to-peer system - the routing program itself is distributed. *The routing program that is invoked at initiation, termination, or abend of a routed transaction is not the same program that was invoked for route selection* - it is the routing program on the target region.
3. The distributed routing program is invoked at more points than the dynamic routing program. When and where the distributed routing program is called shows the points at which the distributed routing program is invoked, and the region on which each invocation occurs.

Writing a distributed routing program

You can use the CICS-supplied default distributed routing program, DFHDSRP, as a model when writing your own distributed routing program.

For general information about user-replaceable programs, and specific information about how to write a distributed routing program, see Writing a distributed routing program.

Important: The distributed routing program must be specified in the **DSRTPGM** system initialization parameter in the routing region and in all potential target regions.

When your routing program is called

For processes and activities started by **RUN ASYNCHRONOUS** commands, your distributed routing program is called at various points on the requesting region and on the target region.

On the requesting region:

1. Either of these activities:
   - For routing the activity. This routing occurs when the transaction associated with the activity is defined as DYNAMIC(YES).
   - For notification of a statically routed activity. This notification occurs when the transaction associated with the activity is defined as DYNAMIC(NO). The routing program is not able to route the activity. It could, however, do other things.
2. If an error occurs in route selection. For example, if the target region returned by the routing program on the route selection call is unavailable. This gives the routing program the opportunity to specify an alternate target. This process iterates until the routing program selects a target that is available or sets a non-zero return code.
3. After CICS has tried (successfully or unsuccessfully) to route the activity to the target region.

   This invocation signals that (unless the requesting region and the target region are one and the same) the responsibility of the requesting region for the transaction has been discharged. The routing program might, for example, use this invocation to release any resources that it has acquired on behalf of the transaction.

On the target region:

These invocations occur only if the routing program on the requesting region has specified that it must be called again on the target region:

1. When the activation starts on the target region (that is, when the transaction that implements the activity starts).
2. If the routed activation (transaction) ends successfully.

3. If the routed activation (transaction) abends.

Figure 56 on page 91 shows the points at which the distributed routing program is called, and the region on which each invocation occurs. The "target region" is not necessarily remote - it could be the local (requesting) region, if the routing program chooses to run the activity locally.

Figure 56. When and where the distributed routing program is called

When it is called on the target region for transaction initiation, termination, or abend, the routing program can update a count of BTS activities that are currently running on that region. When it is called on the requesting region for route selection, the routing program can use the counts maintained by all the regions in the routing set (including itself) as input to its routing decision. This routing decision requires that all the regions in the BTS-set have access to a common data store on which the counts are stored. For further details, see Writing a distributed routing program.

Restrictions on the routing program

The distributed routing program runs outside a unit of work environment. Therefore your program must not alter any recoverable resources or issue file control or temporary storage requests.

Using CICSPlex SM with BTS

You can use CICSPlex SM services to route BTS activities around your BTS-sets, avoiding the need to write your own distributed routing program.

Overview of CICSPlex SM workload management

You can use CICSPlex SM workload management (WLM) functions to route transactions defined as dynamic from a requesting region to a suitable target region selected at the time the transaction is started.

WLM functions

The CICSPlex SM dynamic routing program supports functions:

Workload separation
Routes specific transactions to a group of target regions based on BTS process-type, or a combination of process type and transaction name, or any combination of user ID, terminal ID, and transaction name. For example, using the CICSPlex SM workload separation function, you can specify that transactions beginning with the characters 'SAL' and initiated by members of your sales department must be routed to the group of target regions, SALESGRP, allocated to that department.

Workload routing
Routes transactions among a group of target regions according to the availability and activity levels of those regions. You can use workload routing in addition to, or in place of, workload separation. For example, CICSPlex SM can route the transaction workload among the SALESGRP regions by selecting, as each transaction is initiated, the target region that is likely to deliver the best performance.

Inter-transaction affinity
Requires related transactions to be processed by the same target region. You can use IBM CICS Interdependency Analyzer for z/OS to identify affinities between transactions.

For further introductory information about CICSPlex SM and workload management, see the CICSPlex SM overview.

Using CICSPlex SM to route BTS activities

You can use CICSPlex SM to route BTS Activities around a BTS-set. The routing is based on the workload separation criteria you define. Take note of any affinities that might be introduced.

When routing BTS activities around a BTS-set, CICSPlex SM Workload Management selects a target region based on:

- Any workload separation criteria that you have defined
- The current workloads of the eligible regions
- Any active affinities
- The speed of the communication links to the eligible regions.

The CICSPlex SM component of CICS Transaction Server for OS/390, Version 1 Release 3 understands BTS processes and activities. This makes it possible to separate a BTS workload based on process-type. For example, you could specify that WLM is to route all processes of process-type 'TRAVEL' to one region in the BTS-set, and all processes of type 'PAYROLL' to another region.

CICSPlex SM WLM and the IBM CICS Interdependency Analyzer for z/OS understand affinities between BTS activities and processes. Although BTS itself does not introduce any affinities, and discourages programming techniques that do, it does support legacy code, which might introduce affinities. You must define such affinities to CICSPlex SM WLM, so that it is able to make sensible routing decisions. It is important to specify the lifetime of each affinity; failure to do so might restrict WLM routing options unnecessarily.

Note that:

- A single CICSPlex SM can control routing within multiple BTS-sets. It cannot route activities *across* BTS-sets.
- Workload separation can be performed at two levels:
  1. By creating multiple BTS-sets.
  2. By CICSPlex SM within a BTS-set.

BTS operator commands

You can use the CBAM transaction and the CEMT transaction to inquire about and control CICS business transaction services resources.

CBAM BTS browser
You can use the CBAM transaction to browse the CICS business transaction services objects (process-types, processes, activities, containers, events, and timers) known to your region.

CEMT main terminal
You can use the CEMT transaction to inquire about and control CICS business transaction services resources.

- Use **CEMT INQUIRE PROCESSTYPE** to retrieve information about a CICS business transaction services process-type.

- Use **CEMT INQUIRE TASK** to retrieve information about a user task.
- Use **CEMT SET PROCESSTYPE** to change the attributes of a CICS business transaction services process-type.

CBAM BTS browser

You can use the CBAM transaction to browse the CICS business transaction services objects (process-types, processes, activities, containers, events, and timers) known to your region.

CBAM is a menu-driven transaction. The menus are hierarchically organized. By navigating downwards through the menus, you can display:

1. All the process-types that have been defined to this region with installed PROCESSTYPE definitions.
2. All the processes of a selected process-type. These processes are the processes of the selected type that currently exist on the repository data set pointed to by the installed PROCESSTYPE definition.

   Note: If you are operating BTS in a sysplex and the repository is shared with one or more other regions, some of the processes might have been defined on other regions.
3. The constituent activities of a selected process.
4. The details (program, transaction ID, user ID) of a selected activity.
5. One of the following resources:
   - The containers associated with a selected activity or process, *or*
   - The events in an event pool of the selected activity, *or*
   - The timers defined to a selected activity.

Note: This overview of the CBAM menu hierarchy is slightly simplified. Selectable fields allow you to bypass some screens.

CBAM is a “read-only” transaction - you cannot update any of the displayed attributes by over typing them.

Running the CBAM browser transaction

Start the CBAM browser transaction by typing CBAM on the command line and pressing the ENTER key. This gives you a list of all the process-types that have been defined to this region.

Process-types screen

The *Process-types* screen lists the process-types defined to a region and shows the CICS repository file name, process-type status, and the level of audit logging.

```
CBAM

  Processtype File     Status   Auditlevel

  CBTSAUDA    DFHBARF  Enabled  Activity
  CBTSSHR     DFHBSHR  Enabled  Off
  CBTSSHRF    DFHBSHR  Disabled Activity
  CBTSSHR2    DFHBSHR2 Disabled Off
  MORTLOANS   DFHMORT  Enabled  Process




 Use cursor and Enter for Processes

PF3=Return   7=Back   8=Forward
```

Figure 57. CBAM transaction: initial screen, showing the process-types defined to this region

The displayed fields mean:

Auditlevel
The level of audit logging currently active for processes of this type:

Activity
Activity-level auditing. Audit records are written from:

1. The process audit points
2. The activity primary audit points.

Full
Full auditing. Audit records are written from:

1. The process audit points
2. The activity primary *and* secondary audit points.

Off
No audit records are written.

Process
Process-level auditing. Audit records are written from the process audit points only.

For details of the records that are written from the process, activity primary, and activity secondary audit points, see Specifying the level of audit logging.

File
The CICS repository file on which records for processes of this type are stored.

Status
Whether the PROCESSTYPE is enabled or disabled—that is, whether new processes of this process-type can be defined.

Processtype
The name of a process-type.

If you place the cursor on the name of a process-type, or anywhere on the same line, and press ENTER, you get a list of all the processes of that type that currently exist on the repository data set pointed to by the installed PROCESSTYPE definition—see Figure 58 on page 94.

Processes screen

The *Processes* screen lists the process names, the mode of a process, the completion status, whether a process is suspended, and lists process containers for the process named in the process field.

```
CBAM                                          Processtype MORTLOANS

  Process                              Mode       Comp       Susp    Conts

  MORT000000014                        Dormant    Incomplete No       _
  MORT000000015                        Complete   Forced     No       _
  MORT000000016                        Dormant    Incomplete Sus      _
  MORT000000017                        Active     Incomplete No       _
  PERS000000114                        Initial    Incomplete No       _



 Use cursor and Enter for Activities or Containers (tab to Conts)

PF3=Return   7=Back   8=Forward
```

Figure 58. CBAM transaction: processes screen

The displayed fields mean:

Comp
The completion status of the process:

Abend
The program that implements the root activity abended.

Forced
The process was forced to complete—for example, it was canceled with a CANCEL ACQPROCESS command.

Incomplete
The process is incomplete.

Normal
The process completed normally.

Conts
A selectable field. If you place the cursor on this field and press ENTER, you get a list of the process-containers for the process named in the **Process** field. For an example of the CBAM *Containers* screen, see Figure 61 on page 97.

Mode
The mode of the process. One of:

- Active
- Cancelling
- Complete
- Dormant
- Initial.

For an explanation of each of these modes, see Processing modes.

Process
The name of a process.

Susp
Whether the process is currently suspended:

No
The process is not currently suspended.

Sus
The process is currently suspended.

If you place the cursor on the name of a process and press ENTER, you get a list of the process's constituent activities—see Figure 59 on page 95.

Activities screen

The *Activities* screen lists the activity names, the activity mode, the completion status, and whether a process is suspended.

```
CBAM         Process MORT000000017                Processtype MORTLOANS

  Activity                                       Mode       Comp       Susp

  DFHROOT                                        Dormant    Incomplete No
    NEWMORT                                      Complete   Normal     No
    PAYMENT-RECEIVED                             Complete   Normal     No
    PAYMENT-OVERDUE                              Complete   Normal     No
    INTEREST-CHANGE                              Complete   Normal     No
    CAPITAL-REPAYMNT                             Dormant    Incomplete No
      Credit-Account                             Complete   Normal     No
      Adjust-Interest                            Active     Incomplete No



 Use cursor and Enter for details

PF3=Return   7=Back   8=Forward
```

Figure 59. CBAM transaction: activities screen

The displayed fields mean:

Activity
The name of an activity.

The list of constituent activities is indented. The amount by which an activity is indented represents its level in the process's activity tree.

Comp
The completion status of the activity:

Abend
The activity abended.

Forced
The activity was forced to complete—for example, it was canceled with a CANCEL ACTIVITY command.

Incomplete
The activity is incomplete.

Normal
The activity completed normally.

Mode
The mode of the activity. One of:

- Active
- Cancelling
- Complete
- Dormant
- Initial.

For an explanation of each of these modes, see Processing modes.

Susp
Whether the activity is currently suspended:

No
The activity is not currently suspended.

Sus
The activity is currently suspended.

If you place the cursor on the name of an activity and press ENTER, you get details of the activity—see Figure 60 on page 97.

Activity details screen

The *Activity details* screen lists the name of the program that implements the selected activity, the transaction identifier, and the user ID under which the activity runs.

```
 CBAM         Process MORT000000017                  Processtype MORTLOANS

   Activity DFHROOT

   Program    MORTGAGE
   Transid    MORT
   Userid     CBTSMOR


   Containers
   Events
   Timers



 Use cursor and Enter for Containers, Events or Timers

PF3=Return   7=Back   8=Forward
```

Figure 60. CBAM transaction: activity details screen

There are also three selectable fields:

Containers
Pressing ENTER on this field displays a list of the containers associated with the selected activity. For more information, see Figure 62 on page 98.

Events
Pressing ENTER on this field displays a list of the events in the event pool of the selected activity. For more information, see Figure 63 on page 98.

Timers
Pressing ENTER on this field displays a list of the timers defined to the selected activity. For more information, see Figure 64 on page 99.

Containers screen

The *Containers* screen lists each container associated with a specified process or activity and shows the length, in bytes, of the data contained in it.

```
 CBAM         Process MORT000000017                    Processtype MORTLOANS

   Container        Datalength

   ACCOUNT-NO               36
   BORROWER-INFO          1000



   PF3=Return   7=Back   8=Forward
```

Figure 61. CBAM transaction: containers screen for a process

```
CBAM         Process MORT000000017                      Processtype MORTLOANS

  Activity DFHROOT

  Container        Datalength

  STATUS                  500



 PF3=Return   7=Back   8=Forward
```

Figure 62. CBAM transaction: containers screen for an activity

Events screen

The *Events* screen lists the events in the specified event pool of the activity. The events listed are the events that are currently in the event pool. Events that have been deleted do not appear in the list.

```
CBAM         Process MORT000000017                      Processtype MORTLOANS

  Activity DFHROOT

  Event            Type      Fired  Composite       Timer

  ALL-TIMERS       Composite Yes    OR
  ANNUAL-STATMNT   Timer     Yes    ALL-TIMERS      ANNUAL-STATEMENT
  ANNUAL-ST-DONE   Activity  No
  CAPITAL-REPAYMNT Input     No
  CAP-REPT-DONE    Activity  No
  DFHINITIAL       System    Yes
  INTEREST-CHANGE  Input     No
  PAYMENT-OVRDUE   Timer     No     ALL-TIMERS      PAYMENT-OVERDUE
  PAYMENT-RECEIVED Input     No









PF3=Return   7=Back   8=Forward
```

Figure 63. CBAM transaction: events screen

The displayed fields mean:

Composite
If the event is a composite, the Boolean operator (AND or OR) applied to its predicate.

If the event is a subevent, the name of the composite event of which it forms part.

Event
The name of an event.

Fired
The fire status of the event.

This field shows the *current* fire status of the event, *not* whether the event has ever fired in the past. For example, the fire status of an atomic event that has fired and been retrieved (but not deleted) is shown as 'No', because the act of retrieving the event has reset its fire status to NOTFIRED.

No
Not fired

Yes
Fired

Timer
If the event is a timer event, the name of its associated timer.

Type
The type of the event:

Activity
Activity completion

Composite
Composite

Input
Input

System
System

Timer
Event associated with a timer

Timers screen

The *Timers* screen lists the timers that are currently defined to a specified activity and shows their status.

```
CBAM         Process MORT000000017                      Processtype MORTLOANS

  Activity DFHROOT

  Timer             Status    Event            Date       Time

  ANNUAL-STATMNT   Expired    ANNUAL-STATEMENT 12151998   235959
  PAYMENT-OVRDUE   Unexpired  PAYMENT-OVERDUE  06301999   235959



PF3=Return   7=Back   8=Forward
```

Figure 64. CBAM transaction: timers screen

The displayed fields mean:

Date
The expiry date of the timer, in the form **mmddyyyy**.

Event
The name of the event associated with the timer.

Status
The state of the timer:

Expired
The timer expired normally.

Forced
Expiry of the timer was forced with a FORCE TIMER command.

Unexpired
The timer has not yet expired.

Time
The expiry time of the timer, in the form **hhmmss**.

Timer
The name of a timer.

CEMT INQUIRE PROCESSTYPE

Retrieve information about a CICS business transaction services process-type.

In the CICS Explorer, the Process Types view provides a functional equivalent to this command.

Description

The **INQUIRE PROCESSTYPE** command returns information about the BTS PROCESSTYPE definitions installed on this CICS region. It shows the current state of audit logging for each displayed process-type.

The resource signature

You can use this command to display the resource signature fields. You can use these fields to manage resources by capturing details of when the resource was defined, installed, and last changed. For more information, see Auditing resources. The resource signature fields are CHANGEAGENT, CHANGEAGREL, CHANGETIME, CHANGEUSRID, DEFINESOURCE, DEFINETIME, INSTALLAGENT, INSTALLTIME, and INSTALLUSRID. See Summary of the resource signature field values for detailed information about the content of the resource signature fields.

Input

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

- Type `CEMT INQUIRE PROCESSTYPE` (or suitable abbreviations for the keywords). The resulting display lists the current status.
- Type `CEMT INQUIRE PROCESSTYPE` (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 proc en`, the resulting display shows the details of only those process-types that are enabled.

You can change various attributes in the following ways:

- Overtype your changes on the INQUIRE screen after tabbing to the appropriate field.
- Use the CEMT SET PROCESSTYPE command.

ALL
The default. Information about all process-types is displayed, unless you specify a selection of process-types to be queried.

(value)
The name (1 - 8 characters) of one or more PROCESSTYPE definitions installed in the process-type table (PTT).

Sample screen

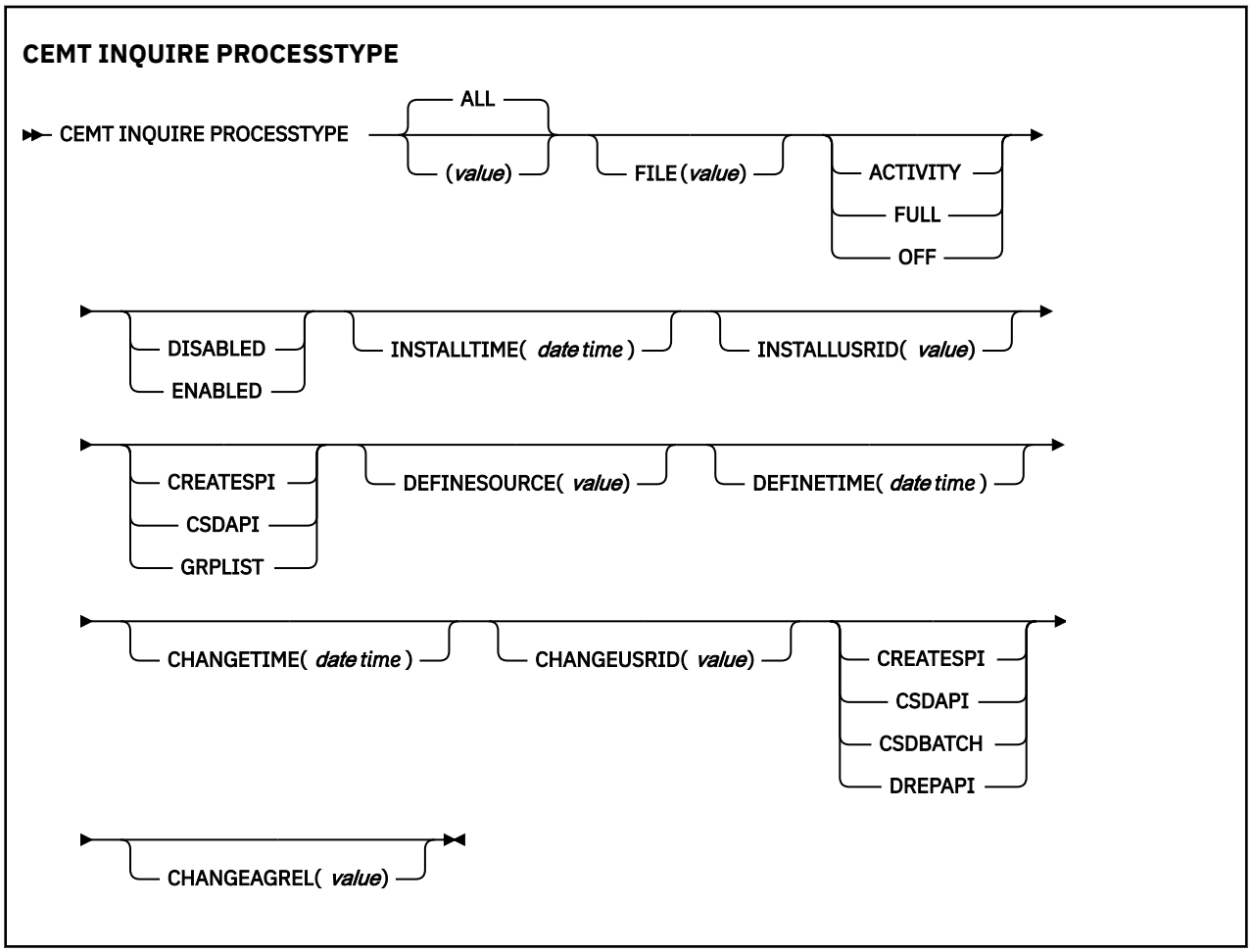
```
I PROC
STATUS:   RESULTS - OVERTYPE TO MODIFY
 Pro(PROCESSTYPE12  ) Fil(FILE12  ) Aud(ADTLOG12) Pro Ena
 Pro(PROCESSTYPE13  ) Fil(FILE13  ) Aud(ADTLOG12) Off Ena
 Pro(PTYPE2B        ) Fil(FILE2B  ) Aud(DFHJ2B  ) Ful Ena
 Pro(PTYPE39        ) Fil(FILE39  ) Aud(DFHJ39  ) Off Ena
 Pro(SALESTYPE1     ) Fil(SALESF1 ) Aud(PLOG51  ) Off Dis
 Pro(SALESTYPE4     ) Fil(SALESF4 ) Aud(PLOG51  ) Act Ena
 Pro(SALESTYPE6     ) Fil(SALESF6 ) Aud(PLOG51  ) Off Ena
```

Figure 65. CEMT INQUIRE PROCESSTYPE screen

If you place the cursor against a specific entry in the list and press ENTER, CICS displays an expanded format as shown in Figure 66 on page 101.

```
I PROC
STATUS:   RESULTS - OVERTYPE TO MODIFY
  Processtype(SALESTYPE4     )
  File(SALESF4 )
  Enablestatus( Enabled )
  Auditlog(PLOG51  )
  Auditlevel(Activity )
```

Figure 66. The expanded display of an individual entry



Displayed fields

AUDITLEVEL
Displays the level of audit logging currently active for processes of this type. The values are as follows:

ACTIVITY
Activity-level auditing. Audit records are written from:

1. The process audit points
2. The activity primary audit points.

FULL
Full auditing. Audit records are written from:

1. The process audit points
2. The activity primary *and* secondary audit points.

OFF
No audit trail records are written.

PROCESS
Process-level auditing. Audit records are written from the process audit points only.

For details of the records that are written from the process, activity primary, and activity secondary audit points, see Specifying the level of audit logging.

AUDITLOG*(value)*
Displays the 8-character name of the CICS journal used as the audit log for processes of this type.

CHANGEAGENT(*value*)
Displays a value that identifies the agent that made the last change to the resource definition. You cannot use CEMT to filter on some of these values because they are duplicated. The possible values are as follows:

CREATESPI
The resource definition was last changed by an **EXEC CICS CREATE** command.

CSDAPI
The resource definition was last changed by a CEDA transaction or the programmable interface to DFHEDAP.

CSDBATCH
The resource definition was last changed by a DFHCSDUP job.

DREPAPI
The resource definition was last changed by a CICSPlex SM BAS API command.

CHANGEAGREL(*value*)
Displays the 4-digit number of the CICS release that was running when the resource definition was last changed.

CHANGETIME(*date time*)
Displays the date and time when the resource definition was last changed. The format of the date depends on the value that you selected for the DATFORM system initialization parameter for your CICS region. The format of the time is hh:mm:ss.

CHANGEUSRID(*value*)
Displays the 8-character user ID that ran the change agent.

DEFINESOURCE(*value*)
Displays the source of the resource definition. The DEFINESOURCE value depends on the CHANGEAGENT option. For details, see Summary of the resource signature field values.

DEFINETIME(*date time*)
Displays the date and time when the resource was created. The format of the date depends on the value that you selected for the DATFORM system initialization parameter for your CICS region. The format of the time is hh:mm:ss.

ENABLESTATUS
Displays whether new processes of this type can be created. The values are as follows:

DISABLED
The installed definition of the process-type is disabled. New processes of this type cannot be defined.

ENABLED
The installed definition of the process-type is enabled. New processes of this type can be defined.

FILE*(value)*
Displays the 8-character name of the CICS repository file on which the process and activity records for processes of this type are stored.

INSTALLAGENT(*value*)
Displays a value that identifies the agent that installed the resource. You cannot use CEMT to filter on some of these values because they are duplicated. The possible values are as follows:

CREATESPI
The resource was installed by an **EXEC CICS CREATE** command.

CSDAPI
The resource was installed by a CEDA transaction or the programmable interface to DFHEDAP.

GRPLIST
The resource was installed by **GRPLIST INSTALL**.

INSTALLTIME(*date time*)
Displays the date and time when the resource was installed. The format of the date depends on the value that you selected for the DATFORM system initialization parameter for your CICS region. The format of the time is hh:mm:ss.

INSTALLUSRID(*value*)
Displays the 8-character user ID that installed the resource.

PROCESSTYPE*(value)*
Indicates that this panel relates to a PROCESSTYPE inquiry and displays the 8-character name of a process-type.

CEMT INQUIRE TASK

Retrieve information about a user task.

In the CICS Explorer, the Tasks view provides a functional equivalent to this command.

Description

INQUIRE TASK returns information about user tasks. Only information about user tasks can be displayed or changed; information about CICS-generated system tasks or subtasks cannot be displayed or changed. System tasks are those tasks started and used internally by CICS, and not as a result of a user transaction.

Input

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

- Type `CEMT INQUIRE TASK` (or suitable abbreviations for the keywords). The resulting display lists the status of every task.
- Type `CEMT INQUIRE TASK` (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 INQ TA TE`, the resulting display shows the details of only those tasks that were started from a terminal.

You can change various attributes in the following ways:

- Overtype your changes on the INQUIRE screen after tabbing to the appropriate field (see Overtyping a display).
- Use the **CEMT SET TASK** command.

(*value*)
The CICS-generated task number, in the range 1 - 99999.

ALL
The default value. The maximum number of tasks displayed is 32000.

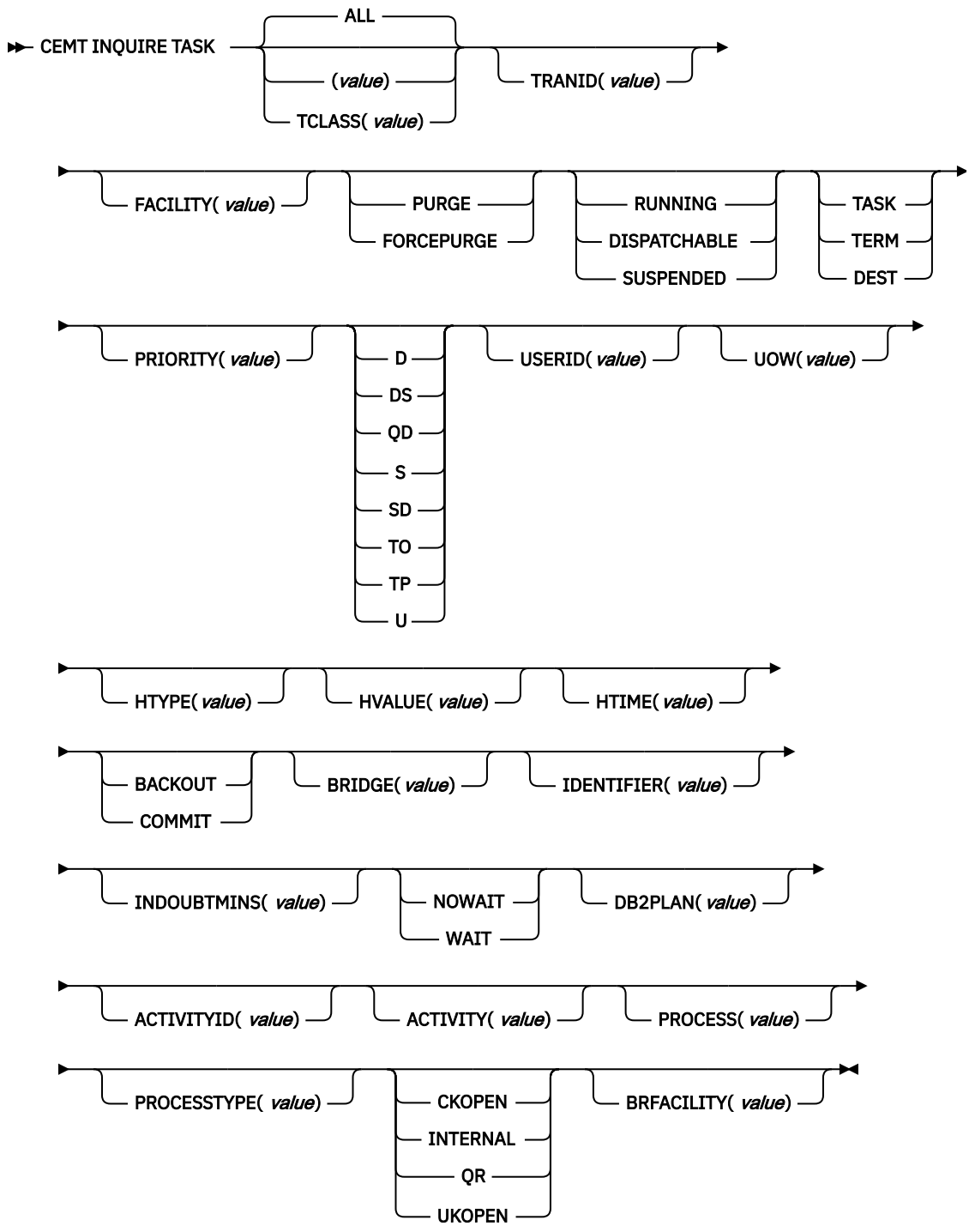
TCLASS (*value*)
The 8-character transaction class name to which the transaction belongs. The maximum number of tasks displayed is 32000.

You cannot specify a list of identifiers, or use the asterisk (*) or plus (+) symbols to specify a family of tasks.

When a value does not apply, or is negative (the value begins with No), the fields on the screen are blank. To modify these fields, locate them by tabbing (they appear in the same sequence as in the expanded format), and overtype with input valid for that field. You might find it more convenient to use the expanded format when setting one of these values.

If you place the cursor against a specific entry in the list and press ENTER, CICS displays an expanded format.

CEMT INQUIRE TASK



Displayed fields

ACTIVITY *(value)*
Displays the 16-character, user-assigned, name of the CICS business transaction services activity that this task is executing on behalf of.

ACTIVITYID *(value)*
Displays the 52-character, CICS-assigned, identifier of the CICS business transaction services activity that this task is executing on behalf of.

BRFACILITY(*value*)
Displays the 8-byte facility token representing the virtual terminal used by the current task if it is used by the 3270 bridge mechanism. If the task is not currently running in the 3270 bridge environment, zeros are returned.

BRIDGE (*value*)
Displays the 4-character name of the bridge monitor transaction if the current task is running in a 3270 bridge environment, and was started by a bridge monitor transaction with a **START BREXIT TRANSID** command. Otherwise, blanks are returned.

DB2PLAN (*value*)
Displays the 1- to 8-character name of the DB2® plan being used by this task, or blanks if no DB2 plan is being used.

FACILITY (*value*)
Displays a 4-character string identifying the name of the terminal or queue that initiated the task. If no FACILITY value is displayed, the task was started without a facility.

FTYPE
Displays the type of facility that initiated this task. The values are as follows:

TASK
The task was initiated from another task.

TERM
The task was initiated from a terminal.

DEST
The task was initiated by a destination trigger level as defined in the TDQUEUE resource definition.

HTIME (*value*)
Displays the time (in seconds) that the task has been in the current suspended state.

HTYPE (*value*)
Displays the reason why the task is suspended. A null value indicates that there is no hold-up, except for the necessity of reaching the head of the queue.

HVALUE (*value*)
Displays a 16-character resource name, such as a file name, or a value such as a TCLASS value.

For information about the values that CICS can return in the HTYPE and HVALUE options, see the resource type and resource name details in The resources on which CICS system tasks can wait in Troubleshooting.

INDOUBTMINS (*value*)
Displays the length of time, in minutes, after a failure during the indoubt period, before the task is to take the action returned in the INDOUBT attribute. The returned value is valid only if the unit of work is indoubt and the value of the INDOUBTWAIT attribute returns WAIT.

INDOUBT (*value*)
Displays the action (based on the ACTION attribute of the TRANSACTION resource definition) to be taken if the CICS region fails or loses connectivity with its coordinator while a unit of work is in the indoubt period.

The action is dependent on the values returned in the INDOUBTWAIT and INDOUBTMINS attributes; if INDOUBTWAIT returns WAIT, the action is not taken until the time returned in INDOUBTWAIT expires.

The values are as follows:

BACKOUT
All changes made to recoverable resources are to be backed out.

COMMIT
All changes made to recoverable resources are to be committed, and the unit of work marked as completed.

INDOUBTWAIT (*value*)
Displays how (based on the WAIT attribute of the TRANSACTION definition) a unit of work (UOW) is to respond if a failure occurs while it is in an indoubt state. The values are as follows:

NOWAIT
The UOW is not to wait, pending recovery from the failure. CICS is to take immediately whatever action is specified on the ACTION attribute of the TRANSACTION definition.

WAIT
The UOW is to wait, pending recovery from the failure, to determine whether recoverable resources are to be backed out or committed.

For further information about the meaning of the ACTION and WAIT attributes of the TRANSACTION definition, see TRANSACTION definition attributes.

IDENTIFIER (*value*)
Displays a 48-character field that contains user data provided by the bridge exit, if the task was initiated in the 3270 bridge environment, or blanks, otherwise. This field is intended to assist in online problem resolution.

For example, this field might contain the WebSphere® MQ correlator for the CICS-MQ bridge, or a web token.

PRIORITY (*value*)
Displays the priority of the task in the range 0 - 255, where 255 is the highest priority.

Note: You can reset this value by typing over it with a different value.

PROCESS *(value)*
Displays the 36-character name of the CICS business transaction services process that this task is executing on behalf of.

PROCESSTYPE *(value)*
Displays the 8-character process-type of the CICS business transaction services process that this task is executing on behalf of.

PURGETYPE (*input only field*)
Specifies whether a task is to be purged or forced to purge. The values are as follows:

PURGE
The task is to be terminated. Termination occurs only when system and data integrity can be maintained.

FORCEPURGE
The task is to be terminated immediately. System integrity is not guaranteed. In some extreme cases, for example if a task is forced to purge during backout processing, CICS terminates abnormally. If you want to terminate a task but do not want to terminate CICS, use PURGE instead of FORCEPURGE.

RUNSTATUS
Displays the status of this task. The values are as follows:

RUNNING
The task is running.

DISPATCHABLE
The task is dispatchable.

SUSPENDED
The task is suspended.

STARTCODE (*value*)
Displays how this task was started. The values are as follows:

D
A distributed program link (DPL) request. The program cannot issue I/O requests against its principal facility or any sync point requests.

DS
A distributed program link (DPL) request, as for code D, with the exception that the program can issue sync point requests.

QD
A transient data trigger level was reached.

S
Start command (no data)

SD
Start command (with data)

TO
The operator typed a transaction code at the terminal.

TP
The transaction was started by presetting the transaction ID for the terminal.

U
User-attached task.

TASK (*value*)
Indicates that this panel relates to a TASK inquiry and displays a CICS-generated task number in the range 1–99999.

TCB (*value*)
Displays the type of TCB under which the task is running. The values are as follows:

CKOPEN
The task is running under a CICS key open TCB.

INTERNAL
The task is running under one of the CICS internal TCBs. An internal TCB can be one of the following:

- The concurrent mode (CO) TCB
- The file-owning mode (FO) TCB
- The resource-owning mode (RO) TCB
- The ONC/RPC mode (RP) TCB
- The sockets listener mode (SL) TCB
- The secure sockets layer mode (SO) TCB
- A sockets mode (S8) TCB
- The FEPI mode (SZ) TCB

QR
The task is running under the CICS QR TCB.

UKOPEN
The task is running under a user key open TCB.

TRANID (*value*)
Displays a 4-character string identifying the transaction name associated with the task.

UOW (*value*)
Displays the 16-character local identifier of the unit of work that is associated with this task.

USERID (*value*)
Displays the user that is currently associated with the task.

CEMT SET PROCESSTYPE

Change the attributes of a CICS business transaction services process-type.

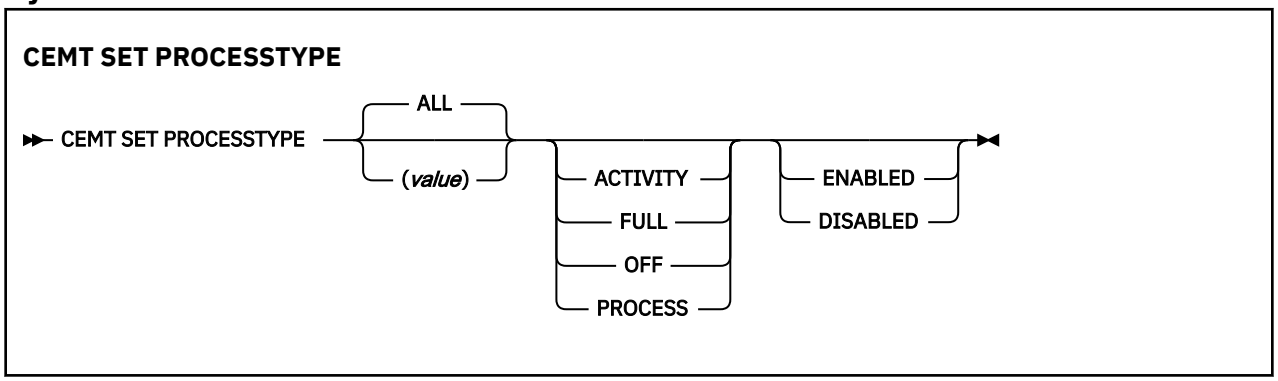
In the CICS Explorer, the Process Types view provides a functional equivalent to this command.

Description

SET PROCESSTYPE enables you to change the current state of audit logging and the enablement status of BTS PROCESSTYPE definitions installed on this CICS region.

Note: Process-types are defined in the process-type table (PTT). CICS uses the entries in this table to maintain its records of processes (and their constituent activities) on external data sets. If you are using BTS in a single CICS region, you can freely use the SET PROCESSTYPE command to modify your process-types. However, if you are using BTS in a CICSplex, it is strongly recommended that you use CICSPlex SM to make such changes. This is because it is essential to keep resource definitions in step with each other, across the CICSplex.

Syntax



Options

ACTIVITY|FULL|OFF|PROCESS
specifies the level of audit logging to be applied to processes of this type.

Note: If the AUDITLOG attribute of the installed PROCESSTYPE definition is not set to the name of a CICS journal, an error is returned if you try to specify any value other than 0FF.

The values are:

ACTIVITY
Activity-level auditing. Audit records will be written from:

1. The process audit points
2. The activity primary audit points.

FULL
Full auditing. Audit records will be written from:

1. The process audit points
2. The activity primary *and* secondary audit points.

OFF
No audit trail records will be written.

PROCESS
Process-level auditing. Audit records will be written from the process audit points only.

For details of the records that are written from the process, activity primary, and activity secondary audit points, see Specifying the level of audit logging.

ALL
specifies that any changes you specify are made to all process-types that you are authorized to access.

ENABLED|DISABLED
specifies whether new processes of this type can be created. The values are:

DISABLED
The installed definition of the process-type is disabled. New processes of this type cannot be defined.

ENABLED
The installed definition of the process-type is enabled. New processes of this type can be defined.

PROCESSTYPE(value)
specifies the 8-character name of the process-type whose attributes are to be changed.

Chapter 5. Troubleshooting BTS

Try these solutions for some of the more common problems that you might encounter when running a BTS system.

Dealing with stuck processes

Consider the causes of a "stuck" process, when it cannot proceed because it is waiting for an event that cannot, or does not, occur. You can use timers or process timers to restart a stuck process.

There are several possible causes:

- A faulty application design - see “Application design errors” on page 111.
- A request to start an activity on a remote system is "unserviceable" - see “Dealing with unserviceable requests” on page 114.
- A CICS region fails - see “Dealing with CICS failures” on page 115.

Application design errors

A stuck process might be caused by a program logic error.

For example, consider the following scenarios:

1. **Outstanding user events:**
   a. One of the activities of the process returns from what it believes to be its final activation. It issues an EXEC CICS RETURN command without the ENDACTIVITY option.
   b. There are no events on the reattachment queue of the activity, but there is a user event in its event pool.
   c. There is no means for the event to be fired. Perhaps it is an input event which has fired, caused reattachment, and been retrieved, but which the activity has neglected to delete.

   In this case, the activity becomes dormant, and there is no way for it to reactivated. The process is stuck.

   The recommended way to prevent this scenario is to add the ENDACTIVITY option to the EXEC CICS RETURN command that ends the final activation of the activity. Coding RETURN ENDACTIVITY deletes any outstanding events—other than activity completion events for child activities, which the activity must deal with properly—and allows the activity to complete normally.
2. **Waiting for an external interaction:**

   A user-related activity returns from its initial activation and becomes dormant, waiting for an external interaction to occur. (User-related activities are described in Activity processing.) However, the expected user input does not happen. Perhaps the clerk is sick, or the data required is not available. The process is stuck.

   You can recover from this scenario by setting a timer which, if the expected external interaction does not occur within a specified period, causes the activity (or its parent) to be reactivated anyway.
3. **Timer error:**

   A programming error results in a timer being set to expire in five days rather than 5 minutes. The process is stuck. See “Restarting stuck processes” on page 112.

   Note: To force a timer to expire before its specified time, use the FORCE TIMER command.

Restarting stuck processes

You can use activity timers or process timers to restart a stuck process. You can use status containers to identify stuck activities. You can use a utility program to help you determine the status of a process.

About this task

For advice on restarting processes that are stuck because of unserviceable requests, see “Dealing with unserviceable requests” on page 114.

For advice on restarting processes that are stuck because of a CICS failure, see “Dealing with CICS failures” on page 115.

Using activity timers

Use activity timers to restart processes that are stuck because of application errors or other reasons. For example, a parent might set a timer that causes it to be reactivated after a specified period, if a particular child activity does not complete.

About this task

The best way to restart processes that are stuck for other reasons - including application errors - is to use timers. The parent names the timer in a way that associates it with a particular child. If the child completes within the specified period, the parent deletes the timer.

One reason for making the application responsible for restarting itself is that it is difficult from *outside* a process to tell whether the process is stuck or merely dormant, particularly if the process is long-lived. Processes of different types can have varying "natural" lifespans; and these lifespans can vary according to system load, availability of remote regions, and so on. The application itself is best placed to know how long each of its activities should run before they can be assumed to be stuck.

You probably do not want to set timers for all your activities. For example, you might think it unnecessary to set a timer for a simple activity that completes its processing in one activation, has no children, and is to be run synchronously. However, you might want to set a timer for an activity to which one or more of the following apply:

- It is to be run asynchronously.
- It requires multiple activations to complete its processing.
- It is long-lived.
- It involves external interaction—for example, user input.

Using process timers

Use process timers to restart a stuck process. You can set timers for individual child activities, for the process, or for both. That is, the root activity can set a timer with an expiry time after the whole process might be expected to complete.

About this task

If the process is short-lived, you might decide not to set any activity timers, but to set a process timer instead.

If the process is long-lived, do not set a process timer without also setting timers for at least some individual activities. Setting timers prevents the possibility of a delay in restarting the process. For example, if a process that is expected to last six months becomes stuck after one day while processing its first activity, and you have set only a process timer, the process could lie dormant for, say, seven months before the root activity is reactivated to deal with the problem.

If the root activity is activated by the process timer, it could, for example:

1. Browse and inquire on each of its descendant activities, checking completion status and mode. For examples of the use of the BTS browsing and inquiry commands, see Browsing examples.
2. If it succeeds in identifying the stuck activity, issue a CANCEL command to cancel it. If the stuck activity is not a child but a lower-level descendant of the root activity, the root must first acquire the stuck activity.
3. The completion event of the stuck activity fires, causing the parent activity to be reactivated. The CHECK ACTIVITY command issued by the parent returns a completion status of FORCED. The parent should be coded to handle the abnormal completion of one of its children. The process is no longer stuck.

Using status containers

Use status containers to make it easier for a root activity to identify which of its descendant activities are stuck.

About this task

Status containers are data-containers that contain information about what an activity is currently doing. Whereas you can use an INQUIRE ACTIVITYID command to discover the mode and completion status of an activity, the information in a status container is likely to be at a more detailed level. For example, each activity in a process might have a data-container called, perhaps, STATUS, which it regularly updates—perhaps at the beginning and end of each activation, and each time it starts new work. A status container might, for instance, contain the date and time, and a string describing the work that the activity has started or ended, or the fact that it is dormant because it is waiting for the completion of a particular child activity.

You can think of an activity as a finite state machine—it is always in one of a limited number of processing states. The "processing states" we refer to here are application-dependent and distinct from the BTS-defined *modes* of an activity. Each activity could regularly update its status container with its current processing state.

Using a utility program

You can use a utility program to help you determine whether a process is stuck or just dormant. CICS supplies the audit trail utility, DFHATUP, and the repository utility, DFHBARUP, or you can write your own.

CICS-supplied utility programs

CICS supplies two utility programs for diagnostic purposes. The audit trail utility, DFHATUP, and the repository utility, DFHBARUP.

The two utility programs for diagnostic purposes are detailed here:

The audit trail utility, DFHATUP
: You can use DFHATUP to print selected audit records from a log stream. If you use auditing to track the progress of your processes across the sysplex, to investigate a stuck process you could print its audit records.

  DFHATUP is described in “Creating a BTS audit trail” on page 116.

The repository utility, DFHBARUP
: You can use DFHBARUP to print selected records from a repository. To investigate a stuck process, you could print its repository records.

  DFHBARUP is described in “Examining BTS repository records” on page 133.

User-written utility programs

You can write a utility program that can check for and restart stuck processes, the utility program is most effective when your activities use status containers.

Your utility program could, for example:

1. Browse all processes of a specified process-type.
2. Browse the descendant activities of each process returned in step 1.
3. Inquire on the status data-container of each activity, and retrieve its contents.
4. Identify a stuck activity from the contents of its status container.
5. Issue an ACQUIRE command to acquire the stuck activity.
6. Issue a CANCEL command to cancel the stuck activity. The completion event for the stuck activity fires, causing its parent to be reactivated. The CHECK ACTIVITY command issued by the parent returns a completion status of FORCED. Ensure that the parent is coded to handle the abnormal completion of one of its children. The process is no longer stuck.

Dealing with activity abends

You can deal with activity abends by coding your application to try the failed activity again or compensate the siblings of the failed activity.

About this task

Compensation is the act of modifying, or compensating for, the effects of a completed activity. For example, compensation can reverse or modify actions taken by preceding activities or terminate business transactions. If a program that implements an activity abends, the parent of the activity receives control. (If the failed activity was run asynchronously, the parent is reactivated.) The **CHECK ACTIVITY** command issued by the parent returns a COMPSTATUS of ABEND - see Dealing with BTS errors and response codes.

Your application should be coded to deal with an activity abend. The parent of the failed activity might, for example, choose to do either of the following:

- Try the failed activity again - see Using a CICS distributed routing program
- Compensate the siblings of the failed activity - see Compensation in BTS.

Dealing with unserviceable requests

An *unserviceable request* is a request that cannot currently be satisfied; for example, an activity that is not available, or the region on which the request is to run is not accessible.

Unserviceable routing requests

If you operate BTS in a sysplex, you can route processes and activities across a set of CICS regions called a *BTS-set*. When a process or activity is started by a RUN ASYNCHRONOUS command, it can be routed either statically or dynamically. Mostly, you might choose dynamic rather than static routing.

For detailed information about routing processes and activities, see Administering BTS.

Static routing

Using static routing, you name the target region to which the activity is to be routed on the REMOTESYSTEM option of the installed transaction definition, for the transaction associated with the activity. If the target region is unavailable at the time the activity is to be started, CICS treats the request as unserviceable.

Dynamic routing

Using dynamic routing, the target region is chosen by your routing program, the distributed routing program or the CICSPlexÂ SM routing program. If the target region that it returns is unavailable, the routing program is invoked again and can select a different target. Alternatively, it can (by setting a non-zero return code) indicate that the request is to be treated as unserviceable.

For definitive information about writing a distributed routing program, see Writing a distributed routing program.

Why classify requests as unserviceable?

Why should your routing program classify requests as "unserviceable"? Why should it not reroute the request to an alternative region, assuming that alternatives are available?

Sometimes, due to a transaction affinity, it might be essential that an activation runs on a specific region, and no other. If so, rather than selecting an alternative target region, your routing program can return the same target (even though it is currently unavailable), and classify the request as unserviceable.

How CICS handles unserviceable requests

CICS identifies unserviceable requests by issuing a DFHSH message and tries the request again every minute.

When a request is "unserviceable", CICS:

1. Issues message DFHSH0105, which identifies the request and indicates that it cannot be serviced.
2. Tries the request again every minute. If the request is successfully serviced, CICS issues message DFHSH0108.
3. Each hour, if the request still cannot be serviced, issues message DFHSH0106. This message indicates the time remaining before CICS purges the request, if it has not been serviced in the meantime.
4. After 24 hours, if the request still cannot be serviced, stops trying to service it and issues message DFHSH0107. The request is discarded.

Resolving unserviceable requests

In many cases, CICS resolves unserviceable requests automatically. If, for example, an unavailable target region becomes available within 24 hours of the request being issued, CICS routes the request correctly.

You should watch for occurrences of DFHSH0105 and DFHSH0106 messages. You should investigate why the request is unserviceable, and take any necessary corrective action. It might be, for example, that a resource required to satisfy the request (an activity or process) is inaccessible; or that a remote region, or a link to it, is unavailable.

Dealing with CICS failures

If one of your CICS regions fails, all BTS processes on the failing region are halted. Processes on other regions might also become stuck, because expected events are not generated. If a CICS region fails, you must perform an emergency restart.

Only in rare circumstances - for example, if the CICS global catalog or system log is corrupted - might it be necessary to perform an initial or cold start after a failure. If it *is* necessary, perform a cold start in preference to an initial start.

Emergency starts

During an emergency restart, CICS automatically restores BTS processes to the state they were in before the failure. Any activities that were active at the time of the failure are rerun. That is, if an activation (transaction) was running, it is backed out and restarted. The activity is sent the same reattachment event that caused the failed activation. Its data-containers contain the same data they held at the start of the failed activation.

Initial and cold starts

During an initial or cold start:

- BTS repository data sets are unchanged.

  Note: Repository data sets are never reinitialized at CICS startup, because they might be shared.
- The local request queue data set is unchanged. All information about BTS timers, pending, and unserviceable requests is preserved. However, it is likely that some of this information is now irrelevant or invalid, because it refers to processes that no longer exist.

Creating a BTS audit trail

You can create an audit trail for BTS processes and activities to track the progress of transactions. You can control the amount of audit logging. Follow the example to see a series of activations.

About this task

It contains:

- "Introduction to BTS audit trails" on page 116
- "Specifying the level of audit logging" on page 117
- "Audit trail constraints when using DASD-only log streams" on page 119
- "Audit trail examples" on page 120
- "Using the audit trail utility program, DFHATUP" on page 123.

Introduction to BTS audit trails

You can create an audit trail for the BTS processes and activities that run in your CICS systems. You can use the audit trail to track the progress of complex business transactions and to diagnose problems in programs that are being developed to form a new business application.

The CICS code contains BTS audit points in much the same way as it contains trace points. However, there are three main differences between audit records and trace entries:

1. Trace entries are written to an internal trace table within the CICS address space. In contrast, the audit trail of a process is written to a CICS journal, which resides on an MVS log stream.
2. Trace entries record the progress of tasks over a relatively short period, typically seconds, minutes, or hours. In contrast, the audit trail of a process can extend to days, weeks, or even months.
3. Trace entries relate to activity in a single CICS region. In contrast, in a sysplex the execution of different parts of a process might take place on different regions within the sysplex. Therefore, each audit record contains system, date, and time information. Typically, an audit record for a BTS activity also contains:

   - The identifier of the activity
   - The process to which the activity belongs
   - Information about the event which caused the activity to be invoked, canceled, suspended, or resumed; or that fired when it completed.

   Because log streams can be shared by more than one region, it is possible to write audit records from different regions to the same log.

There are four, incremental, auditing levels:

1. None
2. Process-level
3. Activity-level
4. Full.

How to specify the levels, and what they mean, is described in "Specifying the level of audit logging" on page 117.

Audit log records are written to an MVS log stream by the CICS Log Manager. You can read the records offline using the CICS audit trail utility program, DFHATUP. DFHATUP allows you to:

- Filter records for specific process-types, processes, and activities
- Interpret records into a readable format.

You can use the CICS journal utility program, DFHJUP, to copy the audit log stream to a backup file and to delete the log stream. By editing the JCL used to run DFHATUP, you can make DFHATUP accept the backup file as input.

Audit records are buffered; they are written to the log stream only when the buffer is full or a sync point occurs. This means that, when multiple CICS regions share the same log stream, audit records might not be in exact date and time order.

Specifying the level of audit logging

You can control the amount of audit logging that CICS performs for each process, using the AUDITLOG and AUDITLEVEL attributes of the PROCESSTYPE definition.

For detailed information about defining process-types, see CEDA DEFINE PROCESSTYPE. However, note the following considerations:

- When a process is first defined, BTS obtains the audit level and audit log information for the process from the installed PROCESSTYPE definition, and copies it into the process record. During the lifetime of the process, this copy of the audit information is used to determine auditing. If the auditing information is changed by, for example, a CEMT SET PROCESSTYPE command, this does not affect existing processes.
- If an installed PROCESSTYPE definition does not specify a CICS journal name in its AUDITLOG field, CICS does not do any audit logging for processes and activities of that type until the definition is replaced with one that does contain the name of an audit log.
- The AUDITLOG field must not specify the SMF data set.
- Several process-types can share the same audit log.
- In a sysplex, different parts of a process might run on different CICS regions. If you want to write audit records for all the parts, you must ensure that all the regions have the same audit log information in their installed PROCESSTYPE definitions. However, see “Audit trail constraints when using DASD-only log streams” on page 119.

The AUDITLEVEL option of the PROCESSTYPE definition allows you to specify one of four logging levels for processes of the defined type:

ACTIVITY
Specifies activity-level auditing. Audit records are written from:

1. The process audit points
2. The activity primary audit points.

That is, an audit record is written:

1. Whenever a process of this type:
   - Is defined
   - Is requested to run
   - Is requested to link
   - Is acquired
   - Completes
   - Is reset
   - Is canceled
   - Is suspended
   - Is resumed

2. Each time data is placed in a process container belonging to a process of this type—that is, each time a PUT CONTAINER PROCESS or PUT CONTAINER ACQPROCESS command is issued against a process of this type
3. Each time a process container belonging to a process of this type is deleted
4. Each time a root activity (DFHROOT) of this type of process is activated.
5. Every time a non-root activity belonging to a process of this type:
   - Is requested to link
   - Is activated
   - Completes.

FULL
Specifies full auditing. Audit records are written from:

1. The process audit points
2. The activity primary *and* secondary audit points.

That is, an audit record is written:

1. Whenever a process of this type:
   - Is defined
   - Is requested to run
   - Is requested to link
   - Is acquired
   - Completes
   - Is reset
   - Is canceled
   - Is suspended
   - Is resumed

   Each time data is placed in a process container belonging to a process of this type

   Each time a process container belonging to a process of this type is deleted

   Each time a root activity (DFHROOT) of this type of process is activated
2. Every time a non-root activity belonging to a process of this type:
   - Is defined
   - Is requested to run
   - Is requested to link
   - Is activated
   - Completes
   - Is acquired
   - Is reset
   - Is canceled
   - Is suspended
   - Is resumed
   - Is deleted.

OFF
Specifies that no audit trail records are written. This is the default value.

PROCESS
Specifies process-level auditing. Audit records are written from the process audit points only. That is, an audit record is written whenever a process of this type:

- Is defined
- Is requested to run
- Is requested to link
- Is acquired
- Completes
- Is reset
- Is canceled
- Is suspended
- Is resumed

Each time data is placed in a process container belonging to a process of this type

Each time a process container belonging to a process of this type is deleted

Each time a root activity (DFHROOT) of this type of process is activated

Note: If you specify any value for AUDITLEVEL other than OFF, you must also specify the AUDITLOG option of the PROCESSTYPE definition.

You must choose a level of auditing that suits your needs. The more records that are written to the audit log, the longer your business transaction takes to run. The fewer records written, the less information there is for auditing or diagnostic purposes.

To reset the AUDITLEVEL attribute of an installed PROCESSTYPE definition, use the CEMT SET PROCESSTYPE command. Changes are preserved across a restart of CICS. Changes to an installed PROCESSTYPE definition do not affect *existing* processes.

If a request to write an audit record fails:

- CICS issues an error message.
- Auditing for processes of this process-type is suspended until the audit error is corrected and a CEMT SET JOURNALNAME(*journal*) ACTION(RESET) command is issued. If the reset completes successfully, auditing is resumed and a CICS message is issued to this effect. Some audit records are lost.

Audit trail constraints when using DASD-only log streams

If you are running BTS in a sysplex, the activities that make up a process might run on different CICS regions. If you want to use audit logging, you must ensure that audit records can be written to a single log stream from any region on which any of the activities run.

About this task

If the CICS regions are in the same MVS image, you can define the log stream to use either a coupling facility structure or DASD-only logging. However, if the CICS regions are on *different* MVS images, the log stream must use a coupling facility structure rather than DASD-only logging. This is because CICS regions on different MVS images cannot access the same DASD-only log stream at the same time.

If the regions are in different MVS images and you use DASD-only logging, you are unable to use shared log streams for your BTS logs. This means that audit records for a single process might be split across several log streams; you must collate them yourself.

Audit trail examples

Follow the sequence of activations of a BTS process, SALES1234567890. The activities that make up the process run on two CICS regions.

For clarity, the example does not show the activations of any other processes that might also be running in these regions.

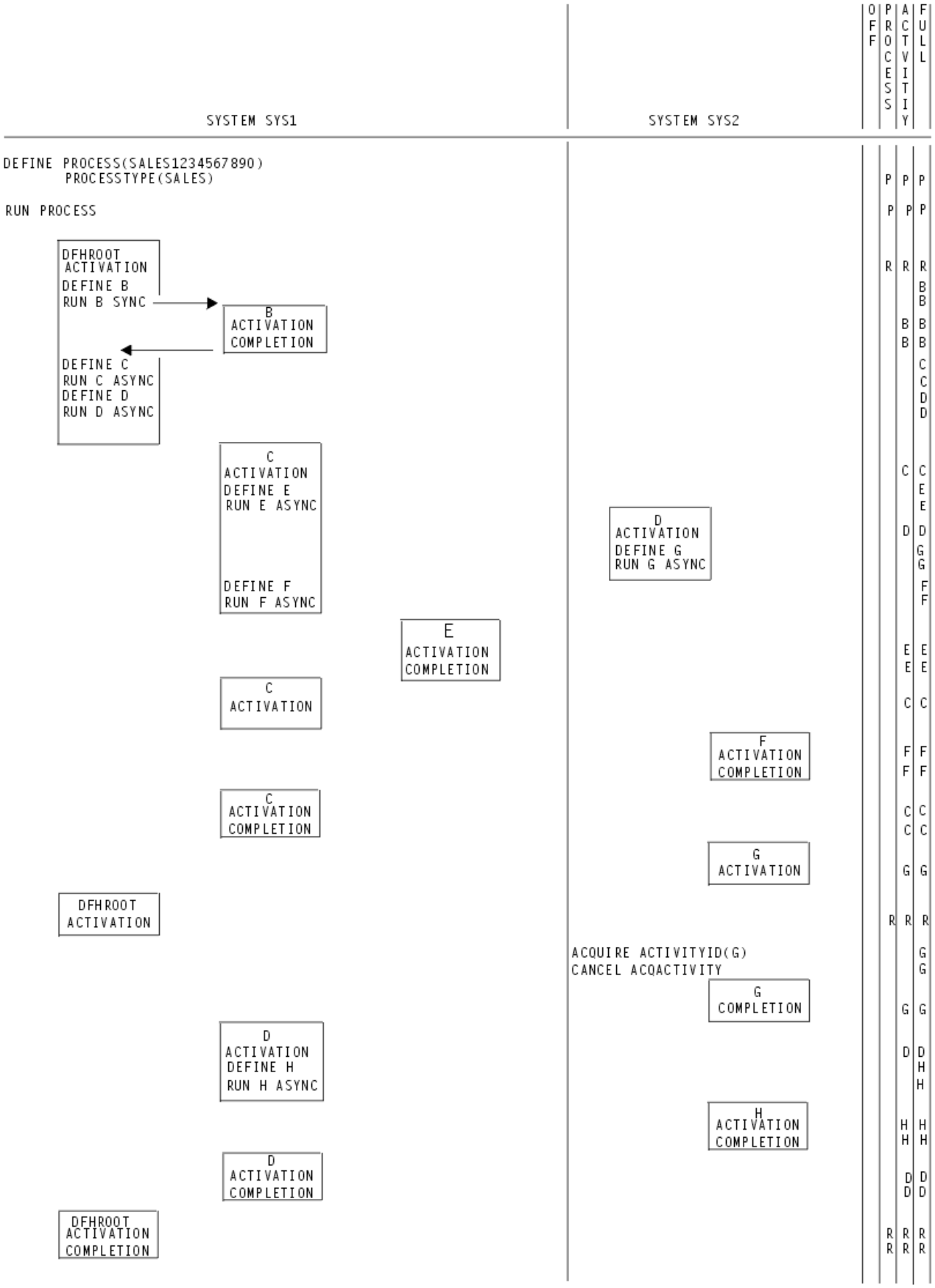


Figure 67. Example audit trails

In this example, an application running on region SYS1 defines a new process, SALES1234567890, and requests it to run. The root activity of the new process begins running on SYS1. It defines and runs an activity B, which executes synchronously. When control returns to the root activity, it defines activities C and D and schedules them to run asynchronously. After the root activity has returned, activity C starts on SYS1 and activity D starts on SYS2.

Activity C schedules child activities E and F to run asynchronously and returns. E and F run on different systems. When each of its child activities completes, C is reactivated and checks the completion status of the child. Lastly, C completes normally, which causes the root activity to be reactivated.

Activity D defines a child activity G and schedules it to run asynchronously. Later, another transaction issues ACQUIRE ACTIVITYID and CANCEL ACQACTIVITY commands against activity G. G completes in a FORCED state. D is reactivated and discovers what has happened to G with a CHECK ACTIVITY command. In response to G's failure, D defines a new activity H and requests it to run asynchronously. D then returns and H runs on the other region. When H completes normally, D is reactivated and completes normally. This causes the root activity to be reactivated. The root activity issues a CHECK ACTIVITY command to see how D completed, and then completes normally, ending the process.

Note: For conciseness, some commands that could result in audit records being written - for example, PUT CONTAINER ACQPROCESS and SUSPEND - are omitted from the example.

Process-level auditing

A setting of PROCESS on the AUDITLEVEL attribute of a PROCESSTYPE definition specifies process-level auditing for processes of the defined type. Records are written from the audit points for processes.

If process-level auditing is set for the process in the example, only six records are written to the audit log (see Figure 67 on page 121):

1. When the process is defined
2. When the process is requested to run
3. When the root activity of the process is activated for the first time
4. When the root activity of the process is activated for the second time
5. When the root activity of the process is activated for the third time
6. When the process completes.

Activity-level auditing

A setting of ACTIVITY on the AUDITLEVEL attribute of a PROCESSTYPE definition specifies activity-level auditing for processes of the defined type.

Records are written from:

- The audit points for processes
- The primary audit points for activities.

If activity-level auditing is set for the process in the example, the following records are written to the audit log:

- The six records described in “Process-level auditing” on page 122.
- Each time one of DFHROOTs descendant activities is activated.
- When each descendant activity completes. This includes the completion of activity G, which has a completion status of FORCED.

  Note: Records are *not* written when an activation ends in an incomplete state. Thus, in the example, a record is not issued when the root activity ends after defining activity D.

Full auditing

Set FULL on the AUDITLEVEL attribute of a PROCESSTYPE definition to write full audit information to the audit log. Note the effect on performance.

Records are written from:

- The audit points for processes
- The primary audit points for activities
- The secondary audit points for activities.

If full auditing is set for the process in the example, the following records are written to the audit log:

- All those written for activity-level auditing
- When each activity is defined
- When each activity is scheduled to run
- When activity G is acquired
- When activity G is canceled.

Note: Full auditing has an adverse effect on performance. It is intended to provide the maximum amount of information to help track down problems when applications are being developed. It is not intended to be used on production systems.

Using the audit trail utility program, DFHATUP

You can use the audit trail utility program, DFHATUP, to read BTS audit records from a log stream and to print them. The DFHATUP utility formats the records to make them easier to interpret. You can also filter selected records.

Using DFHATUP to read audit logs

You can use the DFHATUP batch utility to capture, format, and display records from your audit logs.

About this task

You must run DFHATUP as a batch job against a log stream that is not in use by any CICS regions. If you run it against a log stream that is connected to CICS, DFHATUP is unable to find any records that CICS has in its buffers.

DFHATUP reads the records in the order that they were written to the MVS log stream. By including control statements in the SYSIN data set, you can select the records that DFHATUP writes to the output data set, SYSPRINT. DFHATUP formats the selected records before writing them to SYSPRINT.

DFHATUP ignores any records that it does not recognize as BTS audit records.

Sample job stream to run the DFHATUP program

Follow this example job stream to see how you can use the DFHATUP utility program to process your audit log data.

Figure 68 on page 124 shows an example job stream to run the DFHATUP program. The job stream must include DD statements for the following data sets:

The audit log
The audit log data set to be examined to produce the output data. (Figure 68 on page 124 shows a DD name of 'AUDITLOG'.)

If you do not specify the BLKSIZE parameter its value defaults to 80, which causes audit records to be truncated.

STEPLIB
A partitioned data set (DSORG=PO) that contains the DFHATUP program module. If the module is in a library in the link list, this statement is not required.

SYSIN
The input control data set. This file must be in 80 - byte record format. The control statements that you can use in this data set are described in "SYSIN control statements" on page 125.

Control statements can be continued on to the next line by including any non-blank character in column 72. If the line that follows a continuation character is empty or contains control arguments which conflict with those control arguments that make up the preceding part of the control statement, an error is reported and execution of the utility ends. Any characters which occur beyond column 72 are ignored.

SYSPRINT
The output data set which the formatted audit records and control messages are sent to.

```
//*******************************************************************
//* RUN DFHATUP (AUDIT LOG UTILITY PROGRAM)
//*
//*
//*******************************************************************
//ATUP     EXEC PGM=DFHATUP,PARM='N(EN),P(30),T(M)'
//STEPLIB    DD DSN=CTS130.CICS530.SDFHLOAD,DISP=SHR
//*******************************************************************
//*   The output will go to SYSPRINT
//*******************************************************************
//SYSPRINT DD SYSOUT=A,DCB=RECFM=FBA
//AUDITLOG DD DSN=CICSAA#.CICSDC1.JRNL001,
//            SUBSYS=(LOGR,DFHLGCNV),
//            DCB=BLKSIZE=32760
//SYSIN    DD *
PTYPE(SALES)                                                    +
PROCESS(CUST_SALES_1999.13872977829728.QA)
ACTIVITY(activity-name)
PROCESS(CUST_SALES_1999.11103847635637.QB)                      +
PTYPE(SALES)
/*
//*
```

Figure 68. Sample job to run the DFHATUP utility program

EXEC parameters

You can use the PARM keyword on the EXEC statement to pass national language, pagesize, or translate case parameters to the DFHATUP utility.

The form of the EXEC statement is:

```
EXEC PGM=DFHATUP,PARM='parm1,...,parmn'
```

NATLANG({EN|*CS*v*KA*})
The language in which messages are to be issued.

The minimum abbreviation of this parameter is **N**. The possible values are:

CS
Traditional Chinese

EN
English. This is the default.

KA
Kanji.

PAGESIZE({60|*nn*})
The number of lines to be printed per page, when the output from the utility is sent to a printer. Valid values are in the range 20–99. The default is 60.

The minimum abbreviation of this parameter is **P**.

TRANSLATE({MIXEDCASE|*UPPERCASE*})
Whether the output from the utility is to be in mixed-case or uppercase. The default is mixed-case.

The minimum abbreviation of this parameter is **T**. The minimum abbreviations of MIXEDCASE and UPPERCASE are M and U respectively.

SYSIN control statements

You can use the SYSIN data set to pass information to the audit trail utility program, DFHATUP. For example, you can include statements to select specific sets of records to be formatted.

Comments are identified by an asterisk (*) in the first position, anything entered on the SYSIN card after the asterisk is ignored by DFHATUP. The SYSIN data set must be defined.

Format of the SYSIN control statements

Use the SYSIN control statements to specify the name of an activity, the audit data log, the BTS process, and the process type.

```
SYSIN DD *
[AUDITLOG(name)]
[PTYPE(name) <PROCESS(name)>]
[PROCESS(name)]
[ACTIVITY(name)]
```

An AUDITLOG statement cannot contain additional arguments. Other statements might consist of multiple arguments. When using multiple arguments, put each argument on a separate line; use a non-blank character in column 72 to indicate that this argument and the following one are to be treated as a single control statement. An illegal combination of arguments generates an error message and the utility is not run against the log stream.

ACTIVITY(*name*)
The 1-16 name of an activity. Records for this activity are formatted. No further arguments are needed to make up a control statement; if none are provided, all audit records containing this activity name are selected. To limit the scope of the search, you can add a PTYPE argument, a PROCESS argument, or both on adjoining lines, using a continuation character in column 72.

AUDITLOG(*name*)
The 1-8 character DD name that identifies the audit log data set to be searched. The default is 'AUDITLOG'. This argument must not be specified more than once. It cannot be used with any other in a control statement.

If the specified audit log cannot be located or connected to, or if more than one AUDITLOG statement is found in the SYSIN data set, an error occurs and DFHATUP terminates.

PROCESS(*name*)
The 1-36 character name of a BTS process. No further arguments are needed to make up a control statement; if none are provided, all audit records containing this process name are selected. To limit the scope of the search, you can add a PTYPE argument, an ACTIVITY argument, or both on adjoining lines, using a continuation character in column 72.

PTYPE(*name*)
The 1-8 character name of a BTS process-type. No additional arguments are needed; if none are provided, all audit records containing this process-type are selected. To limit the scope of the search, you can add a PROCESS argument, an ACTIVITY argument, or both on adjoining lines, using a continuation character in column 72.

Example output from the DFHATUP utility

Follow these example outputs of different levels of audit trail auditing from the DFHATUP audit utility.

CICS writes records to an audit log in chronological order. Particularly on busy systems in a sysplex, records from different processes or from different activities in the same process are likely to become interleaved. In order to find out what has taken place during the execution of a specific process, you might want to select particular sets of records.

"Audit trail examples" on page 120 shows the points at which records are written to an audit log, depending on the level of auditing specified for the relevant process-type. The example control statements in Figure 69 on page 126 format all the records written to the audit log for the SALES1234567890 process (which is of the SALES process-type).

```
//SALESLOG DD DSN=CICSAA#.CICSDC1.JRNL001,
//            SUBSYS=(LOGR,DFHLGCNV),
//            DCB=BLKSIZE=32760
//SYSIN    DD *
AUDITLOG(SALESLOG)
PTYPE(SALES)                                                              +
PROCESS(SALES1234567890)
/*
//*
```

Figure 69. Example control statements, to format all the records for the SALES1234567890 process

Example audit trail process-level auditing

Follow this example audit trail to see the output that is produced if the AUDITLEVEL attribute of the PROCESSTYPE definition for the SALES process-type is set to 'PROCESS'.

Extending our previous example, Figure 70 on page 126 shows the output that would be produced if:

- On both regions SYS1 and SYS2, the AUDITLEVEL attribute of the PROCESSTYPE definition for the SALES process-type is set to 'PROCESS'
- The control statements in the SYSIN data set specify that all records for the SALES1234567890 process must be formatted.

```
CBTS Audit Trail Utility - Parameter Validation                                  Date : 29/01/1999 Time : 15:59:30 Page 000001

Exec Parm Options: Natlang   (EN)
                   Translate (mixedcase)
                   Pagesize  (60)
```

```
CBTS Audit Trail Utility - Audit Print                                           Date : 29/01/1999 Time : 15:59:30 Page 000002

Ptype(SALES   ) Function(Define Process  ) Process(SALES1234567890                   ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000072) Activity(DFHROOT        ) Transid(R   ) Program(R       ) Userid(CICSUSER)
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....;f..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3235800CCCDDDE444444444)
                                                              (21469351A172924981B98363339AA51E6014689663000000000)
  Current: Transid(P   ) Program(P       ) Userid(CICSUSER) Date(1999.029) Time(15:59:20.798300)

Ptype(SALES   ) Function(Run Process     ) Process(SALES1234567890                   ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000072) Activity(DFHROOT        ) Asynchronous
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....;f..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3235800CCCDDDE444444444)
                                                              (21469351A172924981B98363339AA51E6014689663000000000)
  Current: Transid(P   ) Program(P       ) Userid(CICSUSER) Date(1999.029) Time(15:59:20.798565)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                   ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000073) Activity(DFHROOT        ) Event(DFHINITIAL      )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....;f..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3235800CCCDDDE444444444)
                                                              (21469351A172924981B98363339AA51E6014689663000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(15:59:20.865320)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                   ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000082) Activity(DFHROOT        ) Event(C               )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....;f..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3235800CCCDDDE444444444)
                                                              (21469351A172924981B98363339AA51E6014689663000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(15:59:25.978683)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                   ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000087) Activity(DFHROOT        ) Event(D               )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....;f..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3235800CCCDDDE444444444)
                                                              (21469351A172924981B98363339AA51E6014689663000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(15:59:26.824560)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                   ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000087) Activity(DFHROOT        ) Compstatus(Normal    )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....;f..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3235800CCCDDDE444444444)
                                                              (21469351A172924981B98363339AA51E6014689663000000000)
  Current: Transid(R   ) Userid(CICSUSER) Date(1999.029) Time(15:59:26.849330)
```

```
CBTS Audit Trail Utility - Selection Results                                     Date : 29/01/1999 Time : 15:59:30 Page 000003

Number of Audit records read :         6
Number of records selected   :         6
  Processing Complete
```

Figure 70. Example audit trail, showing the types of record written for process-level auditing

Example audit trail activity-level auditing

Follow this example audit trail to see the output that is produced if the AUDITLEVEL attribute of the PROCESSTYPE definition for the SALES process-type is set to 'ACTIVITY'.

Figure 71 on page 127 shows the output that is produced if:

- On both regions SYS1 and SYS2, the AUDITLEVEL attribute of the PROCESSTYPE definition for the SALES process-type was set to 'ACTIVITY'
- The control statements in the SYSIN data set specify that all records for the SALES1234567890 process must be formatted.

```
CBTS Audit Trail Utility - Parameter Validation                                  Date : 29/01/1999 Time : 15:24:02 Page 000001

Exec Parm Options: Natlang   (EN)
                   Translate (mixedcase)
                   Pagesize  (60)
```

Figure 71. Example audit trail, showing the types of record written for activity-level auditing (Part 1)

```
CBTS Audit Trail Utility - Audit Print                                           Date : 29/01/1999 Time : 15:24:02 Page 000002

Ptype(SALES   ) Function(Define Process  ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000053) Activity(DFHROOT         ) Transid(R   ) Program(R       ) Userid(CICSUSER)
                                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....F..DFHROOT         )
                                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3349C00CCCDDDE444444444)
                                                                              (21469351A172924981B98363339A2860601468966300000000)
  Current: Transid(P   ) Program(P       ) Userid(CICSUSER) Date(1999.029) Time(15:23:53.323766)

Ptype(SALES   ) Function(Run Process     ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000053) Activity(DFHROOT         ) Asynchronous
                                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....F..DFHROOT         )
                                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3349C00CCCDDDE444444444)
                                                                              (21469351A172924981B98363339A2860601468966300000000)
  Current: Transid(P   ) Program(P       ) Userid(CICSUSER) Date(1999.029) Time(15:23:53.324025)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000054) Activity(DFHROOT         ) Event(DFHINITIAL      )
                                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....F..DFHROOT         )
                                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3349C00CCCDDDE444444444)
                                                                              (21469351A172924981B98363339A2860601468966300000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(15:23:53.433036)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000055) Activity(B               ) Event(DFHINITIAL      )
                                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2........B               )
                                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3360A00C444444444444444)
                                                                              (21469351A172924981B98229672A283CE012000000000000000)
  Current: Transid(B   ) Program(B       ) Userid(CICSUSER) Date(1999.029) Time(15:23:53.440627)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000055) Activity(B               ) Compstatus(Normal    )
                                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2........B               )
                                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3360A00C444444444444444)
                                                                              (21469351A172924981B98229672A283CE012000000000000000)
  Current: Transid(B   ) Userid(CICSUSER) Date(1999.029) Time(15:23:53.440834)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000056) Activity(C               ) Event(DFHINITIAL      )
                                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2....<-..C               )
                                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3364600C444444444444444)
                                                                              (21469351A172924981B98229672A286C0013000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(15:23:53.549149)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000057) Activity(D               ) Event(DFHINITIAL      )
                                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2.....x..D               )
                                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB336FA00C444444444444444)
                                                                              (21469351A172924981B98229672A287C7014000000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(15:23:54.116600)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000058) Activity(E               ) Event(DFHINITIAL      )
                                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..."_K..E               )
                                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3376D00C444444444444444)
                                                                              (21469351A172924981B98229672A28FD2015000000000000000)
  Current: Transid(E   ) Program(E       ) Userid(CICSUSER) Date(1999.029) Time(15:23:54.185211)
```

Figure 72. Example audit trail, showing the types of record written for activity-level auditing (Part 2)

```
CBTS Audit Trail Utility - Audit Print                                        Date : 29/01/1999 Time : 15:24:02 Page 000003

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000058) Activity(E               ) Compstatus(Normal    )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..."_K..E               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3376D00C444444444444444)
                                                              (21469351A172924981B98229672A28FD2015000000000000000)
  Current: Transid(E   ) Userid(CICSUSER) Date(1999.029) Time(15:23:54.185619)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000059) Activity(F               ) Event(DFHINITIAL      )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2...f."..F               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3381700C444444444444444)
                                                              (21469351A172924981B98229672A2865F016000000000000000)
  Current: Transid(F   ) Program(F       ) Userid(CICSUSER) Date(1999.029) Time(15:23:54.198352)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000059) Activity(F               ) Compstatus(Normal    )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2...f."..F               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3381700C444444444444444)
                                                              (21469351A172924981B98229672A2865F016000000000000000)
  Current: Transid(F   ) Userid(CICSUSER) Date(1999.029) Time(15:23:54.198609)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000060) Activity(G               ) Event(DFHINITIAL      )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2....0...G               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB330FC00C444444444444444)
                                                              (21469351A172924981B98229672A2990D017000000000000000)
  Current: Transid(G   ) Program(G       ) Userid(CICSUSER) Date(1999.029) Time(15:23:58.581394)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000061) Activity(C               ) Event(E               )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2....<-..C               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3364600C444444444444444)
                                                              (21469351A172924981B98229672A286C0013000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(15:23:58.591807)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000062) Activity(C               ) Event(F               )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2....<-..C               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3364600C444444444444444)
                                                              (21469351A172924981B98229672A286C0013000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(15:23:58.620666)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000062) Activity(C               ) Compstatus(Normal    )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2....<-..C               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB3364600C444444444444444)
                                                              (21469351A172924981B98229672A286C0013000000000000000)
  Current: Transid(C   ) Userid(CICSUSER) Date(1999.029) Time(15:23:58.636578)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000063) Activity(DFHROOT         ) Event(C               )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....F..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB3349C00CCCDDDE444444444)
                                                              (21469351A172924981B98363339A2860601468966300000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(15:23:58.661620)
```

Figure 73. Example audit trail, showing the types of record written for activity-level auditing (Part 3)

```
CBTS Audit Trail Utility - Audit Print                                    Date : 29/01/1999 Time : 15:24:02 Page 000004

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000064) Activity(G               ) Compstatus(Forced    )
                                                 ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2....0...G               )
                                                           (CCDCCDCF11CCCCDCEC4CEDFECEFB330FC00C444444444444444)
                                                           (21469351A172924981B98229672A2990D017000000000000000)
  Current: Transid(I   ) Program(I       ) Userid(CICSUSER) Date(1999.029) Time(15:24:00.664584)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000065) Activity(D               ) Event(G               )
                                                 ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2.....x..D               )
                                                           (CCDCCDCF11CCCCDCEC4CEDFECEFB336FA00C444444444444444)
                                                           (21469351A172924981B98229672A287C7014000000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(15:24:00.725741)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000066) Activity(H               ) Event(DFHINITIAL      )
                                                 ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2....1...H               )
                                                           (CCDCCDCF11CCCCDCEC4CEDFECEFB3359000C444444444444444)
                                                           (21469351A172924981B98229672A2F737018000000000000000)
  Current: Transid(H   ) Program(H       ) Userid(CICSUSER) Date(1999.029) Time(15:24:00.784073)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000066) Activity(H               ) Compstatus(Normal    )
                                                 ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2....1...H               )
                                                           (CCDCCDCF11CCCCDCEC4CEDFECEFB3359000C444444444444444)
                                                           (21469351A172924981B98229672A2F737018000000000000000)
  Current: Transid(H   ) Userid(CICSUSER) Date(1999.029) Time(15:24:00.784346)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000067) Activity(D               ) Event(H               )
                                                 ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2.....x..D               )
                                                           (CCDCCDCF11CCCCDCEC4CEDFECEFB336FA00C444444444444444)
                                                           (21469351A172924981B98229672A287C7014000000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(15:24:00.813682)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000067) Activity(D               ) Compstatus(Normal    )
                                                 ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2.....x..D               )
                                                           (CCDCCDCF11CCCCDCEC4CEDFECEFB336FA00C444444444444444)
                                                           (21469351A172924981B98229672A287C7014000000000000000)
  Current: Transid(D   ) Userid(CICSUSER) Date(1999.029) Time(15:24:02.478498)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000068) Activity(DFHROOT         ) Event(D               )
                                                 ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....F..DFHROOT         )
                                                           (CCDCCDCF11CCCCDCEC4CECEECFFB3349C00CCCDDDE444444444)
                                                           (21469351A172924981B98363339A2860601468966300000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(15:24:02.511054)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000068) Activity(DFHROOT         ) Compstatus(Normal    )
                                                 ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....F..DFHROOT         )
                                                           (CCDCCDCF11CCCCDCEC4CECEECFFB3349C00CCCDDDE444444444)
                                                           (21469351A172924981B98363339A2860601468966300000000)
  Current: Transid(R   ) Userid(CICSUSER) Date(1999.029) Time(15:24:02.571838)

CBTS Audit Trail Utility - Selection Results                              Date : 29/01/1999 Time : 15:24:02 Page 000005


Number of Audit records read :        24
Number of records selected   :        24
  Processing Complete
```

Figure 74. Example audit trail, showing the types of record written for activity-level auditing (Part 4)

Example audit trail full auditing

Follow this example audit trail to see the output that is produced if the AUDITLEVEL attribute of the PROCESSTYPE definition for the SALES process-type is set to 'FULL'.

Figure 75 on page 129 shows the output that is produced if:

- the AUDITLEVEL attribute of the PROCESSTYPE definition for the SALES process-type is set to 'FULL'
- The control statements in the SYSIN data set specify that all records for the SALES1234567890 process must be formatted.

```
CBTS Audit Trail Utility - Parameter Validation                           Date : 29/01/1999 Time : 14:39:04 Page 000001

Exec Parm Options: Natlang   (EN)
                   Translate (mixedcase)
                   Pagesize  (60)
```

Figure 75. Example audit trail, showing the types of record written for full auditing (Part 1)

```
CBTS Audit Trail Utility - Audit Print                                        Date : 29/01/1999 Time : 14:39:04 Page 000002

Ptype(SALES   ) Function(Define Process  ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000033) Activity(DFHROOT        ) Transid(R   ) Program(R       ) Userid(CICSUSER)
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....v..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB2902A00CCCDDDE444444444)
                                                              (21469351A172924981B98363339A709F5014689663000000000)
  Current: Transid(P   ) Program(P       ) Userid(CICSUSER) Date(1999.029) Time(14:36:12.557162)

Ptype(SALES   ) Function(Run Process     ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000033) Activity(DFHROOT        ) Asynchronous
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....v..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB2902A00CCCDDDE444444444)
                                                              (21469351A172924981B98363339A709F5014689663000000000)
  Current: Transid(P   ) Program(P       ) Userid(CICSUSER) Date(1999.029) Time(14:36:13.921790)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000034) Activity(DFHROOT        ) Event(DFHINITIAL      )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....v..DFHROOT         )
                                                              (CCDCCDCF11CCCCDCEC4CECEECFFB2902A00CCCDDDE444444444)
                                                              (21469351A172924981B98363339A709F5014689663000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.142640)

Ptype(SALES   ) Function(Define Activity ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000034) Activity(B              ) CompletionEvent(B               ) Transid(B   ) Program(B       ) Userid(CICSUSER)
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.....B               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29B0300C444444444444444)
                                                              (21469351A172924981B98229672A7111C012000000000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.295419) Activity(DFHROOT         )

Ptype(SALES   ) Function(Run Activity    ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000034) Activity(B              ) Synchronous
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.....B               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29B0300C444444444444444)
                                                              (21469351A172924981B98229672A7111C012000000000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.295549) Activity(DFHROOT         )

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000035) Activity(B              ) Event(DFHINITIAL      )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.....B               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29B0300C444444444444444)
                                                              (21469351A172924981B98229672A7111C012000000000000000)
  Current: Transid(B   ) Program(B       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.296323)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000035) Activity(B              ) Compstatus(Normal    )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.....B               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29B0300C444444444444444)
                                                              (21469351A172924981B98229672A7111C012000000000000000)
  Current: Transid(B   ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.408739)

Ptype(SALES   ) Function(Define Activity ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000034) Activity(C              ) CompletionEvent(C               ) Transid(C   ) Program(C       ) Userid(CICSUSER)
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.6...C               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29DF500C444444444444444)
                                                              (21469351A172924981B98229672A71C690130000000000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.472960) Activity(DFHROOT         )
```

Figure 76. Example audit trail, showing the types of record written for full auditing (Part 2)

```
CBTS Audit Trail Utility - Audit Print                                        Date : 29/01/1999 Time : 14:39:04 Page 000003

Ptype(SALES   ) Function(Run Activity    ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000034) Activity(C              ) Asynchronous
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.6...C               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29DF500C444444444444444)
                                                              (21469351A172924981B98229672A71C690130000000000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.473066) Activity(DFHROOT         )

Ptype(SALES   ) Function(Define Activity ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000034) Activity(D              ) CompletionEvent(D               ) Transid(D   ) Program(D       ) Userid(CICSUSER)
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29DBF00C444444444444444)
                                                              (21469351A172924981B98229672A71EE901400000000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.482228) Activity(DFHROOT         )

Ptype(SALES   ) Function(Run Activity    ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000034) Activity(D              ) Asynchronous
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29DBF00C444444444444444)
                                                              (21469351A172924981B98229672A71EE901400000000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.482346) Activity(DFHROOT         )

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000036) Activity(C              ) Event(DFHINITIAL      )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.6...C               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29DF500C444444444444444)
                                                              (21469351A172924981B98229672A71C690130000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.556761)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000037) Activity(D              ) Event(DFHINITIAL      )
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB29DBF00C444444444444444)
                                                              (21469351A172924981B98229672A71EE901400000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.569775)

Ptype(SALES   ) Function(Define Activity ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000036) Activity(E              ) CompletionEvent(E               ) Transid(E   ) Program(E       ) Userid(CICSUSER)
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..o..E               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB2906900C444444444444444)
                                                              (21469351A172924981B98229672A72946015000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.656929) Activity(C               )

Ptype(SALES   ) Function(Run Activity    ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000036) Activity(E              ) Asynchronous
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..o..E               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB2906900C444444444444444)
                                                              (21469351A172924981B98229672A72946015000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.657049) Activity(C               )

Ptype(SALES   ) Function(Define Activity ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000036) Activity(F              ) CompletionEvent(F               ) Transid(F   ) Program(F       ) Userid(CICSUSER)
                                                    ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..I..F               )
                                                              (CCDCCDCF11CCCCDCEC4CEDFECEFB290AC00C444444444444444)
                                                              (21469351A172924981B98229672A72BA901600000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.668485) Activity(C               )
```

Figure 77. Example audit trail, showing the types of record written for full auditing (Part 3)

```
CBTS Audit Trail Utility - Audit Print                                          Date : 29/01/1999 Time : 14:39:04 Page 000004

Ptype(SALES   ) Function(Run Activity    ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000036) Activity(F               ) Asynchronous
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..I..F               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB290AC00C444444444444444)
                                                                    (21469351A172924981B98229672A72BA9016000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.668584) Activity(C               )

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000038) Activity(E               ) Event(DFHINITIAL      )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..o..E               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB2906900C444444444444444)
                                                                    (21469351A172924981B98229672A729460150000000000000000)
  Current: Transid(E   ) Program(E       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.757748)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000039) Activity(F               ) Event(DFHINITIAL      )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..I..F               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB290AC00C444444444444444)
                                                                    (21469351A172924981B98229672A72BA9016000000000000000)
  Current: Transid(F   ) Program(F       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.790932)

Ptype(SALES   ) Function(Define Activity ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000037) Activity(G               ) CompletionEvent(G                ) Transid(G   ) Program(G       ) Userid(CICSUSER)
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k.....G               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB2925000C444444444444444)
                                                                    (21469351A172924981B98229672A72F75017000000000000000)
  Current: Transid(D   ) Program(D       ) User
Ptype(SALES   ) Function(Run Activity    ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000037) Activity(G               ) Asynchronous
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k.....G               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB2925000C444444444444444)
                                                                    (21469351A172924981B98229672A72F75017000000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.811377) Activity(D               )

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000040) Activity(G               ) Event(DFHINITIAL      )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k.....G               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB2925000C444444444444444)
                                                                    (21469351A172924981B98229672A72F75017000000000000000)
  Current: Transid(G   ) Program(G       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.844281)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000039) Activity(F               ) Compstatus(Normal    )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..I..F               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB290AC00C444444444444444)
                                                                    (21469351A172924981B98229672A72BA9016000000000000000)
  Current: Transid(F   ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.887329)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000041) Activity(C               ) Event(F               )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.6...C               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB29DF500C444444444444444)
                                                                    (21469351A172924981B98229672A71C690130000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.979781)
```

Figure 78. Example audit trail, showing the types of record written for full auditing (Part 4)

```
CBTS Audit Trail Utility - Audit Print                                          Date : 29/01/1999 Time : 14:39:04 Page 000005

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000038) Activity(E               ) Compstatus(Normal    )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..o..E               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB2906900C444444444444444)
                                                                    (21469351A172924981B98229672A729460150000000000000000)
  Current: Transid(E   ) Userid(CICSUSER) Date(1999.029) Time(14:36:15.070372)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000042) Activity(C               ) Event(E               )
                                                          ActivityId(BAMFILE1..GB
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB29DF500C444444444444444)
                                                                    (21469351A172924981B98229672A71C690130000000000000000)
  Current: Transid(C   ) Program(C       ) Userid(CICSUSER) Date(1999.029) Time(14:36:15.117121)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000042) Activity(C               ) Compstatus(Normal    )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.6...C               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB29DF500C444444444444444)
                                                                    (21469351A172924981B98229672A71C690130000000000000000)
  Current: Transid(C   ) Userid(CICSUSER) Date(1999.029) Time(14:36:15.135971)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000043) Activity(DFHROOT         ) Event(C               )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....v..DFHROOT         )
                                                                    (CCDCCDCF11CCCCDCEC4CECEECFFB2902A00CCCDDDE444444444)
                                                                    (21469351A172924981B98363339A709F501468966300000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:15.169265)

Ptype(SALES   ) Function(Acquire ActId   ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000045) Activity(G               )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k.....G               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB2925000C444444444444444)
                                                                    (21469351A172924981B98229672A72F75017000000000000000)
  Current: Transid(I   ) Program(I       ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.922942)

Ptype(SALES   ) Function(Cancel Activity ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000045) Activity(G               )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k.....G               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB2925000C444444444444444)
                                                                    (21469351A172924981B98229672A72F75017000000000000000)
  Current: Transid(I   ) Program(I       ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.923045)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000045) Activity(G               ) Compstatus(Forced    )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k.....G               )
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB2925000C444444444444444)
                                                                    (21469351A172924981B98229672A72F75017000000000000000)
  Current: Transid(I   ) Program(I       ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.923093)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000046) Activity(D               ) Event(G               )
                                                          ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9...D
                                                                    (CCDCCDCF11CCCCDCEC4CEDFECEFB29DBF00C444444444444444)
                                                                    (21469351A172924981B98229672A71EE901400000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.948512)
```

Figure 79. Example audit trail, showing the types of record written for full auditing (Part 5)

```
CBTS Audit Trail Utility - Audit Print                                    Date : 29/01/1999 Time : 14:39:04 Page 000006

Ptype(SALES   ) Function(Define Activity ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000046) Activity(H              ) CompletionEvent(H              ) Transid(H    ) Program(H       ) Userid(CICSUSER)
                                                        ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..q.X...H               )
                                                                  (CCDCCDCF11CCCCDCEC4CEDFECEFB29FED00C444444444444444)
                                                                  (21469351A172924981B98229672A78F7F018000000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.990993) Activity(D             )

Ptype(SALES   ) Function(Run Activity    ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000046) Activity(H              ) Asynchronous
                                                        ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..q.X...H               )
                                                                  (CCDCCDCF11CCCCDCEC4CEDFECEFB29FED00C444444444444444)
                                                                  (21469351A172924981B98229672A78F7F018000000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.991119) Activity(D             )

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000047) Activity(H              ) Event(DFHINITIAL      )
                                                        ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..q.X...H               )
                                                                  (CCDCCDCF11CCCCDCEC4CEDFECEFB29FED00C444444444444444)
                                                                  (21469351A172924981B98229672A78F7F018000000000000000)
  Current: Transid(H   ) Program(H       ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.052659)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS2) Auditlog(BAMAUDIT)
  Taskno(0000047) Activity(H              ) Compstatus(Normal    )
                                                        ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..q.X...H               )
                                                                  (CCDCCDCF11CCCCDCEC4CEDFECEFB29FED00C444444444444444)
                                                                  (21469351A172924981B98229672A78F7F018000000000000000)
  Current: Transid(H   ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.123737)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000048) Activity(D              ) Event(H               )
                                                        ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D               )
                                                                  (CCDCCDCF11CCCCDCEC4CEDFECEFB29DBF00C444444444444444)
                                                                  (21469351A172924981B98229672A71EE901400000000000000)
  Current: Transid(D   ) Program(D       ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.147332)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000048) Activity(D              ) Compstatus(Normal    )
                                                        ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D               )
                                                                  (CCDCCDCF11CCCCDCEC4CEDFECEFB29DBF00C444444444444444)
                                                                  (21469351A172924981B98229672A71EE901400000000000000)
  Current: Transid(D   ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.162148)

Ptype(SALES   ) Function(Activation      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000049) Activity(DFHROOT        ) Event(D               )
                                                        ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....v..DFHROOT         )
                                                                  (CCDCCDCF11CCCCDCEC4CECEECFFB2902A00CCCDDDE444444444)
                                                                  (21469351A172924981B98363339A709F5014689663000000000)
  Current: Transid(R   ) Program(R       ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.185932)

Ptype(SALES   ) Function(Completion      ) Process(SALES1234567890                    ) System(SYS1) Auditlog(BAMAUDIT)
  Taskno(0000049) Activity(DFHROOT        ) Compstatus(Normal    )
                                                        ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39.....v..DFHROOT         )
                                                                  (CCDCCDCF11CCCCDCEC4CECEECFFB2902A00CCCDDDE444444444)
                                                                  (21469351A172924981B98363339A709F5014689663000000000)
  Current: Transid(R   ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.482472)




CBTS Audit Trail Utility - Selection Results                              Date : 29/01/1999 Time : 14:39:04 Page 000007

Number of Audit records read :        40
Number of records selected   :        40
  Processing Complete
```

Figure 80. Example audit trail, showing the types of record written for full auditing (Part 6)

Note:

1. All times in the audit trails refer to Greenwich Mean Time (GMT).
2. As the example audit trails show, the detailed information within the audit report varies according to the audit point taken.
3. When an activity is activated, in some cases the name of the event that caused the activation is not available. In these cases, the request type and reason for the activation are reported. The possible request types are:
   - Dispatch
   - Cancel
   - Delete

   The possible reasons are:
   - Fire complete
   - Fire input
   - Fire timer
   - Delete command
   - Delete complete
   - Delete reset

- Delete tree
- Cancel command
- Cancel complete
- Cancel force
- Reattach acq
- Unknown. Unknown applies only to dispatch requests. The unknown reason means that the activation has not been triggered by a specific event. This can happen, for example, in any of the following cases:
  - An application issues a RESUME command against a child activity. In this case, BTS does a speculative dispatch, to see if there are any events to be serviced; it does not know, at the time the activation is started, whether there are any.
  - An activation terminates but there are several more fired events that it needs to service. BTS reactivates the activity immediately, but does not regard the activation as being caused by any particular event.
  - A timer is forced. Although a particular timer event fires, this firing occurs in the application that issued the FORCE TIMER command; it is not part of the request that starts the activation.

Examining BTS repository records

You can examine records on a BTS repository data set with the DFHBARUP utility, which takes a "snapshot" of your BTS system.

About this task

It contains:

- “The repository utility program, DFHBARUP” on page 133
- “Using DFHBARUP” on page 134.

The repository utility program, DFHBARUP

You can use the DFHBARUP utility to take a "snapshot" of your BTS system at the time the utility is run. You can filter to print records for a specific process or activity.

By default, DFHBARUP prints all the records currently on the specified repository. If you have more than one repository, it is a snapshot of the processes served by the specified repository.

The state of a repository can change from moment to moment, especially if it is shared across a busy sysplex. For example, records for new processes and activities can be added constantly; conversely, as processes complete and events are deleted their associated records disappear from the repository.

Using DFHBARUP you can filter selected records. For example, you could print only the records associated with a specific process. Doing so would give you the current state of:

- The activities that have been defined to the process, and have not yet been deleted
- The containers associated with the activities - that is, the data they contain
- The events in the event pools of the activities.

Alternatively, you could print only the records associated with a specific activity. Doing so would give you the current state of:

- The activity itself
- The containers associated with the activity
- The events in the event pool of the activity.

DFHBARUP formats the records it extracts, to make them easier to interpret.

Using DFHBARUP

Use the repository utility program, DFHBARUP, to read, format, and print selected records from a specified repository data set.

About this task

Run DFHBARUP as a batch job.

DFHBARUP reads the records in the order they are stored on the repository - that is, in keyed-sequence order. To select the records that DFHBARUP writes to the output data set, SYSPRINT, you include control statements in the SYSIN data set. By default, DFHBARUP prints all records currently on the data set. DFHBARUP formats the selected records before writing them to SYSPRINT.

Sample job stream to run the DFHBARUP program

To help you develop your own job to run the DFHBARUP utility program, use this sample job.

Figure 81 on page 134 shows an example job stream to run the DFHBARUP program. The job stream includes DD statements for the following data sets:

The repository
The repository data set to be examined to produce the output data. (Figure 81 on page 134 shows a DD name of 'REPOS'.)

STEPLIB
A partitioned data set (DSORG=PO) that contains the DFHBARUP program module. If the module is in a library in the link list, this statement is not required.

SYSIN
The input control data set. This file must be in 80- byte record format. The control statements that you can use in this data set are described in “SYSIN control statements” on page 135.

Control statements can be continued on to the next line by including any non-blank character in column 72. If the line that follows a continuation character is empty or contains control arguments which conflict with those that make up the preceding part of the control statement, an error is reported and execution of the utility ends. Any characters which occur beyond column 72 are ignored.

SYSPRINT
The output data set to which the formatted audit records and control messages are to be sent.

```
//*****************************************************************
//* RUN DFHBARUP (REPOSITORY UTILITY PROGRAM)
//*
//*
//*****************************************************************
//ARUP     EXEC PGM=DFHBARUP,PARM='N(EN),P(60),T(M)'
//STEPLIB    DD DSN=CTS130.CICS530.SDFHLOAD,DISP=SHR
//*****************************************************************
//*   The output will go to SYSPRINT
//*****************************************************************
//SYSPRINT DD SYSOUT=A,DCB=RECFM=FBA
//REPOS DD DISP=SHR,DSN=CICS530.CBTS.SALESREP
//SYSIN    DD *
PTYPE(SALES)                                                    +
PROCESS(CUSTSALES1999.13872977829728.QA)                        +
ACTIVITY(ORDER)
/*
//*
```

Figure 81. Sample job to run the DFHBARUP utility program

EXEC parameters

You can use the PARM keyword on the EXEC statement to pass the NATLANG, PAGESIZE, and TRANSLATE parameters to the DFHBARUP utility. Using these parameters you can control the content and format of the DFHBARUP utility output.

The form of the EXEC statement is:

```
EXEC PGM=DFHBARUP,PARM='parm1,...,parmn'
```

NATLANG({EN|*CS*v*KA*})
The language in which messages are to be issued.

The minimum abbreviation of this parameter is **N**. The possible values are:

CS
Traditional Chinese

EN
English. This is the default.

KA
Kanji.

PAGESIZE({60|*nn*})
The number of lines to be printed per page, when the output from the utility is sent to a printer. Valid values are in the range 20–99. The default is 60.

The minimum abbreviation of this parameter is **P**.

TRANSLATE({MIXEDCASE|*UPPERCASE*})
Whether the output from the utility is to be in mixed-case or uppercase. The default is mixed-case.

The minimum abbreviation of this parameter is **T**. The minimum abbreviations of MIXEDCASE and UPPERCASE are M and U respectively.

SYSIN control statements

The SYSIN data set passes information to DFHBARUP. You can include statements to select specific sets of records to be formatted. Define the SYSIN data set with these control statements.

Format of the SYSIN control statements

```
SYSIN DD *
[REPOSITORY(name)]
[PTYPE(name)]
[PROCESS(name)]
[ACTIVITY(name)]
```

Comments are identified by an asterisk in the first position.

The REPOSITORY statement cannot contain additional arguments. Other statements can consist of multiple arguments. When using multiple arguments, put each argument on a separate line; use a non-blank character in column 72 to indicate that this argument and the following one are to be treated as a single control statement. An illegal combination of arguments generates an error message and the utility is not run against the log stream.

ACTIVITY(*name*)
The 1-16 character name of an activity. Only records for activities of this name are formatted. To limit the scope of the search, specify a PROCESS or PTYPE argument with ACTIVITY.

PROCESS(*name*)
The 1-36 character name of a BTS process. No further arguments are needed to make up a control statement; if none are provided, all records containing this process name are selected. To limit the scope of the search, you can add a PTYPE argument on an adjoining line, using a continuation character in column 72.

PTYPE(*name*)
The 1-8 character name of a BTS process-type. No additional arguments are needed; if none are provided, all records containing this process-type are selected. To limit the scope of the search, you can add a PROCESS argument on an adjoining line, using a continuation character in column 72.

REPOSITORY(*name*)

The 1-8 character DD name that identifies the repository data set to be searched. The default is 'REPOS'. This argument must not be specified more than once. It cannot be used with any other in a control statement.

If the specified repository file cannot be opened, or if more than one REPOSITORY statement is found in the SYSIN data set, an error occurs and DFHBARUP terminates.

Example output from the DFHBARUP utility

Follow this example of output produced by the DFHBARUP utility. The example control statements format all the records currently on the SALEREP repository for the SALES1234567890 process, which is of the SALES process-type.

```
.
.
//SALESREP DD DISP=SHR,DSN=CICS530.CBTS.SALESREP
//SYSIN      DD *
REPOSITORY(SALESREP)
PTYPE(SALES)                                                    +
PROCESS(SALES1234567890)
/*
```

Figure 82. Example control statements, to format all records on the SALEREP repository for the SALES1234567890 process

Figure 83 on page 136 shows the output that might be produced by the control statements in Figure 82 on page 136.

```
CICS Business Transaction Services - Parameter Validation                          Date : 29/01/1999 Time : 14:39:04 Page 0001
Exec Parm Options: Natlang   (EN)
                   Translate (mixedcase)
                   Pagesize  (60)

REPOSITORY(SALEREP)
```

Figure 83. Example output from the DFHBARUP utility (Part 1)

```
CICS Business Transaction Services - Repository File Report                        Date : 29/01/1999 Time : 14:39:04 Page 0002
Activity Name : DFHROOT          Id : ..GBIBMIYA.IYCWTC37........DFHROOT          Generation : 0000001
                                      11CCCCDCEC4CECEECFF44042F00CCCDDDE4444444444
                                      A172924981B98363337BB0C1B0146896630000000000
Definitional Attributes
  Program    : ABU081D
  Transid    : RUP4
  Userid     : CICSUSER
  Comp Event :
Current State
  Mode       : Dormant    (Initial, Active, Dormant, Cancelling, Complete)
  Suspended  : No  (Yes, No)
  Generation : 0000001
  Child Count : 0000002
Completion Status
  Completion Response : Incomplete

000000   C1401A11 C7C2C9C2 D4C9E8C1 4BC9E8C3 E6E3C3F3 F74B4B00 4C21FB00 01C4C6C8  *A ..GBIBMIYA.IYCWTC37...<....DFH*
000020   D9D6D6E3 40404040 40404040 40400000 00000004 00004000 000005E0 01500000  *ROOT          ........ ....\.&..*
000040   6EC4C6C8 C2C1C1C3 E3C9E5C9 00000000 FFFFFFFF 01500001 00000000 D740D7E3  *>DFHBAACTIVI.........&......P SA*
000060   E8D7C5F1 4040D7D9 D6C36DC6 D6E4D940 40404040 40404040 40404040 40404040  *LES   SALES1234567890          *
000080   40404040 40404040 40400000 00000000 00000000 00000000 00000000 00000000  *          ......................*
0000A0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  *................................*
0000C0   1A11C7C2 C9C2D4C9 E8C14BC9 E8C3E6E3 C3F3F74B 4B004C21 FB000103 00000000  *..GBIBMIYA.IYCWTC37...<.........*
0000E0   00000000 00000000 00000000 00000000 00000002 00000000 00000000 000003C4  *...............................D*
000100   00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000  *................................*
000120   00000000 10C2011C 10C2011C 00000000 00000000 10C2012C 10C2012C C1C2E4F0  *.....B...B...........B...B..ABU0*
000140   F8F1C440 00000000 00000000 D9E4D7F4 C3C9C3E2 E4E2C5D9 40404040 40404040  *81D ........RUP4CICSUSER        *
000160   40404040 40404040 01404040 40404040 40404040 4003C2C1 D4C1E4C4 C9E30000  *        .            .BAMAUDIT..*
000180   00000000 0000FFFF FFFFFFFF                                                *............                    *
Related BTS Objects
  Process Type : SALES   Name : SALES1234567890
  No Parent
  Child Name : ACT_3            Id : ..GBIBMIYA.IYK2ZFX2........ACT_3            Generation : 0000001
                                     11CCCCDCEC4CEDFECEF440F3A00CCE6F444444444444
                                     A172924981B98229672BBCF9F01133D3000000000000
  Child Name : ACT_ONE          Id : ..GBIBMIYA.IYK2ZFX2........ACT_ONE          Generation : 0000001
                                     11CCCCDCEC4CEDFECEF44068A00CCE6DDC4444444444
                                     A172924981B98229672BBB35A01133D6550000000000

Eventpool
  Event : (Reattach)
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
```

Figure 84. Example output from the DFHBARUP utility (Part 2)

```
CICS Business Transaction Services - Repository File Report                          Date : 29/01/1999 Time : 14:39:04 Page 0003
  Event : DFHINITIAL
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : ACT_ONE
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : ACT_3
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No

Containers

  No Containers
```

Figure 85. Example output from the DFHBARUP utility (Part 3)

```
CICS Business Transaction Services - Repository File Report                          Date : 29/01/1999 Time : 14:39:04 Page 0004
Activity Name : ACT_ONE          Id : ..GBIBMIYA.IYK2ZFX2........ACT_ONE          Generation : 0000001
                                      11CCCCDCEC4CEDFECEF44068A00CCE6DDC4444444444
                                      A172924981B98229672BBB35A01133D6550000000000
Definitional Attributes
  Program    : ABU081E
  Transid    : RUP5
  Userid     : CICSUSER
  Comp Event : ACT_ONE
Current State
  Mode       : Dormant    (Initial, Active, Dormant, Cancelling, Complete)
  Suspended  : No  (Yes, No)
  Generation  : 0000001
  Child Count : 0000000
Completion Status
  Completion Response : Incomplete

000000   C1401A11 C7C2C9C2 D4C9E8C1 4BC9E8D2 F2E9C6E7 F24B4B0B 6385AA00 01C1C3E3  *A ..GBIBMIYA.IYK2ZFX2....e...ACT*
000020   6DD6D5C5 40404040 40404040 40400000 00008081 00004000 00003E88 01500000  *_ONE           .....a.. ....h.&..*
000040   6EC4C6C8 C2C1C1C3 E3C9E5C9 00000000 FFFFFFFF 01500001 00000000 D740D7E3  *>DFHBAACTIVI.........&......P SA*
000060   E8D7C5F1 4040D7D9 D6C36DC6 D6E4D940 40404040 40404040 40404040 40404040  *LES   SALES1234567890           *
000080   40404040 40404040 40400000 0000C140 1A11C7C2 C9C2D4C9 E8C14BC9 E8C3E6E3  *          ....A ..GBIBMIYA.IYCWT*
0000A0   C3F3F74B 4B004C21 FB0001C4 C6C8D9D6 D6E34040 40404040 40404040 00000000  *C37...<....DFHROOT          ....*
0000C0   1A11C7C2 C9C2D4C9 E8C14BC9 E8C3E6E3 C3F3F74B 4B004C21 FB000103 D9E4D7F4  *..GBIBMIYA.IYCWTC37...<.....RUP4*
0000E0   C3C9C3E2 E4E2C5D9 00000000 00000001 00000000 00000000 00000000 000003C4  *CICSUSER.......................D*
000100   00000001 00000003 000084A9 00000000 00000000 00000000 00000000 00000000  *..........dz.....................*
000120   00000000 108F911C 108F911C 00000000 00000000 10C20C10 101465E0 C1C2E4F0  *......j...j..........B.....\ABU0*
000140   F8F1C540 00000000 00000000 D9E4D7F5 C3C9C3E2 E4E2C5D9 C1C3E36D D6D5C540  *81E ........RUP5CICSUSERACT_ONE *
000160   40404040 40404040 01404040 40404040 40404040 4003C2C1 D4C1E4C4 C9E30000  *        .            .BAMAUDIT..*
000180   00000000 0000FFFF FFFFFFFF                                               *............                    *
Related BTS Objects
  Proce  Parent Name : DFHROOT          Id : ..GBIBMIYA.IYCWTC37........DFHROOT          Generation : 0000001
                                      11CCCCDCEC4CECEECFF44042F00CCCDDDE4444444444
                                      A172924981B98363337BB0C1B0146896630000000000
No Children
Eventpool
  Event : (Reattach)
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
```

Figure 86. Example output from the DFHBARUP utility (Part 4)

```
CICS Business Transaction Services - Repository File Report                          Date : 29/01/1999 Time : 14:39:04 Page 0005
  Event : DFHINITIAL
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : ACT1_CONTINUE
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : ACT1_END
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : ACT2_DEF
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : ACT2_CAN
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : ACT2_SUS
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : ACT2_RES
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No

Containers

Container Name : ACT_CONT_1      Container Length : x'00008000'

000000   FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  *................................*
   .
   lines omitted
   .
07FE0   FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  *................................*
Container Name : ACT_CONT_2      Container Length : x'00000400'

000000   22222222 22222222 22222222 22222222 22222222 22222222 22222222 22222222  *................................*
   .
   lines omitted
   .
0003E0   22222222 22222222 22222222 22222222 22222222 22222222 22222222 22222222  *................................*
Container Name : ACT_CONT_3      Container Length : x'00000019'

000000   10101010 10101010 10101010 10101010 10101010 10101010 10                *.........................       *
```

Figure 87. Example output from the DFHBARUP utility (Part 5)

```
CICS Business Transaction Services - Repository File Report                          Date : 29/01/1999 Time : 14:39:04 Page 0006
Activity Name : ACT_3            Id : ..GBIBMIYA.IYK2ZFX2........ACT_3           Generation : 0000001
                                      11CCCCDCEC4CEDFECEF440F3A00CCE6F444444444444
                                      A172924981B98229672BBCF9F01133D3000000000000
Definitional Attributes
  Program    : ABU081Z
  Transid    : RUPZ
  Userid     : CICSUSER
  Comp Event : ACT_3
Current State
  Mode       : Dormant    (Initial, Active, Dormant, Cancelling, Complete)
  Suspended  : No  (Yes, No)
  Generation  : 0000001
  Child Count : 0000000
Completion Status
  Completion Response : Incomplete

000000   C1401A11 C7C2C9C2 D4C9E8C1 4BC9E8D2 F2E9C6E7 F24B4B0C FF39AF00 01C1C3E3  *A ..GBIBMIYA.IYK2ZFX2........ACT*
000020   6DF34040 40404040 40404040 40400000 00000000 00004000 00000550 01500000  *_3           ........ ....&.&..*
000040   6EC4C6C8 C2C1C1C3 E3C9E5C9 00000000 D9D4E4E6 01500001 00000000 D740D7E3  *>DFHBAACTIVI....RMUW.&......P SA*
000060   E8D7C5F1 4040D7D9 D6C36DC6 D6E4D940 40404040 40404040 40404040 40404040  *LES   SALES1234567890           *
000080   40404040 40404040 40400000 0000C140 1A11C7C2 C9C2D4C9 E8C14BC9 E8C3E6E3  *          ....A ..GBIBMIYA.IYCWT*
0000A0   C3F3F74B 4B004C21 FB0001C4 C6C8D9D6 D6E34040 40404040 40404040 00000000  *C37...<....DFHROOT          ....*
0000C0   1A11C7C2 C9C2D4C9 E8C14BC9 E8C3E6E3 C3F3F74B 4B004C21 FB000103 D9E4D7F4  *..GBIBMIYA.IYCWTC37...<.....RUP4*
0000E0   C3C9C3E2 E4E2C5D9 00000000 00000001 00000000 00000000 00000000 000003C4  *CICSUSER.......................D*
000100   00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000  *................................*
000120   00000000 10C2911C 10C2911C 00000000 00000000 10C2912C 10C2912C C1C2E4F0  *.....Bj..Bj.........Bj..Bj.ABU0*
000140   F8F1E940 00000000 00000000 D9E4D7E9 C3C9C3E2 E4E2C5D9 C1C3E36D F3404040  *81Z ........RUPZCICSUSERACT_3   *
000160   40404040 40404040 01404040 40404040 40404040 4003C2C1 D4C1E4C4 C9E30000  *        .            .BAMAUDIT..*
000180   00000000 00000000 00000000                                               *............                    *
Related BTS Objects
  Process Type : SALES   Name : SALES1234567890
  Parent Name : DFHROOT          Id : ..GBIBMIYA.IYCWTC37........DFHROOT          Generation : 0000001
                                      11CCCCDCEC4CECEECFF44042F00CCCDDDE4444444444
                                      A172924981B98363337BB0C1B0146896630000000000

No Children
Eventpool
  Event : (Reattach)
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
```

Figure 88. Example output from the DFHBARUP utility (Part 6)

```
CICS Business Transaction Services - Repository File Report                          Date : 29/01/1999 Time : 14:39:04 Page 0007
  Event : DFHINITIAL
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
  Event : T1
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
    Timer     : TIMER_ONE
  Event : T2
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
    Timer     : TIMER_TWO
  Event : T3
    Type      : Activity
    Fired     : No
    Reattach  : Yes
    Retrieve  : No
    Subevent  : No
    Timer     : TIMER_3
  Timer : TIMER_ONE
    Status    : Unexpired
    Date      : 05/11/1998 Time : 10:23:46
    Event     : T1
  Timer : TIMER_TWO
    Status    : Unexpired
    Date      : 08/11/1998 Time : 10:23:49
    Event     : T2
  Timer : TIMER_3
    Status    : Unexpired
    Date      : 27/11/1998 Time : 10:23:52
    Event     : T3

Containers

  No Containers
```

Figure 89. Example output from the DFHBARUP utility (Part 7)

```
CICS Business Transaction Services - Repository File Report                          Date : 29/01/1999 Time : 14:39:04 Page 0008
Process : SALES1234567890                          Process Type : SALES
Root Id : ..GBIBMIYA.IYCWTC37........DFHROOT
          11CCCCDCEC4CECEECFF44042F00CCCDDDE4444444444
          A172924981B98363337BB0C1B0146896630000000000
Audit Level : Full (Off, Pro, Act, Full)
Audit Log   : BAMAUDIT
000000   D740D7E3 E8D7C5F1 4040D7D9 D6C36DC6 D6E4D940 40404040 40404040 40404040  *P SALES   SALES1234567890       *
000020   40404040 40404040 40404040 40400000 00008004 00004000 00003E14 00A00000  *                ........ ........*
000040   6EC4C6C8 C2C1D7D9 D6C3C5E2 00000000 00000000 00000000 00000000 C1401A11  *>DFHBAPROCES.................A ..*
000060   C7C2C9C2 D4C9E8C1 4BC9E8C3 E6E3C3F3 F74B4B00 4C21FB00 01C4C6C8 D9D6D6E3  *GBIBMIYA.IYCWTC37...<....DFHROOT*
000080   40404040 40404040 40400000 0000776F 10C2003C 00000003 000084A9 00000000  *          .....?.B........dz....*
0000A0   00000000 00000000 00000000 00000000 00000000 108F90AC 108F90AC 00000000  *................................*
0000C0   00000000 10146C90 10146710 03C2C1D4 C1E4C4C9 E3000001 00000000          Containers
Container Name : Container_one   Container Length : x'00008000'

000000   FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  *................................*
       .
       .
       .
        lines omitted
       .
       .
       .

007FE0   FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  *................................*
```

Figure 90. Example output from the DFHBARUP utility (Part 8)

```
CICS Business Transaction Services - Repository File Report                          Date : 29/01/1999 Time : 14:39:04 Page 0009
Container Name : Container_two   Container Length : x'00000400'

000000   22222222 22222222 22222222 22222222 22222222 22222222 22222222 22222222  *................................*
       .
        lines omitted
       .
0003E0   22222222 22222222 22222222 22222222 22222222 22222222 22222222 22222222  *................................*
Container Name : Container_three Container Length : x'00000019'

000000   10101010 10101010 10101010 10101010 10101010 10101010 10                *.........................      *
```

Figure 91. Example output from the DFHBARUP utility (Part 9)

Note: A DFHBARUP report shows activity identifiers in the form they are stored on the repository. Unlike the activity identifiers returned by commands such as ASSIGN and GETNEXT ACTIVITY, those shown by DFHBARUP are not prefixed with the CICS file name of the repository.

BTS messages

You can use BTS messages to help you understand and respond to BTS activities and issues.

BTS messages are identified by the following prefixes:

- DFHBA - Business application manager messages.

- DFHEM - Event manager messages.
- DFHSH - Scheduler services messages.

All CICS messages, including BTS messages, are listed in CICS messages. To discover the meaning of a particular message, refer to that book.

Setting trace levels for BTS

Using system initialization parameters or the CETR transaction, you can use the CICS component codes to specify the level of standard and special tracing that you require for your BTS processes.

About this task

BTS consists of three CICS domains:

Domain name	CICS Component code
Business application manager	BA
Event manager	EM
Scheduler services	SH

See, Trace entries overview for details of all BTS trace points.

For detailed information about using component codes to set the level of tracing to be applied to particular CICS components, see Component names and abbreviations.

Defining tracing levels at system initialization

You can use these system initialization parameters to define, at system startup, the level of tracing that you require for BTS.

You can code any of the following parameters to define, at CICS system initialization time, the level of tracing required for BTS:

- SPCTR system initialization parameter to indicate the level of special tracing required for CICS as a whole.
- SPCTRBA to specify the level of special tracing required for the BTS business application manager domain.
- SPCTREM to specify the level of special tracing required for the BTS event manager domain.
- SPCTRSH to specify the level of special tracing required for the BTS scheduler services domain.
- STNTR system initialization parameter to indicate the level of standard tracing required for CICS as a whole.
- STNTRBA to specify the level of standard tracing required for the BTS business application manager domain.
- STNTREM to specify the level of standard tracing required for the BTS event manager domain.
- STNTRSH to specify the level of standard tracing required for the BTS scheduler services domain.

For more information about system initialization parameters, see CICS system initialization in the IBM Knowledge Center.

Defining tracing levels when CICS is running

You can use the CETR transaction to dynamically define, on a running CICS system, the level of tracing that you require for BTS. Use this CETR screen as an example.

Figure 92 on page 141 shows you what the CETR Component Trace Options screen looks like. To change trace options, you overtype the settings shown on the screen, and then press ENTER.

```
CETR                    Component Trace Options
Overtype where required and press ENTER.                    PAGE 1 OF 2
Component Standard                          Special
--------  --------------------------------  --------------------------
  AP      1                                 1-2
  BA      1                                 1-2
  BF      1                                 OFF
  BM      1                                 OFF
  BR      1                                 1-2
  CP      1                                 1-2
  DC      1                                 OFF
  DD      1                                 1-2
  DI      1                                 1
  DM      1                                 1-2
  DS      1                                 1-2
  DU      1                                 1-2
  EI      1                                 1
  EM      1                                 1-2
  FC      1                                 1-2
  GC      1                                 1-2
  IC      1                                 1

PF:  1=Help   3=Quit   7=Back  8=Forward  9=Messages   ENTER=Change
```

Figure 92. CETR screen for specifying component trace options

With the settings shown, BTS trace entries are made as follows:

- With standard task tracing in effect, from level-1 trace points.
- With special task tracing in effect, from both level-1 and level-2 trace points.

For detailed information about the CETR transaction, see CETR - trace control in the IBM Knowledge Center.

Extracting BTS information from a CICS system dump

You can use dump formatting keywords to extract BTS information from a CICS system dump.

For information about the dump formatting keywords used to extract BTS information from a CICS system dump, see the Using dumps in Problem Determination.

Chapter 6. BTS application programming commands

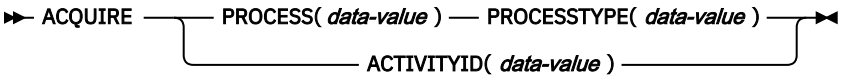
The following CICS API commands support BTS.

Restriction: None of these commands are threadsafe.

ACQUIRE

Acquire access to a BTS activity from outside the process that contains it.

ACQUIRE PROCESS



Conditions: ACTIVITYBUSY, ACTIVITYERR, INVREQ, IOERR, LOCKED, NOTAUTH, PROCESSBUSY, PROCESSERR

Description

ACQUIRE enables a program that is executing outside a particular BTS process to access an activity within the process. It allows the program to:

- Read and write to the activity's data-containers
- Issue various commands, such as RUN and LINK, against the activity.[1]

An activity that a program gains access to by means of an ACQUIRE command is known as an **acquired activity**. A program can acquire only one activity per unit of work. The activity remains acquired until the next syncpoint.

ACQUIRE ACTIVITYID acquires the specified descendant (non-root) activity.

ACQUIRE PROCESS acquires the root activity of the specified process.

Note: When a program defines a process, it is automatically given access to the process's root activity. (This enables the defining program to access the process containers and root activity containers before running the process.) When a program gains access to a root activity by means of *either* a DEFINE PROCESS or an ACQUIRE PROCESS command, the process is known as the **acquired process**.

Rules

1. A program can acquire only one activity within the same unit of work. The activity remains acquired until the next syncpoint. This means, for example, that a program:
   - Cannot issue both a DEFINE PROCESS and an ACQUIRE PROCESS command within the same unit of work.
   - Cannot issue both an ACQUIRE PROCESS and an ACQUIRE ACTIVITYID command within the same unit of work. That is, it can acquire *either* a descendant activity or a root activity, not one of each.
2. If a program is executing as an activation of an activity, it cannot:
   - Acquire an activity in the same process as itself. It cannot, for example, issue ACQUIRE PROCESS for the current process.
   - Use a LINK command to activate the activity that it has acquired.

[1] If the acquired activity is a root activity, against the process.

3. An acquired activity's process is accessible in the same way as the activity itself can access it. Thus, if the acquired activity is a descendant activity:
   - Its process's containers may be read but not updated.
   - The process may not be the subject of any command—such as RUN, LINK, SUSPEND, RESUME, or RESET—that directly manipulates the process or its root activity.

   Conversely, if the acquired activity is a root activity:
   - Its process's containers may be both read and updated.
   - The process may be the subject of commands such as RUN, LINK, SUSPEND, RESUME, or RESET. The ACQPROCESS keyword on the command identifies the subject process as the one the program that issues the command has acquired in the current unit of work.

Options

ACTIVITYID(data-value)
specifies the identifier (1–52 characters) of the descendant activity to be acquired.

PROCESS(data-value)
specifies the name (1–36 characters) of the process whose root activity is to be acquired.

PROCESSTYPE(data-value)
specifies the process-type (1–8 characters) of the process whose root activity is to be acquired.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

109 ACTIVITYERR
RESP2 values:

8
The activity referred to by the ACTIVITYID option could not be found.

16 INVREQ
RESP2 values:

22
The unit of work that issued the ACQUIRE command has already acquired an activity; a unit of work can acquire only one activity.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access the file associated with the BTS repository data set on which details of the process are stored.

106 PROCESSBUSY
RESP2 values:

13
The request timed out. It may be that another task using this process-record has been prevented from ending.

108 PROCESSERR
RESP2 values:

5
The process named in the PROCESS option could not be found.

9
The process-type named in the PROCESSTYPE option could not be found.

Usage examples

ACQUIRE ACTIVITYID can be used to implement user-related activities. For example, on its first activation an activity might:

1. Define an input event to represent a particular user-interaction
2. Issue an ASSIGN command to obtain the identifier of its own activity-instance
3. Save the input event and activity identifier on a data base
4. Return without completing.

Later, when a user is ready to process the work represented by the activity, he or she starts a transaction. This transaction, which executes outside the BTS process:

1. Retrieves the input event and activity identifier from the data base
2. Uses the ACQUIRE ACTIVITYID command to acquire access to the activity
3. Places the information required to complete the activity in an input data-container, and runs the activity. The INPUTEVENT option of the RUN command tells the activity why it is being activated.

ACQUIRE PROCESS can be used to implement client/server processing. For example, a client program might use the DEFINE PROCESS and RUN commands to create and run a server process, which carries out some work, defines one or more input events, and returns without completing. The client issues a syncpoint or returns. To run the same server process again, the client uses the ACQUIRE PROCESS and RUN commands.

ADD SUBEVENT

Add a sub-event to a BTS composite event.

ADD SUBEVENT

Conditions: EVENTERR, INVREQ

Description

ADD SUBEVENT adds a sub-event to a BTS composite event. The sub-event:

- Must be an atomic (not a composite) event
- Cannot be a system event
- Must not currently be part of a composite event
- Cannot, if the predicate of the composite event uses the AND Boolean operator, be an input event.

Adding a sub-event causes the composite's predicate to be re-evaluated.

Options

EVENT(data-value)
specifies the name (1–16 characters) of the composite event. This must previously have been defined to the current activity, using the DEFINE COMPOSITE EVENT command.

SUBEVENT(data-value)
specifies the name (1–16 characters) of the atomic event to be added to the composite event as a sub-event. The sub-event must previously have been defined to the current activity, using one of the following commands:

- DEFINE ACTIVITY
- DEFINE INPUT EVENT
- DEFINE TIMER

It:

- Must not currently be part of a composite event
- Cannot, if the predicate of the composite event uses the AND Boolean operator, be an input event.

Conditions

111 EVENTERR
RESP2 values:

4
The event specified on the EVENT option is not recognized by BTS.

5
The sub-event specified on the SUBEVENT option is not recognized by . BTS.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of an activity.

2
The event specified on the EVENT option is invalid—it is not a composite event.

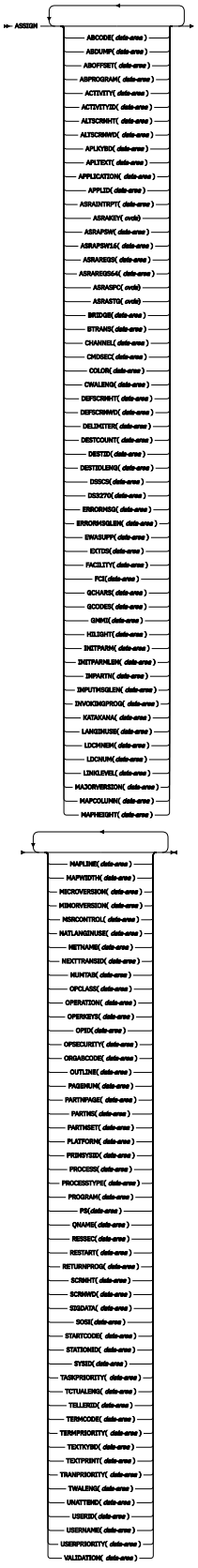
3
The sub-event specified on the SUBEVENT option is invalid. Specifying any of the following as a sub-event produces this error:

- A composite event
- A system event
- A sub-event of another composite event
- A sub-event of this composite event—that is, an atomic event that has already been added to this composite event
- An input event, if the composite uses the AND Boolean operator.

ASSIGN

Request values from outside the local environment of the application program.

ASSIGN



Condition: INVREQ

This command is threadsafe.

Description

The **ASSIGN** command gets values from outside the local environment of the application program. The data obtained depends on the specified options. You can specify up to 16 options in one **ASSIGN** command. When you specify multiple options on the ASSIGN command, if any of the specified options fails with the exception condition INVREQ, the other specified options are still populated with the requested information.

For options that apply to terminals or terminal-related data, the reference is always to the principal facility.

If the principal facility is a remote terminal, the data returned is obtained from the local copy of the information; the request is not routed to the system to which the remote terminal is attached.

Transaction routing is, as far as possible, transparent to the **ASSIGN** command. In general, the values returned are the same whether the transaction is local or remote.

For more details on these options, see Developing in an intersystem environment.

Options

ABCODE(*data-area*)
Returns a 4-character current abend code. Abend codes are documented in Transaction abend codes. If an abend has not occurred, the variable is set to blanks.

ABDUMP(*data-area*)
Returns a 1-byte value. X'FF' indicates that an **EXEC CICS ABEND ABCODE** command was issued without the NODUMP option and that ABCODE contains an abend code. X'00' indicates that either no dump was produced, or ABCODE contains blanks.

ABOFFSET(*data-area*)
Returns a fullword binary offset in bytes of an abend when the latest abend with a code ASRA, ASRB, or ASRD occurred. If the abend is outside the current program, a value of X'FFFFFFFF' is returned. The data area is set to binary zeros if no ASRA, ASRB, or ASRD abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server program.

ABPROGRAM(*data-area*)
Returns an 8-character name of the failing program for the latest abend.

If the abend originally occurred in a DPL server program running in a remote system, ABPROGRAM returns the DPL server program name.

This field is set to binary zeros if it is not possible to determine the failing program at the time of the abend.

When the latest abend is an APCT (resulting from an unsuccessful attempt to load a program, mapset or partitionset), the name is taken from the program, mapset, or partitionset that was not loaded.

ACTIVITY(*data-area*)
Returns, if this program is running on behalf of a CICS business transaction services (BTS) activity, the 16-character name of the activity.

BTS is described in Overview of BTS.

ACTIVITYID(*data-area*)
Returns, if this program is running on behalf of a BTS activity, the 52-character, CICS-assigned, identifier of the activity-instance.

If a program that is running outside the current process wants to acquire control of this activity-instance, it must specify this identifier on an **ACQUIRE ACTIVITYID** command.

BTS is described in Overview of BTS.

ALTSCRNHT(*data-area*)
Returns the alternate screen height defined for the terminal as a halfword binary variable. If the task is not initiated from a terminal, INVREQ occurs.

ALTSCRNWD(*data-area*)
Returns the alternate screen width defined for the terminal as a halfword binary variable. If the task is not initiated from a terminal, INVREQ occurs.

APLKYBD(*data-area*)
Returns a 1-byte indicator that shows whether the terminal keyboard has the APL keyboard feature (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

APLTEXT(*data-area*)
Returns a 1-byte indicator that shows whether the terminal keyboard has the APL text feature (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

APPLICATION(*data-area*)
Returns the 64 character name of the current application associated with the task. It is part of the application context that is made up of the application name, the platform name, the operation name and the major, minor and micro version number of the application. If there is no application context associated with the task, then blanks are returned.

APPLID(*data-area*)
Returns an 8-character APPLID of the CICS system that owns the transaction.

If your system is using XRF, the value returned is the generic APPLID. An application program is unaffected by a takeover from the active to the alternate.

ASRAINTRPT(*data-area*)
Returns an 8-character data-area that contains the ILC (instruction length code) and the PIC (program interrupt code) at the point when the latest abend with a code of AICA, ASRA, ASRB, ASRD, or ASRE occurred. The field contains binary zeros if no AICA, ASRA, ASRB, ASRD, or ASRE abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server program. When valid, the contents of the 8 bytes returned are as follows:

- ILC (2 bytes binary)
- PIC (2 bytes binary)
- filler (4 bytes binary, always zero)

ASRAKEY(*cvda*)
Returns the execution key at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend, if any. CVDA values are as follows:

CICSEXECKEY
This value is returned if the task was running in CICS-key at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend. Note that if CICS subsystem storage protection is not active, all programs run in CICS key.

USEREXECKEY
This value is returned if the task was running in user-key at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend.

NONCICS
This value is returned if the execution key at the time of the last abend was not one of the CICS keys; for example, not key 8 or key 9.

NOTAPPLIC
This value is returned if there was not an AEYD, AEYF, AICA, ASRA, or ASRB abend.

ASRAPSW(*data-area*)
Returns an 8-byte data area that contains the program status word (PSW) at the point when the latest abend with a code of AICA, ASRA, ASRB, ASRD, or ASRE occurred.

The field contains binary zeros if no AICA, ASRA, ASRB, ASRD, or ASRE abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server program.

ASRAPSW16(*data-area*)
Returns a 16-byte data area that contains the 128-bit program status word (PSW) at the point when the latest abend with a code of AICA, ASRA, ASRB, ASRD, or ASRE occurred.

The field contains binary zeros if no AICA, ASRA, ASRB, ASRD, or ASRE abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server program.

ASRAREGS(*data-area*)
Returns the contents of general registers 0 - 15 at the point when the latest AICA, ASRA, ASRB, ASRD, or ASRE abend occurred.

The contents of the registers are returned in the data area (64 bytes long) in the order 0, 1, ..., 14, 15.

The data area is set to binary zeros if no AICA, ASRA, ASRB, ASRD, or ASRE abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server program.

ASRAREGS64(*data-area*)
Returns the contents of the 64-bit general registers 0 - 15 at the point when the latest AICA, ASRA, ASRB, ASRD, or ASRE abend occurred.

The contents of the registers are returned in the data area (128 bytes long) in the order 0, 1, ..., 14, 15.

The data area is set to binary zeros if no AICA, ASRA, ASRB, ASRD, or ASRE abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server program.

ASRASPC(*cvda*)
Returns the type of space in control at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend, if any. CVDA values are as follows:

SUBSPACE
This value is returned if the task was running in either its own subspace or the common subspace at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend.

BASESPACE
This value is returned if the task was running in the base space at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend. All tasks run in base space if transaction isolation is not active.

NOTAPPLIC
This value is returned if there was not an AEYD, AEYF, AICA, ASRA, or ASRB abend.

ASRASTG(*cvda*)
Returns the type of storage being addressed at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend, if any. The CVDA values are as follows:

CICS
This value is returned if the storage being addressed is CICS-key storage. This can be in one of the CICS dynamic storage areas (CDSA, ECDSA, ETDSA, or GCDSA). This can be in one of the read-only dynamic storage areas (RDSA or ERDSA) when CICS is running with the NOPROTECT option on the **RENTPGM** system initialization parameter, or when storage protection is not active.

USER
This value is returned if the storage being addressed is user-key storage in one of the user dynamic storage areas (UDSA, EUDSA, or GUDSA).

READONLY
This value is returned if the storage being addressed is read-only storage in one of the read-only dynamic storage areas (RDSA or ERDSA) when CICS is running with the PROTECT option on the **RENTPGM** system initialization parameter.

NOTAPPLIC
This value is returned in the following conditions:

- There is no AEYD, AEYF, AICA, ASRA, or ASRB abend found for this task.
- The affected storage in an abend is not managed by CICS.
- The ASRA abend is not caused by an 0C4 abend.

BRIDGE(*data-area*)
Returns the 4-character TRANSID of the bridge monitor transaction that issued a **START BREXIT TRANSID** command to start the user transaction that issued this command. Blanks are returned in the following situations:

- The user transaction was not started by a bridge monitor transaction.
- This command was issued by a program started by a distributed program link (DPL) request.

Note: If the **START BREXIT** command was issued from a bridge exit, the TRANSID returned is the that of the bridge monitor that issued a **START BREXIT** naming the bridge exit.

BTRANS(*data-area*)
Returns a 1-byte indicator that shows whether the terminal is defined as having the background transparency capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

CHANNEL(*data-area*)
Returns the 16-character name of the current channel of the program, if one exists; otherwise blanks.

CMDSEC(*data-area*)
Returns a 1-byte indicator that shows whether command security checking is defined for the current task. (X for yes, blank for no.)

COLOR(*data-area*)
Returns a 1-byte indicator that shows whether the terminal is defined as having the extended color capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

CWALENG(*data-area*)
Returns a halfword binary field that indicates the length of the common work area (CWA). If no CWA exists, a zero length is returned.

DEFSCRNHT(*data-area*)
Returns a halfword binary variable that contains the default screen height defined for the terminal. If the task is not initiated from a terminal, INVREQ occurs.

DEFSCRNWD(*data-area*)
Returns a halfword binary variable that contains the default screen width defined for the terminal. If the task is not initiated from a terminal, INVREQ occurs.

DELIMITER(*data-area*)
Returns a 1-byte data-link control character for a 3600. Possible values are as follows:

X'80'
Input ended with end-of-text (ETX).

X'40'
Input ended with end-of-block (ETB).

X'20'
Input ended with inter-record separator (IRS).

X'10'
Input ended with start of header (SOH).

X'08'
Transparent input.

If the task is not initiated from a terminal, INVREQ occurs.

DESTCOUNT(*data-area*)
Returns a halfword binary field. This option has the following uses:

- Following a BMS **ROUTE** command, it shows that the value required is the number of different terminal types in the route list, and hence the number of overflow control areas that might be required.
- Within BMS overflow processing, it shows that the value required is the relative overflow control number of the destination that has encountered overflow. If this option is specified when overflow

processing is not in effect, the value obtained is meaningless. If no BMS commands have been issued, INVREQ occurs.

DESTID(*data-area*)
Returns an 8-byte identifier of the outboard destination, padded with blanks on the right to eight characters. If this option is specified before a batch data interchange command is issued in the task, INVREQ occurs.

DESTIDLENG(*data-area*)
Returns a halfword binary length of the destination identifier obtained by DESTID. If this option is specified before a batch data interchange command is issued in the task, INVREQ occurs.

DSSCS(*data-area*)
Returns a 1-byte indicator that shows whether the principal facility is a basic SCS data stream device (X'FF') or not (X'00').

If the task is not initiated from a terminal, INVREQ occurs.

DS3270(*data-area*)
Returns a 1-byte indicator that shows whether the principal facility is a 3270 data stream device (X'FF') or not (X'00').

If the task is not initiated from a terminal, INVREQ occurs.

ERRORMSG(*data-area*)
Returns the error message up to a maximum of 500 bytes that is currently referenced in the transaction abend control block for the CICS task. Following a failure of a DPL request, the message is that returned from the remote system. For messages shorter than 500 bytes the message is padded with nulls.

If no message is present, the 500 byte area contains nulls.

ERRORMSGLEN(*data-area*)
Returns a halfword binary value representing the length of the message returned for ERRORMSG. If the message referenced in the transaction abend control block exceeds 500 bytes, the message is truncated and the length is set to 500.

If no message is present the length returned is 0.

EWASUPP(*data-area*)
Returns a 1-byte indicator that shows whether Erase Write Alternative is supported (X'FF') or not (X'00').

If the task is not initiated from a terminal, INVREQ occurs.

EXTDS(*data-area*)
Returns a 1-byte indicator that shows whether the terminal accepts the 3270 extended data stream, (X'FF') or not (X'00'). Extended data stream capability is required for a terminal that supports the query feature, color, extended highlighting, programmed symbols or validation. A terminal that accepts the query structured field command also has this indicator set. If extended data stream is on, the device supports the write structured field COMMAND and Outbound Query Structured field.

For guidance information about query structured fields, see the IBM 3270 Data Stream Programmers Reference.

If the task is not initiated from a terminal, INVREQ occurs.

FACILITY(*data-area*)
Returns a 4-byte identifier of the principal facility that initiated the transaction issuing this command. If this option is specified, and there is no allocated facility, INVREQ occurs.

Note: You can use the QNAME option to get the name of the transient data intrapartition queue if the transaction was initiated by expiry of a transient data trigger level.

FCI(*data-area*)
Returns a 1-byte facility control indicator. For more information, see Codes returned by ASSIGN. This indicates the type of facility associated with the transaction; for example, X'01' indicates a terminal or logical unit. The obtained value is always returned.

GCHARS(*data-area*)
Returns a halfword binary graphic character set global identifier (the GCSGID). The value is a number in the range 1 through 65534 representing the set of graphic characters that can be input or output at the terminal. If the task is not initiated from a terminal, INVREQ occurs.

GCODES(*data-area*)
Returns a halfword binary code page global identifier (the CPGID). The value is a number in the range 1 through 65534 representing the EBCDIC or ASCII code page defining the code points for the characters that can be input or output at the terminal. If the task is not initiated from a terminal, INVREQ occurs.

GMMI(*data-area*)
Returns a 1-byte indicator that shows whether a "good morning" message applies to the terminal associated with the running transaction (X'FF') or not (X'00'). If this option is specified and the current task is not associated with a terminal, the INVREQ condition occurs.

HILIGHT(*data-area*)
Returns a 1-byte indicator that shows whether the terminal is defined as having the extended highlight capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

INITPARM(*data-area*)
Returns the 60-character data-area that contains any initialization parameters that are specified for the program on the **INITPARM** system initialization parameter. The values are only returned if the name of the program that issues the command matches a program name that is specified on the **INITPARM** system initialization parameter. If there are no parameters for the program, the area is not updated and its contents are undefined. Use INITPARMLEN with **INITPARM** to determine whether a parameter was specified or not; the value that is returned by **ASSIGN INITPARM** alone cannot be used to indicate whether a system initialization parameter was specified.

INITPARMLEN(*data-area*)
Returns a halfword binary length of the INITPARM. If there is no parameter for it, INITPARMLEN contains binary zeros.

INPARTN(*data-area*)
Returns the 1- or 2-character name of the most recent input partition. If no map is yet positioned, or if BMS routing is in effect, or if the task is not initiated from a terminal, INVREQ occurs.

INPUTMSGLEN(*data-area*)
Returns a halfword binary length of the terminal input string, in bytes. If there is no terminal input data, a length of zero is returned.

INVOKINGPROG(*data-area*)
Returns the 8-character name of the application program that used the **LINK** or **XCTL** command to link or transfer control to the current program:

- If you issue the **ASSIGN INVOKINGPROG** command in a remote program that was invoked by a distributed program link (DPL) command, CICS returns the name of the program that issued the DPL command.
- If you issue the **ASSIGN INVOKINGPROG** command in an application program at the highest level, CICS returns eight blanks.
- If you issue the **ASSIGN INVOKINGPROG** command in a user-replaceable program, a Bridge Exit program or a program list table program, CICS returns eight blanks.
- If you issue the **ASSIGN INVOKINGPROG** command from a global user exit, task-related exit, or application program linked to from such an exit, CICS returns the name of the most recent invoking program that was not a global user exit or task-related user exit.

KATAKANA(*data-area*)
Returns a 1-byte indicator that shows whether the principal facility supports Katakana (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

LANGINUSE(*data-area*)
Returns a 3-byte mnemonic code that shows the language in use. The 3-byte mnemonic has a 1:1 correspondence with the 1-byte NATLANGINUSE option. See National language codes for possible values of the code.

LDCMNEM(*data-area*)
Returns a 2-byte logical device code (LDC) mnemonic of the destination that has encountered overflow. If this option is specified when overflow processing is not in effect, the value obtained not significant. If no BMS commands have been issued, INVREQ occurs.

LDCNUM(*data-area*)
Returns a 1-byte LDC numeric value of the destination that has encountered overflow. This indicates the type of the LDC, such as printer or console. If this option is specified when overflow processing is not in effect, the value obtained is not significant.

LINKLEVEL(*data-area*)
Returns a halfword binary value representing the program link level in the local system. The topmost link level is level one and for each EXEC CICS LINK the link level is incremented by one. The link level is not incremented for a language CALL statement. If a program is the target of a DPL request, the link level returned is that within the CICS region it is executing and not the wider distributed transaction. If a program is DPLed to, then link level one will be the CICS mirror program DFHMIRS.

MAJORVERSION(*data-area*)
Returns the fullword binary value representing the major version of the current application associated with the task, which is part of the application context. If there is no application context associated with the task, then -1 is returned.

MAPCOLUMN(*data-area*)
Returns a halfword binary number of the column on the display that contains the origin of the most recently positioned map. If no map is yet positioned, or if BMS routing is in effect, or if the task is not initiated from a terminal, INVREQ occurs.

MAPHEIGHT(*data-area*)
Returns a halfword binary height of the most recently positioned map. If no map is yet positioned, or if BMS routing is in effect, or if the task is not initiated from a terminal, INVREQ occurs.

MAPLINE(*data-area*)
Returns a halfword binary number of the line on the display that contains the origin of the most recently positioned map. If no map is yet positioned, or if BMS routing is in effect, or if the task is not initiated from a terminal, INVREQ occurs.

MAPWIDTH(*data-area*)
Returns a halfword binary width of the most recently positioned map. If no map is yet positioned, or if BMS routing is in effect, or if the task is not initiated from a terminal, INVREQ occurs.

MICROVERSION(*data-area*)
Returns the fullword binary value representing the micro version of the current application associated with the task, which is part of the application context. If there is no application context associated with the task, then -1 is returned.

MINORVERSION(*data-area*)
Returns the fullword binary value representing the minor version of the current application associated with the task, which is part of the application context. If there is no application context associated with the task, then -1 is returned.

MSRCONTROL(*data-area*)
Returns a 1-byte indicator that shows whether the terminal supports magnetic slot reader (MSR) control (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

NATLANGINUSE(*data-area*)
Returns a 1-byte mnemonic code that shows the national language associated with the USERID for the current task (which could be the default USERID). Refer to the **SIGNON** command for an explanation of how this value is derived. (NATLANGINUSE does not show the system default language as specified on the **NATLANG** system initialization parameter.)

See National language codes for possible values of the code.

NETNAME(*data-area*)
Returns the 8-character name of the logical unit in the z/OS Communications Server network. If the task is not initiated from a terminal, INVREQ occurs. If the principal facility is not a local terminal, CICS no longer returns a null string but the netname of the remote terminal.

If this command was issued by a user transaction that was started by a 3270 bridge transaction, the value returned is the termid of the bridge facility.

If the CICS region supports z/OS Communications Server LU aliases, the NETNAME returned by CICS could be an LU alias, either dynamically allocated by z/OS Communications Server or predefined on the **LUALIAS** parameter of a CDRSC definition.

NEXTTRANSID(*data-area*)
Returns the 4-character next transaction identifier as set by **SET NEXTTRANSID** or **RETURN TRANSID**. It returns blanks if there are no more transactions.

NUMTAB(*data-area*)
Returns a 1-byte number of the tabs required to position the print element in the correct passbook area of the 2980. If the task is not initiated from a terminal, INVREQ occurs.

OPCLASS(*data-area*)
Returns, in a 24-bit string, the operator class used by BMS for routing terminal messages, as defined in the CICS segment of the External Security Manager.

OPERATION(*data-area*)
Returns the 64 character name of the current operation associated with the task, which is part of the application context. If there is no application context associated with the task, then blanks are returned.

OPERKEYS(*data-area*)
This option is accepted for compatibility with previous releases. If specified, a 64-bit null string is returned.

OPID(*data-area*)
Returns the 3-character operator identification. This is used by BMS for routing terminal messages, as defined in the CICS segment of the External Security Manager.

If the task is initiated from a remote terminal, the OPID returned by this command is not necessarily that associated with the user that is signed on at the remote terminal. If you want to know the OPID of the signed on user, use the INQUIRE TERMINAL system programming command.

The OPID can also be different from the OPID of the user currently signed on, if it was changed with the SET TERMINAL command.

OPSECURITY(*data-area*)
This option is accepted for compatibility with previous releases. If specified, a 24-bit null string is returned.

ORGABCODE(*data-area*)
Returns a 4-byte original abend code in cases of repeated abends.

OUTLINE(*data-area*)
Returns a 1-byte indicator that shows whether the terminal is defined as having the field outlining capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

PAGENUM(*data-area*)
Returns a halfword binary current page number for the destination that has encountered an overflow. If this option is specified when overflow processing is not in effect, the value obtained is meaningless. If no BMS commands have been issued, INVREQ occurs.

PARTNPAGE(*data-area*)
Returns a 2-byte name of the partition that most recently caused page overflow. If no BMS commands have been issued, INVREQ occurs.

PARTNS(*data-area*)
Returns a 1-byte indicator that shows whether the terminal supports partitions (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

PARTNSET(*data-area*)
Returns the name (1–6 characters) of the application partition set. A blank value is returned if there is no application partition set. If the task is not initiated from a terminal, INVREQ occurs.

PLATFORM(*data-area*)
Returns the 64 character name of the platform associated with the task, which is part of the application context. If there is no application context associated with the task, then blanks are returned.

PRINSYSID(*data-area*)
Returns the 4-character name by which the other system is known in the local system; that is, the CONNECTION definition that defines the other system. For a single-session APPC device defined by a terminal definition, the returned value is the terminal identifier.

This only applies when the principal facility is one of the following:

- An MRO session to another CICS system
- An LU6.1 session to another CICS or IMS system
- An APPC session to another CICS system, or to another APPC system or device

If the principal facility is not an MRO, LU6.1, or APPC session, or if the task has no principal facility, INVREQ occurs.

Note: Special considerations apply generally when transaction routing. In particular an **ASSIGN PRINSYSID** command cannot be used in a routed transaction to find the name of the terminal-owning region. For more information, see CICS transaction routing.

PROCESS(*data-area*)
Returns, if this program is running on behalf of a CICS business transaction services (BTS) activity, the 36-character name of the BTS process that contains the activity.

BTS is described in Overview of BTS.

PROCESSTYPE(*data-area*)
Returns, if this program is running on behalf of a BTS activity, the 8-character process type of the BTS process that contains the activity.

BTS is described in Overview of BTS.

PROGRAM(*data-area*)
Returns an 8-character name of the currently running program.

PS(*data-area*)
Returns a 1-byte indicator that shows whether the terminal is defined as having the programmed symbols capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

QNAME(*data-area*)
Returns a 4-character name of the transient data intrapartition queue that caused this task to be initiated by reaching its trigger level. If the task is not initiated by automatic transaction initiation (ATI), INVREQ occurs.

RESSEC(*data-area*)
Returns a 1-byte indicator that shows whether resource security checking is defined for the transaction running. (X for yes, blank for no.)

RESTART(*data-area*)
Returns a 1-byte indicator that shows whether a restart of the task (X'FF'), or a normal start of the task (X'00'), has occurred.

RETURNPROG(*data-area*)
Returns the 8-character name of the program to which control is to be returned when the current program has finished running. The values returned depend on how the current program was given control, as follows:

- If the current program was invoked by a **LINK** command, including a distributed program link, RETURNPROG returns the same name as INVOKINGPROG.

- If the current program was invoked by an **XCTL** command, RETURNPROG returns the name of the application program in the chain that last issued a LINK command.

  If the program that invoked the current program with an **XCTL** command is at the highest level, CICS returns eight blanks.
- If the **ASSIGN RETURNPROG** command is issued in the program at the top level, CICS returns eight blanks.
- If the **ASSIGN RETURNPROG** command is issued in a user-replaceable module, or a program list table program, CICS returns eight blanks.
- If the **ASSIGN RETURNPROG** command is issued in a global user exit, task-related exit, or application program linked to from such an exit, CICS returns the name of the program that control is returned to when all intermediate global user exit and task-related user exit programs have completed.

Example:

```
Program A links to program B
Program B links to program C
Program C uses XCTL to transfer control to program D
Program D issues an ASSIGN RETURNPROG command, and CICS returns the name of Program B.
```

SCRNHT(*data-area*)
Returns a halfword binary variable that contains the height of the 3270 screen defined for the current task. If the task is not initiated from a terminal, INVREQ occurs.

SCRNWD(*data-area*)
Returns a halfword binary variable that contains the width of the 3270 screen defined for the current task. If the task is not initiated from a terminal, INVREQ occurs.

SIGDATA(*data-area*)
Returns a 4-byte character string that contains the inbound signal data received from a logical unit. If the task is not initiated from a terminal, INVREQ occurs.

SOSI(*data-area*)
Returns a 1-byte indicator that shows whether the terminal is defined as having the mixed EBCDIC/DBCS fields capability (X'FF') or not (X'00'). The DBCS subfields within an EBCDIC field are delimited by SO (shift-out) and SI (shift-in) characters. If the task is not initiated from a terminal, INVREQ occurs.

STARTCODE(*data-area*)
Returns a 2-character value that indicates how the transaction that issued the request was started. Possible values are as follows:

Code
Transaction started by

D
A distributed program link (DPL) request that did not specify the SYNCONRETURN option. The task cannot issue I/O requests against its principal facility, or any sync point requests.

DS
A distributed program link (DPL) request, as in code D, that did specify the SYNCONRETURN option. The task can issue sync point requests.

QD
Transient data trigger level.

S
START command that did not pass data in the FROM option. It might or might not have passed a channel.

SD
START command that passed data in the FROM option.

SZ
FEPI START command.

TD
Terminal input or permanent transid.

U
User-attached task.

STATIONID(*data-area*)
Returns a 1-byte station identifier of a 2980. If the task is not initiated from a terminal, INVREQ occurs.

SYSID(*data-area*)
Returns the 4-character name given to the local CICS system. This value can be specified in the SYSID option of a file control, interval control, temporary storage, or transient data command, in which case the resource to be accessed is assumed to be on the local system.

TASKPRIORITY(*data-area*)
Returns a halfword binary filed that indicates the current priority of the issuing task (0–255). When the task is first attached, this is the sum of the user, terminal, and transaction priorities. This value can be changed during execution by a **CHANGE TASK** command.

TCTUALENG(*data-area*)
Returns a halfword binary length of the terminal control table user area (TCTUA). If no TCTUA exists, a zero length is returned.

TELLERID(*data-area*)
Returns a 1-byte teller identifier of a 2980. If the task is not initiated from a terminal, INVREQ occurs.

TERMCODE(*data-area*)
Returns a 2-byte code giving the type and model number of the terminal associated with the task.

The first byte is a code identifying the terminal type, derived from the TERMINAL resource. For a description of the resource attributes, see TERMINAL attributes . The second byte is a single-character model number as specified in the TERMMODEL attribute.

The meanings of the type codes are given in Codes returned by ASSIGN.

TERMPRIORITY(*data-area*)
Returns a halfword binary terminal priority (0–255).

TEXTKYBD(*data-area*)
Returns a 1-byte indicator that shows whether the principal facility supports TEXTKYBD (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

TEXTPRINT(*data-area*)
Returns a 1-byte indicator that shows whether the principal facility supports TEXTPRINT (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

TRANPRIORITY(*data-area*)
Returns a halfword binary transaction priority (0–255).

TWALENG(*data-area*)
Returns a halfword binary length of the transaction work area (TWA). If no TWA exists, a zero length is returned.

UNATTEND(*data-area*)
Returns a 1-byte indicator that shows whether the mode of operation of the terminal is unattended, that is to say that no person is attending the terminal. These indicators are X'FF' for unattended and X'00' for attended. If the task is not initiated from a terminal, INVREQ occurs.

USERID(*data-area*)
Returns an 8-byte user ID of the signed-on user. If no user is explicitly signed on, CICS returns the default user ID. Special considerations apply if you are using an intercommunication environment. See the Getting started with intercommunication for more information about the **ASSIGN** command for LUTYPE6.1, APPC, and MRO.

USERNAME(*data-area*)
Returns a 20-character name of the user obtained from the external security manager (ESM).

USERPRIORITY(*data-area*)
Returns a halfword binary operator priority (0–255).

VALIDATION(*data-area*)
Returns a 1-byte indicator that shows whether the terminal is defined as having the validation capability (X'FF') or not (X'00'). Validation capability consists of the mandatory fill, mandatory enter, and trigger attributes. If the task is not initiated from a terminal, INVREQ occurs.

Conditions

The ASSIGN command always returns the exception condition INVREQ under CECI or in a REXX program. Even though CECI or the REXX program might return the information you requested correctly, it also attempts to get information from other options, some of which are invalid.

16 INVREQ
RESP2 values:

2
No BMS command has yet been issued, BMS routing is in effect, or no map has yet been positioned.

3
No batch data interchange (BDI) command has yet been issued.

4
The task is not initiated by automatic transaction initiation (ATI).

5
The task is not associated with a terminal; or the task has no principal facility; or the principal facility is not an MRO, LU6.1, or APPC session.

6
A CICS BTS request was issued from outside the CICS BTS environment. Therefore, the transaction is not running on behalf of a BTS activity.

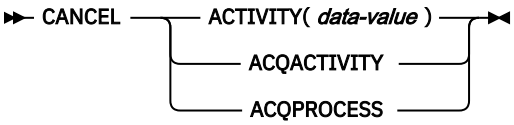
200
Command syntax options are not allowed in a server program invoked by a distributed program link.

Default action: terminate the task abnormally.

CANCEL (BTS)

Cancel a BTS activity or process.

CANCEL (BTS)



Conditions: ACTIVITYBUSY, ACTIVITYERR, INVREQ, IOERR, LOCKED, NOTAUTH, PROCESSBUSY, PROCESSERR

Description

CANCEL (BTS) forces a BTS activity or process, and all its descendant activities, into COMPLETE mode.

Options

ACQACTIVITY
specifies that the activity to be canceled is the one that the current unit of work has acquired by means of an ACQUIRE ACTIVITYID command.

ACQPROCESS
specifies that the process that the current unit of work has acquired is to be canceled.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the child activity to be canceled.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
One or more of the descendant activities of the activity to be canceled are inaccessible or in CANCELLING mode.

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

14
The activity to be canceled is not in INITIAL or DORMANT mode.

16 INVREQ
RESP2 values:

4
The ACTIVITY option was used to name a child activity, but the command was issued outside the scope of a currently-active activity.

15
The ACQPROCESS option was used, but the issuing task has not acquired a process.

24
The ACQACTIVITY option was used, but the issuing task has not acquired an activity.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access the file associated with the BTS repository data set on which details of the process or activity are stored.

106 PROCESSBUSY
RESP2 values:

13
One or more of the activities that make up the process to be canceled are inaccessible or in CANCELLING mode.

108 PROCESSERR
RESP2 values:

9
The process—type could not be found.

14
The process to be canceled is not in INITIAL, DORMANT, or COMPLETE mode.

Activities

The only activities a program can cancel are as follows:

- If it is running as the activation of an activity, its own child activities. It can cancel several of its child activities within the same unit of work.
- The activity it has acquired, by means of an ACQUIRE ACTIVITYID command, in the current unit of work.

To be canceled successfully, an activity must be in INITIAL or DORMANT mode. CICS tries to cancel activities synchronously. However, if one or more descendant activities of the activity to be canceled are inaccessible (due, for example, to the failure of a communications link):

- The subtree of descendant activities is canceled asynchronously.
- The activity to be canceled is placed in CANCELLING mode.

The completion event associated with a canceled activity is not deleted from the parent's event pool. On normal completion of this command, the activity still exists, and can be reset and run again, if necessary.

When an acquired activity is canceled, its parent is reactivated because of the firing of the canceled activity's completion event.

Processes

The only process a program can cancel is the one it has acquired in the current unit of work. If it does so, it cannot acquire another process within the current unit of work.

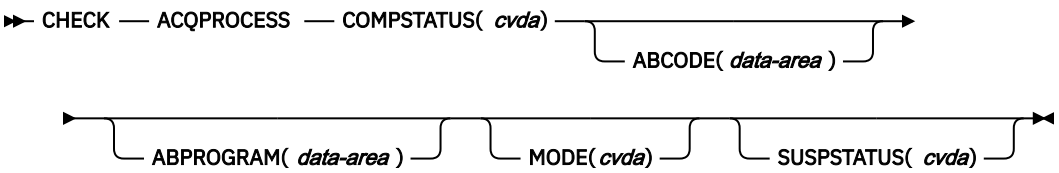
To be canceled successfully, a process must be in INITIAL, DORMANT, or COMPLETE mode.

CICS tries to cancel the process synchronously, in the way described for activities.

CHECK ACQPROCESS

Check the completion status of a BTS process.

CHECK ACQPROCESS



Conditions: INVREQ

Description

CHECK ACQPROCESS returns the completion status of the currently-acquired BTS process. Typically, it is used to check the success of a previous RUN ACQPROCESS or LINK ACQPROCESS command. It allows the requestor to discover whether the process completed successfully, or whether, for example, it needs to be reactivated in order to complete its processing.

The only process a program can check is the one that it has acquired in the current unit of work - see Acquiring processes and activities .

The RESP and RESP2 options on this command reflect whether the command is understood by CICS - for example, PROCESSERR occurs if the process is not currently acquired by the requestor.

The COMPSTATUS option returns a CVDA value indicating the completion status of the process's root activity - for example, NORMAL is returned if the root activity has successfully completed all its processing steps, while INCOMPLETE is returned if it has returned from an activation but needs to be reattached in order to complete its processing.

Options

ABCODE(data-area)
returns, if the process's root activity terminated abnormally, the 4-character abend code.

ABPROGRAM(data-area)
returns, if the process's root activity terminated abnormally, the 8-character name of the program that was in control at the time of the abend.

ACQPROCESS
specifies that the process that is currently acquired by the requestor is to be checked.

COMPSTATUS(cvda)
indicates the completion status of the process. CVDA values are:

ABEND
The program that implements the process's root activity abended. Any children of the root activity have been canceled.

FORCED
The process was forced to complete—for example, it was canceled with a CANCEL ACQPROCESS command.

INCOMPLETE
The process is incomplete. This could mean:

- That it has not yet been run
- That it has returned from one or more activations but needs to be reattached in order to complete all its processing steps
- That it is currently active.

NORMAL
The process completed successfully.

MODE(cvda)
indicates the processing state of the process. CVDA values are:

ACTIVE
An activation of the process is running.

CANCELLING
CICS is waiting to cancel the process. A CANCEL ACQPROCESS command has been issued, but CICS cannot cancel the process immediately because one or more of the root activity's children are inaccessible.

COMPLETE
The process has completed.

DORMANT
The process is waiting for an event to fire its next activation.

INITIAL
No RUN or LINK command has yet been issued against the process.

SUSPSTATUS(cvda)
indicates whether the process is currently suspended. CVDA values are:

SUSPENDED
The process is currently suspended. If a reattachment event occurs, it will not be reactivated.

NOTSUSPENDED
The process is not currently suspended. If a reattachment event occurs, it will not be reactivated.

Conditions

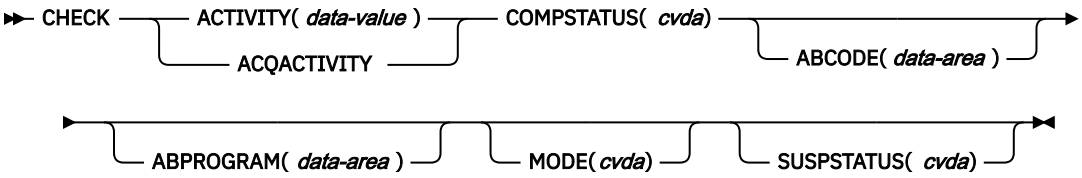
16 INVREQ
RESP2 values:

15
The unit of work that issued the request has not acquired a process.

CHECK ACTIVITY

Check the completion status of a BTS activity.

CHECK ACTIVITY



Conditions: ACTIVITYBUSY, ACTIVITYERR, INVREQ, IOERR, LOCKED

Description

CHECK ACTIVITY returns the completion status of a BTS activity. Typically, it is used to check the success of a previous RUN ACTIVITY or LINK ACTIVITY command. It allows the requestor to discover whether an activity completed successfully, or whether, for example, it needs to be reactivated in order to complete its processing.

CHECK ACTIVITY can be issued:

1. By a parent activity, to check the completion status of one of its children
2. By a program that has acquired an activity by means of an ACQUIRE ACTIVITYID command.

It can be used to check descendant (not root) activities:

- That have completed
- That have not completed
- That were requested to run asynchronously
- That were requested to run synchronously.

The RESP and RESP2 options on this command reflect whether the command is understood by CICS—for example, ACTIVITYERR occurs if the child named on the ACTIVITY option has not been defined to the parent.

The COMPSTATUS option returns a CVDA value indicating the completion status of the activity—for example, NORMAL is returned if the activity has successfully completed all its processing steps, while INCOMPLETE is returned if it has returned from an activation but needs to be reattached in order to complete its processing.

If this command is issued by a parent activity in respect of one of its children, and the child has completed, on return from the command CICS deletes the child's completion event from the parent's event pool.

For further guidance on the use of the CHECK ACTIVITY command, see Dealing with BTS errors and response codes .

Options

ABCODE(data-area)
returns, if the activity terminated abnormally, the 4-character abend code.

ABPROGRAM(data-area)
returns, if the activity terminated abnormally, the 8-character name of the program that was in control at the time of the abend.

ACQACTIVITY
specifies that the activity to be checked is the one that the current unit of work has acquired by means of an ACQUIRE ACTIVITYID command.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the activity to be checked.

Use this option to check the state of a child of the current activity.

COMPSTATUS(cvda)
indicates the completion status of the activity. CVDA values are:

ABEND
The program that implements the activity abended. Any children of the activity have been canceled.

The activity's completion event is deleted from the parent's event pool.

FORCED
The activity was forced to complete—for example, it was canceled with a CANCEL ACTIVITY command.

The activity's completion event is deleted from the parent's event pool.

INCOMPLETE
The named activity is incomplete. This could mean:

- That it has not yet been run
- That it has returned from one or more activations but needs to be reattached in order to complete all its processing steps
- That it is currently active.

The activity's completion event is **not** deleted from the parent's event pool.

NORMAL
The named activity completed successfully.

The activity's completion event is deleted from the parent's event pool.

MODE(cvda)
indicates the processing state of the activity. CVDA values are:

ACTIVE
An activation of the activity is running.

CANCELLING
CICS is waiting to cancel the activity. A CANCEL ACTIVITY command has been issued, but CICS cannot cancel the activity immediately because one or more of the activity's children are inaccessible.

COMPLETE
The activity has completed.

DORMANT
The activity is waiting for an event to fire its next activation.

INITIAL
No RUN or LINK command has yet been issued against the activity; or the activity has been reset by means of a RESET ACTIVITY command.

SUSPSTATUS(cvda)
indicates whether the activity is currently suspended. CVDA values are:

SUSPENDED
The activity is currently suspended. If a reattachment event occurs, it will not be reactivated.

NOTSUSPENDED
The activity is not currently suspended. If a reattachment event occurs, it will be reactivated.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

109 ACTIVITYERR
RESP2 values:

8
The activity named in the ACTIVITY option could not be found.

16 INVREQ
RESP2 values:

4
The ACTIVITY option was used to name a child activity, but the command was issued outside the scope of a currently-active activity.

24
The ACQACTIVITY option was used, but the unit of work that issued the request has not acquired an activity.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

CHECK TIMER

Check the status of a BTS timer.

CHECK TIMER

Conditions: INVREQ, IOERR, TIMERERR

Description

CHECK TIMER returns the status of a BTS timer. It allows the requestor to discover whether a timer has expired and, if so, whether it expired normally or whether its expiry was forced by means of a FORCE TIMER command.

On return from this command, if the timer has expired its associated event is deleted from the current activity's event pool.

The only timers a program can check are those owned by the current activity.

Options

STATUS(cvda)
indicates the status of the timer. CVDA values are:

EXPIRED
The timer expired normally.

Its associated event is deleted from the current activity's event pool.

FORCED
The timer expired because a FORCE TIMER command was issued against it.

Its associated event is deleted from the current activity's event pool.

UNEXPIRED
The timer has not yet expired.

Its associated event is not deleted from the current activity's event pool.

TIMER(data-value)
specifies the name (1–16 characters) of the timer to be checked.

Conditions

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of a currently-active activity.

17 IOERR
An input/output error has occurred on the repository file.

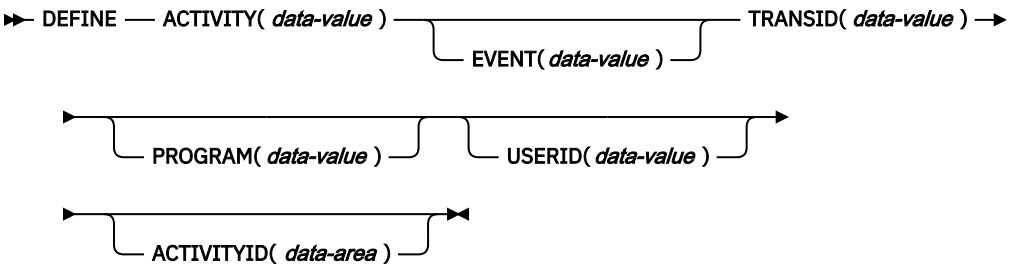
115 TIMERERR
RESP2 values:

13
The timer specified on the TIMER option does not exist.

DEFINE ACTIVITY

Define a CICS business transaction services activity.

DEFINE ACTIVITY



Conditions: ACTIVITYERR, EVENTERR, INVREQ, IOERR, NOTAUTH, TRANSIDERR

Description

DEFINE ACTIVITY defines an activity to CICS business transaction services. It is used to add a child activity to the current activity.

The name of the program used in the execution of the new activity is taken either from the PROGRAM option, or, if PROGRAM is not specified, from the transaction definition pointed to by the TRANSID option.

The transaction attributes specified on the TRANSID and USERID options take effect when the activity is activated by a RUN command, but *not* if it is activated by a LINK command—see “Context-switching” on page 224.

BTS does not commit the addition of the activity until the requesting transaction has taken a successful syncpoint.

Options

ACTIVITY(data-value)
specifies the name (1–16 characters) of the new activity. The name must not be the name of another child activity of the activity that issues the DEFINE command.

The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

ACTIVITYID(data-area)
returns the 52-character identifier assigned by CICS to the newly-defined activity. This identifier is unique across the sysplex.

EVENT(data-value)
specifies the name (1–16 characters) of the completion event for the activity. The completion event is sent to the activity's parent when the activity completes.

If EVENT is not specified, the completion event is given the same name as the activity itself.

The acceptable characters are A-Z a-z 0-9 $ @ # . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

PROGRAM(data-value)
specifies the name (1–8 characters) of the program for the activity being defined. If no program is specified, the name is taken from the TRANSID definition.

TRANSID(data-value)
specifies the name (1–4 characters) of the transaction under which the activity is to run, when it is activated by a RUN command.

Note: If the activity is activated by a LINK command, it is run under the TRANSID of the transaction that issues the LINK.

The transaction must be defined in the CICS region in which the process is running.

USERID(data-value)
specifies the userid (1–8 characters) under whose authority the activity is to run, when it is activated by a RUN command.

Note: If the activity is activated by a LINK command, it is run under the userid of the transaction that issues the LINK.

The value of this field is known as the *defined userid*.

If you omit USERID, the defined userid defaults to the userid under which the transaction that issues the DEFINE command is running—we can call this the *command userid*.

If USERID is specified, CICS performs (at define time) a surrogate security check to verify that the command userid is authorized to use the defined userid. Thus, if you specify USERID, you must authorize the command userid as a surrogate user of the defined userid.

Conditions

109 ACTIVITYERR
RESP2 values:

3
The name specified on the ACTIVITY option has already been used to name another child of the current activity.

111 EVENTERR
RESP2 values:

7
The completion event specified on the EVENT option has already been defined to the current activity's event pool.

16 INVREQ
RESP2 values:

4
The DEFINE ACTIVITY command was issued outside the scope of a currently-active activity.

17
The activity name specified on the ACTIVITY option, or the event name specified on the EVENT option, is invalid.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access the file associated with the BTS repository data set on which details of the activity are to be stored.

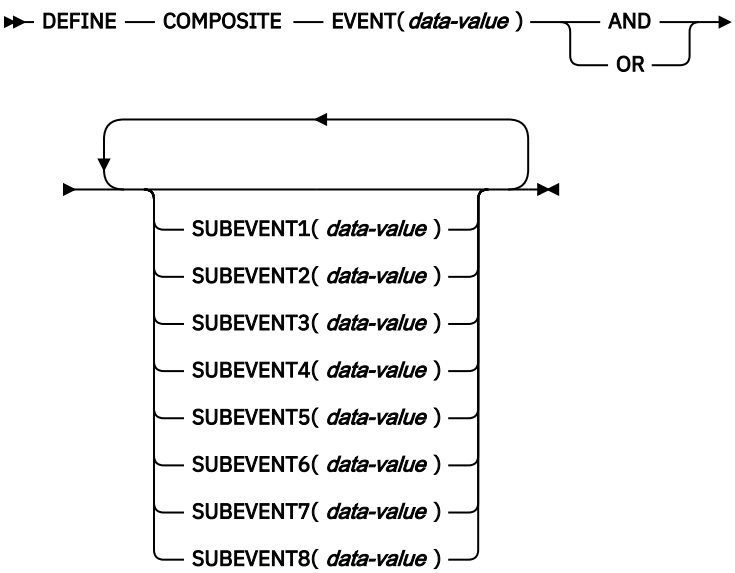
102
The user associated with the issuing task is not authorized as a surrogate of the defined userid specified on the USERID option.

28 TRANSIDERR
The transaction identifier specified on the TRANSID option is not defined to CICS.

DEFINE COMPOSITE EVENT

Define a BTS composite event.

DEFINE COMPOSITE EVENT



Conditions: EVENTERR, INVREQ

Description

DEFINE COMPOSITE EVENT defines a composite event to BTS. A composite event is formed from zero or more atomic events known as sub-events.

DEFINE COMPOSITE EVENT defines a *predicate*, which is a logical expression involving sub-events. At all times, the composite event's fire status (FIRED or NOTFIRED) reflects the value of the predicate. When the predicate becomes true, the composite event fires; when it becomes false, the composite's fire status reverts to NOTFIRED.

The logical operator that is applied to the sub-events in the composite event's predicate is one of the Boolean operators AND or OR. *AND and OR cannot both be used*.

You can specify up to 8 sub-events to be added to the composite event when the composite is created. If you do not specify any sub-events, the composite event is defined as "empty"—that is, as containing no sub-events.

To add sub-events to a composite event after the composite has been defined, use the ADD SUBEVENT command. There is no limit to the number of sub-events that you can add using ADD SUBEVENT.

Note: The following *cannot* be added as sub-events to a composite event:

- Composite events
- System events
- Sub-events of other composite events
- Input events, if the composite uses the AND operator.

To remove sub-events from a composite event, use the REMOVE SUBEVENT command.

Options

AND
specifies that the Boolean operator to be associated with this composite's predicate is AND. This means that the composite event will fire when *all* of its sub-events have fired.

Note: The fire status of an empty composite event that uses the AND operator is always FIRED (true).

EVENT(data-value)
specifies the name (1–16 characters) of the composite event being defined. The acceptable characters are A-Z a-z 0-9 $ @ # . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

OR
specifies that the Boolean operator to be associated with this composite's predicate is OR. This means that the composite event will fire when *any* of its sub-events fires.

Note: The fire status of an empty composite event that uses the OR operator is always NOTFIRED (false).

SUBEVENTn(data-value)
specifies the name (1–16 characters) of a sub-event to be added to the composite event when the composite is created. The acceptable characters are A-Z a-z 0-9 $ @ # . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

You can specify this option up to 8 times; *n* must be in the range 1–8.

The sub-events that you specify must previously have been defined to the current activity by means of DEFINE INPUT EVENT, DEFINE ACTIVITY, or DEFINE TIMER commands. They must not be sub-events of existing composite events.

Conditions

111 EVENTERR
RESP2 values:

6
The event name specified on the EVENT option is invalid.

7
The event name specified on the EVENT option has already been defined to this activity.

21–28
One or more of the sub-events named on the SUBEVENTn option does not exist. The RESP2 value indicates the first sub-event that does not exist.

16 INVREQ
RESP2 values:

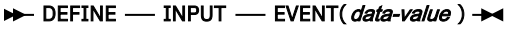
1
The command was issued outside the scope of an activity.

31–38
One or more of the sub-events names specified on the SUBEVENTn option is invalid. The RESP2 value indicates the first invalid sub-event name.

DEFINE INPUT EVENT

Define a BTS input event.

DEFINE EVENT



Conditions: EVENTERR, INVREQ

Description

DEFINE INPUT EVENT defines an input event to BTS. Typically, an input event is passed to an activity by its parent, causing the activity to be activated. (Sometimes, however, the input event originates from outside the process.)

Most events fire on the completion of something, such as an activity or a specified time interval. An input event is different in that it fires after a RUN command that names it is issued.

An activity defines an input event in order to receive notification (via the INPUTEVENT option of the RUN or LINK ACTIVITY commands) of why it has been activated.

Note: System events such as DFHINITIAL are a special type of input event. They are recognized by all activities and do not need to be defined.

Options

EVENT(data-value)
specifies the name (1–16 characters) of the input event being defined. The acceptable characters are A-Z a-z 0-9 $ @ # . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

Conditions

111 EVENTERR
RESP2 values:

6
The event name specified on the EVENT option is invalid.

7
The event name specified on the EVENT option has already been defined to this activity.

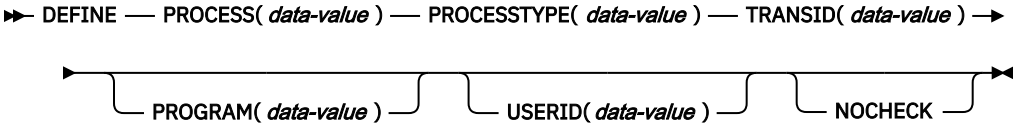
16 INVREQ
RESP2 values:

1
The command was issued outside the scope of an activity.

DEFINE PROCESS

Define a CICS business transaction services process.

DEFINE PROCESS



Conditions: INVREQ, IOERR, NOTAUTH, PROCESSERR, TRANSIDERR

Description

DEFINE PROCESS defines a BTS process. It:

- Adds a new process (for example, a new instance of a business transaction) to the CICS business transaction services system
- Creates the process's root activity.

The name of the program used in the execution of the new process is taken either from the PROGRAM option, or, if PROGRAM is not specified, from the transaction definition pointed to by the TRANSID option.

The transaction attributes specified on the TRANSID and USERID options take effect when the process is activated by a RUN command, but *not* if it is activated by a LINK command—see "RUN" on page 223.

BTS does not commit the addition of the process until the requesting transaction has taken a successful syncpoint.

Options

NOCHECK
specifes that no record is to be written to the repository data set to reserve the name of the process.

Note that the process name must be unique in the repository—see the PROCESS and PROCESSTYPE options—and that BTS does not commit the addition of the process until the requesting transaction has taken a successful syncpoint.

You can use this option to improve BTS performance by removing the write to the repository and its associated logging. However, if you do so be aware that the error of specifying a non-unique process name no longer causes a PROCESSERR condition to be returned on the DEFINE PROCESS command. The error may not be discovered until much later—when syncpoint occurs—making it much harder to debug.

PROCESS(data-value)
specifies a name (1–36 characters) to identify the new process (business transaction instance). The name must be unique within the BTS repository data set on which details of the process are to be stored—see the PROCESSTYPE option. For example, it is valid to issue a DEFINE command on which the PROCESS option specifies a name that is currently in use by another process, *provided that* the PROCESSTYPE option maps to a different underlying repository data set from that on which the first process is defined.

The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are also permitted.

If the name is specified as a literal string that is less than 36 characters long, it is padded with trailing blanks up to 36 characters. If the name is specified as a variable whose value is less than 36 characters long, no padding occurs.

PROCESSTYPE(data-value)
specifies the type (1–8 characters) of the new process.

Each process-type maps to a VSAM data set (the repository), on which information about processes of the named type is stored. That is, information about the state of a process (and of its constituent activities) is stored on the repository associated with the process-type to which it belongs. Records for multiple process-types can be stored on the same repository data set.

You can categorize your processes by assigning them to different process-types.

PROGRAM(data-value)
specifies the name (1–8 characters) of the program for the process being added. If no program is specified, the name is taken from the TRANSID definition.

TRANSID(data-value)
specifies the name (1–4 characters) of the transaction under which the process is to run when it is activated by a RUN command.

Note: If the process is activated by a LINK command, it is run under the TRANSID of the transaction that issues the LINK.

The transaction must be defined in the CICS region in which the DEFINE PROCESS command is executed.

USERID(data-value)
specifies the userid (1–8 characters) under whose authority the process is to run when it is activated by a RUN command.

Note: If the process is activated by a LINK command, it is run under the userid of the transaction that issues the LINK.

The value of this field is known as the *defined userid*.

If you omit USERID, the defined userid defaults to the userid under which the transaction that issues the DEFINE command is running—we can call this the *command userid*.

If USERID is specified, CICS performs (at define time) a surrogate security check to verify that the command userid is authorized to use the defined userid. Thus, if you specify USERID, you must authorize the command userid as a surrogate user of the defined userid.

Conditions

16 INVREQ
RESP2 values:

12
The installed PROCESSTYPE is not enabled.

22
The unit of work that issued the DEFINE PROCESS command has already acquired an activity.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access the file associated with the BTS repository data set on which details of the process are to be stored.

102
The user associated with the issuing task is not authorized as a surrogate of the defined userid specified on the USERID option.

108 PROCESSERR
RESP2 values:

2
The process name specified on the PROCESS option is already in use on the BTS repository data set associated with the PROCESSTYPE option.

9
The process-type specified on the PROCESSTYPE option could not be found.

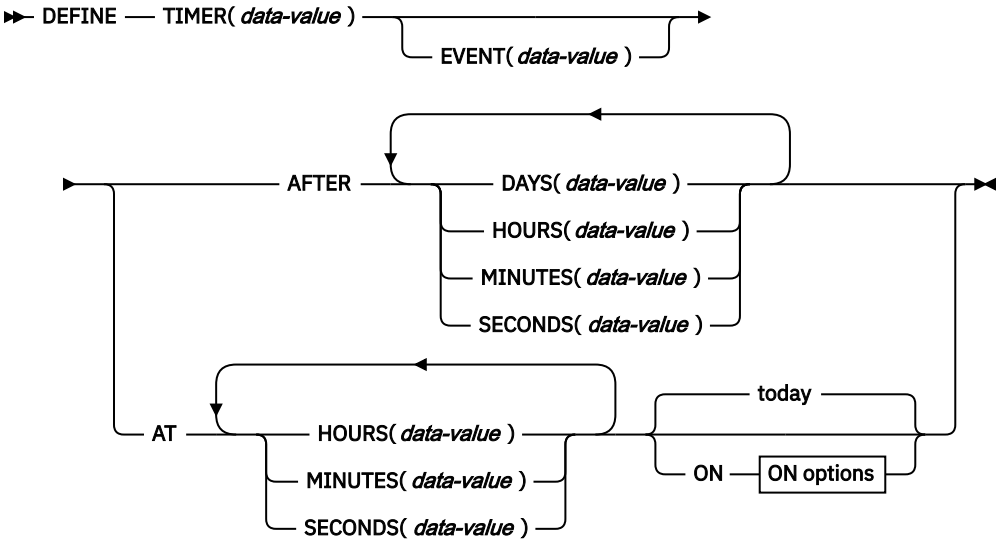
16
The process name specified on the PROCESS option contains an invalid character or characters.

28 TRANSIDERR
The transaction identifier specified on the TRANSID option is not defined to CICS.

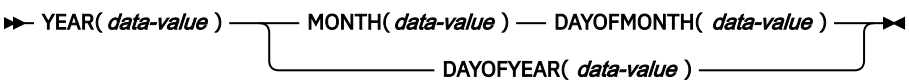
DEFINE TIMER

Define a BTS timer.

DEFINE TIMER



ON options



Conditions: EVENTERR, INVREQ, TIMERERR

Description

DEFINE TIMER defines a BTS timer that will expire after a specified interval, or at a specified time and date. When a timer is defined, an associated event is also defined, in the event pool of the current activity. The name of the associated event defaults to the name of the timer. When the timer expires, its associated event fires.

Note:

1. All dates and times refer to the local time.
2. A timer that specifies a time and date that has already passed expires immediately. Similarly, if the requested interval is zero, the timer expires immediately.

Options

AFTER

Specifies the interval of time that is to elapse before the timer is to expire.

You must specify one or more of DAYS(0–999), HOURS(0–23), MINUTES(0–59), and SECONDS(0–59). For example, HOURS(1) SECONDS(3) means one hour and three seconds (the minutes default to zero).

AT

Specifies the time at which the timer is to expire.

You must specify one or more of HOURS(0–23), MINUTES(0–59), and SECONDS(0–59). For example:

- HOURS(1) means 1 a.m.
- HOURS(15) MINUTES(15) means 3:15 p.m.
- MINUTES(15) means 0:15 a.m.

DAYOFMONTH(*data-value*)
Specifies, as a fullword binary value in the range 1–31, the day-of-the-month on which the timer is to expire.

DAYOFYEAR(*data-value*)
Specifies, as a fullword binary value in the range 1–366, the day-of-the-year on which the timer is to expire. For example, DAYOFYEAR(1) specifies 1st January.

DAYS(*data-value*)
Specifies a fullword binary value in the range 0–999. This is a suboption of the AFTER option. For its use and meaning, see AFTER.

The default value is zero.

EVENT(data-value)
Specifies the name (1–16 characters) of the event to be associated with the timer. The acceptable characters are A-Z a-z 0-9 $ @ # . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

The default event name is the name of the timer.

HOURS(*data-value*)
Specifies a fullword binary value in the range 0–23. This is a suboption of the AFTER and AT options. For its use and meaning, see these options.

The default value is zero.

MINUTES(*data-value*)
Specifies a fullword binary value in the range 0–59. This is a suboption of the AFTER and AT options. For its use and meaning, see these options.

The default value is zero.

MONTH(*data-value*)
Specifies, as a fullword binary value in the range 1–12, the month in which the timer is to expire.

ON
Specifies the date at which the timer is to expire, as a combination of the YEAR, MONTH, DAYOFMONTH, and DAYOFYEAR options.

If the ON option is not specified, the default date is today.

SECONDS(*data-value*)
Specifies a fullword binary value in the range 0–59. This is a suboption of the AFTER and AT options. For its use and meaning, see these options.

The default value is zero.

TIMER(*data-value*)
Specifies the name (1–16 characters) of the timer. The acceptable characters are A-Z a-z 0-9 $ @ # . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

YEAR(*data-value*)
Specifies, as a fullword binary value in the range 0–2040, the year in which the timer is to expire.

Conditions

111 EVENTERR
RESP2 values:

6
The event name specified on the EVENT option is invalid.

7
The event name specified on the EVENT option (or the default event name taken from the timer name) has already been defined to this activity.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of a currently-active activity.

11
An invalid interval was specified.

12
An invalid date or time was specified.

115 TIMERERR
RESP2 values:

14
The timer name specified on the TIMER option is invalid.

15
The timer name specified on the TIMER option has already been defined to this activity.

Examples

```
DEFINE TIMER() AT HOURS(15)
```

defines a timer that will expire at 3 p.m. today (or immediately if the local time is already later than 3 p.m.).

```
DEFINE TIMER() AT HOURS(15) ON YEAR(2001) MONTH(11) DAYOFMONTH(3)
```

defines a timer that will expire at 3 p.m. on 3rd November 2001.

```
DEFINE TIMER() AT HOURS(15) ON YEAR(2001) DAYOFYEAR(32)
```

defines a timer that will expire at 3 p.m. on 1st February 2001.

```
DEFINE TIMER() AT HOURS(8) ON YEAR(1997) MONTH(1) DAYOFMONTH(1)
```

defines a timer that expires immediately.

DELETE ACTIVITY

Delete a BTS child activity.

DELETE

Conditions: ACTIVITYBUSY, ACTIVITYERR, INVREQ, IOERR, LOCKED

Description

DELETE ACTIVITY removes a child activity from the BTS repository data set on which it is defined. The child activity's completion event is removed from the parent's event pool. Any descendants of the child activity are also deleted.

The activity to be deleted must be a child of the activity that issues the DELETE command. To be eligible for deletion, the child activity must be in one of the following processing states (modes):

- COMPLETE—completed normally, abnormally, or previously canceled.
- INITIAL—not yet run, or reset by means of a RESET ACTIVITY command.

For a description of all possible processing states, see Processing modes .

Note: A child activity that is not deleted explicitly by means of a DELETE ACTIVITY command is deleted automatically by CICS when its parent completes.

Options

ACTIVITY(data-value)
specifies the name (1–16 characters) of the child activity to be deleted.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

14
The child activity named on the ACTIVITY option is not in COMPLETE or INITIAL mode, and is therefore ineligible for deletion.

16 INVREQ
RESP2 values:

4
The DELETE ACTIVITY command was issued outside the scope of a currently-active activity.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

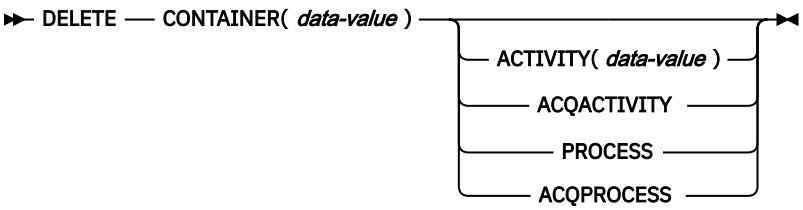
30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

DELETE CONTAINER (BTS)

Delete a named BTS data-container.

DELETE CONTAINER (BTS)



Conditions: ACTIVITYERR, CONTAINERERR, INVREQ, IOERR, LOCKED, PROCESSBUSY

Description

DELETE CONTAINER (BTS) deletes a BTS data-container and discards any data that it contains.

The container is identified by name and by the process or activity for which it is a container—the process or activity that "owns" it. The activity that owns the container can be identified:

- Explicitly, by specifying one of the PROCESS- or ACTIVITY-related options.
- Implicitly, by omitting the PROCESS- and ACTIVITY-related options. If these are omitted, the current activity is implied.

Note: Process containers can be deleted only by the root activity or by a program that has acquired the process.

Options

ACQACTIVITY
specifies either of the following:

- If the program that issues the command has acquired a process, that the container is owned by the root activity of that process.
- Otherwise, that the container is owned by the activity that the program has acquired by means of an ACQUIRE ACTIVITYID command.

ACQPROCESS
specifies that the container is owned by the process that the program that issues the command has acquired in the current unit of work.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the activity that owns the container. This must be a child of the current activity.

CONTAINER(data-value)
specifies the name (1–16 characters) of the container to be deleted.

PROCESS
specifies that the container to be deleted is owned by the current process—that is, the process that the program that issues the command is executing on behalf of.

Conditions

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

110 CONTAINERERR
RESP2 values:

10
The container named on the CONTAINER option could not be found.

26
The process container named on the CONTAINER option is read-only. (Process containers can be deleted only by the root activity or by a program that has acquired the process.)

16 INVREQ
RESP2 values:

4
The command was issued outside the scope of a currently-active activity.

15
The ACQPROCESS option was used, but the unit of work that issued the request has not acquired a process.

24
The ACQACTIVITY option was used, but the unit of work that issued the request has not acquired an activity.

25
The PROCESS option was used, but the command was issued outside the scope of a currently-active process.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

31
The record on the repository file is in use.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

106 PROCESSBUSY
RESP2 values:

13
The request could not be satisfied because the process record is locked by another task.

DELETE EVENT

Delete a BTS event.

DELETE EVENT

Conditions: EVENTERR, INVREQ

Description

DELETE EVENT deletes a BTS event that is no longer needed. The event is removed from the current activity's event pool. An event can be deleted whether it has fired or not.

DELETE EVENT can be used to delete only the following types of event:

- Input
- Composite.

DELETE EVENT *cannot* be used to delete:

- Activity completion events. These are implicitly deleted when a response from the completed activity is acknowledged by a CHECK ACTIVITY command issued by the activity's parent; or when a DELETE ACTIVITY command is issued.
- Timer events. These are implicitly deleted when the expiry of the associated timer is acknowledged by a CHECK TIMER command; or when a DELETE TIMER command is issued.
- System events.

Note:

1. If the event to be deleted is included in the predicate of a composite event, it is removed from the predicate's Boolean expression. The fire status of the composite event (FIRED or NOTFIRED) is re-evaluated.
2. Deleting a composite event has no effect on its sub-events.

Options

EVENT(data-value)
specifies the name (1–16 characters) of the event to be deleted.

Conditions

111 EVENTERR
RESP2 values:

4
The event specified on the EVENT option is not recognized by BTS.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of an activity.

2
The event specified on the EVENT option cannot be deleted because it is a system, timer, or activity completion event.

DELETE TIMER

Delete a BTS timer.

DELETE TIMER

Conditions: INVREQ, TIMERERR

Description

DELETE TIMER deletes a BTS timer. If an event is associated with the timer, the event is also deleted and removed from the current activity's event pool. (There will be no event associated with the timer if the timer has expired and a CHECK TIMER command has been issued.)

The only timers a program can delete are those owned by the current activity. A timer can be deleted whether it has expired or not.

Options

TIMER(data-value)
specifies the name (1–16 characters) of the timer to be deleted.

Conditions

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of a currently-active activity.

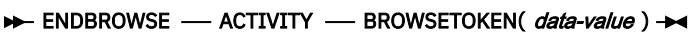
115 TIMERERR
RESP2 values:

13
The timer specified on the TIMER option does not exist.

ENDBROWSE ACTIVITY

End a browse of the child activities of a BTS activity, or of the descendant activities of a BTS process.

ENDBROWSE ACTIVITY



Conditions: ILLOGIC, TOKENERR

Description

ENDBROWSE ACTIVITY ends a browse of the child activities of a BTS activity (or of the descendant activities of a BTS process), and invalidates the browse token.

Options

BROWSETOKEN(data-value)
specifies, as a fullword binary value, the browse token to be deleted.

Conditions

21 ILLOGIC
RESP2 values:

1
The value specified in the BROWSETOKEN option matches a current browse token, but not one that is being used for an activity browse.

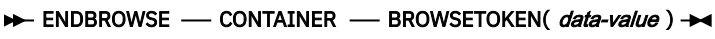
112 TOKENERR
RESP2 values:

3
The browse token is not valid.

ENDBROWSE CONTAINER (BTS)

End a browse of the containers associated with a BTS activity or process.

ENDBROWSE CONTAINER



Conditions: ILLOGIC, TOKENERR

Description

ENDBROWSE CONTAINER ends a browse of the containers associated with a BTS activity or process, and invalidates the browse token.

Options

BROWSETOKEN(data-value)
specifies, as a fullword binary value, the browse token to be deleted.

Conditions

21 ILLOGIC
RESP2 values:

1
The value specified in the BROWSETOKEN option matches a current browse token, but not one that is being used for a browse of containers.

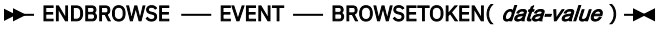
112 TOKENERR
RESP2 values:

3
The browse token is not valid.

ENDBROWSE EVENT

End a browse of the events known to a BTS activity.

ENDBROWSE EVENT



Conditions: TOKENERR

Description

ENDBROWSE EVENT ends a browse of the events that are within the scope of a BTS activity, and invalidates the browse token.

Options

BROWSETOKEN(data-value)
specifies, as a fullword binary value, the browse token to be deleted.

Conditions

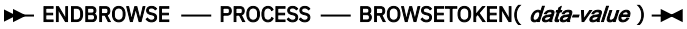
112 TOKENERR
RESP2 values:

3
The browse token is not valid.

ENDBROWSE PROCESS

End a browse of processes of a specified type within the CICS business transaction services system.

ENDBROWSE PROCESS



Conditions: ILLOGIC, TOKENERR

Description

ENDBROWSE PROCESS ends a browse of the processes of a specified type within the CICS business transaction services system, and invalidates the browse token.

Options

BROWSETOKEN(data-value)
specifies, as a fullword binary value, the browse token to be deleted.

Conditions

21 ILLOGIC
RESP2 values:

1
The value specified in the BROWSETOKEN option matches a current browse token, but not one that is being used for a process browse.

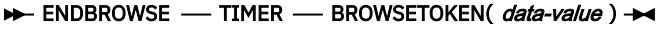
112 TOKENERR
RESP2 values:

3
The browse token is not valid.

ENDBROWSE TIMER

End a browse of a BTS timer.

ENDBROWSE TIMER



Conditions: ILLOGIC, INVREQ, TOKENERR

Description

ENDBROWSE TIMER ends a browse of a BTS timer.

Options

BROWSETOKEN(data-value)
Specifies, as a fullword binary value, the browse token to be deleted.

Conditions

21 ILLOGIC
RESP2 values:

1
The value specified in the BROWSETOKEN option matches a current browse token, but not one that is being used for a browse of a timer.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of a currently—active activity.

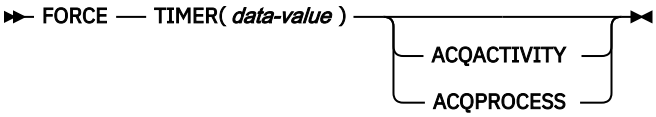
112 TOKENERR
RESP2 values:

3
The browse token is not valid.

FORCE TIMER

Force the early expiry of a BTS timer.

FORCE TIMER



Conditions: INVREQ, TIMERERR

Description

FORCE TIMER forces a BTS timer that has not yet expired to expire immediately. This causes the event associated with the timer to fire.

If the timer has already expired, the command has no effect.

The activity that owns the timer can be identified:

- Explicitly, by specifying either the ACQPROCESS or ACQACTIVITY option.
- Implicitly, by omitting the ACQPROCESS and ACQACTIVITY options. If these are omitted, the current activity is implied.

Options

ACQACTIVITY
specifies either of the following:

- If the program that issues the command has acquired a process, that the timer is owned by the root activity of that process.
- Otherwise, that the timer is owned by the activity that the program has acquired by means of an ACQUIRE ACTIVITYID command.

ACQPROCESS
specifies that the timer is owned by the process that the program that issues the command has acquired in the current unit of work.

TIMER(data-value)
specifies the name (1–16 characters) of the timer to be forced.

Conditions

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of a currently-active activity.

16
The ACQPROCESS option was specified, but there is no acquired process.

17
The ACQACTIVITY option was specified, but there is no acquired activity.

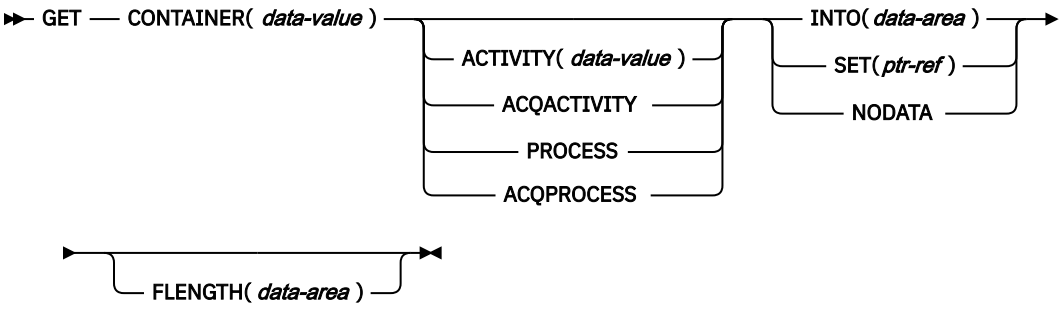
115 TIMERERR
RESP2 values:

13
The timer named on the TIMER option does not exist.

GET CONTAINER (BTS)

Retrieve data from a named BTS data-container.

GET CONTAINER



Conditions: ACTIVITYERR, CONTAINERERR, INVREQ, IOERR, LENGERR, LOCKED, PROCESSBUSY

Description

GET CONTAINER reads the data associated with a specified BTS activity or process into working storage.

The container which holds the data is identified by name and by the process or activity for which it is a container—the process or activity that "owns" it. The activity that owns the container can be identified:

- Explicitly, by specifying one of the PROCESS- or ACTIVITY-related options.
- Implicitly, by omitting the PROCESS- and ACTIVITY-related options. If these are omitted, the current activity is implied.

See also "PUT CONTAINER (BTS)" on page 210 and "MOVE CONTAINER (BTS)" on page 208.

Options

ACQACTIVITY
specifies either of the following:

- If the program that issues the command has acquired a process, that the container is owned by the root activity of that process.
- Otherwise, that the container is owned by the activity that the program has acquired by means of an ACQUIRE ACTIVITYID command.

ACQPROCESS
specifies that the container is owned by the process that the program that issues the command has acquired in the current unit of work.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the activity that owns the container. This must be a child of the current activity.

CONTAINER(data-value)
specifies the name (1–16 characters) of the container that holds the data to be retrieved.

FLENGTH(data-area)
As an input field, FLENGTH specifies, as a fullword binary value, the length of the data to be read. As an output field, FLENGTH returns the length of the data in the container. Whether FLENGTH is an input or an output field depends on which of the INTO, SET, or NODATA options you specify.

INTO option specified
FLENGTH is both an input and an output field.

On input, FLENGTH specifies the maximum length of the data that the program accepts. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. If the length of

the data is less than the value specified, the data is copied with no padding and the LENGERR condition occurs.

FLENGTH need not be specified if the length can be generated by the compiler from the INTO variable. If you specify both INTO and FLENGTH, FLENGTH specifies the maximum length of the data that the program accepts.

On **output** (that is, on completion of the retrieval operation) CICS sets the data area, if specified, to the actual length of the data in the container.

SET or NODATA option specified
FLENGTH is an output-only field. It must be specified and specified as a data-area.

On completion of the retrieval operation, the data area is set to the actual length of the data in the container.

INTO(data-area)
specifies an area of working storage into which the retrieved data is to be placed.

NODATA
specifies that no data is to be retrieved. Use this option to discover the length of the data in the container (returned in FLENGTH).

PROCESS
specifies that the container to be retrieved is owned by the current process—that is, the process that the program that issues the command is executing on behalf of.

SET(ptr-ref)
specifies a data area in which the address of the retrieved data is returned. The data area is maintained by CICS until a subsequent GET CONTAINER command with the SET option is issued by the task, or until the task ends.

If your application needs to keep the data it should move it into its own storage.

Conditions

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

110 CONTAINERERR
RESP2 values:

10
The container named on the CONTAINER option could not be found.

16 INVREQ
RESP2 values:

2
The INTOCCSID option was specified without the CHANNEL option, and there is no current channel (because the program that issued the command was not passed one.) INTOCCSID is valid only on GET CONTAINER commands that specify (explicitly or implicitly) a channel. It is not valid on GET CONTAINER (BTS) commands.

4
The command was issued outside the scope of a currently-active activity.

15
The ACQPROCESS option was used, but the unit of work that issued the request has not acquired a process.

24
The ACQACTIVITY option was used, but the unit of work that issued the request has not acquired an activity.

25
The PROCESS option was used, but the command was issued outside the scope of a currently-active process.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

31
The record on the repository file is in use.

22 LENGERR
RESP2 values:

11
The length of the program area is not the same as the length of the data in the container. If the area is smaller, the data is truncated to fit into it. If the area is larger, the data is copied to the program area but no padding is added.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

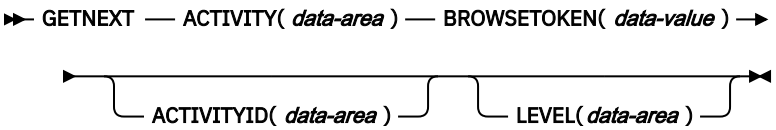
106 PROCESSBUSY
RESP2 values:

13
The request could not be satisfied because the process record is locked by another task.

GETNEXT ACTIVITY

Browse the child activities of a BTS activity, or the descendant activities of a BTS process.

GETNEXT ACTIVITY



Conditions: ACTIVITYERR, END, ILLOGIC, IOERR, TOKENERR

Description

GETNEXT ACTIVITY returns either:

- The name and identifier of the next child activity of a BTS activity (if the PROCESS and PROCESSTYPE options were omitted from the STARTBROWSE ACTIVITY command)
- The name and identifier of the next descendant activity of a BTS process (if the PROCESS and PROCESSTYPE options were specified on the STARTBROWSE ACTIVITY command).

You can use the INQUIRE ACTIVITYID command to query the identified activity.

Options

ACTIVITYID(data-area)
returns the 52-character identifier of the next activity.

ACTIVITY(data-area)
returns the 16-character name of the next activity.

BROWSETOKEN(data-value)
specifies, as a fullword binary value, a browse token returned on a previous STARTBROWSE ACTIVITY command.

LEVEL(data-area)
returns a fullword value indicating the depth in the activity-tree at which the next activity lies.

On a browse of the descendant activities of a process, a value of '0' indicates the root activity, '1' a child of the root activity, '2' a grandchild of the root activity, and so on.

On a browse of the child activities of an activity, the value returned is always 0.

Conditions

109 ACTIVITYERR
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

83 END
RESP2 values:

2
There are no more resource definitions of this type.

21 ILLOGIC
RESP2 values:

1
The value specified in the BROWSETOKEN option matches a current browse token, but not one that is being used for an activity browse.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

112 TOKENERR
RESP2 values:

3
The browse token is not valid.

GETNEXT CONTAINER (BTS)

Browse the containers associated with a BTS activity or process.

GETNEXT CONTAINER

Conditions: END, ILLOGIC, TOKENERR

Description

GETNEXT CONTAINER returns the name of the next container associated with a BTS activity or process. You can use the INQUIRE CONTAINER command to query the returned container.

Note:

1. You can use successive GETNEXT CONTAINER commands to retrieve the names of all the process's or activity's containers that existed at the time the STARTBROWSE CONTAINER command was executed. However, the names of any containers that are deleted after the STARTBROWSE and before they have been returned by a GETNEXT are not returned.
2. The names of any containers that are created on (or moved to) this process or activity after the STARTBROWSE command is executed may or may not be returned.
3. The order in which containers are returned is undefined.

Options

BROWSETOKEN(data-value)
specifies, as a fullword binary value, a browse token returned on a previous STARTBROWSE CONTAINER command.

CONTAINER(data-area)
returns the 16-character name of the next data-container.

Conditions

83 END
RESP2 values:

2
The are no more containers for this process or activity.

21 ILLOGIC
RESP2 values:

1
The value specified in the BROWSETOKEN option matches a current browse token, but not one that is being used for a browse of containers.

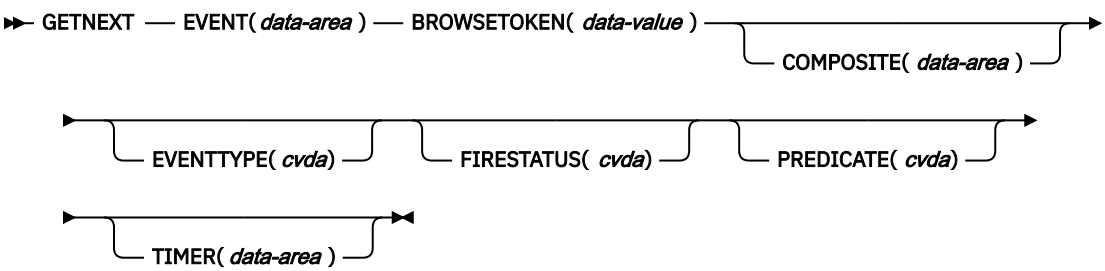
112 TOKENERR
RESP2 values:

3
The browse token is not valid.

GETNEXT EVENT

Browse the events known to a BTS activity.

GETNEXT EVENT



Conditions: END, TOKENERR

Description

GETNEXT EVENT returns the attributes of the next event, or sub-event, that is within the scope of a BTS activity.

Options

BROWSETOKEN(data-value)
specifies, as a fullword binary value, a browse token returned on a previous STARTBROWSE EVENT command.

COMPOSITE(data-area)
returns, if the named event is a sub-event, the 16-character name of the composite event that it is part of.

EVENT(data-area)
returns the 16-character name of the next event. This may be:

- An atomic event. An atomic event returned on this command may or may not be a sub-event.
- A composite event.
- A system event.

EVENTTYPE(cvda)
indicates the type of the named event. CVDA values are:

ACTIVITY
Activity completion

COMPOSITE
Composite

INPUT
Input

SYSTEM
System

TIMER
Timer.

FIRESTATUS(cvda)
indicates the state of the named event. CVDA values are:

FIRED
The event has fired normally.

NOTFIRED
The event has not fired.

PREDICATE(cvda)
indicates, if the named event is composite, the Boolean operator applied to its predicate. CVDA values are:

AND
The Boolean operator applied to the predicate is AND.

OR
The Boolean operator applied to the predicate is OR.

TIMER(data-area)
returns, if the named event is a timer event, the 16-character name of its associated timer.

Conditions

83 END
RESP2 values:

2
There are no more resource definitions of this type.

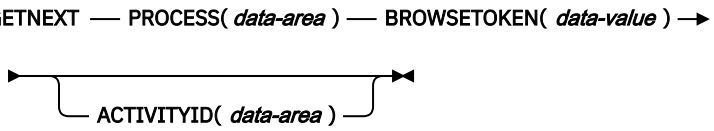
112 TOKENERR
RESP2 values:

3
The browse token is not valid.

GETNEXT PROCESS

Browse all processes of a specified type within the CICS business transaction services system.

GETNEXT PROCESS



Conditions: END, ILLOGIC, IOERR, PROCESSERR, TOKENERR

Description

GETNEXT PROCESS returns the name of the next process of a specified type within the CICS business transaction services system.

Options

ACTIVITYID(data-area)
returns the 52-character identifier of the next process's root activity.

BROWSETOKEN(data-value)
specifies, as a fullword binary value, a browse token returned on a previous STARTBROWSE PROCESS command.

PROCESS(data-area)
returns the 36-character name of the next process.

Conditions

83 END
RESP2 values:

2
There are no more resource definitions of this type.

21 ILLOGIC
RESP2 values:

1
The value specified in the BROWSETOKEN option matches a current browse token, but not one that is being used for a process browse.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

108 PROCESSERR
RESP2 values:

13
The request timed out. It may be that another task using this process-record has been prevented from ending.

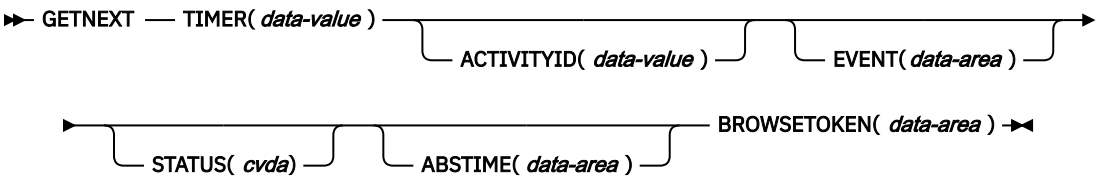
112 TOKENERR
RESP2 values:

3
The browse token is not valid.

GETNEXT TIMER

Browse the BTS timers associated with a BTS activity.

GETNEXT TIMER



Conditions: ACTIVITYERR, INVREQ, IOERR, NOTAUTH, TIMERERR

Description

GETNEXT TIMER returns the name of the next BTS timer associated with a BTS activity.

Options

ABSTIME(data-area)
Returns, in packed decimal format, the time at which the timer will expire, expressed in milliseconds since 00:00 on 1 January 1900 (rounded to the nearest hundredth of a second).

You can use **FORMATTIME** to change the data into other familiar formats.

ACTIVITYID(data-value)
Specifies the identifier (1–52 characters) of the activity with which the timer is associated.

If this option is omitted, the current activity is assumed.

BROWSETOKEN(data-area)
Specifies a fullword binary data area, into which CICS will place the browse token.

EVENT(data-area)
Returns the 16-character name of the event (if any) associated with the timer.

STATUS(cvda)
Indicates the state of the timer. CVDA values are:

EXPIRED
The timer expired normally.

FORCED
Expiry of the timer was forced by means of a **FORCE TIMER** command.

UNEXPIRED
The timer has not yet expired.

TIMER(data-value)
Specifies the name (1–16 characters) of the BTS timer.

Conditions

109 ACTIVITYERR
RESP2 values:

3
The activity indicated by the ACTIVITYID option could not be found.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of a currently—active activity.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

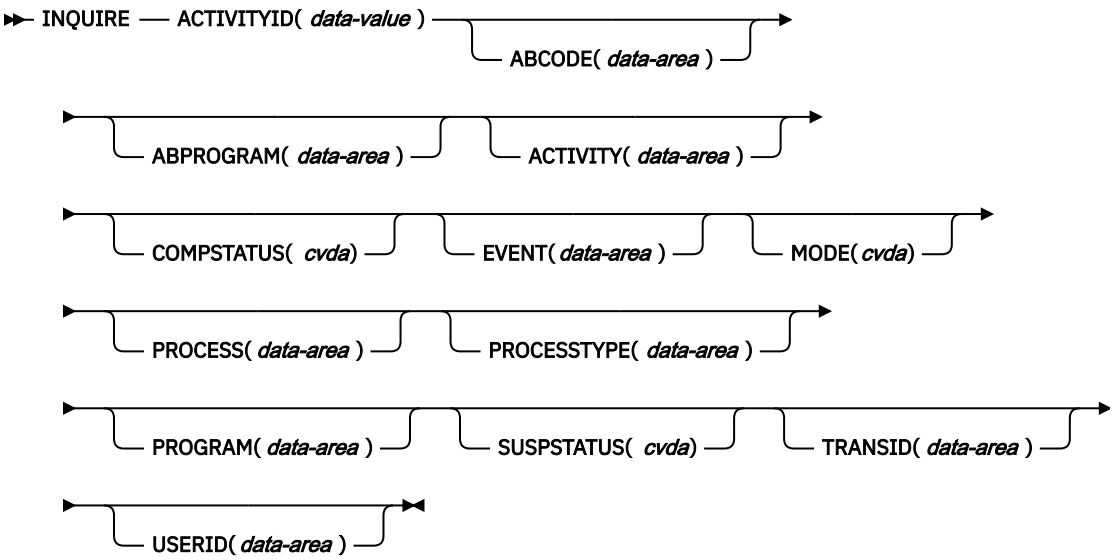
115 TIMERERR
RESP2 values:

1
The timer specified on the TIMER option could not be found.

INQUIRE ACTIVITYID

Retrieve the attributes of a BTS activity.

INQUIRE ACTIVITYID



Conditions: ACTIVITYERR, NOTAUTH

Description

INQUIRE ACTIVITYID returns the attributes of a specified BTS activity.

You can use this command to get details of an activity whose identifier has been retrieved during a browse operation.

Options

ABCODE(data-area)
returns, if the activity terminated abnormally, the 4-character abend code.

ABPROGRAM(data-area)
returns, if the activity terminated abnormally, the 8-character name of the program that was in control at the time of the abend.

ACTIVITY(data-area)
returns the 16-character name of the activity being queried.

ACTIVITYID(data-value)
specifies the identifier (1–52 characters) of the activity to be queried. (Typically, the activity identifier will have been retrieved by a GETNEXT ACTIVITY command, during an activity browse.)

COMPSTATUS(cvda)
indicates the completion status of the activity. CVDA values are:

ABEND
The program that implements the activity abended. Any children of the activity have been canceled.

FORCED
The activity was forced to complete—for example, it was canceled with a CANCEL ACTIVITY command.

INCOMPLETE
The named activity is incomplete. This could mean:

- That it has not yet been run
- That it has returned from one or more activations but needs to be reattached in order to complete all its processing steps
- That it is currently active.

NORMAL
The named activity completed successfully.

EVENT(data-area)
returns the 16-character name of the completion event that is sent to the requestor of this activity when the activity completes asynchronously with the requestor.

MODE(cvda)
indicates the current state (mode) of the activity. CVDA values are:

ACTIVE
An activation of the activity is running.

CANCELLING
CICS is waiting to cancel the activity. A CANCEL ACTIVITY command has been issued, but CICS cannot cancel the activity immediately because one or more of the activity's children are inaccessible.

No further operations on the activity are permitted until it has been canceled.

COMPLETE
The activity has completed, either successfully or unsuccessfully. The value returned on the COMPSTATUS option tells you how it completed.

DORMANT
The activity is waiting for an event to fire its next activation.

INITIAL
No RUN or LINK command has yet been issued against the activity; or the activity has been reset by means of a RESET ACTIVITY command.

PROCESS(data-area)
returns the 36-character name of the process to which this activity belongs.

PROCESSTYPE(data-area)
returns the 8-character name of the process-type to which the process that contains this activity belongs.

PROGRAM(data-area)
returns the 8-character name of the program that executes when this activity is run.

SUSPSTATUS(cvda)
indicates whether the activity is currently suspended. CVDA values are:

SUSPENDED
The activity is currently suspended. If a reattachment event occurs, it will not be reactivated.

NOTSUSPENDED
The activity is not currently suspended. If a reattachment event occurs, it will be reactivated.

TRANSID(data-area)
returns the 4-character transaction identifier under which this activity runs.

USERID(data-area)
returns the 8-character identifier of the user under whose authority this activity runs.

Conditions

109 ACTIVITYERR
RESP2 values:

1
The activity identifier specified on the ACTIVITYID option does not relate to any activity that is within the scope of this task.

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

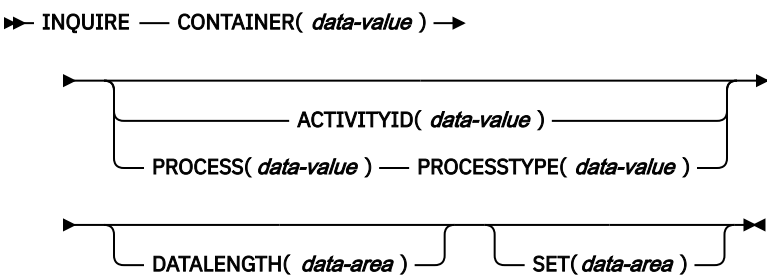
70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

INQUIRE CONTAINER

Retrieve the attributes of a BTS data-container.

INQUIRE CONTAINER



Conditions: ACTIVITYERR, CONTAINERERR, IOERR, NOTAUTH, PROCESSERR

Description

INQUIRE CONTAINER returns a pointer to the contents of a named BTS data-container, plus the length of the data.

To inquire upon a container associated with the current activity, omit the ACTIVITYID and PROCESS options.

To inquire upon a container associated with another activity, specify the ACTIVITYID option. (The activity identifier specified on the ACTIVITYID option may, for example, have been returned on a GETNEXT ACTIVITY command during a browse operation.)

To inquire upon a process container (including one associated with the *current* process), specify the PROCESS and PROCESSTYPE options.

Note:

1. Inquiring on a container of the current activity returns details of the in-storage version, rather than the committed version on the repository. This means that it's possible to see:
   - Containers that are not yet on the repository
   - Container contents that differ from those on the repository.
2. Inquiring on a container not owned by the current activity returns details of the committed version on the repository. However, the read of the repository record is “dirty”—the record is not locked. So, if the record is being updated by another task, it's possible for the returned data to be unreliable.

Options

ACTIVITYID(data-value)
specifies the identifier (1–52 characters) of the activity which the data-container is associated with.

If both this and the process options are omitted, the current activity is assumed.

CONTAINER(data-value)
specifies the name (1–16 characters) of the data-container being inquired upon.

DATALENGTH(data-area)
returns the fullword length of the data contained in the named data-container.

PROCESS(data-value)
specifies the name (1–36 characters) of the process which the data-container is associated with.

If both this and the ACTIVITYID option are omitted, the current activity is assumed.

PROCESSTYPE(data-value)
specifies the process-type (1–8 characters) of the process named in the PROCESS option.

SET(data-area)
returns a pointer to the contents of the data-container.

Conditions

109 ACTIVITYERR
RESP2 values:

2
The activity indicated by the ACTIVITYID option could not be found.

3
Because neither the ACTIVITYID nor the PROCESS options were specified, an inquiry on the current activity was implied—but there is no current activity associated with the request.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

110 CONTAINERERR
RESP2 values:

1
The container specified on the CONTAINER option could not be found.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

108 PROCESSERR
RESP2 values:

2
The process-type specified on the PROCESSTYPE option could not be found.

4
The process specified on the PROCESS option could not be found.

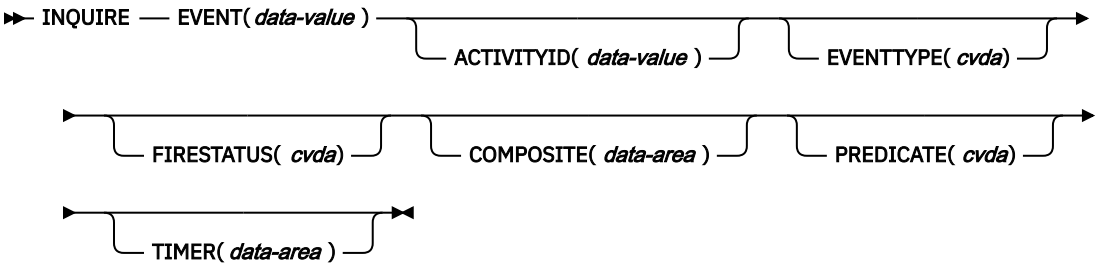
13
The request timed out. It may be that another task using this process-record has been prevented from ending.

33
The process specified on the PROCESS option has not yet been committed.

INQUIRE EVENT

Retrieve the attributes of a BTS event.

INQUIRE EVENT



Conditions: ACTIVITYERR, EVENTERR, INVREQ, IOERR, NOTAUTH

Description

INQUIRE EVENT returns the attributes of a named BTS event.

To inquire upon an event associated with the current activity, omit the ACTIVITYID option. To inquire upon an event associated with another activity, specify the ACTIVITYID option. (The activity identifier specified on the ACTIVITYID option may, for example, have been returned on a GETNEXT ACTIVITY command during a browse operation.)

Options

ACTIVITYID(data-value)
specifies the identifier (1–52 characters) of the activity which the event is associated with.

If this option is omitted, the current activity is assumed.

COMPOSITE(data-area)
returns, if the named event is a sub-event, the 16-character name of the composite event that it is part of.

EVENT(data-value)
specifies the name (1–16 characters) of the event being inquired upon.

EVENTTYPE(cvda)
indicates the type of the named event. CVDA values are:

ACTIVITY
Activity completion

COMPOSITE
Composite

INPUT
Input

SYSTEM
System

TIMER
Timer.

FIRESTATUS(cvda)
indicates the state of the named event. CVDA values are:

FIRED
The event has fired normally.

NOTFIRED
The event has not fired.

PREDICATE(cvda)
indicates, if the named event is composite, the Boolean operator applied to its predicate. CVDA values are:

AND
The Boolean operator applied to the predicate is AND.

OR
The Boolean operator applied to the predicate is OR.

TIMER(data-area)
returns, if the named event is a timer event, the 16-character name of the timer.

Conditions

109 ACTIVITYERR
RESP2 values:

3
The activity indicated by the ACTIVITYID option could not be found.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

111 EVENTERR
RESP2 values:

1
The event specified on the EVENT option could not be found.

16 INVREQ
RESP2 values:

1
There is no current activity within the scope of this task.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

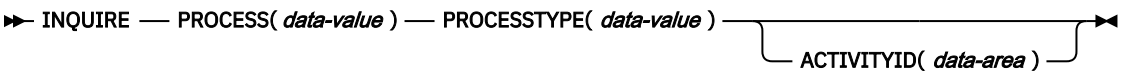
70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

INQUIRE PROCESS

Retrieve the attributes of a BTS process.

INQUIRE PROCESS



Conditions: ILLOGIC, NOTAUTH, PROCESSERR

Description

INQUIRE PROCESS returns the attributes of a named BTS process. It can be used, for example, to obtain the identifier of the root activity of a process, in order to start a browse of the root activity's child activities, containers, or events.

Options

ACTIVITYID(data-area)
returns the 52-character identifier of the root activity of the process that is being queried.

PROCESS(data-value)
specifies the name (1–36 characters) of the process to be queried.

PROCESSTYPE(data-value)
specifies the process-type (1–8 characters) of the process to be queried.

Conditions

21 ILLOGIC
RESP2 values:

1
A browse of this resource type is already in progress.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

108 PROCESSERR
RESP2 values:

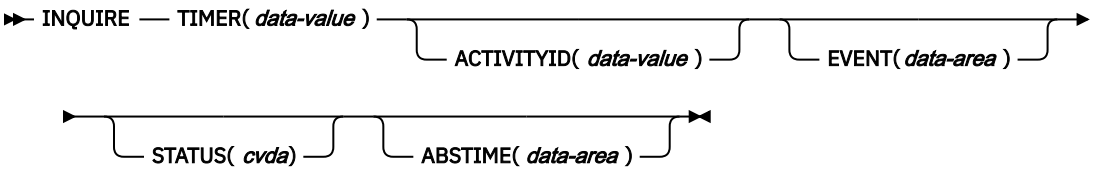
1
The process specified on the PROCESS option could not be found.

4
The process-type specified on the PROCESSTYPE option could not be found.

INQUIRE TIMER

Retrieve the attributes of a BTS timer.

INQUIRE TIMER



Conditions: ACTIVITYERR, INVREQ, IOERR, NOTAUTH, TIMERERR

Description

INQUIRE TIMER returns the attributes of a named BTS timer.

To inquire upon a timer associated with the current activity, omit the ACTIVITYID option. To inquire upon a timer associated with another activity, specify the ACTIVITYID option. (The activity identifier specified on the ACTIVITYID option may, for example, have been returned on a **GETNEXT ACTIVITY** command during a browse operation.)

Options

ABSTIME(data-area)
Returns, in packed decimal format, the time at which the timer will expire, expressed in milliseconds since 00:00 on 1 January 1900 (rounded to the nearest hundredth of a second).

You can use FORMATTIME to change the data into other familiar formats.

ACTIVITYID(data-value)
Specifies the identifier (1–52 characters) of the activity with which the timer is associated.

If this option is omitted, the current activity is assumed.

EVENT(data-area)
Returns the 16-character name of the event (if any) associated with the timer.

STATUS(cvda)
Indicates the state of the timer. CVDA values are:

EXPIRED
The timer expired normally.

FORCED
Expiry of the timer was forced by means of a FORCE TIMER command.

UNEXPIRED
The timer has not yet expired.

TIMER(data-value)
specifies the name (1–16 characters) of the timer.

Conditions

109 ACTIVITYERR
RESP2 values:

3
The activity indicated by the ACTIVITYID option could not be found.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of a currently—active activity.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

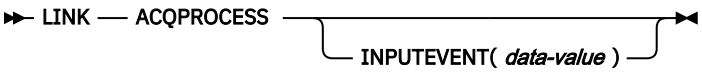
115 TIMERERR
RESP2 values:

1
The timer specified on the TIMER option could not be found.

LINK ACQPROCESS

Execute a CICS business transaction services process synchronously without context-switching.

LINK ACQPROCESS



Conditions: EVENTERR, INVREQ, IOERR, NOTAUTH, PGMIDERR, PROCESSBUSY, PROCESSERR

Description

LINK ACQPROCESS executes the CICS business transaction services process currently acquired by the requestor. The process is executed synchronously with the requestor, with no context-switching.

The only process that a program can link is the one that it has acquired in the current unit of work. (Note, however, that if the program is running as the activation of an activity, it must use a RUN, not a LINK, command to activate the process it has acquired.) See Acquiring processes and activities .

To check the response from the process, the CHECK ACQPROCESS command must be used. This is because the response to the request to activate the process does not contain any information about the success or failure of the process itself—only about the success or failure of the request to activate it. Typically, the CHECK command is issued immediately after the LINK command.

LINK ACQPROCESS causes BTS to invoke the process's root activity and send it an input event. If the root activity is in its initial state—that is, if this is the first time it is to be run—CICS sends it the DFHINITIAL system event. If the root activity is not in its initial state, the input event must be specified on the INPUTEVENT option.

No context-switching

When an process is activated by a LINK ACQPROCESS command, it is invoked synchronously with the requestor and:

- In the same unit of work as the requestor
- With the transaction attributes (TRANSID and USERID) of the requesting transaction.

In other words, there is no context-switch. To invoke a process synchronously *with* context-switching—that is, in a separate UOW from that of the requesting transaction and with the TRANSID and USERID attributes specified on its DEFINE PROCESS command—use the RUN ACQPROCESS SYNCHRONOUS command.

Note: A context-switch always occurs when a process is run asynchronously.

If performance is more important than failure isolation, recoverability, and security, use LINK ACQPROCESS rather than RUN ACQPROCESS SYNCHRONOUS.

Options

ACQPROCESS
specifies that the process currently acquired by the requestor is to be run.

INPUTEVENT(data-value)
specifies the name (1–16 characters) of the event that causes the process to be attached.

You *must not* specify this option if the process's root activity is in its initial state; that is, if this is the first time the process is to be run. In this case, CICS sends the root activity the DFHINITIAL system event.

You *must* specify this option if the root activity is not in its initial state; that is, if it has been activated before.

If you specify INPUTEVENT, for the LINK command to be successful the root activity must have defined the named event as an input event.

Conditions

111 EVENTERR
RESP2 values:

7
The event named on the INPUTEVENT option has not been defined by the root activity of the process to be run as an input event; or its fire status is FIRED.

16 INVREQ
RESP2 values:

15
The task that issued the LINK command has not defined or acquired a process.

23
The process is suspended, and therefore cannot be run synchronously.

40
The program that implements the process to be run is remote.

44
A LINK has been attempted to a Java™ program, but the JVM pool is disabled.

45
A LINK has been attempted to a Java program, but the JVM profile cannot be found.

46
A LINK has been attempted to a Java program, but the JVM profile is not valid.

47
A LINK has been attempted to a Java program, but the system properties file cannot be found.

48
A LINK has been attempted to a Java program, but the user class cannot be found.

49
The shared class cache is STOPPED and autostart is disabled, so a Java program requesting use of the shared class cache cannot be executed.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to run the process.

27 PGMIDERR
RESP2 values:

1
A program has no installed resource definition and either program autoinstall was switched off, or the program autoinstall user program indicated that the program should not be autoinstalled.

2
A program is disabled.

3
A program could not be loaded because:

- This was the first load of the program and the program load failed, usually because the load module could not be found.
- This was a subsequent load of the program, but the first load failed.

In order to reset the load status the load module must be in the DFHRPL or dynamic LIBRARY concatenation, and a SET PROGRAM NEWCOPY will be required.

21
The program autoinstall user program failed either because the program autoinstall user program is incorrect, incorrectly defined, or as a result of an abend in the program autoinstall user program. Program autoinstall is disabled and message DFHPG0202 or DFHPG0203 written to the CSPL.

22
The model returned by the program autoinstall user program was not defined to CICS, or was not enabled.

23
The program autoinstall user program returned invalid data.

24
Define for the program failed due to autoinstall returning an invalid program name or definition.

106 PROCESSBUSY
RESP2 values:

13
The request timed out. It may be that another task using this process-record has been prevented from ending.

108 PROCESSERR
RESP2 values:

6
Another process is current. That is, the program that issued the LINK command cannot link to the process it has acquired because it is itself running as an activation of a process.

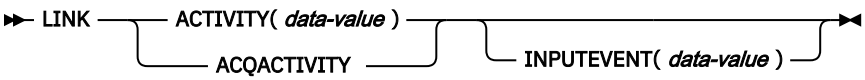
9
The process-type could not be found.

14
The root activity of the process to be run is not in INITIAL or DORMANT mode.

LINK ACTIVITY

Execute a CICS business transaction services activity synchronously without context-switching.

LINK ACTIVITY



Conditions: ACTIVITYBUSY, ACTIVITYERR, EVENTERR, INVREQ, IOERR, LOCKED, NOTAUTH, PGMIDERR

Description

LINK ACTIVITY executes a CICS business transaction services activity synchronously with the requestor, with no context-switching. The activity must previously have been defined to BTS.

LINK ACTIVITY causes BTS to invoke the activity and send it an input event. If the activity is in its initial state—that is, if this is the first time it is to be run, or if it has been reset by a RESET ACTIVITY command—CICS sends it the DFHINITIAL system event. If the activity is not in its initial state, the input event must be specified on the INPUTEVENT option.

The only activities a program can link to are as follows:

- If it is running as the activation of an activity, its own child activities. It can link to several of its child activities within the same unit of work.
- The activity it has acquired, by means of an ACQUIRE ACTIVITYID command, in the current unit of work. (Note, however, that if the program is running as the activation of an activity, it must use a RUN, not a LINK, command to activate the activity it has acquired.)

To check the response from the activity, the CHECK ACTIVITY command must be used. This is because the response to the request to activate the activity does not contain any information about the success or failure of the activity itself—only about the success or failure of the request to activate it. Typically, the CHECK command is issued immediately after the LINK command.

No context-switching

When an activity is activated by a LINK ACTIVITY command, it is invoked synchronously with the requestor and:

- In the same unit of work as the requestor
- With the transaction attributes (TRANSID and USERID) of the requesting transaction.

In other words, there is no **context-switch**. To invoke an activity synchronously *with* context-switching —that is, in a separate UOW from that of the requesting transaction and with the TRANSID and USERID attributes specified on its DEFINE ACTIVITY command—use the RUN ACTIVITY SYNCHRONOUS command.

Note: A context-switch always occurs when an activity is run asynchronously.

If performance is more important than failure isolation, recoverability, and security, use LINK ACTIVITY rather than RUN ACTIVITY SYNCHRONOUS.

Options

ACQACTIVITY
specifies that the activity to be run is the one that the current unit of work has acquired by means of an ACQUIRE ACTIVITYID command.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the activity to be run. The name must be that of a child of the current activity.

INPUTEVENT(data-value)
specifies the name (1–16 characters) of the event that causes the activity to be attached.

You *must not* specify this option if the activity is in its initial state; that is, if this is the first time it is to be run, or if it has been reset by a RESET ACTIVITY command. In this case, CICS sends the activity the DFHINITIAL system event.

You *must* specify this option if the activity is not in its initial state; that is, if it has been activated before, and has not been reset by a RESET ACTIVITY command.

If you specify INPUTEVENT, for the LINK command to be successful the activity to be attached must have defined the named event as an input event.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

14
The target activity is not in the correct mode to process the specified event option. If the INPUTEVENT option was not specified, the activity must be in INITIAL mode. If the INPUTEVENT option was specified, the activity must be in DORMANT mode.

111 EVENTERR
RESP2 values:

7
The event named on the INPUTEVENT option has not been defined by the activity to be run as an input event; or its fire status is FIRED.

16 INVREQ
RESP2 values:

4
The ACTIVITY option was used to name a child activity, but the command was issued outside the scope of a currently-active activity.

21
The activity is suspended, and therefore cannot be run synchronously.

24
The ACQACTIVITY option was used, but the issuing task has not acquired an activity.

40
The program that implements the activity is remote.

44
A LINK has been attempted to a Java program, but the JVM pool is disabled.

45
A LINK has been attempted to a Java program, but the JVM profile cannot be found.

46
A LINK has been attempted to a Java program, but the JVM profile is not valid.

47
A LINK has been attempted to a Java program, but the system properties file cannot be found.

48
A LINK has been attempted to a Java program, but the user class cannot be found.

49
The shared class cache is STOPPED and autostart is disabled, so a Java program requesting use of the shared class cache cannot be executed.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to run the activity.

27 PGMIDERR
RESP2 values:

1
A program has no installed resource definition and either program autoinstall was switched off, or the program autoinstall user program indicated that the program should not be autoinstalled.

2
A program is disabled.

3
A program could not be loaded because:

- This was the first load of the program and the program load failed, usually because the load module could not be found.
- This was a subsequent load of the program, but the first load failed.

In order to reset the load status the load module must be in the DFHRPL or dynamic LIBRARY concatenation, and a SET PROGRAM NEWCOPY will be required.

21
The program autoinstall user program failed either because the program autoinstall user program is incorrect, incorrectly defined, or as a result of an abend in the program autoinstall user program. Program autoinstall is disabled and message DFHPG0202 or DFHPG0203 written to the CSPL.

22
The model returned by the program autoinstall user program was not defined to CICS, or was not enabled.

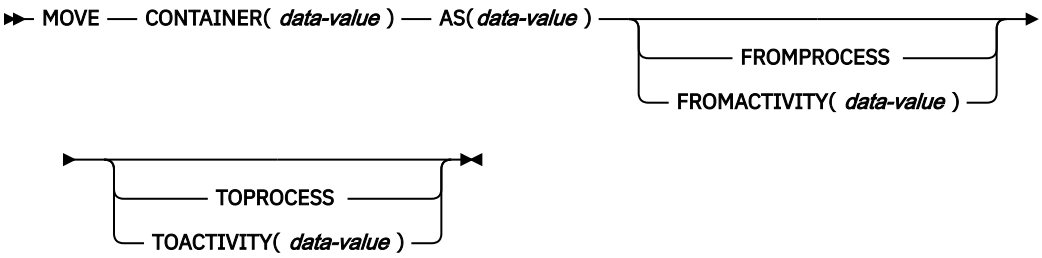
23
The program autoinstall user program returned invalid data.

24
Define for the program failed due to autoinstall returning an invalid program name or definition.

MOVE CONTAINER (BTS)

Move a BTS data-container (and its contents) from one activity to another.

MOVE CONTAINER (BTS)



Conditions: ACTIVITYERR, CONTAINERERR, INVREQ, IOERR, LOCKED

Description

MOVE CONTAINER (BTS) moves a data-container (and its contents) from one BTS activity to another. After the move, the source container is destroyed.

The source and target containers are identified by name and by the activities that own them. The activity that owns the source container can be identified:

- Explicitly, by specifying the FROMPROCESS or FROMACTIVITY option.
- Implicitly, by omitting the FROMPROCESS and FROMACTIVITY options. If these are omitted, the current activity is implied.

Similarly, the activity that owns the target container can be identified:

- Explicitly, by specifying the TOPROCESS or TOACTIVITY option.
- Implicitly, by omitting the TOPROCESS and TOACTIVITY options. If these are omitted, the current activity is implied.

You can move a container:

- From the current activity to a child of the current activity
- From a child of the current activity to the current activity
- From the current activity to the current activity (thus renaming the container)
- From one child of the current activity to another

In addition, *if the current activity is the root activity*, you can move a container:

- From the current process to the current (root) activity
- From the current process to a child of the current activity
- From the current process to the current process (thus renaming the container)

- From the current activity to the current process
- From a child of the current activity to the current process

You can use MOVE CONTAINER, instead of GET CONTAINER and PUT CONTAINER, as a more efficient way of transferring data between activities—for an explanation, see Container commands .

Note:

1. If the source container does not exist, an error occurs.
2. If the target container does not already exist, it is created. If the target container already exists, its previous contents are overwritten.
3. You cannot move containers from one process to another. Both the source and target containers must be within the scope of the current process.
4. Only the root activity can specify a process-container as the source or target of a MOVE CONTAINER command.

   A process's containers are *not* the same as its root activity's containers.

See also "GET CONTAINER (BTS)" on page 186 and "PUT CONTAINER (BTS)" on page 210.

Options

AS(data-value)
specifies the name (1–16 characters) of the target container. If the target container already exists, its contents are overwritten.

CONTAINER(data-value)
specifies the name (1–16 characters) of the source container that is to be moved.

FROMACTIVITY(data-value)
specifies the name (1–16 characters) of the activity that owns the source container. If specified, this option must name a child of the current activity (or the current activity itself).

FROMPROCESS
specifies that the source container is owned by the current process—that is, the process that the program that issues the command is executing on behalf of.

TOACTIVITY(data-value)
specifies the name (1–16 characters) of the activity that owns the target container. If specified, this option must name a child of the current activity (or the current activity itself).

TOPROCESS
specifies that the target container is owned by the current process—that is, the process that the program that issues the command is executing on behalf of.

Conditions

109 ACTIVITYERR
RESP2 values:

8
The activity named on the FROMACTIVITY or TOACTIVITY option could not be found.

110 CONTAINERERR
RESP2 values:

10
The container named on the CONTAINER option could not be found.

26
The process container named on the CONTAINER option is read-only.

16 INVREQ
RESP2 values:

4
The command was issued outside the scope of a currently-active activity.

25
The FROMPROCESS or TOPROCESS option was used, but the command was issued outside the scope of a currently-active process.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

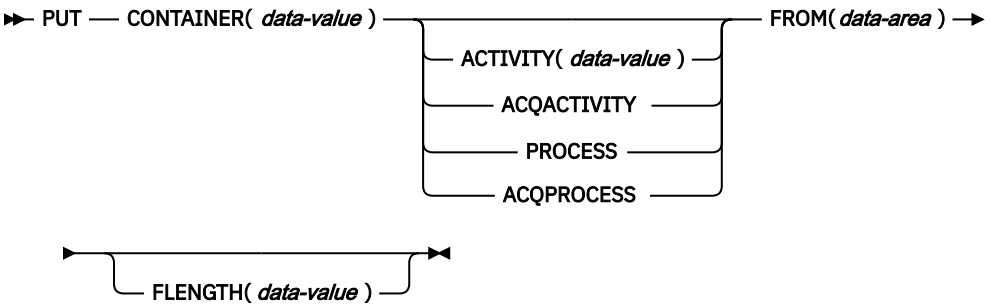
31
The record on the repository file is in use.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

PUT CONTAINER (BTS)

Save data in a named BTS data-container.

PUT CONTAINER (BTS)



Conditions: ACTIVITYERR, CONTAINERERR, INVREQ, IOERR, LOCKED, PROCESSBUSY

Description

PUT CONTAINER (BTS) saves data and places it in a container associated with a specified BTS activity or process.

The container is identified by name. The process or activity that owns the container can be identified:

- Explicitly, by specifying one of the PROCESS- or ACTIVITY-related options.
- Implicitly, by omitting the PROCESS- and ACTIVITY-related options. If these are omitted, the current activity is implied.

Note:

1. There is no limit to the number of containers that can be associated with an activity.
2. Different activities can own identically-named containers—these are different containers.
3. If the named container does not already exist, it is created. If the named container already exists, its previous contents are overwritten.
4. Containers owned by a process (*process-containers*) can be read by every activity in the process. However, they can be updated only by the root activity, or by a program that has acquired the process.

   A process's containers are *not* the same as its root activity's containers.

See also “GET CONTAINER (BTS)” on page 186 and “MOVE CONTAINER (BTS)” on page 208.

Options

ACQACTIVITY
specifies either of the following:

- If the program that issues the command has acquired a process, that the container is owned by the root activity of that process.
- Otherwise, that the container is owned by the activity that the program has acquired by means of an ACQUIRE ACTIVITYID command.

ACQPROCESS
specifies that the container is owned by the process that the program that issues the command has acquired in the current unit of work.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the activity that owns the container. This must be a child of the current activity.

CONTAINER(data-value)
specifies the name (1–16 characters) of the container into which data is to be placed.

The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

FLENGTH(data-value)
specifies, as a fullword binary value, the length of the data area from which data is to be read.

FROM(data-area)
specifies an area of working storage from which the data to be saved is to be read.

PROCESS
specifies that the container into which data is to be placed is owned by the current process—that is, the process that the program that issues the command is executing on behalf of.

Conditions

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

110 CONTAINERERR
RESP2 values:

10
The container named on the CONTAINER option could not be found.

18
The name specified on the CONTAINER option contains an illegal character or combination of characters.

26
The process container named on the CONTAINER option is read-only.

16 INVREQ
RESP2 values:

1
The DATATYPE option was specified without the CHANNEL option, and there is no current channel (because the program that issued the command was not passed one.) DATATYPE is valid only on PUT CONTAINER commands that specify (explicitly or implicitly) a channel. It is not valid on PUT CONTAINER (BTS) commands.

2
The FROMCCSID option was specified without the CHANNEL option, and there is no current channel (because the program that issued the command was not passed one.) FROMCCSID is valid only on PUT CONTAINER commands that specify (explicitly or implicitly) a channel. It is not valid on PUT CONTAINER (BTS) commands.

4
The command was issued outside the scope of a currently-active activity.

15
The ACQPROCESS option was used, but the unit of work that issued the request has not acquired a process.

24
The ACQACTIVITY option was used, but the unit of work that issued the request has not acquired an activity.

25
The PROCESS option was used, but the command was issued outside the scope of a currently-active process.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

31
The record on the repository file is in use.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

106 PROCESSBUSY
RESP2 values:

13
The request could not be satisfied because the process record is locked by another task.

REMOVE SUBEVENT

Remove a sub-event from a BTS composite event.

REMOVE SUBEVENT

Conditions: EVENTERR, INVREQ

Description

REMOVE SUBEVENT removes a sub-event from a named BTS composite event.

This call does not delete the removed event. Nor does it reset the event's fire status. Note that, after this call, the removed event—because it is no longer a sub-event—will cause the current activity to be reattached if it fires.

Removing a sub-event causes the composite's predicate to be re-evaluated.

Options

EVENT(data-value)
specifies the name (1–16 characters) of the composite event.

SUBEVENT(data-value)
specifies the name (1–16 characters) of the event which is to be removed from the named composite event.

Conditions

111 EVENTERR
RESP2 values:

4
The event specified on the EVENT option is not recognized by BTS.

5
The sub-event specified on the SUBEVENT option is not recognized by BTS.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of an activity.

2
The event specified on the EVENT option is not a composite event.

3
The event specified on the SUBEVENT option is not a sub-event of the composite event specified on the EVENT option.

RESET ACQPROCESS

Reset a BTS process to its initial state.

RESET ACQPROCESS

Conditions: INVREQ, IOERR, LOCKED, NOTAUTH, PROCESSBUSY, PROCESSERR

Description

RESET ACQPROCESS resets the currently-acquired BTS process to its initial state. Any descendant activities of the root activity are deleted.

Note: RESET has no effect on the process containers, nor on the root activity's containers, the contents of which are unchanged.

Issue this command, before a second RUN command, when a process needs to be retried. When the process is re-run, the root activity is sent a DFHINITIAL event.

To be eligible to be reset, a process must:

1. Have been acquired in the current unit of work—that is, it must be the currently-acquired process.
2. Be in one of the following modes:
   - COMPLETE. This is the usual case. Perhaps the process has completed abnormally, and needs to be reset before being retried.
   - INITIAL. The process has not yet been run.

Options

ACQPROCESS
specifies that the process that is currently acquired by the requestor is to be reset.

Conditions

16 INVREQ
RESP2 values:

15
The unit of work that issued the request has not acquired a process.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to reset the process.

106 PROCESSBUSY
RESP2 values:

13
The request timed out. It may be that another task using this process-record has been prevented from ending.

108 PROCESSERR
RESP2 values:

14
The process to be reset is not in COMPLETE or INITIAL mode.

RESET ACTIVITY

Reset a BTS activity to its initial state.

RESET ACTIVITY

Conditions: ACTIVITYBUSY, ACTIVITYERR, INVREQ, IOERR, LOCKED, NOTAUTH

Description

RESET ACTIVITY resets a BTS child activity to its initial state. Its completion event is added to the parent's event pool, with the fired status set to NOTFIRED. If the activity has children of its own, they are deleted.

Note: RESET has no effect on the contents of the activity's data containers, which are unchanged.

Issue this command, before a second RUN command, when an activity needs to be retried. When the activity is re-run, it is sent a DFHINITIAL event.

To be eligible to be reset, an activity must:

1. Be a child of the activity that issues the RESET command.
2. Be in one of the following modes:

- COMPLETE. This is the usual case. Perhaps the activity has completed abnormally, and needs to be reset before being retried.
- INITIAL. The activity has not yet been run.

Options

ACTIVITY(data-value)
specifies the name (1–16 characters) of the activity to be reset. This must be a child of the current activity.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

109 ACTIVITYERR
RESP2 values:

8
The activity named in the ACTIVITY option is not a child of the current activity.

14
The activity to be reset is not in COMPLETE or INITIAL mode.

16 INVREQ
RESP2 values:

4
The RESET ACTIVITY command was issued outside the scope of a currently-active activity.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

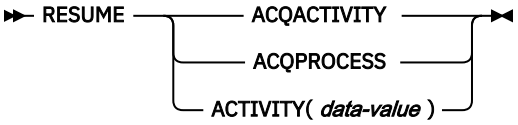
70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to reset the activity.

RESUME

Resume a suspended BTS process or activity.

RESUME



Conditions: ACTIVITYBUSY, ACTIVITYERR, INVREQ, IOERR, LOCKED, PROCESSERR

Description

RESUME resumes a BTS process or activity that has previously been suspended (by means of a SUSPEND command). That is, it allows the process or activity to be reattached when events in its event pool are fired. If events that would normally have caused reattachment have occurred during the time the process or activity was suspended, the latter is reattached for all these events.

The only process a program can resume is the one it has acquired in the current unit of work.

The only activities a program can resume are as follows:

- If it is running as the activation of an activity, its own child activities. It can resume several of its child activities within the same unit of work.
- The activity it has acquired, by means of an ACQUIRE ACTIVITYID command, in the current unit of work.

Options

ACQACTIVITY
specifies that the activity to be resumed is the one that the current unit of work has acquired by means of an ACQUIRE ACTIVITYID command.

ACQPROCESS
specifies that the process that is currently acquired by the requestor is to be resumed.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the child activity to be resumed.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

14
The activity is in COMPLETE or CANCELLING mode, and therefore cannot be resumed.

16 INVREQ
RESP2 values:

4
The ACTIVITY option was used to name a child activity, but the command was issued outside the scope of a currently-active activity.

15
The ACQPROCESS option was used, but the unit of work that issued the request has not acquired a process.

24
The ACQACTIVITY option was used, but the unit of work that issued the request has not acquired an activity.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

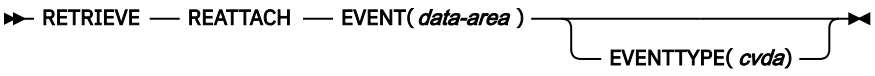
108 PROCESSERR
RESP2 values:

14
The process is in COMPLETE or CANCELLING mode, and therefore cannot be resumed.

RETRIEVE REATTACH EVENT

Retrieve the name of an event that caused the current BTS activity to be reattached.

RETRIEVE REATTACH EVENT



Conditions: END, INVREQ

Description

RETRIEVE REATTACH EVENT:

- Returns the name of the next event in the current BTS activity's reattachment queue.
- If the retrieved event is atomic, resets its fire status to NOTFIRED. (Composite events are not reset by this command, but only when their predicates become false.)

Use this command to find the name of the event that caused the activity to be reattached. In some cases, reattachment could result from the firing of more than one event—if, for example, the activity has previously been suspended, and reattachment events occurred while it was suspended; or if two or more timer events fire simultaneously. The event name or names are placed on the reattachment queue, from where they can be retrieved by issuing one or more RETRIEVE REATTACH EVENT commands.

Each time it is activated, an activity must deal with at least one reattachment event. That is, it must issue at least one RETRIEVE REATTACH EVENT command, and (if this is not done automatically by CICS) reset the fire status of the retrieved event to NOTFIRED—see Resetting and deleting reattachment events . Failure to do so results in the activity completing abnormally, because it has made no progress—it has not reset any reattachment events and is therefore in danger of getting into an unintentional loop.

If there are multiple events on its reattachment queue, an activity can, by issuing multiple RETRIEVE REATTACH EVENT commands, deal with several or all of them in a single activation. Alternatively, it can deal with them singly, by issuing only one RETRIEVE command per activation and returning; it is then reactivated to deal with the next event on its reattachment queue. Which approach you choose is a matter of program design. Bear in mind, if you deal with several reattachment events in the same activation, that a syncpoint does not occur until the activation returns.

Note: The retrieval of a composite event from the reattachment queue does not reset the state of the composite event to NOTFIRED. Thus, if it retrieves a composite reattachment event, the activity program

may need to issue one or more RETRIEVE SUBEVENT commands, to retrieve (and reset) the sub-event or sub-events that have fired. This in turn causes the fire status of the composite event to be re-evaluated.

Options

EVENT(data-area)
returns the 16-character name of the event which caused this activity to be reattached.

EVENTTYPE(cvda)
returns the type of the reattachment event. CVDA values are:

ACTIVITY
Activity completion.

COMPOSITE
Composite.

INPUT
Input

SYSTEM
The BTS system event, DFHINITIAL.

TIMER
Timer.

Conditions

83 END
RESP2 values:

8
There are no more events to retrieve.

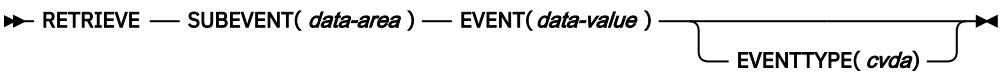
16 INVREQ
RESP2 values:

1
The command was issued outside the scope of an activity.

RETRIEVE SUBEVENT

Retrieve the name of the next sub-event in a BTS composite event's sub-event queue.

RETRIEVE SUBEVENT



Conditions: END, EVENTERR, INVREQ

Description

RETRIEVE SUBEVENT:

- Retrieves the name of the next sub-event in a BTS composite event's sub-event queue.
- Resets the retrieved sub-event's fire status to NOTFIRED.
- Causes the composite event's fire status to be re-evaluated.

The firing of a composite event results from the firing of a set of one or more sub-events. The names of sub-events that have fired are placed on the composite event's sub-event queue, from which they can be retrieved, in sequence, by issuing successive RETRIEVE SUBEVENT commands.

You can use this command to discover which sub-event or sub-events caused a composite event to fire.

Note:

1. The presence of events on the sub-event queue does not imply that the composite event has fired. (Some sub-events in the set required to fire the composite event may still be in NOTFIRED state, and not yet on the sub-event queue.) To discover whether a composite event has fired, use the TEST EVENT command.
2. Retrieval is destructive; when the name of a fired sub-event is retrieved, that sub-event cannot be retrieved again.
3. Because it resets the fire status of the sub-event, RETRIEVE SUBEVENT causes the fire status of the composite event to be re-evaluated.

Options

EVENT(data-value)
specifies the name (1–16 characters) of the composite event.

EVENTTYPE(cvda)
returns the type of the sub-event. CVDA values are:

ACTIVITY
Activity completion.

INPUT
Input

TIMER
Timer.

SUBEVENT(data-area)
returns the 16-character name of the sub-event at the head of the sub-event queue.

Conditions

83 END
RESP2 values:

9
There are no more sub-events to retrieve.

10
The composite event contains no sub-events (it is empty).

111 EVENTERR
RESP2 values:

4
The event specified on the EVENT option is not recognized by BTS.

16 INVREQ
RESP2 values:

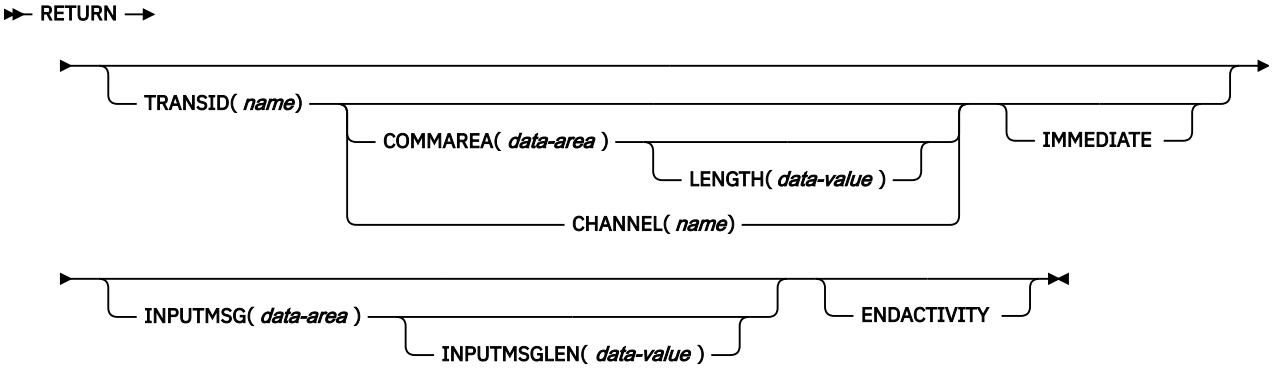
1
The command was issued outside the scope of an activity.

2
The event specified on the EVENT option is invalid. It is not a composite event.

RETURN

Return program control.

RETURN



Conditions: CHANNELERR, INVREQ, LENGERR

This command is threadsafe.

Description

RETURN returns control from an application program either to an application program at the next higher logical level, or to CICS.

When returning a communications area (COMMAREA), the LENGTH option specifies the length of the data to be passed. The LENGTH value being passed must not be greater than the length of the data area specified in the COMMAREA option. If it is, the results are unpredictable and may result in a LENGERR condition, as described in the section about passing data to other programs in Passing data to other programs.

The valid range for the COMMAREA length is 0 through 32 763 bytes. If the length provided is outside this range, the LENGERR condition occurs.

The COMMAREA, IMMEDIATE, and CHANNEL options can be used only when the RETURN command is returning control to CICS; otherwise, the INVREQ condition occurs.

No resource security checking occurs on the RETURN TRANSID command. However, transaction security checking is still available when CICS attaches the returned transaction.

Options

CHANNEL(*name*)

specifies the name (1–16 characters) of a channel that is to be made available to the next program that receives control. The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters. If the channel does not exist, it is created. This new channel remains in scope until the link level changes. For more information about channel scope, see The scope of a channel.

Channel names are always in EBCDIC. The allowable set of characters for channel names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that, if a channel is to be shipped between regions (that is, if the transaction named on the TRANSID option is remote), the characters used in naming it should be restricted to A-Z a-z 0-9 & : = , ; < > . - and _.

You can specify the channel name DFHTRANSACTION to use a transaction channel. A transaction channel does not go out of scope when the link level changes: it is always accessible in the transaction. For more information, see Channels and containers.

The program that issues the RETURN command can do one of the following:

- Have already created the channel by means of one or more PUT CONTAINER CHANNEL commands
- Specify its current channel, by name
- Name a non-existent channel, in which case a new, empty, channel is created

This option is valid only on a RETURN command issued by a program at the highest logical level; that is, a program returning control to CICS.

COMMAREA(*data-area*)
specifies a communication area that is to be made available to the next program that receives control. In a COBOL receiving program, you must give this data area the name DFHCOMMAREA. See Sharing data across transactions for more information about the CICS COMMAREA. Because the data area is freed before the next program starts, a copy of the data area is created and a pointer to the copy is passed.

The communication area specified is passed to the next program that runs at the terminal. To ensure that the communication area is passed to the correct program, include the IMMEDIATE option.

This option is valid only on a RETURN command issued by a program at the highest logical level, that is, a program returning control to CICS.

ENDACTIVITY
This option is for use by programs that implement CICS business transaction services (BTS) activities. It specifies that the current activity is completing, and is not to be reactivated.

If there are no user events in the activity's event pool, the activity completes normally.

If there are user events (fired or unfired) in the activity's event pool:

- If one or more of the events are activity completion events, the activity abends. Trying to force an activity to complete before it has dealt with one or more of its child activities is a program logic error.
- If none of the events are activity completion events, the events are deleted and the activity completes normally.

For information about BTS in general and the ENDACTIVITY option in particular, see Activity completion.

This option is ignored outside the CICS BTS environment.

IMMEDIATE
ensures that the transaction specified in the TRANSID option is attached as the next transaction regardless of any other transactions enqueued by ATI for this terminal. The next transaction starts immediately and appears to the operator as having been started by terminal data. If the terminal is using bracket protocol, the terminal is also held in bracket. This option is valid only on a RETURN command issued by a program at the highest logical level, that is a program returning control to CICS.

Note that in a multi region environment, using IMMEDIATE does not affect the transaction definition as this is still found in the terminal-owning region (TOR).

INPUTMSG(*data-area*)
specifies data to be passed either to another transaction, identified by the TRANSID option, or to a calling program in a multiprogram transaction. You can also use INPUTMSG when returning control to CICS from a user-written dynamic transaction routing program, when you might want to modify the initial input.

In all cases, the data in the INPUTMSG data area is passed to the first program to issue a RECEIVE command following the RETURN.

See INPUTMSG for more information and illustrations about the use of INPUTMSG.

INPUTMSGLEN(*data-value*)
specifies a halfword binary value to be used with INPUTMSG.

LENGTH(*data-value*)
specifies a halfword binary value that is the length in bytes of the COMMAREA. For a description of a safe upper limit, see LENGTH options in CICS commands.

TRANSID(*name*)
specifies the transaction identifier (1–4 characters) to be used with the next input message entered from the terminal with which the task that issued the RETURN command has been associated. The specified name must have been defined as a transaction to CICS.

If TRANSID is specified for a program running on a terminal that is defined with a permanent transaction ID, the terminal's permanent transaction is initiated next rather than the transaction specified on the RETURN.

If you specify a TRANSID of binary zeros, the transaction identifier for the next program to be associated with the terminal may be determined from subsequent input from the terminal. Issuing a RETURN with a TRANSID of binary zeros and a COMMAREA can cause unpredictable results if the next transaction is not coded to handle the COMMAREA or if it receives a COMMAREA not intended for it.

If you specify TRANSID on a program that is not at the highest level, and there is a subsequent error on COMMAREA, INPUTMSG, or CHANNEL on the final RETURN, the TRANSID is cleared.

The next transaction identifier is also cleared on an abnormal termination of the transaction.

If IMMEDIATE is specified with this option, control is passed to the transaction specified in the TRANSID option in preference to any transactions enqueued by ATI.

If IMMEDIATE is not specified with this option, an ATI initiated transaction of the same name enqueued to the terminal nullifies this option.

This option is not valid if the transaction issuing the RETURN command is not associated with a terminal, or is associated with an APPC logical unit.

Conditions

122 CHANNELERR
RESP2 values:

1
The name specified on the CHANNEL option contains an illegal character or combination of characters.

16 INVREQ
RESP2 values:

1
A RETURN command with the TRANSID option is issued in a program that is not associated with a terminal.

2
A RETURN command with the CHANNEL, COMMAREA, or IMMEDIATE option is issued by a program that is not at the highest logical level.

4
A RETURN command with the TRANSID option is issued in a program that is associated with an APPC logical unit.

8
A RETURN command with the INPUTMSG option is issued for a program that is not associated with a terminal, or that is associated with an APPC logical unit, or an IRC session.

30
PG domain not initialized. Parameters are not allowed on the EXEC RETURN statement in first stage PLT programs.

200
A RETURN command is issued with an INPUTMSG option by a program invoked by DPL.

203
The CHANNEL option was specified but the remote region to which control is returned does not support channels.

Default action: terminate the task abnormally.

22 LENGERR
RESP2 values:

11
The COMMAREA length is less than 0 or greater than 32763.

26
The COMMAREA ADDRESS passed was zero, but the commarea length was non-zero.

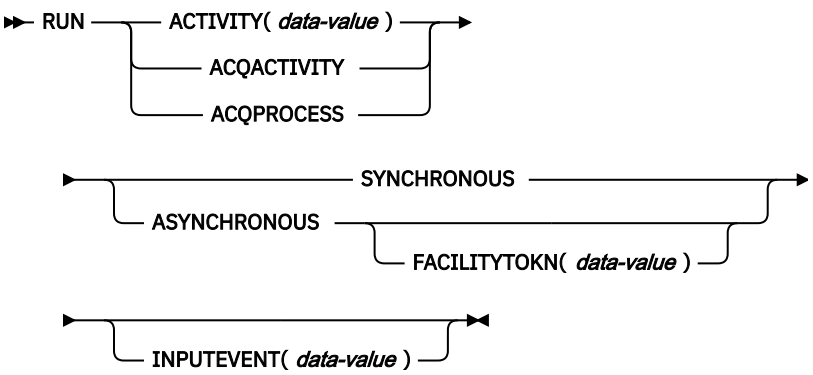
27
The INPUTMSG LENGTH was less than 0 or greater than 32767.

Default action: terminate the task abnormally.

RUN

Execute a CICS business transaction services process or activity synchronously or asynchronously, with context-switching.

RUN



Conditions: ACTIVITYBUSY, ACTIVITYERR, EVENTERR, INVREQ, IOERR, LOCKED, NOTAUTH, PROCESSBUSY, PROCESSERR

Description

RUN executes a CICS business transaction services process or activity synchronously or asynchronously with the requestor, with context-switching. The process or activity must previously have been defined to BTS.

RUN causes BTS to attach the process or activity, by sending it an input event. If the process or activity is in its initial state—that is, if this is the first time it is to be run, or if the activity has been reset by a RESET ACTIVITY command—CICS sends it the DFHINITIAL system event. If the process or activity is dormant—that is, waiting for a reattachment event to occur—the input event must be specified on the INPUTEVENT option.

If the process or activity is in any mode other than INITIAL or DORMANT, it cannot be run.

The SYNCHRONOUS and ASYNCHRONOUS options allow you to specify whether the process or activity should be executed synchronously or asynchronously with the requestor.

Context-switching

When a process or activity is activated by a RUN command, it is run:

- In a separate unit of work from the requestor.
- With the transaction attributes (TRANSID and USERID) specified on the DEFINE PROCESS or DEFINE ACTIVITY command.

In other words, a **context-switch** takes place. The relationship of the process or activity to the requestor is as between separate transactions, except that:

- Data can be passed between the two units of work
- The start and finish of the activity is related to the requestor's syncpoints.

To run a process or activity *without* context-switching—that is, in the same UOW and with the same TRANSID and USERID attributes as the requesting transaction—use the LINK ACQPROCESS, LINK ACQACTIVITY, or LINK ACTIVITY command. This is possible only if the process or activity is run synchronously.

If the ability to isolate a failure is more important than performance, use RUN SYNCHRONOUS rather than LINK.

Activities

The only activities a program can run are as follows:

- If it is running as the activation of an activity, its own child activities. It can run several of its child activities within the same unit of work.
- The activity it has acquired, by means of an ACQUIRE ACTIVITYID command, in the current unit of work.

To check the response from the activity, the CHECK ACTIVITY command must be used. This is because the response to the request to run the activity does not contain any information about the success or failure of the activity itself—only about the success or failure of the request to run it.

Typically, if the activity is run synchronously, the CHECK command is issued immediately after the RUN command. If it is run asynchronously, the CHECK command could be issued:

- When the activity's parent is reattached due to the firing of the activity's completion event
- When the requestor is reattached due to the expiry of a timer.

The activity's completion event is one of the following:

1. The event named on the EVENT option of the DEFINE command for the activity.
2. If the DEFINE command did not specify a completion event, an event of the same name as the activity.

To retry an activity:

1. Issue a RESET ACTIVITY command to reset the activity to its initial state.
2. Issue a RUN command.

Processes

The only process that a program can run is the one that it has acquired in the current unit of work—see Acquiring processes and activities.

To check the response from the process, the CHECK ACQPROCESS command must be used. This is because the response to the request to run the process does not contain any information about the success or failure of the process itself—only about the success or failure of the request to run it.

Typically, if the process is run synchronously, the CHECK command is issued immediately after the RUN command. If the process is run asynchronously, the CHECK command could be issued when the requestor is reattached due to the expiry of a timer.

Options

ACQACTIVITY
specifies that the activity to be run is the one that the current unit of work has acquired by means of an ACQUIRE ACTIVITYID command.

ACQPROCESS
specifies that the process currently acquired by the requestor is to be run.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the activity to be run. The name must be that of a child of the current activity.

ASYNCHRONOUS
specifies that the process or activity is to be executed asynchronously with the requestor.

FACILITYTOKN(data-value)
specifies an 8-byte bridge facility token.

This option applies when a BTS client activity runs a 3270-based pseudoconversational transaction. To ensure that the existing bridge facility is reused for the next transaction in the pseudoconversation, the client passes its token to the next child activity. This is explained in more detail in Reusing existing 3270 applications in BTS .

INPUTEVENT(data-value)
specifies the name (1–16 characters) of the event that causes the process or activity to be attached.

You *must not* specify this option if the process or activity is in its initial state; that is, if this is the first time it is to be run, or if the activity has been reset by a RESET ACTIVITY command. In this case, CICS sends the process or activity the DFHINITIAL system event.

You *must* specify this option if the process or activity is not in its initial state; that is, if it has been activated before, and has not been reset by a RESET ACTIVITY command.

If you specify INPUTEVENT, for the RUN command to be successful the process or activity to be attached must have defined the named event as an input event.

If you issue multiple asynchronous RUN commands against the same activity within the same unit of work:

- If you specify *the same input event*, each RUN command after the first fails.
- If you specify *different input events*, the activity may or may not be invoked as many times as the number of RUN requests—the only guarantee is that it will be invoked at least once. For example, if , within the same unit of work, you issue five asynchronous RUN requests for the same activity, specifying different input events, the activity might be invoked twice. At the first invocation, three input events might be presented, and at the second two.

SYNCHRONOUS
specifies that the process or activity is to be executed synchronously with the requestor.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

14
The activity to be run is not in INITIAL or DORMANT mode.

27
The activity named on the RUN SYNCHRONOUS command has abended.

111 EVENTERR
RESP2 values:

7
The event named on the INPUTEVENT option has not been defined by the activity or process to be run as an input event; or its fire status is FIRED.

16 INVREQ
RESP2 values:

4
The ACTIVITY option was used to name a child activity, but the command was issued outside the scope of a currently-active activity.

15
The task that issued the RUN ACQPROCESS command has not defined or acquired a process.

20
The SYNCHRONOUS option was used, but the activity to be run is suspended.

24
The ACQACTIVITY option was used, but the unit of work that issued the request has not acquired an activity.

28
CICS could not attach the transaction associated with the process or activity to be run. (This response occurs only on RUN SYNCHRONOUS commands.)

32
The SYNCHRONOUS option was used, but the transaction associated with the process or activity to be run is defined as remote. You cannot run a process or activity synchronously if its transaction is remote.

40
The program that implements the process or activity to be run is remote.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to run the process or activity.

106 PROCESSBUSY
RESP2 values:

13
The request timed out. It may be that another task using this process-record has been prevented from ending.

108 PROCESSERR
RESP2 values:

6
You cannot run the current process.

9
The process-type could not be found.

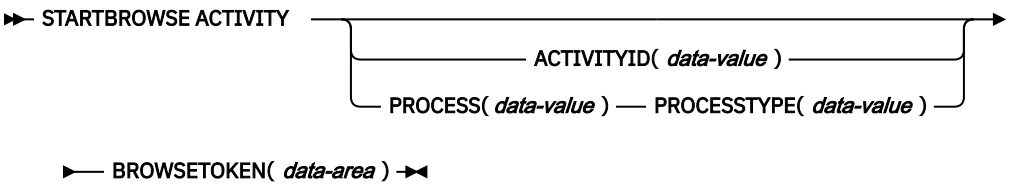
14
The process to be run is not in INITIAL or DORMANT mode.

27
The process named on the RUN SYNCHRONOUS command has abended.

STARTBROWSE ACTIVITY

Start a browse of the child activities of a BTS activity, or of the descendant activities of a process.

STARTBROWSE ACTIVITY



Conditions: ACTIVITYERR, NOTAUTH, PROCESSERR

Description

STARTBROWSE ACTIVITY initializes a browse token which can be used to identify either:

- Each child activity of a specified BTS parent activity
- Each descendant activity of a specified BTS process.

If you specify the ACTIVITYID option, the children (but not the grandchildren nor other descendants) of the specified activity can be browsed. This option takes as its argument an activity identifier. This identifier may, for example, have been returned on a previous GETNEXT ACTIVITY command. If it was, the command starts a browse of child activities one level down the activity tree.

If you specify the PROCESS and PROCESSTYPE options, all the descendant activities of the specified process can be browsed. This type of browse is known as a **flat browse**. A flat browse is one which can return every descendant activity exactly once. A parent activity is always returned before its children. The value returned in the LEVEL option of a GETNEXT ACTIVITY command indicates the depth at which the activity lies in the process's activity-tree, with the root activity having a level of zero.

If you specify neither the ACTIVITYID nor the PROCESS and PROCESSTYPE options, the children of the current activity can be browsed.

Options

ACTIVITYID(data-value)
specifies the identifier (1–52 characters) of the activity whose child activities are to be browsed.

Typically, the activity identifier specified on this option has been returned on a previous GETNEXT ACTIVITY command (or, in the case of a root activity, on a GETNEXT PROCESS command). ACTIVITYID allows you to start a browse of child activities one level down the activity tree.

If you omit both this and the PROCESS option, the children of the current activity are browsed.

BROWSETOKEN(data-area)
specifies a fullword binary data area, into which CICS will place the browse token.

PROCESS(data-value)
specifies the name (1–36 characters) of the process whose descendant activities are to be browsed.

PROCESSTYPE(data-value)
specifies the process-type (1–8 characters) of the process named on the PROCESS option.

Conditions

109 ACTIVITYERR
RESP2 values:

1
The activity indicated by the ACTIVITYID option could not be found.

2
Because neither the ACTIVITYID nor the PROCESS options were specified, a browse of the children of the current activity was implied—but there is no current activity associated with the request.

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access the file whose data set contains the records to be browsed.

108 PROCESSERR
RESP2 values:

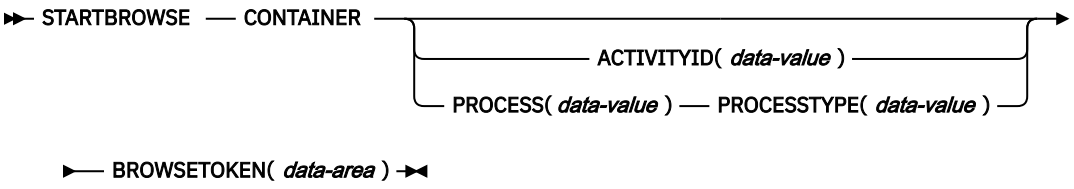
3
The process specified on the PROCESS option could not be found.

4
The process-type specified on the PROCESSTYPE option could not be found.

STARTBROWSE CONTAINER (BTS)

Start a browse of the containers associated with a BTS activity or process.

STARTBROWSE CONTAINER



Conditions: ACTIVITYERR, IOERR, NOTAUTH, PROCESSERR

This command is threadsafe.

Description

STARTBROWSE CONTAINER initializes a browse token which can be used to identify the name of each data-container associated with a specified BTS activity or process.

Note: The browse token should be used only by the program that issues the STARTBROWSE command.

If you do not specify the ACTIVITYI or the PROCESS option, CICS examines the context (channel or BTS) of the request. If a current channel exists, the command is assumed to be a STARTBROWSE CONTAINER (CHANNEL) command. If a current activity exists, its containers are browsed. If neither exists, an ACTIVITYERR 2 is raised: see the description of the ACTIVITYERR condition, below.

Options

ACTIVITYID(data-value)
specifies the identifier (1–52 characters) of the activity whose containers are to be browsed.

Typically, the identifier specified on this option has been returned on a previous GETNEXT ACTIVITY command.

BROWSETOKEN(data-area)
specifies a fullword binary data area, into which CICS will place the browse token.

PROCESS(data-value)
specifies the name (1–36 characters) of the process whose containers are to be browsed.

Note: The containers associated with a process (*process containers*) are globally available throughout the process. They are not the same as the root activity's containers.

PROCESSTYPE(data-value)
specifies the process-type (1–8 characters) of the process named on the PROCESS option.

Conditions

109 ACTIVITYERR
RESP2 values:

1
The activity indicated by the ACTIVITYID option could not be found.

2
None of the ACTIVITYID, PROCESS, or CHANNEL options were specified and there is no current channel and no current activity associated with the request.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

108 PROCESSERR
RESP2 values:

3
The process specified on the PROCESS option could not be found.

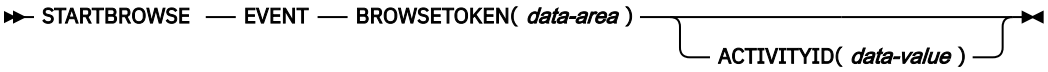
4
The process-type specified on the PROCESSTYPE option could not be found.

13
The request timed out. It may be that another task using this process-record has been prevented from ending.

STARTBROWSE EVENT

Start a browse of events known to a BTS activity.

STARTBROWSE EVENT



Conditions: ACTIVITYERR, INVREQ, IOERR, NOTAUTH

Description

STARTBROWSE EVENT initializes a browse token which can be used to identify each event (including each sub-event and system event) that is within the scope of a specified BTS activity. If you do not specify an activity, events within the scope of the current activity are browsed.

A browse started by STARTBROWSE EVENT returns:

- Atomic events. An atomic event returned on this command may or may not be included in the predicate of a composite event—that is, it may or may not be a sub-event.
- Composite events.
- System events.

Options

ACTIVITYID(data-value)
specifies the identifier (1–52 characters) of the activity whose events are to be browsed.

If you omit this option, events known to the current activity are browsed.

BROWSETOKEN(data-area)
specifies a fullword binary data area, into which CICS will place the browse token.

Conditions

109 ACTIVITYERR
RESP2 values:

1
The activity identifier specified on the ACTIVITYID option does not relate to any activity that is within the scope of this task.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

16 INVREQ
RESP2 values:

1
There is no current activity within the scope of this task.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

STARTBROWSE PROCESS

Start a browse of all processes of a specified type within the CICS business transaction services system.

STARTBROWSE PROCESS

Conditions: IOERR, NOTAUTH, PROCESSERR

Description

STARTBROWSE PROCESS initializes a browse token which can be used to identify each process of a specified type within the CICS business transaction services system.

When you add a process to the BTS system, you use the PROCESSTYPE option of the DEFINE PROCESS command to categorize it. You specify the name of a PROCESSTYPE resource definition, which in turn names a CICS file definition that maps to a physical VSAM data set (the repository) on which details of the process and its constituent activities will be stored. (Records for multiple process-types can be stored on the same repository data set.)

The STARTBROWSE PROCESS command enables you to start a browse of processes of a specified type.

Options

BROWSETOKEN(data-area)
specifies a fullword binary data area, into which CICS will place the browse token.

PROCESSTYPE(data-value)
specifies the process-type (1–8 characters) of the processes to be browsed.

Conditions

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

108 PROCESSERR
RESP2 values:

1
No processes of this process-type could be found.

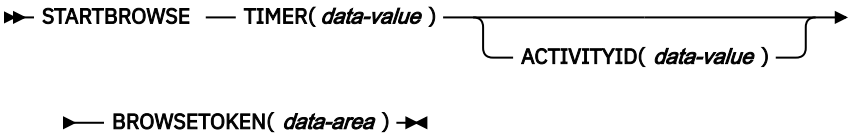
4
The process-type specified on the PROCESSTYPE option could not be found.

13
The request timed out. It may be that another task using this process-record has been prevented from ending.

STARTBROWSE TIMER

Start a browse of a BTS timer.

STARTBROWSE TIMER



Conditions: ACTIVITYERR, INVREQ, IOERR, NOTAUTH, TIMERERR

Description

STARTBROWSE TIMER initializes a browse token that can be used to identify a BTS timer.

Options

ACTIVITYID(data-value)
Specifies the identifier (1–52 characters) of the activity with which the timer is associated.

If this option is omitted, the current activity is assumed.

BROWSETOKEN(data-area)
Specifies a fullword binary data area, into which CICS will place the browse token.

TIMER(data-value)
Specifies the name (1–16 characters) of the timer.

Conditions

109 ACTIVITYERR
RESP2 values:

3
The activity indicated by the ACTIVITYID option could not be found.

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of a currently—active activity.

17 IOERR
RESP2 values:

30
An input/output error has occurred on the repository file.

70 NOTAUTH
RESP2 values:

101
The user associated with the issuing task is not authorized to access this resource in the way requested.

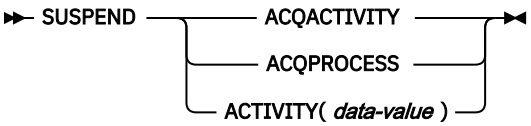
115 TIMERERR
RESP2 values:

1
The timer specified on the TIMER option could not be found.

SUSPEND (BTS)

Suspend a BTS process or activity.

SUSPEND (BTS)



Conditions: ACTIVITYBUSY, ACTIVITYERR, INVREQ, IOERR, LOCKED, PROCESSERR

Description

SUSPEND (BTS) prevents a BTS process or activity being reattached when events in its event pool are fired.

The only process a program can suspend is the one that it has acquired in the current unit of work.

The only activities a program can suspend are as follows:

- If it is running as the activation of an activity, its own child activities. It can suspend several of its child activities within the same unit of work.
- The activity it has acquired, by means of an ACQUIRE ACTIVITYID command, in the current unit of work.

To resume a suspended process or activity, a RESUME command must be issued.

Options

ACQACTIVITY
specifies that the activity to be suspended is the one that the current unit of work has acquired by means of an ACQUIRE ACTIVITYID command.

ACQPROCESS
specifies that the process that is currently acquired by the requestor is to be suspended.

ACTIVITY(data-value)
specifies the name (1–16 characters) of the child activity to be suspended.

Conditions

107 ACTIVITYBUSY
RESP2 values:

19
The request timed out. It may be that another task using this activity-record has been prevented from ending.

109 ACTIVITYERR
RESP2 values:

8
The activity named on the ACTIVITY option could not be found.

16 INVREQ
RESP2 values:

4
The ACTIVITY option was used to name a child activity, but the command was issued outside the scope of a currently-active activity.

14
The activity is in COMPLETE or CANCELLING mode, and therefore cannot be suspended.

15
The ACQPROCESS option was used, but the unit of work that issued the request has not acquired a process.

24
The ACQACTIVITY option was used, but the unit of work that issued the request has not acquired an activity.

17 IOERR
RESP2 values:

29
The repository file is unavailable.

30
An input/output error has occurred on the repository file.

100 LOCKED
The request cannot be performed because a retained lock exists against the relevant record on the repository file.

108 PROCESSERR
RESP2 values:

5
The process could not be found.

TEST EVENT

Test whether a BTS event has fired.

TEST EVENT

Conditions: EVENTERR, INVREQ

Description

TEST EVENT tests whether a named BTS event has occurred (fired).

Options

EVENT(data-value)
specifies the name (1–16 characters) of the event to test for completion.

FIRESTATUS(cvda)
FIRESTATUS returns the fire status of the event. CVDA values are:

FIRED
The event has fired.

NOTFIRED
The event has not fired.

Conditions

111 EVENTERR
RESP2 values:

4
The event specified on the EVENT option is not recognized by BTS.

16 INVREQ
RESP2 values:

1
The command was issued outside the scope of an activity.

Chapter 7. BTS System Programming Reference

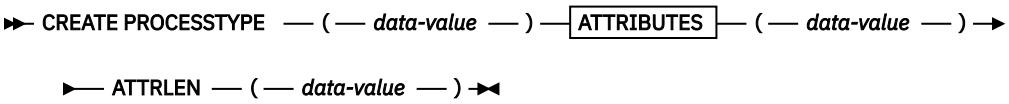
Use the CICS business transaction services system programming commands to work on BTS process-types.

You can use CICS command security to restrict access to the commands described in this section.

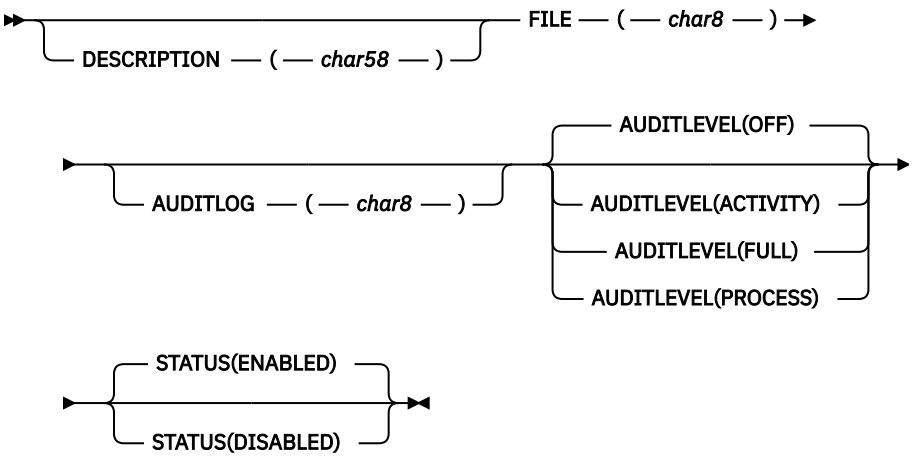
CREATE PROCESSTYPE

Use the **CREATE PROCESSTYPE** command to add the definition of a BTS process-type to the local CICS region.

CREATE PROCESSTYPE



ATTRIBUTES



Conditions: ILLOGIC, INVREQ, LENGERR, NOTAUTH

Description

CREATE PROCESSTYPE adds the definition of a BTS process-type to the local CICS region. The definition is built without reference to data stored on the CSD file. If there is already a process-type by the name you specify in the local CICS region, the new definition replaces the old one; if not, the new definition is added.

A syncpoint is implicit in CREATE PROCESSTYPE processing, except when an exception condition is detected early in processing the command. Uncommitted changes to recoverable resources made up to that point in the task are committed if the CREATE executes successfully and rolled back if not. For other general rules about CREATE commands, see Creating resource definitions.

Options

ATTRIBUTES(data-value)
specifies the attributes of the PROCESSTYPE being added. The list of attributes must be coded as a single character string using the syntax shown in **PROCESSTYPE attributes**. For general rules for specifying attributes, see The ATTRIBUTES option.For details of specific attributes, see CEDA DEFINE PROCESSTYPE.

ATTRLEN(data-value)
specifies the length in bytes of the character string supplied in the ATTRIBUTES option, as a halfword binary value. The length can be from 0 to 32767.

PROCESSTYPE(data-value)
specifies the name (1-8 characters) of the PROCESSTYPE definition to be added to the CICS region. The acceptable characters are A-Z a-z 0-9 $ @ # . / -_ % & ? ! : | " = ¬ , ; < >. Leading and embedded blank characters are not permitted. If the name supplied is less than eight characters, it is padded with trailing blanks up to eight characters.

Conditions

ILLOGIC
RESP2 values:

2
The command cannot be executed because an earlier CONNECTION or TERMINAL pool definition has not yet been completed.

INVREQ
RESP2 values:

n
There is a syntax error in the ATTRIBUTES string, or an error occurred during either the discard or resource definition phase of the processing.

200
The command was executed in a program defined with an EXECUTIONSET value of DPLSUBSET or a program invoked from a remote system by a distributed program link without the SYNCONRETURN option.

LENGERR
RESP2 values:

1
The length you have specified in ATTRLEN is negative.

NOTAUTH
RESP2 values:

100
The user associated with the issuing task is not authorized to use this command.

101
The user associated with the issuing task is not authorized to create a PROCESSTYPE definition with this name.

102
The caller does not have surrogate authority to install the resource with the particular userid.

DISCARD PROCESSTYPE

Use the **DISCARD PROCESSTYPE** command to remove the definition of a specified process-type from the local CICS region.

DISCARD PROCESSTYPE

Conditions: INVREQ, NOTAUTH, PROCESSERR

Description

DISCARD PROCESSTYPE removes the definition of a specified process-type from the local CICS region.

Note:

1. Only disabled process-types can be discarded.
2. If you are using BTS in a single CICS region, you can use the DISCARD PROCESSTYPE command to remove process-types. However, if you are using BTS in a sysplex, it is strongly recommended that you use CICSPlex SM to remove them. If you don't use CICSPlex SM, problems could arise if Scheduler Services routes to this region work that requires a discarded definition.

Options

PROCESSTYPE(data-value)
specifies the name (1-8 characters) of the process-type to be removed.

Conditions

INVREQ
RESP2 values:

2
The process-type named in the PROCESSTYPE option is not disabled.

NOTAUTH
RESP2 values:

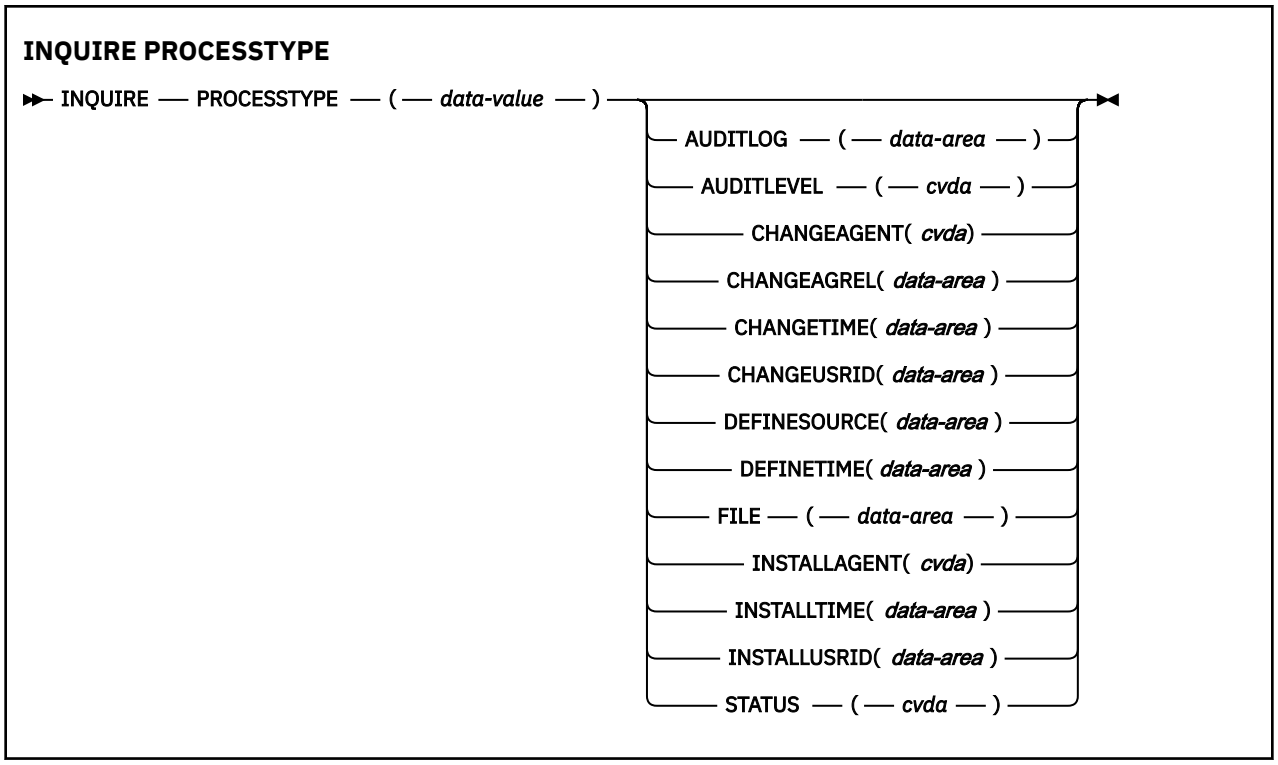
100
The user associated with the issuing task is not authorized to use this command.

PROCESSERR
RESP2 values:

1
The process-type named in the PROCESSTYPE option is not defined in the process-type table (PTT).

INQUIRE PROCESSTYPE

Use the **INQUIRE PROCESSTYPE** command to retrieve the attributes of a process-type.



Conditions: NOTAUTH, PROCESSERR

Description

INQUIRE PROCESSTYPE returns the attributes of a specified process-type.

The resource signature

You can use this command to retrieve the resource signature fields. You can use these fields to manage resources by capturing details of when the resource was defined, installed, and last changed. For more information, see Auditing resources. The resource signature fields are BUNDLE, CHANGEAGENT, CHANGEAGREL, CHANGETIME, CHANGEUSRID, DEFINESOURCE, DEFINETIME, INSTALLAGENT, INSTALLTIME, and INSTALLUSRID. See Summary of the resource signature field values for detailed information about the content of the resource signature fields.

Options

AUDITLEVEL(*cvda*)

Indicates the level of audit currently active for processes of the specified type. CVDA values are as follows:

ACTIVITY

Activity-level auditing. Audit records are written from the following points:

- The process audit points
- The activity primary audit points.

FULL

Full auditing. Audit records are written from the following points:

- The process audit points
- The activity primary *and* secondary audit points.

OFF

No audit trail records are written.

PROCESS
Process-level auditing. Audit records are written from the process audit points only.

For details of the records that are written from the process, activity primary, and activity secondary audit points, see Specifying the level of audit logging.

AUDITLOG(*data-area*)
Returns the 8-character name of the CICS journal used as the audit log for processes of the specified type.

CHANGEAGENT(*cvda*)
Returns a CVDA value that identifies the agent that made the last change to the resource definition. The possible values are as follows:

CREATESPI
The resource definition was last changed by an **EXEC CICS CREATE** command.

CSDAPI
The resource definition was last changed by a CEDA transaction or the programmable interface to DFHEDAP.

CSDBATCH
The resource definition was last changed by a DFHCSDUP job.

DREPAPI
The resource definition was last changed by a CICSPlex SM BAS API command.

CHANGEAGREL(*data-area*)
Returns a 4-digit number of the CICS release that was running when the resource definition was last changed.

CHANGETIME(*data-area*)
Returns an ABSTIME value that represents the time stamp when the resource definition was last changed. For more information about the format of the ABSTIME value, see FORMATTIME.

CHANGEUSRID(*data-area*)
Returns the 8-character user ID that ran the change agent.

DEFINESOURCE(*data-area*)
Returns the 8-character source of the resource definition. The DEFINESOURCE value depends on the CHANGEAGENT value. For more information, see Summary of the resource signature field values.

DEFINETIME(*data-area*)
Returns an ABSTIME value that represents the time stamp when the resource definition was created.

FILE(*data-area*)
Returns the 8-character name of the CICS file associated with the process-type.

INSTALLAGENT(*cvda*)
Returns a CVDA value that identifies the agent that installed the resource. The possible values are as follows:

CREATESPI
The resource was installed by an **EXEC CICS CREATE** command.

CSDAPI
The resource was installed by a CEDA transaction or the programmable interface to DFHEDAP.

GRPLIST
The resource was installed by **GRPLIST INSTALL**.

INSTALLTIME(*data-area*)
Returns an ABSTIME value that represents the time stamp when the resource was installed.

INSTALLUSRID(*data-area*)
Returns the 8-character user ID that installed the resource.

PROCESSTYPE(*data-value*)
Specifies the name (1 - 8 characters) of the process-type being inquired on.

STATUS(*cvda*)
Indicates whether new processes of the specified type can currently be defined. CVDA values are as follows:

DISABLED
The installed definition of the process-type is disabled. New processes of this type cannot be defined.

ENABLED
The installed definition of the process-type is enabled. New processes of this type can be defined.

Conditions

NOTAUTH
RESP2 values:

100
The user associated with the issuing task is not authorized to use this command.

PROCESSERR
RESP2 values:

1
The process-type specified on the PROCESSTYPE option could not be found.

Browsing process-type definitions

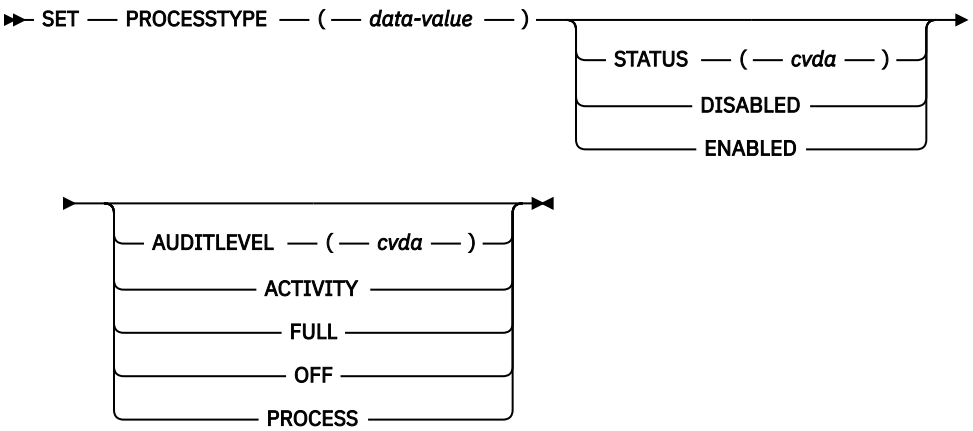
You can browse through all the process-type definitions in your system by using the browse options (START, NEXT, and END) on **INQUIRE PROCESSTYPE** commands.

In browse mode, the definitions are returned in alphabetic order. For general information about browsing, syntax, exception conditions, and examples, see Browsing resource definitions.

SET PROCESSTYPE

Use the **SET PROCESSTYPE** command to change the attributes of a process-type.

SET PROCESSTYPE



Conditions: INVREQ, NOTAUTH, PROCESSERR

Description

SET PROCESSTYPE allows you to change the current state of audit logging and the enablement status of PROCESSTYPE definitions installed on this CICS region.

Note: Process-types are defined in the process-type table (PTT). CICS uses the entries in this table to maintain its records of processes (and their constituent activities) on external data sets. If you are using BTS in a single CICS region, you can use the SET PROCESSTYPE command to modify your process-types. However, if you are using BTS in a sysplex, it is strongly recommended that you use CICSPlex SM to make such changes. This is because it is essential to keep resource definitions in step with each other, across the sysplex.

Options

AUDITLEVEL(cvda)
specifies the level of audit logging to be applied to processes of this type.

Note: If the AUDITLOG attribute of the installed PROCESSTYPE definition is not set to the name of a CICS journal, an error is returned if you try to specify any value other than OFF.

The CVDA values are:

ACTIVITY
Activity-level auditing. Audit records will be written from:

1. The process audit points
2. The activity primary audit points.

FULL
Full auditing. Audit records will be written from:

1. The process audit points
2. The activity primary *and* secondary audit points.

OFF
No audit trail records will be written.

PROCESS
Process-level auditing. Audit records will be written from the process audit points only.

For details of the records that are written from the process, activity primary, and activity secondary audit points, see Specifying the level of audit logging.

PROCESSTYPE(value)
specifies the 8-character name of a process-type defined in the process-type table (PTT), whose attributes are to be changed.

STATUS(cvda)
specifies whether new processes of this type can be created. The CVDA values are:

DISABLED
The installed definition of the process-type is disabled. New processes of this type cannot be defined.

ENABLED
The installed definition of the process-type is enabled. New processes of this type can be defined.

Conditions

INVREQ
RESP2 values:

2
The process-type is not disabled, and therefore cannot be enabled.

3
You have specified an invalid CVDA value on the AUDITLEVEL option.

5
You have specified an invalid CVDA value on the STATUS option.

6
You have specified a value of FULL, PROCESS, or ACTIVITY on the AUDITLEVEL option, but the AUDITLOG attribute of the PROCESSTYPE definition does not specify an audit log.

NOTAUTH
RESP2 values:

100
The user associated with the issuing task is not authorized to use this command.

PROCESSERR
RESP2 values:

1
The process-type named in the PROCESSTYPE option is not defined in the process-type table (PTT).

Notices

This information was developed for products and services offered in the United States of America. 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 character set (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
United States of America

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 Client Relationship Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

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

IBM 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 (CICS TS 5.4) are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- Securing overview
- Developing for external interfaces
- Application development reference
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS TS 5.4, 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 TS 5.4 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
- CICS Transactions
- CICSPlex System Manager (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 TS 5.4, 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.

Apache, Apache Axis2, Apache Maven, Apache Ivy, the Apache Software Foundation (ASF) logo, and the ASF feather logo are trademarks of Apache Software Foundation.

Gradle and the Gradlephant logo are registered trademark of Gradle, Inc. and its subsidiaries 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.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Red Hat®, and Hibernate® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Spring Boot is a trademark of Pivotal Software, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Zowe™, the Zowe logo and the Open Mainframe Project™ are trademarks of The Linux Foundation.

The Stack Exchange name and logos are trademarks of Stack Exchange Inc.

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 (PII) is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect PII. If this Software Offering uses cookies to collect PII, 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 PII 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 PII 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 do not collect PII. 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 PII 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

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y