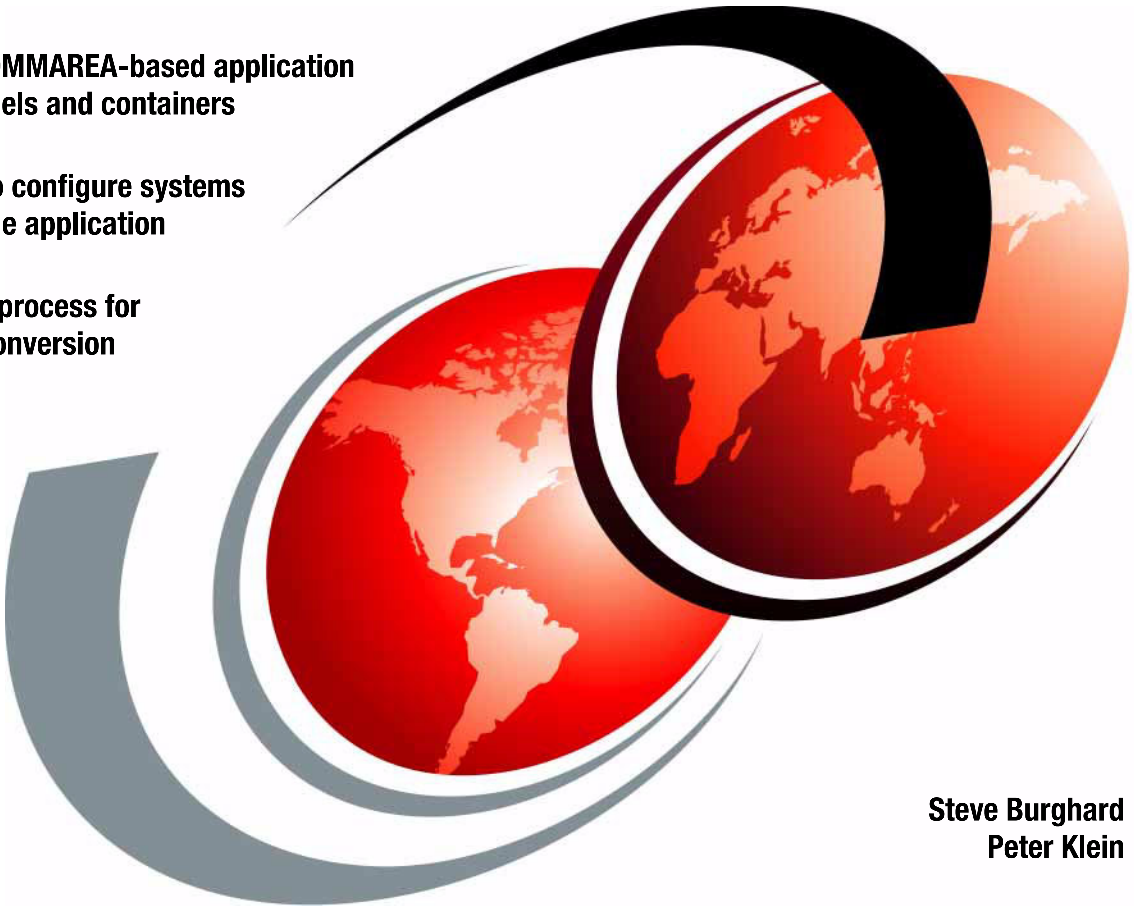
IBM

Using IBM CICS Transaction Server Channels and Containers

Convert a COMMAREA-based application to use channels and containers

Learn how to configure systems with a sample application

Simplify the process for code page conversion



Steve Burghard
Peter Klein

Redbooks

ibm.com/redbooks

International Technical Support Organization

Using IBM CICS Transaction Server Channels and Containers

March 2015

SG24-7227-01

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

Second Edition (March 2015)

This edition applies to version 5, release 2 of CICS Transaction Server for z/OS, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Make sure that you are using the correct edition for the level of the product.

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication describes the new channels and containers support in IBM Customer Information Control System (CICS®) Transaction Server V5.2. The book begins with an overview of the techniques used to pass data between applications running in CICS.

This book describes the constraints that these data techniques might be subject to, and how a channels and containers solution can provide solid advantages alongside these techniques. These capabilities enable CICS to fully comply with emerging technology requirements in terms of sizing and flexibility.

The book then goes on to describe application design, and looks at implementing channels and containers from an application programmer point of view. It provides examples to show how to evolve channels and containers from communication areas (COMMAREAs).

Next, the book explains the channels and containers application programming interface (API). It also describes how this API can be used in both traditional CICS applications and a Java CICS (JCICS) applications.

The business transaction services (BTS) API is considered as a similar yet recoverable alternative to channels and containers. Some authorized program analysis reports (APARs) are introduced, which enable more flexible web services features by using channels and containers.

The book also presents information from a systems management point of view, describing the systems management and configuration tasks and techniques that you must consider when implementing a channels and containers solution.

The book chooses a sample application in the CICS catalog manager example, and describes how you can port an existing CICS application to use channels and containers rather than using COMMAREAs.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie, NY.

Steve Burghard is a member of the CICS 390 Change Team based in Hursley, UK. He has 31 years of experience in CICS. He has been with IBM for 18 years, working as a CICS Technical Solutions Specialist before joining the Change Team. He has contributed widely to other ITSO CICS Redbooks publications, and has a vast range of experience in all areas of IBM CICSPlex® System Manager and IBM CICS Transaction Server for OS/390®.

Peter Klein is a CICS Team Leader at the IBM Germany Customer Support Center. He has 26 years of experience working as a technical support specialist with IBM software products. His expertise includes IBM WebSphere® MQ (IBM MQ), CICSPlex System Manager, and distributed transaction systems. Peter has contributed to several other IBM Redbooks publications and ITSO projects sponsored by IBM Learning Services.

Thanks to the following authors of the previous edition of this book:

- Chris Rayns
- Pietro De Angelis
- Steve Burghard
- David Carey
- Scott Clee
- Peter Klein
- Erhard Woerner

Thanks to the following people for their contributions to this project:

Mark Cocker, CICS Technical Strategy and Planning
IBM Hursley

Rich Conway, ITSO Technical Support
International Technical Support Organization, Poughkeepsie Center

Andy Bates, CICS Product Manager
IBM Hursley

Colin Penfold, CICS Transaction Server, Technical Planner
IBM Hursley

Phil Wakelin, IBM CICS Transaction Gateway, Technical Planner
IBM Hursley

Andy Wright, CICS Transaction Server, Senior Software Engineer,

IBM Hursley

Steve Zemblowski, Software Specialist, Advanced Technical Support
IBM Dallas

Now you can become a published author, too

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time. Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run two - six weeks, and you can participate either in person or as a remote resident working from your home base.

Learn more about the residency program, browse the residency index, and apply online:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us.

We want our IBM Redbooks publication to be as helpful as possible. Send us your comments about this or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review IBM Redbooks publication form:

  `ibm.com/redbooks`

- Send your comments in an email:

  `redbook@us.ibm.com`

- Mail your comments:

  IBM Corporation, International Technical Support Organization
  Dept. HYJ; HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`

- Follow us on Twitter:

  `http://twitter.com/ibmredbooks`

- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7227-01
for Using IBM CICS Transaction Server Channels and Containers
as created or updated on March 17, 2015.

March 2015, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described in the following section.

New information

- CICS Transaction Server V3R2
  - Channels and containers enhanced to include 64-bit use
    - A 64-bit CICS Storage Manager was created, the first user of which was the CICS channels and containers support
- CICS Transaction Server V4R1
  - Introduction of Internet Protocol interconnectivity (IPIC)-based connectivity, along with channels and containers, for improved interoperability between CICS Transaction Server and IBM TXSeries®
- CICS Transaction Server V5R1
  - CICS-WebSphere MQ Bridge support for channels and containers
  - GET64 CONTAINER application programming interface (API) introduced
    - This includes the **`BYTEOFFSET`** parameter (also for GET CONTAINER)
  - PUT64 CONTAINER API introduced
    - This includes the **`APPEND`** parameter (also for PUT CONTAINER)
- CICS Transaction Server V5R2
  - DFHTRANSACTION channel

1

Introduction to channels and containers

This introduction provides an overview of the techniques traditionally used to pass data between applications running in IBM Customer Information Control System (CICS). We describe the constraints that they might have been subject to, and how a channels and containers solution was developed to provide solid advantages, alongside these techniques, which enabled CICS to fully comply with modern technology requirements in terms of sizing and flexibility.

1.1 Communication area for data passing

In 1975, command-level programming was introduced in CICS using an in-memory process called the communication area (COMMAREA) as a method of passing data from one application to another. The COMMAREA was limited in size to 32 kilobytes (). However, this was not a concern, because application data was IBM 3270-based and so not resource intensive. The 32 KB limit was enough to manage data propagation between applications.

Characteristics of a COMMAREA

A COMMAREA is a facility used to transfer information between two programs within a transaction, or to transfer information between two transactions from the same terminal. Information in a COMMAREA is available only to the two participating programs, unless those programs take explicit steps to make the data available to other programs, which might be started later in the transaction.

When one program links to another, the COMMAREA can be any data area to which the linking program has access. It is often in the working storage or *Linkage Section* of this program. In this area, the linking program can both pass data to the program it is starting and receive results from the specified program. When one program uses an **XCTL** command to transfer control to another program, CICS might copy the specified COMMAREA into a new area of storage.

This is because the starting program and its control blocks might no longer be available after it transfers control. In either case, the address of the area is passed to the program that is receiving control, and the CICS command-level interface (CLI) sets up addressability.

The processing required for using COMMAREA in a **LINK** command is minimal. However, it is significantly more for a distributed program link (DPL), because it can return the maximum storage. It is slightly more with the **XCTL** and **RETURN** commands, when CICS creates the COMMAREA from a larger area of storage used by the program.

Figure 1-1 shows a diagram representing a link using a COMMAREA.

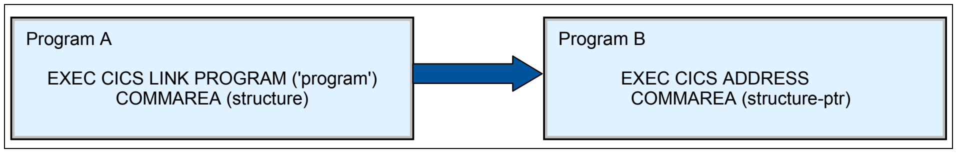

Program A
EXEC CICS LINK PROGRAM ('program') COMMAREA (structure)
Program B
EXEC CICS ADDRESS COMMAREA (structure-ptr)

Figure 1-1 Sample of a link using COMMAREA

The length option of a CICS command is generally expressed as a signed halfword binary value. This puts a theoretical upper limit of 32,763 bytes on length, meaning the maximum length for an area that the **`link`**, **`xctl`**, and **`return`** commands pass is 32 KB. For DPL, the suggested maximum length is 24 KB to enable additional CICS control blocks to be transferred.

The following list provides a summary of some of the characteristics of a COMMAREA:

- Required processing is low.
- COMMAREA is not recoverable.
- CICS holds a COMMAREA in CICS main storage until the terminal user responds with the next transaction.
- A COMMAREA is available only to the first program in the next transaction, unless this program explicitly passes the data to another program or a succeeding transaction.
- The use of the COMMAREA option on the **`RETURN`** command is the principal example of a safe programming technique that you can use to pass data between successive transactions in a CICS pseudo-conversational transaction.

1.2 The requirement for change

Considering that the COMMAREA was the most popular technique used to pass data between applications that run in CICS, the question was: Why did we need to change such a valid method that historically demonstrates its strength, and based on which the major part of CICS applications have been built?

The answer was, we were not changing at all, but merely enhancing the technique of passing data. This was for the benefit of the new evolution of application design, which is based on web technologies, and which required large amounts of data to pass between applications. This enhanced design also provided the possibility to dynamically add or remove parts of the passed data.

1.2.1 Memory constraints

CICS applications have traditionally passed parameter data using the **`EXEC CICS LINK`** application programming interface (API) specifying a COMMAREA. Previously, CICS applications were highly optimized to be memory-efficient, and were easily able to pass the required parameter data within the size constraints imposed by the CICS API on COMMAREA memory size.

However, modern CICS applications are required to process large quantities of structured parameter data, in both Extensible Markup Language (XML) and non-XML formats, such as JavaScript Object Notation (JSON).

This is because the integration of CICS applications with the elements of enterprise solutions, outside the bounds of the CICS environment, is becoming the norm. The consequence of effectively extending CICS applications to new enterprise solutions is that the constraints imposed on the COMMAREA size by the system might be too inflexible.

1.2.2 Flexibility of the channels and containers approach

The COMMAREA is a large and contiguous block of data that contains all of the data to be passed to the called program, even if only part of this data is required. This might not be acceptable for the modern application design, where the flexibility of data structure is a basic component that offers the possibility of accommodating future business requirements.

If an XML document has to be exchanged, the parameter data contained within it is different from the data format previously known to CICS programmers.

By design, XML is extensible, so you can add further data elements when application requirements change. XML structures can accommodate a varied mix of data types, and it is common for an XML document to contain large binary objects, such as images, in addition to character and numeric payload data.

When you add the lengthy tag descriptions, XML-based parameter data areas can require large areas of memory when passed by value between program elements. Therefore, the 32 KB limit, together with the static structure imposed on the traditional CICS COMMAREA, is insufficient for such applications.

Although the COMMAREA remains the basic way to pass data between CICS application programs with the recognized qualities mentioned earlier, a new way to exchange data is required alongside the COMMAREA solution. This provides an alternative way to satisfy the requirements introduced by the usage of new application programming styles.

A new approach that provides an easy and more flexible mechanism for exchange of large volumes of structured parameter data between CICS programs was introduced in CICS Transaction Server V3.1.

This new approach is known as the *channels and containers* model.

1.3 COMMAREA constraints and alternative solutions

The following section provides a more detailed description of the COMMAREA, to help you understand its constraints and the techniques that you might consider using to bypass them.

1.3.1 32 KB size limit

The size limit of the COMMAREA can be explained as follows:

- The **`EXEC CICS`** API constrains the size of a COMMAREA to a maximum of 32 KB. It is in the processing of large XML documents that the constraints of the traditional COMMAREA application become more prominent. To deal with this type of content, circumvention methods are required.
- The 32 KB COMMAREA size constraint is applicable to both **`LINK`** and **`XCTL`** commands in a single region. These constraints are also applicable to those COMMAREAs that programs participating in a DPL use, between two CICS regions.
- This 32 KB constraint can also affect the exchange of data between multiple CICS tasks in the following cases:
  - When data is passed between two tasks using the **`EXEC CICS START TRANSID FROM`** command
  - In CICS transactions involved in a pseudo-conversational sequence where you can exchange data by using the **`EXEC CICS RETURN TRANSID COMMAREA`** command
- The 32 KB COMMAREA restriction also applies to the external CICS interface (EXCI) and the external call interface (ECI) used by the IBM CICS Transaction Gateway and the IBM CICS Universal Clients.

1.3.2 Methods of passing data larger than 32 KB before channels and containers

There are several options you might consider implementing to circumvent the 32 KB COMMAREA restriction, both within a single CICS region and between multiple CICS regions. These can include:

- Using the `FLENGTH` option of the **`GETMAIN`** command to acquire a storage area larger than 32 KB, and passing the address of the large storage area in the COMMAREA:
  - This solution, although simple, only works in a single CICS address space. A region affinity between the two programs or transactions is created.
  - Must ensure that programs use the same data key and data lock.
- Using Virtual Storage Access Method (VSAM) files or IBM DB2® data.

  This technique involves the allocation of program-specific managed resources, such as VSAM files or DB2 data, to contain large parameter payloads, and passing the resource name within the COMMAREA structure.
- Passing the name of a temporary storage (TS) queue in the COMMAREA.

  By placing the data in TS, more than 32 KB of data can be passed between programs or tasks. If the TS queue is placed in a TS, owning region, or a shared TS server, the data can be accessed across multiple CICS regions.
- Passing the name of an IBM WebSphere MQSeries® queue in the COMMAREA.

  By placing the data in IBM MQ queues and only passing the queue name, a larger amount of data can be passed between the communicating programs or tasks.

The following considerations apply to many cases in the previously mentioned techniques:

- Applications can create new managed resources of the types listed, using the appropriate **`EXEC CICS`** API.
- Depending on the configured resource characteristics, these can be accessible across multiple CICS regions.
- These solutions require the application to manage unique resource naming conventions and implement lifecycle management for the resource used.
- Applications might also be required to implement the resource setup and initialization, perhaps introducing extra processing demands.
- More systems configuration and management might be required, depending on the nature of the shared managed resource.

1.3.3 The evolution of the 32 KB COMMAREA

It is worth mentioning, with the various techniques available to bypass the 32 KB restriction, the COMMAREA remains the most used method to pass data between applications running under CICS. Therefore, the simplest solution to this problem would have been to make the COMMAREA capable of handling lengths greater than 32 KB. Although such an implementation would ease the 32 KB restriction, it would not resolve all of the *problems* that exist in exchanging data today.

Also, it would exacerbate some of the problem areas. The following list describes some of these issues:

- Overloading of the copybooks

  The current copybooks used in the exchange of data today tend to be overloaded. Also, these structures are redefined several times, depending on whether the copybook is passing input, output, or error information. This leads to confusion on exactly when the fields are valid.

- Inefficient transmission of overloaded COMMAREAs:

  - The current overloaded COMMAREA structure does not lend itself to being efficiently transmitted between CICS regions. The COMMAREA structure size must account for the maximum size of the data that can be returned.

    By addressing the COMMAREA structure directly, CICS cannot determine if you have changed the data contents. CICS must always return the full COMMAREA structure from a DPL, even if nothing has been changed.

  - The current COMMAREA structure does not support easy separation of binary and character data. By contrast, the channel construct offers a simple way to get character data returned in the code page that your application program requires.

  Note: Further information regarding this argument is presented in 2.3, “Overloaded COMMAREAs” on page 38.

- Loss of compatibility

  Merely changing the COMMAREA length to a fullword length could result in the loss of object and source code compatibility for existing CICS programs. In this case, a program would have difficulty determining if EIBCALEN was valid or whether to check a new EXEC interface block (EIB) field.

- Business transaction services (BTS) use

  This approach addresses many of the concerns associated with the previous circumvention techniques, and is available on all currently supported CICS versions. However, it requires the adoption of a new programming approach for CICS applications, with consequential application re-engineering. This can be an ambitious undertaking for a mature, critical CICS business application suite.

- XML requirement

  In the contemporary enterprise solution context, a CICS application can expect to exchange parameter data in the form of an XML document, either from a system component external to CICS or as part of the request processing path with other CICS applications. Depending on the nature of the request, XML documents can be large, while containing large volumes of character, numeric, and binary payload data.

The adoption of channels and containers in existing applications can preserve the existing application architecture while overcoming the most common constraints revealed by the new data format. This enables CICS components to process XML or other structure data in places where the 32 KB COMMAREA limit cannot satisfy the application requirements.

1.4 Channels and containers concepts

CICS Transaction Server Version 3.1 introduced a new approach that provided an easy and more flexible mechanism for exchange of large volumes of structured parameter data between CICS programs. This new approach was provided by two new capabilities known as *channels* and *containers*.

A container is a named reference to a storage area managed by CICS that can hold any form of application data. A container can be any size, and can hold data in any format that the application requires. An application can reference any number of containers. CICS provides `EXEC` API verbs to create, delete, reference, access, and manipulate a container, and to associate it with a channel.

A channel is a uniquely named reference to a collection of containers. A channel is analogous to a COMMAREA, but it does not have the constraints of a COMMAREA. CICS provides an `EXEC` API, which associates a named channel with a collection of one or more containers.

This is an easy way of grouping parameter data structures that might pass to a called application. CICS deletes a channel and its containers when it can no longer be referenced (when a channel becomes *out of scope*).

1.4.1 General concepts

The following list describes general concepts about channels and containers:

- Containers are named blocks of data designed for passing information between programs. You can think of them as *named COMMAREAs*.
- Container size is limited only by the amount of storage available.
- Container data is stored “above the bar”. However, the data must be copied below the bar for applications to access it.
- Containers are grouped together in sets called *channels*. Programs can pass a single channel between them. You can think of a channel as a *parameter list*. The same channel can be passed from one program to another.
- To create named containers and assign them to a channel, a program uses the following command:

  ```
  EXEC CICS PUT CONTAINER(container-name) CHANNEL(channel-name)
  ```

  The program can then pass the channel and its containers to a second program using the `CHANNEL(channel-name)` option of the **`EXEC CICS LINK`**, **`XCTL`**, **`START`**, or **`RETURN`** commands.

  Example 1-1 shows how to pass a channel on a link.

  Example 1-1 Passing a channel on a link

  ```
  EXEC CICS PUT CONTAINER(structure-name)
            CHANNEL(channel-name)
            FROM(structure)
  EXEC CICS LINK PROGRAM(PROG2)
            CHANNEL(channel-name)
  ```

- The second program can read containers passed to it using the following command:

  ```
  EXEC CICS GET CONTAINER(container-name)
  ```

  This command reads the named container belonging to the channel that the program was started with. See Example 1-2.

  Example 1-2 Receiving a container

  ```
  EXEC CICS GET CONTAINER(structure-name)
       CHANNEL(channel-name)
       INTO(structure)
  ```

- If the second program is started by a **`LINK`** command, it can also return containers to the calling program. It can do this by creating new containers, or by reusing existing containers.

Example 1-3 shows how to return a container.

Example 1-3 Returning a container

```
EXEC CICS PUT CONTAINER(structure-name)
     FROM(structure)
EXEC CICS RETURN
```

Note: Addressing mode 64 programs, called *AMODE(64)*, can use channels and containers to transfer data in 64-bit storage in the same way, by using the following commands:

```
EXEC CICS PUT64 CONTAINER(container-name) CHANNEL(channel-name)
EXEC CICS GET64 CONTAINER(container-name) CHANNEL(channel-name)
```

These commands are for use only in non-Language Environment AMODE(64) assembly language application programs. CICS BTS containers are *not* supported.

- Channels and containers are visible only to the program that creates them, and to the programs that they are passed to. When these programs end, CICS automatically deletes the containers and their storage. The exception to this is containers that are defined within a channel created with the name DFHTRANSACTION, which we describe later.
- Channels and containers are not recoverable. If you require to use recoverable containers, use CICS BTS containers. The relationship between channel and BTS containers is described in 1.4.5, “Channels and business transaction services” on page 19.
- Channels and COMMAREAs **`EXEC CICS LINK`**, **`EXEC CICS XCTL`**, and **`EXEC CICS RETURN`** can only pass a channel *or* a COMMAREA. However, Program A can pass data in a COMMAREA to Program B, which then creates a channel to pass the data on to Program C. Because Program B receives the data returned from Program C in a container, Program B moves the container data into a COMMAREA, and this is where Program A expects to find it.

1.4.2 Channels

A channel is a uniquely named reference to a collection of application parameter data held in containers. It’s analogous to a COMMAREA, but is not subject to the constraints of a COMMAREA.

You can choose a channel name that is a meaningful representation of the data structures that the channel is associated with. For example, in a human resource application, a channel name might be `<employee-info>`.

This collection of application parameter data serves as a standard mechanism to exchange data between CICS programs. CICS Transaction Server provides an **EXEC** API that associates a named channel with a collection of one or more containers, offering an easy way to group parameter data structures that can pass to a called application. CICS Transaction Server removes a channel when it can no longer be referenced (when it becomes out of scope).

The current channel

A program's current channel (if any) is the channel with which it was started. The current channel is set by the calling program or transaction, by transferring the control to the called program through a **LINK**, **XCTL**, **START**, and pseudo-conversational return with the **CHANNEL** parameter.

Although the program can create other channels, the current channel for a particular invocation of a particular program never changes. It is analogous to a parameter list. If a channel is not explicitly specified, the current channel is used as the default value for the **CHANNEL ('*<channel-name>*')** parameter on the **EXEC CICS** command. Figure 1-2 shows the previously mentioned process.

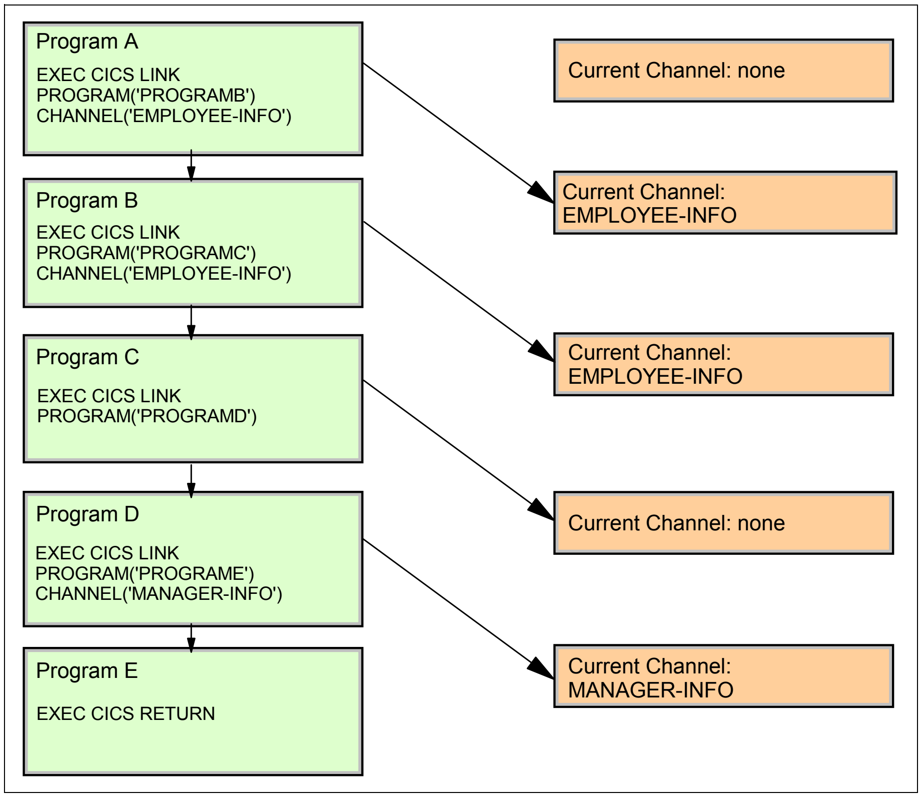


Figure 1-2 The current channel

Typically, programs that exchange a channel are written to handle that channel. Therefore, both client and server programs know the name of the channel, and the names and number of the containers in that channel.

However, for example, if a server program or component is written to handle more than one channel, on invocation, it must discover which of the possible channels it has been passed.

A program can discover its current channel, meaning the channel with which it was started, if it issues the following command:

```
EXEC CICS ASSIGN CHANNEL
```

If there is no current channel, the command returns blanks.

The program can also retrieve the names of the containers in its current channel by browsing, but typically this is not necessary. A program written to handle several channels is often coded to be aware of the names and number of the containers in each possible channel.

Important: A browse does not guarantee the order in which containers are returned.

To get the names of the containers in the current channel, use the browse commands, as Example 1-4 shows.

Example 1-4 Browsing containers in a channel

```
EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(data-area)
EXEC CICS GETNEXT CONTAINER(data-area) BROWSETOKEN(token)
EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(token)
```

Having retrieved the name of its current channel and, if necessary, the names of the containers in the channel, a server program can adjust its processing to suit the kind of data that it has been passed.

The browsing of a channel is described in detail in 2.4, "STARTBROWSE application programming interface" on page 40.

Tip: For a program creating a channel, the **`assign channel`** command returns blanks unless it was started using **`start`**, **`link`**, or **`xctl`**, specifying the channel name.

The scope of a channel

The scope of a channel is the code (the program or programs) from which you can access it.

See Figure 1-3, which illustrates the scope of channel.

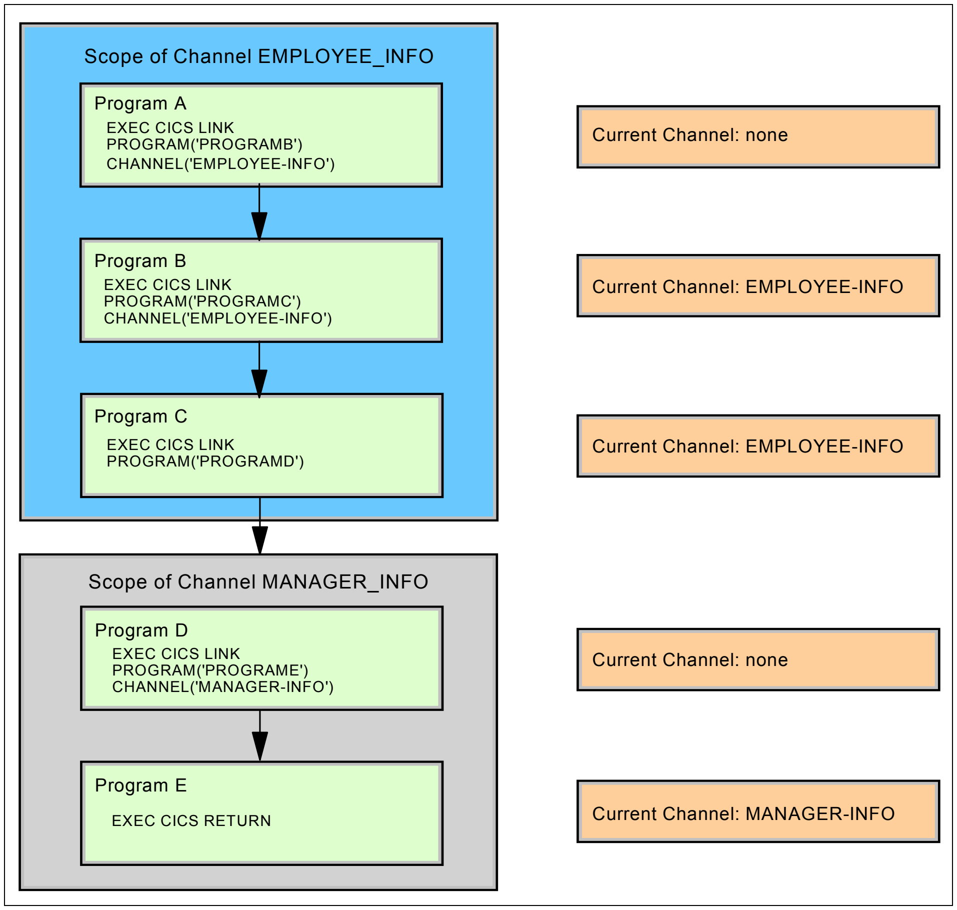


Figure 1-3 Example showing the scope of a channel

Figure 1-3 shows the scope of the EMPLOYEE-INFO channel, which consists of Program A (the program that created it), Program B (for which it is the current channel), and Program C (for which it is also the current channel). Additionally, Figure 1-3 shows the scope of the MANAGER-INFO channel, which consists of Program D (which created it) and Program E (for which it is the current channel).

None of these channels is the DFHTRANSACTION transaction channel, the scope of which would be the whole transaction.

Lifetime of a channel

A channel is created when it is named on an **EXEC CICS** command. The usual command to create a channel is the following command:

```
EXEC CICS PUT CONTAINER
```

In the previously given command, specifying the **CHANNEL** parameter creates the channel, and also associates the container with it.

A channel is deleted when it goes out of scope to the programs in the linkage stack, which means that no programs are able to access it. This causes CICS to delete the channel.

However, an exception to this is the DFHTRANSACTION transaction channel, which does not go out of scope until the end of the transaction, to any program in the linkage stack. This is true if each of those programs is not running in a CICS region at a version earlier than CICS Transaction Server Version 5.2.

Figure 1-4 shows the APIs used to create and manage a channel. Each of the following commands creates a channel if there is not one already created with that name.

```
EXEC CICS PUT CONTAINER CHANNEL
Creates a channel and places data into a container within the
channel
EXEC CICS PUT64 CONTAINER CHANNEL
Places data from 64-bit storage into a container that is associated
with a specified channel
EXEC CICS GET CONTAINER CHANNEL
Retrieves the container data passed to the called program
EXEC CICS GET64 CONTAINER CHANNEL
Retrieves data from a named channel container into 64-bit storage.
EXEC CICS MOVE CONTAINER CHANNEL AS TOCHANNEL
Moves a container from one channel to another channel
EXEC CICS DELETE CONTAINER CHANNEL
Deletes a container
EXEC CICS ASSIGN CHANNEL
Returns the name of the program's current channel, if one exists
EXEC CICS LINK PROGRAM CHANNEL
Links to the program, on a local or remote system, passing the
channel and container data
EXEC CICS XCTL PROGRAM CHANNEL
Transfers control to the program passing the channel and container
data
EXEC CICS START TRANSID CHANNEL
Starts a task, on a local or remote system, copying the named
channel and containers and passing it to the started task
EXEC CICS RETURN TRANSID CHANNEL
Returns control to CICS, passing the channel and containers to the
next transaction
```

Figure 1-4 API to create and manage a channel

API used to manage channels and containers are further described in 3.1, “EXEC CICS application programming interface” on page 66.

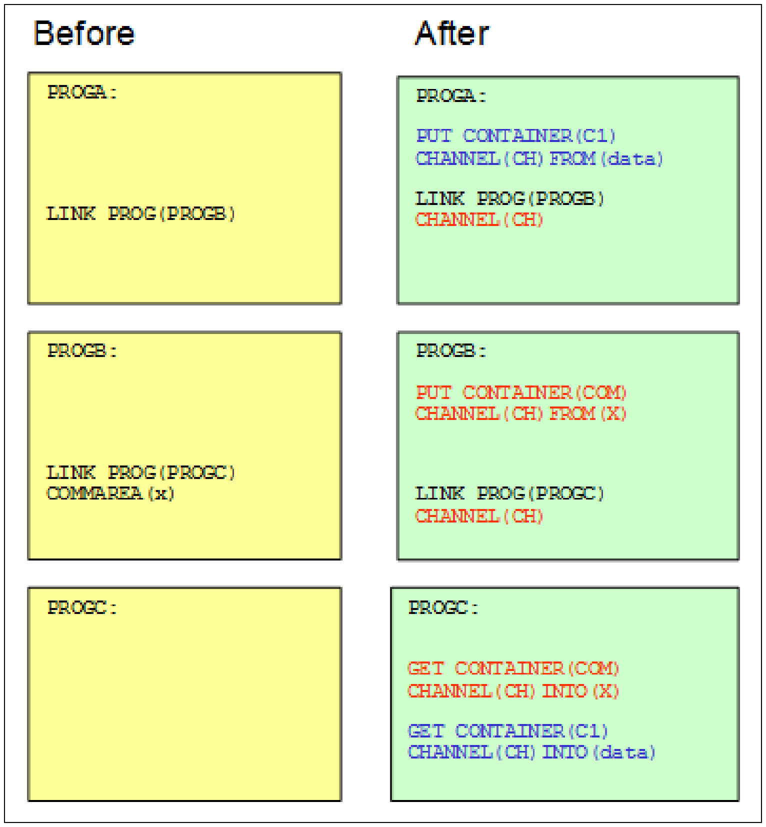
1.4.3 The DFHTRANSACTION transaction channel

Channels normally go out of scope when the link level changes. They might, therefore, not be available to all of the programs in a transaction. If you create a channel with the name DFHTRANSACTION, it does not go out of scope when the link level changes.

DFHTRANSACTION is therefore available to all programs in a transaction, including any exit points that are API-enabled. However, the transaction channel cannot be passed to any CICS region at a version earlier than CICS Transaction Server Version 5.2. DFHTRANSACTION can be used in all API commands that accept a channel name.

In Figure 1-5, we show how a simple application using dynamic program links to pass a COMMAREA would need to be amended to use channels and containers before the implementation of the DFHTRANSACTION channel in CICS Transaction Server Version 5.2.

Programs at all link levels in the DPL stack would need to be amended to pass channel CH.



```
PROGA:

LINK PROG(PROGB)
```

```
PROGA:
PUT CONTAINER(C1)
CHANNEL(CH) FROM(data)
LINK PROG(PROGB)
CHANNEL(CH)
```

```
PROGB:

LINK PROG(PROGC)
COMMAREA(x)
```

```
PROGB:
PUT CONTAINER(COM)
CHANNEL(CH) FROM(X)

LINK PROG(PROGC)
CHANNEL(CH)
```

```
PROGC:
```

```
PROGC:

GET CONTAINER(COM)
CHANNEL(CH) INTO(X)

GET CONTAINER(C1)
CHANNEL(CH) INTO(data)
```

Figure 1-5 Implementing channels and containers without DFHTRANSACTION

In Figure 1-6, we show how, using the DFHTRANSACTION channel, the only change needed to the application is to amend the front-end program to create the DFHTRANSACTION channel with a **`put container`** command. Also, to amend the program that uses the data from the container to read the container with a **`get container`** command.

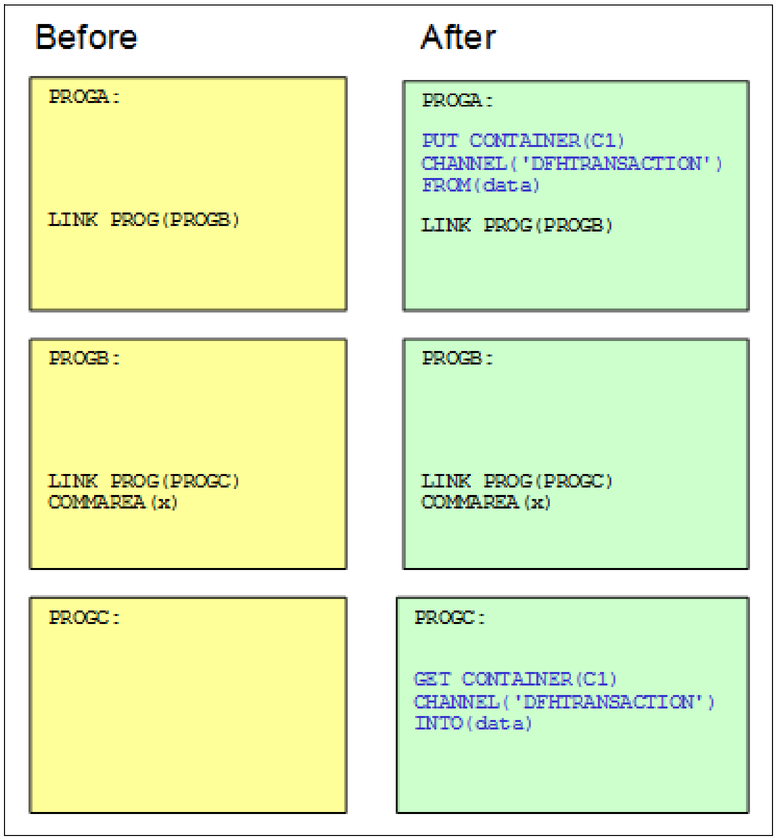


Figure 1-6 Implementing channels and containers with DFHTRANSACTION

Remember: When using the DFHTRANSACTION channel, the front-end program in the DPL stack should be the program that creates the channel, even if it creates an empty channel.

1.4.4 Containers

A container is a uniquely named block of data that can be passed to a subsequent program or transaction. It refers to a particular parameter data structure that exists within a collection of virtually any form of application parameter data.

You can choose a container name that has a meaningful representation of the data structure. For example, in a human resource application, the container name might be `<employee-name>`.

CICS Transaction Server provides **`EXEC`** API verbs to create, delete, reference, access, and manipulate a container, and also to associate it with a channel. See Figure 1-7 for more details.

```
EXEC CICS PUT CONTAINER CHANNEL
Creates a channel and places data into a container within the
channel
EXEC CICS PUT64 CONTAINER CHANNEL
Creates a channel and places data from 64-bit storage into a
container within the channel
EXEC CICS GET CONTAINER CHANNEL
Retrieves the container data passed to the called program into
64-bit storage
EXEC CICS GET64 CONTAINER CHANNEL
Retrieves the container data passed to the called program
EXEC CICS MOVE CONTAINER CHANNEL AS TOCHANNEL
Moves a container from one channel to another channel
EXEC CICS DELETE CONTAINER CHANNEL
Deletes a container from a channel
EXEC CICS STARTBROWSE CONTAINER
Start a browse of the containers associated with a channel
EXEC CICS GETNEXT CONTAINER
Return the name of the next container associated to the channel
EXEC CICS ENDBROWSE CONTAINER
Ends the browse of the containers associated with the channel
```

Figure 1-7 Container-related API

A container can be any length, and a container size is constrained only by the available user storage in the CICS address space. It can include data in any format that an application requires. An application can create any number of containers and can use separate containers for different data types, such as binary and character data. This capability helps ensure that each container structure is based on a unique area of memory.

It also minimizes the potential for errors that commonly arise when parameter data for multiple applications is overloaded in a single memory area. The potential errors are minimized by isolating different data structures, and making the association between data structure and purpose clear.

CICS read-only containers

CICS can create channels and containers for its own use, and pass them to user programs. In some cases (in particular when CICS uses containers for security information), CICS marks these containers as read-only. This is so that the user program cannot modify data that CICS requires on return from the user program.

User programs cannot create read-only containers.

You cannot overwrite, move, or delete a read-only container. Therefore, if you specify a read-only container on a **`put container`**, **`move container`**, or **`delete container`** command, you receive an `INVREQ` condition.

1.4.5 Channels and business transaction services

The **`put container`**, **`get container`**, **`move container`**, and **`delete container`** commands that are used to build and interact with a channel are similar to those used in CICS BTS applications. BTS implemented containers as a way of passing information between activities and processes. There is no limit to the size of a container in BTS.

The containers used in the channel context are similar to those used in BTS, and the commands used to access the container data are similar (for example, **`get`**, **`put`**, **`move`**, and **`delete`**).

It is possible to have the same server program started in both a channel and a BTS context. To accomplish this, the server program must avoid the use of options that specifically identify the context.

The server program must *call* CICS to determine the context of a command. When a container command is run, CICS first checks to see if there is a current channel. If there is a current channel, the context of the command is `channel`. If there is no current channel, CICS checks to see if this is part of a BTS activity. If this is part of a BTS activity, the context is `BTS`. If the program has no channel context and no BTS context, an `INVREQ` is raised.

Therefore, a program that issues container commands can be used, without change, as part of a channel application or as part of a BTS activity.

The BTS approach requires the adoption of a new programming approach for CICS applications, with consequential application re-engineering. This can be an ambitious undertaking for a mature, critical CICS business application suite. The channels and containers approach is more simple, and does not require as much effort to change applications.

Important: Channel containers are not recoverable. If you require to use recoverable containers, use CICS BTS containers.

1.4.6 Channels and CICS Java

CICS provides **`EXEC`** API support for channels and containers in all supported CICS programming languages. In the CICS Java environment, CICS Java (JCICS) classes are provided to enable channels and containers to be used as the mechanism to exchange data between CICS Java applications and traditional CICS procedural applications.

See Figure 1-8 for details about the JCICS classes that CICS Java programs can use to pass and receive channels.

```
com.ibm.cics.server.Channel
A Channel class used to create new containers in a channel
com.ibm.cics.server.Container
A Container class used to place data in a container
com.ibm.cics.server.ContainerIterator
A ContainerInterator class used to browse the current channel
```

Figure 1-8 JCICS classes managing channels

CICS also provides the exception classes for handling errors shown in Figure 1-9.

```
com.ibm.cics.server.CCSIDErrorException
Class that represents the CICS CCSIDERR condition
com.ibm.cics.server.ChannelErrorException
Class that represents the CICS CHANNELERR condition
com.ibm.cics.server.ContainerErrorException
Class that represents the CICS CONTAINERERR condition
```

Figure 1-9 JCICS classes for channels error handling

Details about this item are further provided in 3.2, “CICS Java” on page 76.

1.4.7 Data conversion

There are two types of container: BIT and CHAR. CHAR containers enable application programs to convert data between different encodings. The data conversion model that channel applications use is much simpler than the model that COMMAREA applications use.

This is because the system programmer controls the data conversion in COMMAREA applications. However, the application programmer controls the data conversion in channel applications, using simple API commands.

The following cases are examples of when data conversion is necessary:

- When character data is passed between platforms that use different encoding standards, for example, Extended Binary Coded Decimal Interchange Code (EBCDIC) and American Standard Code for Information Interchange (ASCII).
- When you want to change the encoding of some character data from one coded character set identifier (CCSID) to another.

Applications that use channels to exchange data use a simple data conversion model. Frequently, no conversion is required and, when conversion is required, you can use a single programming instruction to tell CICS to handle it automatically.

Further information about data conversion is presented in 2.6, "Data conversion and code page conversion" on page 44.

Conversion models

There are two conversion model applications.

Using COMMAREA

For applications that use the COMMAREAs to exchange data, the conversion takes place under the control of the system programmer. This uses the DFHCNV conversion table, the DFHCCNV conversion program, and the DFHUCNV user-replaceable conversion program, which is optional.

Using channels

Channel applications use the data conversion model, which is much simpler than the one that the COMMAREA applications use. The data in channel and containers is converted under the control of the application programmer, using API commands.

The following items specify the processes that the application programmer supports:

- The application programmer is responsible only for the conversion of user data, which is the data in containers that the application programs create. CICS converts the system data automatically, where it is necessary.
- The application programmer is concerned only with the conversion of character data. The conversion of binary data, between big-endian and little-endian, is *not* supported.
- Applications can use the container API as a simple means of converting character data from one code page to another. Example 1-5 converts data from codepage1 to codepage2.

> **Tip:** When using Example 1-5, remember to add a temporary channel, such as CHANNEL(temp). If you fail to add this temporary channel, you are relying solely on the program to be called along with a channel. Additionally, if the data is large, you must perform a **`delete container`** operation immediately.

Example 1-5 API to convert code page

```
EXEC CICS PUT CONTAINER(temp) DATATYPE(CHAR)
              FROMCCSID(codepage1) FROM(input-data)
EXEC CICS GET CONTAINER(temp) INTOCCSID(codepage2)
              SET(data-ptr) FLENGTH(data-len)
```

1.5 Benefits of using channels and containers

The lifecycle and scope of channels and containers are completely under the control of the CICS system, ensuring data integrity and storage resource management.

The following list describes the major benefits obtained by applications using the capabilities of the channels and containers methodology:

- An unconstrained, CICS-supported method of passing parameter data
- Segregation of parameter data structures, each part represented by a named container structure
- A loose functional coupling approach
- The freedom to dynamically determine the nature of the passed data, and to select the appropriate processing required

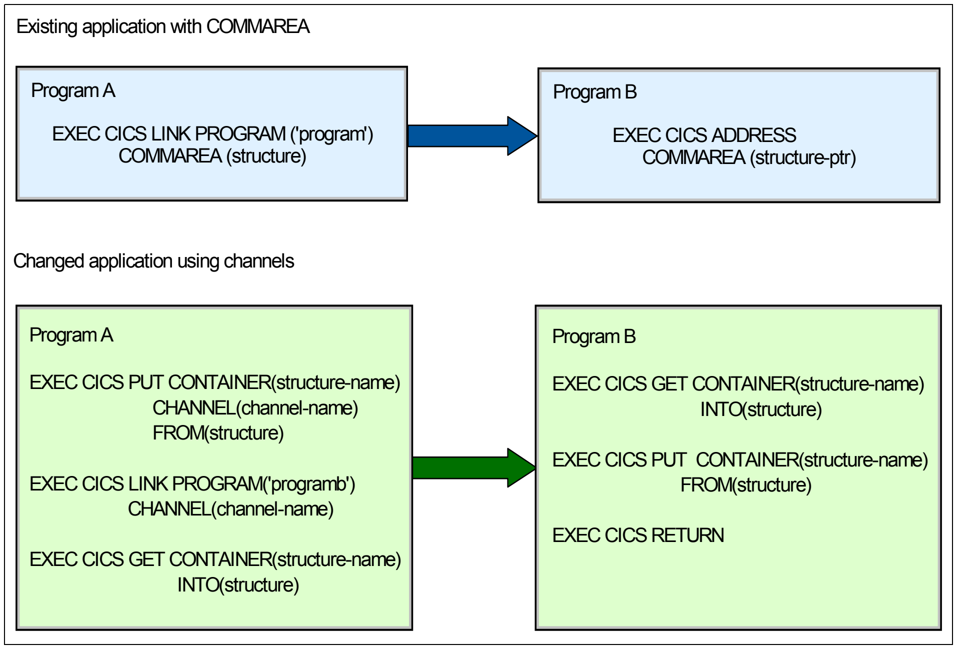
- A CICS-standard API for optimized exchange of data between CICS programs implemented in any CICS-supported language
- CICS-managed lifecycle for channel and container resources
- Ease of parameter passing by use of unique named references
- Ease of understanding by use of unique named references to parameter payload
- Explicit code page conversion operations

The internal CICS implementation of channels and containers is optimized for efficient memory management and data transfer. CICS ensures that only the necessary new and modified containers are transferred between the calling applications. This is to optimize the performance of the calling mechanism.

Assuming that the containers separate the different parameter structures, the calling applications benefit from complete access to the data content in all containers that are in scope.

1.6 Porting COMMAREA to channels and containers

To port programs that are exchanging data through a COMMAREA on a **`link`** command, the format of the command must be changed, and proper commands must be added to use channels and containers. Figure 1-10 shows you an example of this.



Existing application with COMMAREA
Program A
Program B
Changed application using channels
Program A
Program B

Existing application with COMMAREA

Program A:

```
EXEC CICS LINK PROGRAM ('program')
          COMMAREA (structure)
```

Program B:

```
EXEC CICS ADDRESS
     COMMAREA (structure-ptr)
```

Changed application using channels

Program A:

```
EXEC CICS PUT CONTAINER(structure-name)
              CHANNEL(channel-name)
              FROM(structure)

EXEC CICS LINK PROGRAM('programb')
          CHANNEL(channel-name)

EXEC CICS GET CONTAINER(structure-name)
              INTO(structure)
```

Program B:

```
EXEC CICS GET CONTAINER(structure-name)
              INTO(structure)

EXEC CICS PUT  CONTAINER(structure-name)
              FROM(structure)

EXEC CICS RETURN
```

Figure 1-10 Changes from COMMAREA to channels using link

The same applies to programs using the **`start`** command with the COMMAREA. Figure 1-11 on page 25 shows an example of the changes necessary to convert an application program using an **`EXEC CICS START`** command with data to start passing a channel. The example shows the commands that must be added or changed.

A program can issue multiple starts with data for a single transaction ID. CICS starts one instance of the transaction. The program can issue multiple retrieves to get the data. When using the channel option on the start, CICS starts one transaction for each *start* request. The started transaction is able to access the contents of a single channel, and also get a copy of the channel.

Figure 1-11 shows an example of converting an application program.

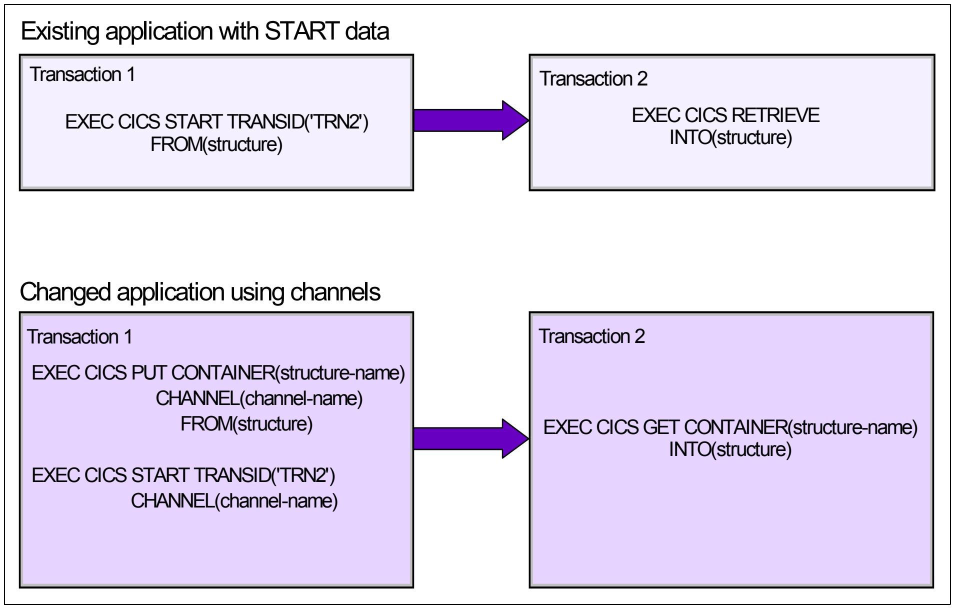

Existing application with START data
Transaction 1
EXEC CICS START TRANSID('TRN2')
FROM(structure)
Transaction 2
EXEC CICS RETRIEVE
INTO(structure)
Changed application using channels
Transaction 1
EXEC CICS PUT CONTAINER(structure-name)
CHANNEL(channel-name)
FROM(structure)
EXEC CICS START TRANSID('TRN2')
CHANNEL(channel-name)
Transaction 2
EXEC CICS GET CONTAINER(structure-name)
INTO(structure)

Figure 1-11 Changes from COMMAREA to channels using start

Consider the following items when porting from a COMMAREA to channels and containers:

- CICS application programs, which use traditional COMMAREAS to exchange data, continue to work as before.
- `EXEC CICS LINK` and `EXEC CICS START` commands, which can pass either COMMAREAs or channels, can be dynamically routed.
- If you employ a user-written dynamic or distributed routing program for workload management, rather than IBM CICSPlex System Manager, you must modify your program to handle the new values that it can pass in the DYRLEVEL, DYRTYPE, and DYRVER fields of the DFHDYPDS communications area.
- It is possible to replace a COMMAREA using a channel with a single container. Although this might seem the simplest way to move from COMMAREAs to channels and containers, it is *not* a good practice to do this.
- In addition, be aware that a channel might use more storage than a COMMAREA designed to pass the same data. Because you are taking the time to change your application programs to use this new function, you must implement the *best practices* for channels and containers.

- Channels have several advantages over COMMAREAs, and it is advantageous to design your channels to make the most of these improvements.
- In previous releases, because the size of COMMAREAs is limited to 32 KB and channels were not available, some applications used TS queues to pass more than 32 KB of data from one program to another. Typically, this involved multiple writes to, and reads from, a TS queue. If you port one of these applications to use channels, be aware of the following points:
  - If the TS queue used by your existing application is in main storage, the storage requirements of the new, ported application are likely to be similar to those of the existing application.
  - If the TS queue used by your existing application is in auxiliary storage, the storage requirements of the ported application are likely to be greater than those of the existing application. This is because container data is held in storage rather than being written to disk.

Details about COMMAREA migration to channels and containers are provided in 2.9.3, “Porting from COMMAREAs to channels” on page 60.

2

Application design and implementation

In this chapter we provide information about implementing channels and containers from an application programmer’s point of view.

Examples are provided to show how to evolve channels and containers from communication areas (COMMAREAs).

When using channels and containers, the code page conversion, formerly an issue for the systems programmer, is now an issue for an application programmer.

2.1 Container usage as a replacement to COMMAREAs

This chapter presents examples that show how to replace a COMMAREA with a channel and container solution.

2.1.1 Basic COMMAREA example

This section shows an example of how to replace a COMMAREA by a channel and container solution. The first part of Example 2-1 shows the classic COMMAREA solution. A program calls a subroutine providing two variables grouped into a COMMAREA structure. The first variable is treated as the input field for the subroutine, and the second variable receives the output the called program creates.

The second part in Example 2-1 shows a name as input. The subroutine changes the name into a phonetic string and returns this string to the caller. For example, if you call the subroutine with different names, all of which sound the same but are spelled differently, the subroutine returns the same phonetic string for all names.

For instance, the subroutine returns MAYR when it is called for Meyer, Mayer, Meier, or Maier. Therefore, Example 2-1 is a program fragment that demonstrates how to call the subroutine `PHONETIC` with a COMMAREA.

Example 2-1 COMMAREA example

```
01 MYCOMMAREA.
  03 LNAME PIC X(40).
  03 PNAME PIC X(40).
01 PHONETIC PIC X(8) VALUE IS 'PHONETIC'.
  .
  .
  MOVE 'MEYER' TO LNAME.
  EXEC CICS LINK PROGRAM (PHONETIC)
      COMMAREA(MYCOMMAREA)
  END-EXEC.
  DISPLAY PNAME.
*** PNAME CONTAINS ‘MAYR’.
```

2.1.2 Basic channel example

Example 2-2 shows how to transform the small piece of code into a channel and container solution. In Example 2-2, you can see that the input to the subroutine can now be treated as two separate objects. This means that the two variables can be defined independently. We advise that you perform this data separation, even though it is still possible to send both variables in one container.

Example 2-2 Channel and container example

```
01 LNAME PIC X(40).
01 PNAME PIC X(40).
01 INPUTCONTAINER  PIC X(16) VALUE IS 'MYINPUTCONTAINER'.
01 OUTPUTCONTAINER PIC X(16) VALUE IS 'MYOUTPTCONTAINER'.
01 CHANNELNAME     PIC X(16) VALUE IS 'MYCHANNELNAME'.
01 PHONETIC PIC X(8) VALUE IS 'PHONETIC'.
    .
    MOVE 'MEYER' TO LNAME.
    EXEC CICS PUT CONTAINER(INPUTCONTAINER)
        FROM(LNAME)
        CHANNEL(CHANNELNAME)
    END-EXEC.
    EXEC CICS LINK PROGRAM (PHONETIC)
        CHANNEL(CHANNELNAME)
    END-EXEC.
    EXEC CICS GET CONTAINER(OUTPUTCONTAINER)
        INTO(PNAME)
        CHANNEL(CHANNELNAME)
    END-EXEC.
    DISPLAY PNAME.
*** PNAME CONTAINS ‘MAYR’.
```

2.1.3 Channel name character set

A channel name consists of 1-16 characters. The following list contains the acceptable characters:

- A-Z
- a-z
- 0-9
- $
- @
- #
- /
- %
- &
- ?
- !
- :
- |
- "
- =
- ¬
- ,
- ;
- < >
- .
- -
- _

You cannot use leading and embedded blank characters. If the name supplied has fewer than 16 characters, it is padded with trailing blanks.

2.1.4 Creating a channel

You can create a channel if you name and place it with one of the following commands to transfer the control to another program or transaction:

- `EXEC CICS LINK   PROGRAM CHANNEL`
- `EXEC CICS START  TRANSID CHANNEL`
- `EXEC CICS XCTL   PROGRAM CHANNEL`
- `EXEC CICS RETURN  TRANSID CHANNEL`

Important: **`Start`** makes a copy of the channel. If there are subsequent changes in the program that issued the **`start`**, these changes are *not* reflected in the copy of the channel.

Normally, a **PUT** or **MOVE** is used to create a channel:

- `EXEC CICS PUT  CONTAINER CHANNEL`
- `EXEC CICS PUT64  CONTAINER CHANNEL`
- `EXEC CICS MOVE CONTAINER CHANNEL TOCHANNEL`

If the channel does not already exist in the current program scope, it is created.

Attention: If the channel name on the previous commands does not already exist within the current program scope, an *empty* channel is created. If this is done unintentionally, it leads to unpredictable results.

A simple means to create a channel, and populate it with containers of data, is to issue a succession of commands with the following information:

```
EXEC CICS PUT CONTAINER(<container-name>) CHANNEL(<channel-name>)
FROM(<data-area>)
```

The first **PUT** command creates the channel, if the channel does not already exist, and adds a container to it. The subsequent commands add further containers to the channel. If the containers already exist, their contents are overwritten by the new data unless you specify the `APPEND` option, which indicates that the data passed to the container is appended to the end of the existing data in the container. The following command shows an example of this:

```
EXEC CICS PUT CONTAINER(<container-name>) CHANNEL(<channel-name>)
FROM(<data-area>) APPEND
```

An alternative way to add containers to a channel is to move them from another channel. To do this, use the following command:

```
EXEC CICS MOVE CONTAINER(<container-name>) AS(<container-new-name>)
               CHANNEL(<channel-name1>) TOCHANNEL(<channel-name2>)
```

You can use **`move container`**, rather than **`get container`** and **`put container`**, as a more efficient way of transferring data between channels.

Note: The following items are a summary of points that you must remember:

- If the `CHANNEL` or `TOCHANNEL` option is not specified, the current channel is implied.
- The source channel must be in the program scope.
- If the target channel does not exist within the current program scope, it is created.
- If the source container does not exist, an error occurs.
- If the target container does not exist, it is created. If it exists, its contents are overwritten.

When subroutines are called from various programs that use both COMMAREA and channel techniques, the subroutines must be able to recognize which technique is being used in the current invocation. The suggested method to check whether a channel is being used is to use the IBM CICS API command **`EXEC CICS ASSIGN CHANNEL`**.

If the returned value is not equal to spaces, a channel has been passed and you can process it. Otherwise, use a COMMAREA.

It is feasible to do the check if you reverse the process. Checking the `EIBCALEN` field for greater than zero indicates that this is a COMMAREA version. However, we do *not* recommend this, because if it is a pseudo conversational program the COMMAREA length is always zero on the first invocation. Therefore, in this case, you cannot trust this as an indication for COMMAREA or channel technique.

In Example 2-3 we show the back-end program phonetic, which has been written to use both techniques.

Example 2-3 Decide channel or COMMAREA: PHONETIC

```
WORKING-STORAGE SECTION.
01 CURRENTCHANNELNAME PIC X(16).
01 MYBROWSETOKEN PIC 9(8) BINARY.
01 INPUTCONTAINER  PIC X(16) VALUE IS 'MYINPUTCONTAINER'.
01 OUTPUTCONTAINER PIC X(16) VALUE IS 'MYOUTPTCONTAINER'.

01 LASTNAME PIC X(40).

LINKAGE SECTION.
01 DFHCOMMAREA.
  03 LNAME PIC X(40).
  03 PNAME PIC X(40).
```

```
PROCEDURE DIVISION USING DFHCOMMAREA.
  EXEC CICS ASSIGN CHANNEL(CURRENTCHANNELNAME)
  END-EXEC.
  IF CURRENTCHANNELNAME IS EQUAL TO SPACES
          THEN
*         /* CONTINUE WITH COMMAREA */
               CONTINUE
          ELSE
* GET THE CONTAINER NAME
          EXEC CICS STARTBROWSE CONTAINER
               CHANNEL(CURRENTCHANNELNAME)
               BROWSETOKEN(MYBROWSETOKEN)
          END-EXEC
          EXEC CICS GETNEXT CONTAINER(INPUTCONTAINER)
               BROWSETOKEN(MYBROWSETOKEN)
          END-EXEC
          EXEC CICS ENDBROWSE CONTAINER
               BROWSETOKEN(MYBROWSETOKEN)
          END-EXEC
          EXEC CICS GET CONTAINER(INPUTCONTAINER)
               CHANNEL(CURRENTCHANNELNAME)
               INTO(LASTNAME)
          END-EXEC
  END-IF
```

The `STARTBROWSE` example, shown previously, only works when the subroutine is called with one container in the channel. We advise that you access containers directly, as described in the following section. A `STARTBROWSE` loop makes sense, where a subroutine is written to handle more than one channel. Therefore, it must be able to discover which of the possible channels it has been passed. To learn more about this, read 2.4, “STARTBROWSE application programming interface” on page 40.

Current channel

If the container name that the subroutine uses is already known, you can access the container directly. If you know that you are always using channels, the **`assign channel`** statement is *not* required. In this case, the channel used to start the subroutine is known as the *current channel*. Even though the subroutine might create other channels, the current channel does not change.

A main program started for the first time does not have a current channel. You must reinvoke this program with an **`EXEC CICS RETURN TRANSID`** command with a channel, for this main program to have a current channel. Subroutines have a current channel when they are started with a channel.

Example 2-4 shows the process that you can use to get the input container directly, without using a browse loop. In this case, the name must be standard and not changed. Always check the response code for unexpected events, for example, `CONTAINERERROR`, which means that the specified container could not be found. See Example 2-4.

Example 2-4 Get container directly

```
WORKING-STORAGE SECTION.
01 L PIC 9(8) BINARY.
01 LASTNAME PIC X(40).
01 RC PIC 9(8) BINARY.
01 INPUTCONTAINER PIC X(16) VALUE IS 'MYINPUTCONTAINER'.
01 CURRENTCHANNELNAME PIC X(16).

LINKAGE SECTION.
01 DFHCOMMAREA.
   03 LNAME PIC X(40).
   03 PNAME PIC X(40).

PROCEDURE DIVISION USING DFHCOMMAREA.
   EXEC CICS ASSIGN CHANNEL(CURRENTCHANNELNAME)
   END-EXEC.
   IF CURRENTCHANNELNAME IS EQUAL TO SPACES
      THEN
*         /* CONTINUE WITH COMMAREA */
            CONTINUE
      ELSE
* GET THE CONTAINER NAME
            EXEC CICS GET CONTAINER(INPUTCONTAINER)
                      CHANNEL(CURRENTCHANNELNAME)
                      INTO(LASTNAME)
                      FLENGTH(L)
                      RESP(RC)
            END-EXEC
            IF RC NOT = DFHRESP(NORMAL)
               THEN
*                    *** DO SOME ERROR PROCESSING HERE ***
                    CONTINUE
               END-IF
      END-IF.
```

2.2 Flexible way to pass multiple pieces of data

There is more to coding with the channel technique than there is with the COMMAREA technique. As Example 2-7 on page 37 shows, it could be questioned which is better.

We do *not* recommend replacing all COMMAREAs with channels, especially existing running applications. However, there are situations where the channel technique is much easier to implement than the COMMAREA. One such instance is explained in the following paragraph.

Assume that there are applications that call the previously defined phonetic subroutine, providing a name as input. However, the subroutine does not simply return a phonetic string, it also translates the name in a phonetic and reads from an input source, for example, DB2, Virtual Storage Access Method (VSAM), and so on. This is a realistic situation. It is conceivable that, in this situation, the data return by the subroutine could exceed the limit of a COMMAREA.

Furthermore, the question arises how to structure the data in the COMMAREA. You require to provide input data, output data with an unpredictable size, status data, and error data.

The following sections compare this application scenario in both variations.

2.2.1 COMMAREA solution

In addition to the mandatory input data, you also require some kind of return information data, such as return codes and reason codes. The return codes and reason codes indicate whether the request was successful, partially successful, or not successful.

Partially successful means that the name has been successfully translated into a phonetic, but the result set is too large and does not fit into the largest available COMMAREA. This is because the largest COMMAREA you can have is 32 kilobytes (KB). This must be reflected in the return codes and reason codes, because further processing is necessary to retrieve all of the data.

On a subsequent call to the subprogram, additional status data is required. For instance, the subroutine must receive the key of the latest record returned, so that this key can be used to start again.

An example of how this COMMAREA would look is provided in Example 2-5.

Example 2-5 COMMAREA example for a complex request

```
01 MYCOMMAREA.
       03 NAME PIC X(40).
       03 RETURNCODE PIC 9(8) BINARY.
       03 REASONCODE PIC 9(8) BINARY.
       03 STATUSDATA PIC 9(15).
       03 I PIC 9(4) BINARY.
       03 CUSTOMERRECORD PIC X(800) OCCURS 0 TO 40 TIMES
                                    DEPENDING ON I.
```

Because you cannot determine the size of the returned data, you must set up a maximum COMMAREA size. Although, the COMMAREA here is not exactly the maximum size of 32 KB, you have insufficient space remaining to add another client record of 800 bytes.

If the subroutine returns the COMMAREA and the return codes and reason codes indicate that the request was successful, but there are more client records available that could not be returned because of the size limit of the COMMAREA, additional calls are required. Therefore, a loop is necessary on each return, to check whether more data is still available or not. You can see this in Example 2-6. This process is processor-intensive, complex to program, and error-prone.

Example 2-6 COMMAREA programming example

```
EXEC CICS LINK PROGRAM(PHONETIC)
             COMMAREA(MYCOMMAREA)
END-EXEC.

* PROCESS ALL RETURNED CUSTOMER RECORDS IN A LOOP
     PERFORM VARYING J FROM 1 BY 1 UNTIL J IS GREATER THAN I
*         ...
          CONTINUE
     END-PERFORM.

* CHECK WHETHER ADDITIONAL CUSTOMER RECORDS ARE AVAILABLE
     IF RETURNCODE = 4 AND REASONCODE = 1
          THEN
              PERFORM UNTIL RETURNCODE = ZERO
*                 CALL PHONETIC AGAIN
*                 PROCESS RESULTS
                  CONTINUE
              END-PERFORM
     END-IF.
```

2.2.2 Channel solution

The program coding requirement to use channels and containers for the application scenario mentioned previously, is much easier and more logically structured than the COMMAREA solution shown in Example 2-6 on page 36. See Example 2-7 for the channel code.

The input data is provided in a container to the subroutine, and the subroutine returns the result in another container. Error data could optionally return, but this is not mandatory, because the existence of an output container would indicate success. If no container is returned, there is no client data available. If a container is returned, all of the client records are in this container.

Example 2-7 Channel programming example

```
WORKING-STORAGE SECTION.
01 LNAME PIC X(40).
01 INPUTCONTAINER  PIC X(16) VALUE IS 'MYINPUTCONTAINER'.
01 OUTPUTCONTAINER PIC X(16) VALUE IS 'MYOUTPTCONTAINER'.
01 CHANNELNAME     PIC X(16) VALUE IS 'MYCHANNEL'.
01 PHONETIC PIC X(8) VALUE IS 'PHONETIC'.
01 P POINTER.
01 L PIC 9(8) BINARY.

LINKAGE SECTION.
01 CUSTOMERRECORD PIC X(800).

PROCEDURE DIVISION.
    EXEC CICS PUT CONTAINER(INPUTCONTAINER)
         FROM(LNAME)
         CHANNEL(CHANNELNAME)
    END-EXEC.
    EXEC CICS LINK PROGRAM(PHONETIC)
         CHANNEL(CHANNELNAME)
    END-EXEC.
    EXEC CICS GET CONTAINER(OUTPUTCONTAINER)
         CHANNEL(CHANNELNAME)
         SET(P) FLENGTH(L)
    END-EXEC.

* CHECK HERE L FOR GREATER THAN ZERO (RECORDS RETURNED) AND THAN
* PROCESS THE RETURNED TABLE STARTING AT POINTER P
```

In Example 2-7, there is no logical requirement to check the availability of records left. No loop is necessary to process additional subroutine invocations. It is much easier and more straightforward.

2.3 Overloaded COMMAREAs

The current copybooks, used in the exchange of data, tend to overload. The structures are redefined several times, depending on whether the copybook is passing input, output, or error information. This leads to confusion about the validity of the various fields at any particular time.

Example 2-8 is an expanded replica of the previously mentioned fragment code in Example 2-1 on page 28. On comparison of the two examples, it can be seen that the complexity increases from one to the other. The `FLAG` variable indicates whether the following data is input data, output data, or error data.

Redefinitions are necessary to work with meaningful variable names. Otherwise, the name of the variable is misleading. Differentiation between `LNAME` and `PNAME` is only possible by using redefinitions. An alternative to redefinitions is the use of dummy sections (see the `ERROR-STRUCTURE` variable).

Before using dummy sections, a `SET ADDRESS` is required to establish addressability. Also, checking for an invalid flag is necessary. The subroutine could be wrong, returning an unexpected value in this variable. See Example 2-8.

Example 2-8 Overloaded COMMAREA: FIELD is for input and for output

```
WORKING-STORAGE SECTION.
  01 MYCOMMAREA.
     03 FLAG  PIC X(1).
     03 FIELD PIC X(40).
     03 LNAME REDEFINES FIELD PIC X(40).
     03 PNAME REDEFINES FIELD PIC X(40).
  01 PHONETIC PIC X(8) VALUE IS 'PHONETIC'.

  LINKAGE SECTION.
  01 ERROR-STRUCTURE.
     03 EFLAG PIC X(1).
     03 EMSG  PIC X(40).

  PROCEDURE DIVISION.
     MOVE 'MEYER' TO LNAME.
*          I = INPUT
     MOVE 'I'     TO FLAG
     EXEC CICS LINK PROGRAM (PHONETIC)
           COMMAREA(MYCOMMAREA)
     END-EXEC.
*               E = ERROR
     IF FLAG = 'E'
        THEN
```

```
*        DO SOME ERROR PROCESSING.
           SET ADDRESS OF ERROR-STRUCTURE
             TO ADDRESS OF MYCOMMAREA
*            ...
             CONTINUE
        ELSE
*                       O = OUTPUT
             IF FLAG = 'O'
                THEN
*                    PNAME SHOULD CONTAIN 'MAYR'.
                     DISPLAY PNAME
                ELSE
*                    FLAG CONTAINS UNEXPECTED VALUE.
*                    DO SOME OTHER ERROR PROCESSING.
                     CONTINUE
             END-IF
  END-IF.
```

When using channels and containers, you can avoid the confusion of redefinitions and dummy sections. The program code is more straightforward. In Example 2-9, we have expanded fragment code in Example 2-2 on page 29 by introducing an error container.

Example 2-9 Introduction of an error container

```
IDENTIFICATION DIVISION.
PROGRAM-ID. RESCCO5.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LNAME  PIC X(40).
01 PNAME  PIC X(40).
01 ERRMSG PIC X(40).
01 INPUTCONTAINER  PIC X(16) VALUE IS 'MYINPUTCONTAINER'.
01 OUTPUTCONTAINER PIC X(16) VALUE IS 'MYOUTPTCONTAINER'.
01 ERRORCONTAINER  PIC X(16) VALUE IS 'MYERRORCONTAINER'.
01 CHANNELNAME     PIC X(16) VALUE IS 'MYCHANNELNAME'.
01 PHONETIC PIC X(8) VALUE IS 'PHONETIC'.
01 RC PIC 9(8) BINARY.
PROCEDURE DIVISION.
   MOVE 'MEYER' TO LNAME.
   EXEC CICS PUT CONTAINER(INPUTCONTAINER)
        FROM(LNAME)
        CHANNEL(CHANNELNAME)
   END-EXEC.
   EXEC CICS LINK PROGRAM (PHONETIC)
        CHANNEL(CHANNELNAME)
   END-EXEC.
   EXEC CICS GET CONTAINER(ERRORCONTAINER)
        INTO(ERRMSG)
        CHANNEL(CHANNELNAME)
        RESP(RC)
   END-EXEC.
   IF RC = DFHRESP(NORMAL)
      THEN
*          Do some error processing here, because an
*          error container exists. Exit.
        CONTINUE
   END-IF.
   EXEC CICS GET CONTAINER(OUTPUTCONTAINER)
        INTO(PNAME)
        CHANNEL(CHANNELNAME)
   END-EXEC.
   DISPLAY PNAME.
   EXEC CICS RETURN
   END-EXEC.
   GOBACK.
END PROGRAM RESCC05.
```

2.4 STARTBROWSE application programming interface

Typically, programs that exchange a channel are written to handle that channel. Therefore, both client and server programs know the name of the channel, and the names and number of the containers in the channel. However, if, for example, a server program or component is written to handle more than one channel, on invocation, it must discover which of the possible channels it has been passed.

A program can discover its current channel (the channel with which it was started), by issuing an **EXEC CICS ASSIGN CHANNEL** command. If there is no current channel, the command returns blanks.

The program can browse to get the names of the containers in its current channel. Typically, this is not necessary. A program written to handle several channels is often coded to be aware of the names and number of the containers in each possible channel.

To get the names of the containers in the current channel, use the following browse commands:

- `EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(data-area)`
- `EXEC CICS GETNEXT CONTAINER(data-area) BROWSETOKEN(token)`
- `EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(token)`

Having retrieved the name of its current channel and, if necessary, the names of the containers in the channel, a server program can adjust its processing to suit the kind of data that it has been passed.

A `STARTBROWSE` loop is provided in Example 2-10. The loop displays all container names in the channel and their lengths.

Example 2-10 Display all container names in a channel

```
IDENTIFICATION DIVISION.
PROGRAM-ID. RESCC04.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CONTAINERNAME PIC X(16).
01  XRESP PIC 9(8) BINARY.
01  XTOKEN PIC 9(8) BINARY.
01  XLENGTH PIC 9(8) BINARY.
PROCEDURE DIVISION.
    EXEC CICS STARTBROWSE CONTAINER
              BROWSETOKEN(XTOKEN)
              RESP(XRESP)
    END-EXEC.
    IF XRESP = DFHRESP(NORMAL)
       THEN
             EXEC CICS GETNEXT CONTAINER(CONTAINERNAME)
                       BROWSETOKEN(XTOKEN)
                       RESP(XRESP)
             END-EXEC
             PERFORM UNTIL XRESP = DFHRESP(END)
                 EXEC CICS GET CONTAINER(CONTAINERNAME)
                           NODATA FLENGTH(XLENGTH)
                           RESP(XRESP)
                 END-EXEC
                 DISPLAY ' CONTAINER NAME= ' CONTAINERNAME
                    ' CONTAINER LENGTH= ' XLENGTH
                 EXEC CICS GETNEXT CONTAINER(CONTAINERNAME)
                           BROWSETOKEN(XTOKEN)
                           RESP(XRESP)
```

```
                END-EXEC
            END-PERFORM
            EXEC CICS ENDBROWSE CONTAINER
                      BROWSETOKEN(XTOKEN)
                      RESP(XRESP)
            END-EXEC
      ELSE
*           DO SOME ERROR PROCESSING.
            CONTINUE
    END-IF
    EXEC CICS RETURN
    END-EXEC.
    GOBACK.
END PROGRAM RESCC04.
```

On program return, the starting program knows the name of the channel that has been returned, because it is the same name used to start the subroutine. However, it does not necessarily know the names of the containers in the channel.

The question to be answered is: *Does the returned channel contain the same containers as the passed channel, or has the subroutine deleted some or created others?* In this case, we advise browsing through the channel, as shown in Example 2-10 on page 41.

Important: Do not rely on the order in which the container names are returned. This might be unpredictable, because it can depend on various factors, such as whether the STARTBROWSE runs locally (where the channel has been created) or remotely. Also, the order might change between releases.

It is more efficient to perform a GET for a container, rather than browse through the containers (STARTBROWSE), to see if it exists. The NODATA option can be used if you only need to know that the container exists, not what it holds.

2.5 Channels and containers in called subroutines

In the previous sections, we have described how to create, transfer, and return data in channels and containers using the CICS API. In the following section, we describe how dynamically called subroutines can participate.

Processing containers in a called subroutine

When using either a dynamic or static Common Business Oriented Language (COBOL) call, because CICS treats both the calling and called program as the same logical link level, channels that were passed to or created by the calling program are available to the called program. However, you must note the importance of specifying the channel name on any container-related **`EXEC CICS`** command issued in the called program.

If the calling program has a current channel, the called program, by default, assumes this as the channel name. Therefore, if the called program issues a container-related **`EXEC CICS`** command with no channel name specified, the calling program's current channel is assumed. Therefore, the calling program's current channel is also the called program's current channel.

If the calling program does *not* have a current channel, the called program also does not have a current channel. In this case, if the calling program creates its own channel, this channel is available to the called program.

However, any container-related **`EXEC CICS`** command that the called program issues to access this channel *must* specify the channel name. Failure to provide a channel name in this case causes the container-related **`EXEC CICS`** command to return an `INVREQ` with a `RESP2=4(no current channel)`.

If the calling program has a current channel, the called program also has a current channel. If you want to access any other channel, apart from the current channel in the called program, you need to provide the other channel's name in any container-related **`EXEC CICS`** command that you use in the called program.

Therefore, unless you want to use the current channel, you must ensure that you always provide the correct channel name on any container-related command that you issue in a called program.

2.6 Data conversion and code page conversion

CICS is now able to convert received data to another encoding format, and later convert results back to the appropriate form on return. Data for conversion is expected to come from Hypertext Markup Language (HTML) or Extensible Markup Language (XML), but this function is not limited to these. Input data from any interface can be processed with suitable conversion.

2.6.1 Data conversion with channels

Applications that use channels to exchange data use a simple data conversion model. Frequently, no conversion is required and, when it is, a single programming instruction can be used to tell CICS to handle it automatically.

Note the following points:

- Usually, when a (non-Java) CICS Transaction Server program calls another (non-Java) CICS Transaction Server program, no data conversion is required, because both programs use Extended Binary Coded Decimal Interchange Code (EBCDIC) encoding.

  For example, if a CICS Transaction Server C-language program calls a CICS Transaction Server COBOL program to pass it some containers holding character data, the only reason for using conversion would be the unusual one of wanting to change the coded character set identifier (CCSID) of the data.
- The data conversion model that a channel application uses is much simpler than that a COMMAREA application uses. Applications that use COMMAREAs to exchange data use the traditional data conversion model. Conversion is done under the control of the system programmer, using the DFHCNV conversion table, the DFHCCNV conversion program, and the DFHUCNV user-replaceable conversion program, which is optional.

  In contrast, the data in channel containers is converted under the control of the application programmer, using API commands.
- The application programmer is responsible only for the conversion of user data, which is the data in containers that application programs create. When necessary, CICS converts system data automatically.
- Containers only support the conversion of character data. The conversion of binary data, between big-endian and little-endian, is not supported.
- Your applications can use the container API as a simple means of converting character data from one code page to another.

For data conversion purposes, CICS recognizes two types of data:

- CHAR

  Character data is a text string. The data in the container is converted, if necessary, to the code page of the application that retrieves it. If the application that retrieves the data is a client on an American Standard Code for Information Interchange (ASCII)-based system, this is an ASCII code page. If it is a CICS Transaction Server for IBM z/OS® application, it is an EBCDIC code page.

  All the data in a container is converted as though it were a single character string. For single-byte character set (SBCS) code pages this is not a problem, because a structure consisting of several character fields can be interpreted as a single-byte character string.

  However, for double-byte character set (DBCS) code pages this is not the case. This is because DBCS code pages can consist of character strings made up of double-byte characters, or a mixture of single-byte and double-byte characters.

  To ensure that it is obvious that you are using DBCS, and that the data conversion works correctly, you must put each character string of a DBCS code page into a separate container.

- BIT

  All non-character data is everything that is not designated as being of type CHAR. The data in the container cannot be converted. This is the default value, unless `FROMCCSID` is specified, in which case CHAR is the default.

The following API commands are used for data conversion:

- `EXEC CICS PUT CONTAINER [<CHANNEL>] [<DATATYPE>] [FROMCCSID]`
- `EXEC CICS GET CONTAINER [<CHANNEL>] [INTOCCSID]`

2.6.2 How to cause CICS to convert data automatically

In the client program, use the `DATATYPE(DFHVALUE(CHAR))` option of the **`put container`** or **`put64 container`** command to specify that a container holds character data, and that the data is eligible for conversion. The following example shows how to put data into a container eligible for conversion:

```
EXEC CICS PUT CONTAINER(<cont_name>) CHANNEL(<channel_name>)
          FROM(<data1>) DATATYPE(DFHVALUE(CHAR))
```

There is no requirement to specify the `FROMCCSID` or `FROMCODEPAGE` option, unless the data is not in the default CCSID of the client platform. The default CCSID is implied.

Remember: For CICS Transaction Server regions, the default CCSID is specified in the **`LOCALCCSID`** system initialization parameter.

For `DATATYPE(DFHVALUE(CHAR))` you can specify `CHAR`.

In the following example, you find the **`get container`** command to retrieve the data from the program's current channel. The data is returned in the default CCSID of the server platform. There is no requirement to specify the `INTOCCSID` or `INTOCODEPAGE` option unless you want the data to be converted to a CCSID other than the default.

If client and server platforms are different, data conversion takes place automatically. Data retrieved from a container is automatically converted, as the following command shows:

```
EXEC CICS GET CONTAINER(<cont_name>) INTO(<data_area1>)
```

Important: Consider that the `BYTEOFFSET` option of **`GET CONTAINER`** or **`GET64 CONTAINER`** is specified, and you use a code page with multibyte characters. Depending on the `BYTEOFFSET` value that you specify, the data returned might have partial characters at the beginning, end, or both. In this situation, your application program must be able to handle and interpret the data returned.

The return is the same. In the following sample, you can see that the server program issues a **`put container`** command to return a value to the client. Put data in a container and make the data eligible for conversion as follows:

```
EXEC CICS PUT CONTAINER(<status>) FROM(<data_area2>) CHAR
```

In the example that follows, the client program issues a **`get container`** command to retrieve the status returned by the server program. The status is returned in the default CCSID of the client platform. There is no requirement to specify the `INTOCCSID` or `INTOCODEPAGE` option unless you want the data to be converted to a CCSID other than the default.

If the client and server platforms are different, data conversion takes place automatically. Get the data from a container and convert it automatically, as shown in the following example:

```
EXEC CICS GET CONTAINER(<status>) CHANNEL(<channel_name>)
          INTO(<status_area>)
```

Note: We have used INTO for the **get** commands for code page conversion. This is logical when you know exactly how much data is being returned, which might not be the case. A programmer must use SET if the length of the data returned is not known.

2.6.3 Using containers to do code page conversion

Your application can use the container API as a simple means of converting character data from one code page to another. The code shown in Figure 2-1 converts data from codepage1 to codepage2.

```
EXEC CICS PUT CONTAINER(temp) CHAR
              FROMCCSID(ccsid1) FROM(inputdata)
END-EXEC

EXEC CICS GET CONTAINER(temp) INTOCCSID(ccsid2)
              SET(data-ptr) FLENGTH(data-len)
END-EXEC
```

Figure 2-1 Code page data converter

The input data length can differ from the output data length. We suggest that you use the SET option (locate mode), rather than the INTO option (move mode), unless the converted length is known. You can issue a **get container** with the NODATA and FLENGTH option to retrieve the length of the converted data. The length of the output data is calculated, but no data is returned.

When using the container API in this way, note the following tips:

- If you prefer to specify a supported Internet Assigned Numbers Authority (IANA) charset name for the code pages, rather than the decimal CCSIDs, or if you want to specify a CCSID alphanumerically, use the FROMCODEPAGE and INTOCODEPAGE options rather than the FROMCCSID and INTOCSSID options.
- To avoid storage resource requirements after conversion, copy the converted data and delete the container.
- To avoid shipping the channel, use a temporary channel.
- All-to-all conversion is not possible. A code page conversion error occurs if a specified code page and the channel’s code page are an unsupported combination.

2.6.4 SOAP example

A CICS Transaction Server SOAP application performs the following tasks:

1. Retrieve a Unicode Transformation Format 8 (UTF8) or UTF16 message from a socket or IBM MQ message queue.
2. Put the message into a container in UTF8 format.
3. Put EBCDIC data structures into other containers on the same channel.
4. Make a distributed program link (DPL) call to a handler program, on a back-end application-owning region (AOR), passing the channel.

The back-end handler program, also running on CICS Transaction Server, can use the **`EXEC CICS GET CONTAINER`** command to retrieve the EBCDIC data structures or messages. It can get the message in UTF8, or UTF16, or in its own EBCDIC code page, or the region's EBCDIC code page. Similarly, it can use **`EXEC CICS PUT CONTAINER`** commands to place data into the containers, in UTF8, UTF16, or EBCDIC formats.

To retrieve one of the messages in the region's EBCDIC code page, the handler can issue the following command to get data from a container in the default CCSID:

```
EXEC CICS GET CONTAINER(<input_msg>) INTO(<msg>)
```

Because the `INTOCCSID` and `INTOCODEPAGE` options are not specified, the message data is automatically converted to the region's EBCDIC code page. This assumes that the **`put container`** command used to store the message data in the channel specified is a `DATATYPE` of `CHAR`. If it specified a `DATATYPE` of `BIT`, which is the default, no conversion is possible.

To return output to the region's EBCDIC code page, the handler can issue the following command to put data into a container default CCSID:

```
EXEC CICS PUT CONTAINER(<output>) FROM(<output_msg>)
```

Because `CHAR` is not specified, no data conversion is permitted. Because the `FROMCCSID` or `FROMCODEPAGE` options are not specified, the message data is selected to be in the region's EBCDIC code page.

To retrieve one of the messages in UTF8, the `INTOCCSID` or `INTOCODEPAGE` option must be specified to identify the code page and prevent automatic conversion of the data to the region's EBCDIC code page. The handler can issue the following command to retrieve the data and convert it:

```
EXEC CICS GET CONTAINER(<input_msg>) INTO(<msg>) INTOCCSID(<utf8>)
```

In this case, <*utf8*> is a variable that is defined as a fullword, and is initialized to 1208, which is the CCSID for UTF8. If you prefer to use an IANA charset name for the code page, you can use the INTOCODEPAGE option rather than the INTOCCSID option:

```
EXEC CICS GET CONTAINER(<input_msg>) INTO(<msg>) INTOCODEPAGE(<utf8>)
```

In this case, utf8 is a variable that is defined as a character string of length 56, and is initialized to ‘utf-8’.

To return some output in UTF8, the server program can issue the following command to specify a CCSID on returning the data:

```
EXEC CICS PUT CONTAINER(<output>) FROM(<output_msg>) FROMCCSID(<utf8>)
```

or alternatively:

```
EXEC CICS PUT CONTAINER(<output>) FROM(<output_msg>)
FROMCODEPAGE(<utf8>)
```

In this case, the variable <*utf8*> is defined and initialized in the same way as for INTOCCSID and INTOCODEPAGE.

The FROMCCSID or FROMCODEPAGE option specifies that the message data is currently in UTF8 format. Because FROMCCSID or FROMCODEPAGE is specified, a DATATYPE of CHAR is implied, so conversion is permitted.

2.6.5 File example

A CICS Transaction Server file application performs the following actions:

1. Read a VSAM key-sequenced data set (KSDS) file with client records.
2. Put the EBCDIC data into a container.
3. Get the data from the container in ASCII format.
4. Write a transient data (TD) queue to externalize the file in ASCII format.

The file interface has no code page conversion capabilities. So, if you want to externalize a file from an EBCDIC code page to an ASCII file, for example, it is easy to use the channels and containers.

To read the file records in the region’s EBCDIC code page, the program can issue the following command:

```
EXEC CICS READ FILE(<filename>) INTO(<data-area1>)
RIDFLD(<keydata-area>)
```

To copy the data and move the record into a container and make it eligible for conversion, use the following command:

```
EXEC CICS PUT CONTAINER(<containername>) CHANNEL(<channelname>)
FROM(<data-area1>) CHAR
```

The `CHAR` option specifies that the container holds character data, and that the data is eligible for conversion. The `FROMCCSID` option has been omitted, so the region's default CCSID is implied.

The following command converts the container from the current local EBCDIC code page to ASCII:

```
EXEC CICS GET CONTAINER(<containername>) CHANNEL(<channelname>)
INTO(<data-area2>) INTOCCSID(858)
```

Figure 2-2 shows an example of how to use a variable for `INTOCCSID`.

```
01 ASCIICODEPAGE pic 9(8) BINARY.
.
MOVE 858 TO ASCIICODEPAGE
EXEC CICS GET CONTAINER(containername) CHANNEL(channelname)
              INTO(data-area2) INTOCCSID(ASCIICODEPAGE)
```

Figure 2-2 Use a variable to hold the code page number

The following command is to externalize the ASCII record to a TD queue file:

```
EXEC CICS WRITEQ TD QUEUE(<queuename>) from(<data-area2>)
```

2.6.6 Command-level interpreter CICS-supplied transaction example

Use a CICS command-level interpreter (CECI) transaction to put EBCDIC data into a container and receive the container in ASCII. Do *not* press **PF3** after putting the data into a container with CECI, because this stops CECI and the current unit of work (UOW). A new session does not have access to the container, so simply overwrite the CECI input string.

In Figure 2-3, you can see how to put that data into a container, and what the CICS system returns.

```
CECI PUT CONTAINER(TEMP) CHANNEL(MYCHANNELNAME) FROM(12345) CHAR

returns:

PUT CONTAINER(TEMP) CHANNEL(MYCHANNELNAME) FROM(12345) CHAR
STATUS:  COMMAND EXECUTION COMPLETE                         NAME=
 EXEC CICS  PUT COntainer( 'TEMP            ' )
  < ACTivity() | ACQActivity | Process | ACQProcess
    | CHANnel( 'MYCHANNELNAME   ' ) >
  FROM( '12345' )
  < FLength( +0000000005 ) >
  < Datatype() | Bit | CHAR >
  < FROMCCsid() | FROMCOdepage() >
  < APpend >

in hex:

PUT CONTAINER(TEMP) CHANNEL(MYCHANNELNAME) FROM(12345) CHAR
STATUS:  COMMAND EXECUTION COMPLETE                         NAME=
 EXEC CICS  PUT COntainer( X'E3C5D4D7404040404040404040404040' )
  < ACTivity() | ACQActivity | Process | ACQProcess
    | CHANnel( X'D4E8C3C8C1D5D5C5D3D5C1D4C5404040' ) >
  FROM( X'F1F2F3F4F5' )
  < FLength( X'00000005' ) >
  < Datatype() | Bit | CHAR >
  < FROMCCsid() | FROMCOdepage() >
  < APpend >
```

Figure 2-3 Put X'F1F2F3F4F5' into a container

In Figure 2-4 you can see that CICS returns the ASCII representation of the digits “12345”.

```
GET CONTAINER(TEMP) CHANNEL(MYCHANNELNAME) INTO(&OUT)
INTOCODEPAGE(858)

returns:

GET CONTAINER(TEMP) CHANNEL(MYCHANNELNAME) INTO(&OUT)
INTOCODEPAGE(858)
STATUS:  COMMAND EXECUTION COMPLETE                       NAME=
 EXEC CICS  GET CONtainer( 'TEMP           ' )
  < ACTivity() | ACQActivity | Process | ACQProcess
    | CHannel( 'MYCHANNELNAME   ' ) >
  ( Set() | INTO( '.....' ) | NOData )
  < Flength( +0000000005 ) >
  < INTOCCsid() | INTOCOdepage( '858
' )
    | (( COnvertst() | NOConvert ) < CCsid() > > )
  < Byteoffset() >

in hex:

GET CONTAINER(TEMP) CHANNEL(MYCHANNELNAME) INTO(&OUT)
INTOCODEPAGE(858)
STATUS:  COMMAND EXECUTION COMPLETE                       NAME=
 EXEC CICS  GET CONtainer( X'E3C5D4D7404040404040404040404040' )
  < ACTivity() | ACQActivity | Process | ACQProcess
    | CHannel( X'D4E8C3C8C1D5D5C5D3D5C1D4C5404040' ) >
  ( Set() | INTO( X'3132333435' ) | NOData )
  < Flength( X'00000005' ) >
  < INTOCCsid()
    | INTOCOdepage(
X'F8F5F84040404040404040404040404040404040404040' ... )
    | (( COnvertst() | NOConvert ) < CCsid() > > )
  < Byteoffset() >
```

Figure 2-4   Get container returns ASCII digits X'3132333435'

Restriction: Replacing a container with a different data type is not possible using **`PUT CONTAINER`**. Also, you cannot replace a container using different CCSIDs. If you need to do this, delete the container and re-create it with a **`PUT CONTAINER`**.

2.7 Storage

Container storage is created in 64-bit data. However, storage for containers must be copied below the bar to access it. When using channels and containers with sizes significantly above 32 KB, you must pay attention to the following factors:

- Dynamic storage area (DSA) size

  Using big containers requires big DSAs, which could lead to short-on-storage (SOS) conditions, if you need to get the whole contents of the container in one go. You can use **`BYTEOFFSET`** to **`GET`** containers in sections, and you can use **`APPEND`** to **`PUT`** containers in sections.

- Addressing mode (AMODE)

  A program running in AMODE 24 cannot receive containers greater than 16 megabytes (MB), unless `OFFSET` is used.

  You must specify AMODE(64) *only in a non-Language Environment assembler language program* if you want to use **`EXEC CICS GET64 CONTAINER`** and **`EXEC CICS PUT64 CONTAINER`** commands.

- Data location

  You must read any programs with data location that cannot receive containers greater than 16 MB.

- Performance

  When an **`EXEC CICS LINK`** command is sent over an intersystem communication (ISC) connection and a big container is specified, it can lead to performance issues. Sending big containers over ISC links can take a significant amount of time.

- Queuing

  If links are busy transmitting big containers, it can lead to queuing in the CICS system.

- Using the `SET` option

  Specifying a `SET` value on an **`EXEC CICS GET CONTAINER`** command always gets the whole container contents below the bar.

The storage model for containers is different than that of COMMAREAs. COMMAREAs are shared storage between programs, therefore there is only one copy of the data, except in certain circumstances, such as programs with different keys and AMODEs. For container storage, there are three copies of the data:

- The working storage of each of the programs
- The CICS copy of the container

If you have transactions that use very large containers, you might consider controlling such transactions by the use of `TCLASS` to circumvent the possibility of your system going into an SOS condition.

2.8 Dynamic routing application considerations

You might need to dynamically route workload based on information in the COMMAREA. The normal procedure you can use to do this is if you provide additional system-related information in a header. The application program places the header at the top of the COMMAREA before the application data. Application programming and systems programming must mutually agree on the header information.

The systems programmer provides a user-replaceable module, which gets control for every transaction or program defined as dynamic to CICS. This module has access to the COMMAREAs, so it can read the header information and make a routing decision based on this information. The default routing program for CICS is DFHDYP. If you use IBM CICSPlex System Manager, it is mandatory to use EYU9XLOP as the routing program.

An application header could have information for lengths, terminal information, application information, time stamps, and also as an option, user ID, password information, and so on. The following example does not provide a complete header, but is a more simple example based on the previously introduced phonetic application.

Figure 2-5 shows the smallest system configuration for the dynamic routing environment, at least two AORs connected to a terminal-owning region (TOR).

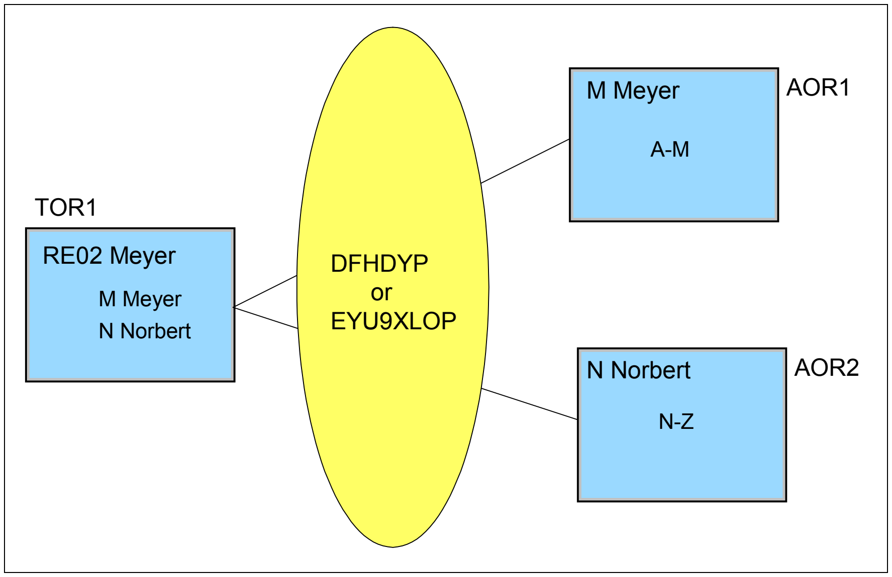


Figure 2-5 DFHDYP or EYU9XLOP influence the AOR selection

The example application has a 40-byte name as input, and a 40-byte phonetically translated name as output. Additionally, we have a one-byte header, which is the first character of the name. The exit influences the routing decision based on this header. If the first character in the name falls between A and M, AOR1 is eligible for routing, otherwise AOR2 is selected.

2.8.1 COMMAREA

In the COMMAREA solution, prefix the application data with the header. The dynamic routing exit has access to the COMMAREA. The CICS system provides the address of the COMMAREA to the exit. Note that in this simple example you can avoid the header, and the routing exit can directly access the application data.

However, this is not good in practice, because the routing exit must know all of the structures of all of the applications. Therefore, we use a header.

In Example 2-11 you can see the COMMAREA instance.

Example 2-11 Provide a header in the COMMAREA for the dynamic routing program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EX211.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 MYCOMMAREA.
   03 HEADER PIC X(1).
   03 LNAME  PIC X(40).
   03 PNAME  PIC X(40).
01 PHONETIC PIC X(8) VALUE IS 'PHONETIC'.
PROCEDURE DIVISION.
   MOVE 'MEYER' TO LNAME.
   EXEC CICS LINK PROGRAM (PHONETIC)
        COMMAREA(MYCOMMAREA)
   END-EXEC.
   DISPLAY PNAME.
*         PNAME CONTAINS 'MAYR'.
   EXEC CICS RETURN
   END-EXEC.
   GOBACK.
END PROGRAM EX211.
```

2.8.2 Channel

When using a channel, Example 2-11 looks slightly different. As previously mentioned, we advise you to separate the data in different containers. Therefore, put the name in an input container, to get back an output container. The container data has to be available to the routing program. However, the channels and containers API is not valid in this context, and the **`EXEC CICS GET CONTAINER`** command would result in an *invalid request*.

Therefore, the container data must be put in a special container that the routing program can read. The mandatory name of this container is DFHROUTE. For further information about DFHROUTE, see Chapter 4, “Systems management and configuration” on page 99.

In our example, we create an input container and a container for routing information in one channel. On the **`EXEC CICS LINK`** command, we provide the channel name. Because the subroutine is defined as dynamic, the routing exit gets control. This exit has direct access to the DFHROUTE container only. The routing decision is made based on the content in this container.

In Example 2-12, you can find the code necessary to create the DFHROUTE container.

Example 2-12 Provide the header in a DFHROUTE container

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EX212.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LNAME                     PIC X(40).
01 ROUTEINFO REDEFINES LNAME PIC X(1).
01 PNAME                     PIC X(40).
01 INPUTCONTAINER   PIC X(16) VALUE IS 'MYINPUTCONTAINER'.
01 OUTPUTCONTAINER  PIC X(16) VALUE IS 'MYOUTPTCONTAINER'.
01 ROUTINGCONTAINER PIC X(16) VALUE IS 'DFHROUTE'.
01 CHANNELNAME      PIC X(16) VALUE IS 'MYCHANNELNAME'.
01 PHONETIC PIC X(8) VALUE IS 'PHONETIC'.
PROCEDURE DIVISION.
   MOVE 'MEYER' TO LNAME.
   EXEC CICS PUT CONTAINER(INPUTCONTAINER)
      FROM(LNAME)
      CHANNEL(CHANNELNAME)
   END-EXEC.
   EXEC CICS PUT CONTAINER(ROUTINGCONTAINER)
      FROM(ROUTEINFO)
      CHANNEL(CHANNELNAME)
   END-EXEC.
   EXEC CICS LINK PROGRAM (PHONETIC)
      CHANNEL(CHANNELNAME)
   END-EXEC.
   EXEC CICS GET CONTAINER(OUTPUTCONTAINER)
      INTO(PNAME)
      CHANNEL(CHANNELNAME)
   END-EXEC.
   EXEC CICS RETURN
   END-EXEC.
   GOBACK.
END PROGRAM EX212.
```

Remember: The name of the container including the routing information must be DFHROUTE. The routing exit has access to the DFHROUTE container, but no access to other containers.

2.9 Best practices

This section provides information about leading practices when using channels and containers.

2.9.1 Designing a channel

It is possible to use containers to pass data in the same manner as COMMAREAs have traditionally been used. However, channels have several advantages over COMMAREAs, and it is advantageous to design your own channels to make the most of these improvements.

At the end of a DPL call, if a container has changed, it is necessary for CICS to send its contents back to the calling region. Input containers whose contents have been changed by the server program, and containers created by the server program, are returned. Therefore, for optimal DPL performance, abide by the following practices:

- Use separate containers for input and output data.
- The server program, not the client, must create the output containers.
- Use separate containers for read-only and read/write data.
- Use separate containers for CHAR data and BIT data.
- If a structure is optional, make it a separate container.
- Use dedicated containers for error information.

The following hints provide general guidelines for designing a channel. They include and expand on the suggestions for achieving optimal DPL performance:

- Use separate containers for input and output data. This provides the following advantages:
  - Better encapsulation of the data, making your programs easier to maintain
  - Greater efficiency when a channel is passed on a DPL call, because smaller containers flow in each direction
- Do not initialize containers in a client program if the server program does not require the data as input data. If the client program has any large containers that the server program does not require, use a separate channel for the DPL request. Use `MOVE` to move the containers between the current channel and the channel passed to the server.

- Use separate containers for read-only and read/write data. This provides the following advantages:
  - Simplified copybook structure makes your programs easier to understand
  - Avoidance of the problems with REORDER overlays
  - Greater transmission efficiency between CICS regions, because read-only containers sent to a server region are not returned
- Use separate containers for each structure. This provides the following advantages:
  - Better encapsulation of the data, making your programs easier to understand and maintain
  - Greater ease in changing one of the structures, because you do not require to recompile the entire component
  - The ability to pass a subset of the channel to subcomponents if you use the **`move container`** command to move containers between channels
- If a structure is optional, make it a separate container. This leads to greater efficiency, because the structure is passed only if the container is present.
- Use dedicated containers for error information. This provides the following advantages:
  - Better documentation of what is error information.
  - Greater efficiency:
    - The structure containing the error information is passed back only if an error occurs.
    - It is more efficient to check for the presence of an error container if you issue a **`get container (`*`<known-error-container-name>`*`)`** or **`get container64 (`*`<known-error-container-name>`*`)`** command and possibly receive a `NOTFOUND` condition, than it is to initiate a browse of the containers in the channel.
- When you need to pass data of different types, for example binary data and character data, use separate containers for each type, rather than one container with a complicated structure. This improves your ability to move between different code pages.
- Do *not* create too many large containers, because it limits the amount of storage available to other applications.
- Channels and containers use storage below 2 gigabytes (GB) (below the bar) and some 64-bit (above-the-bar) storage. Their use of 64-bit storage influences the value that you choose for the z/OS **`MEMLIMIT`** parameter that applies to the CICS region. You must also consider other CICS facilities that use 64-bit storage.

2.9.2 Naming a channel

Before linking to the subprogram, you must move the data to a container and connect the container to a channel. Note that the channel and container names are case-sensitive. For example, *MyChannel* differs from *mychannel* and *MYCHANNEL*. Therefore, putting a container into *MyChannel* and then linking to a program with a container name of *mychannel* does not give error messages during compile, but might lead to unexpected results when you run the program.

CICS creates a new channel on first reference. Therefore, if there is a typo in the channel name specified on the `EXEC CICS LINK` command, a new channel is created and transferred to the subroutine without any containers.

To avoid this pitfall, we suggest that you do not use strings, such as container or channel names included in quotation marks. Instead, move the channel name and the container name into a previously defined variable that you can use in the API. This ensures that your channel or container name is always correct.

2.9.3 Porting from COMMAREAs to channels

The following sections describe the various processes of porting from COMMAREAs to channels.

Porting existing functions

You should take the following points into consideration:

- CICS application programs that use traditional COMMAREAS to exchange data continue to work as before.
- If you employ a user-written dynamic or distributed routing program for workload management, rather than CICSPlex System Manager, you must modify your program to handle the new values that it can be passed in the DYRTYPE field of the DFHDYPDS communications area. See the *CICS Transaction Server for z/OS V5.2 CICS Customization Guide*, SC34-7269.

Porting to the channels and containers function

This section describes how you can port several types of existing applications to use channels and containers rather than COMMAREAs.

It is possible to replace a COMMAREA with a channel that consists of a single container. We do not recommend this practice, even though it might seem to be the simplest means to move from COMMAREAs to channels and containers.

You must implement the leading practices for channels and containers, because you are taking the time to change your application programs to use this function. See 2.9.1, “Designing a channel” on page 58.

Channels have several advantages over COMMAREAs. See 1.5, “Benefits of using channels and containers” on page 22. It is advantageous to design your channels to make the most of these improvements.

Porting link commands that pass COMMAREAs

To port two programs that use a COMMAREA on a `link` command to exchange a structure, change the instructions as shown in Table 2-1.

Table 2-1 Porting link commands that pass COMMAREAs

Program	Before	After

PROG1	EXEC CICS LINK
	PROGRAM(PROG2)
	COMMAREA(structure)
		EXEC CICS PUT
		CONTAINER(contr-name)
		CHANNEL(channel-name)
		FROM(structure)
		
		EXEC CICS LINK
		PROGRAM(PROG2)
		CHANNEL(channel-name)
		.
		.
		.
		EXEC CICS GET
		CONTAINER(contr-name)
		CHANNEL(channel-name)
		INTO(structure)

PROG2	EXEC CICS ADDRESS
	COMMAREA(structurePtr)
	...
	RETURN
		EXEC CICS GET
		CONTAINER(contr-name)
		INTO(structure)
		...
		EXEC CICS PUT
		CONTAINER(contr-name)
		FROM(structure)
		
		EXEC CICS RETURN

Note: In the COMMAREA example, PROG2, having put data in the COMMAREA, only has to issue a `return` command to return the data to PROG1. In the channel example, to return data, PROG2 must issue a `put container` command before the `return`.

Porting xctl commands that pass COMMAREAs

To port two programs, which use a COMMAREA on an `xctl` command to pass a structure, change the instructions as shown in Table 2-2.

Table 2-2 Porting xctl commands that pass COMMAREAs

Program	Before	After
PROG1	EXEC CICS XCTL PROGRAM(PROG2) COMMAREA(structure)	EXEC CICS PUT CONTAINER(contr-name) CHANNEL(channel-name) FROM(structure) EXEC CICS XCTL PROGRAM(PROG2) CHANNEL(channel-name)
PROG2	EXEC CICS ADDRESS COMMAREA(structurePtr)	EXEC CICS GET CONTAINER(contr-name) INTO(structure)

Porting pseudo-conversational transactions COMMAREAs

To port two programs, which use COMMAREAs to exchange a structure as part of a pseudo-conversation, change the instructions as shown in Table 2-3.

Table 2-3 Porting pseudo-conversational COMMAREAs on return command

Program	Before	After
PROG1	EXEC CICS RETURN TRANSID(TRAN2) COMMAREA(structure)	EXEC CICS PUT CONTAINER(contr-name) CHANNEL(channel-name) FROM(structure) EXEC CICS RETURN TRANSID(TRAN2) CHANNEL(channel-name)
PROG2	EXEC CICS ADDRESS COMMAREA(structurePtr)	EXEC CICS GET CONTAINER(contr-name) INTO(structure)

Porting start data

To port two programs, which use start data to exchange a structure, change the instructions as shown in Table 2-4.

Table 2-4 Porting start data

Program	Before	After

PROG1	EXEC CICS START
	TRANSID(TRAN2)
	FROM(structure)
		EXEC CICS PUT
		CONTAINER(contr-name)
		CHANNEL(channel-name)
		FROM(structure)
		
		EXEC CICS START
		TRANSID(TRAN2)
		CHANNEL(channel-name)

PROG2	EXEC CICS RETRIEVE
	INTO(structure)
		EXEC CICS GET
		CONTAINER(contr-name)
		INTO(structure)

Note: The new version of PROG2 is the same as that in the pseudo-conversational example.

Porting dynamically routed applications

`EXEC CICS LINK` and `EXEC CICS START` commands, which can pass either COMMAREAs or channels, can be dynamically routed.

When a `link` or `start` command passes a COMMAREA rather than a channel, the routing program can, depending on the type of request, inspect or change the COMMAREA's contents. For link requests and transactions, determining which terminal-related start requests begin is handled by the dynamic routing program.

The routing program is given the address of the application's COMMAREA in the DYRACMAA field of its communication area, and can inspect and change its contents. The same rule is not valid for non-terminal-related start requests, which are handled by the distributed routing program.

Important: The routing programs communication area is mapped by the DFHDYPDS DSECT.

If you port a dynamically-routed **EXEC CICS LINK** or **EXEC CICS START** command to use a channel rather than a COMMAREA, the routing program is passed in the DYRCHANL field of DFHDYPDS, which is the name of the channel. Note that the routing program is given the name of the channel, not its address, and so is unable to use the DYRCHANL field to inspect or change the contents of the channel's containers.

To give the routing program the same kind of functionality with channels, an application that uses a channel can create, within the channel, a special container named DFHROUTE.

If the application issues a link or terminal-related start request that is to be dynamically routed, the dynamic routing program is given in the DYRACMAA field of DFHDYPDS, the address of the DFHROUTE container, and can inspect and change its contents. This rule is not valid for a non-terminal-related **start** request.

If you are porting a program to pass a channel rather than a COMMAREA, you could use its existing COMMAREA structure to map DFHROUTE.

See Chapter 4, "Systems management and configuration" on page 99 for more information.

3

Programming

This chapter provides information about the channels and containers application programming interface (API) and how you can use it in a traditional IBM Customer Information Control System (CICS) application and also a CICS Java (JCICS) application. It also describes the business transaction services (BTS) API as a similar but recoverable alternative to channels and containers.

3.1 EXEC CICS application programming interface

CICS provides a set of API commands that you can use to perform actions in channels and containers. The following sections describe the various actions that you can perform using the API, and some of the nuances involved in doing so.

Information: For a detailed breakdown of the **`EXEC CICS`** API, see the CICS Transaction Server for z/OS V5.2 Knowledge Center on the following website:

http://www-01.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.home.doc/welcomePage/welcomePage.html

3.1.1 Creating a channel

There is no explicit **`EXEC CICS`** API command for creating an instance of a channel. Instead, an instance of a channel is implicitly created when performing certain commands. Example 3-1 shows a set of commands that instantiates a named channel if it does not already exist.

Example 3-1 Creating a channel

```
EXEC CICS LINK PROGRAM CHANNEL
EXEC CICS MOVE CONTAINER CHANNEL TOCHANNEL
EXEC CICS PUT CONTAINER CHANNEL
EXEC CICS PUT64 CONTAINER CHANNEL
EXEC CICS RETURN TRANSID CHANNEL
EXEC CICS START TRANSID CHANNEL
EXEC CICS XCTL PROGRAM CHANNEL
```

Tip: If the named channel does not exist on a **`link`**, **`return`**, **`start`**, or **`xctl`** command, an empty channel is created and passed to the program. You can use this with **`link`** as a location to place values that are to be returned from the link to the program.

3.1.2 Placing data in a container

The **`put container`** and **`put64 container`** commands are used to add data to a container. If the container named on the command does not exist, it is created and the data inserted. Its contents are overwritten if the container already exists. The length of a container is specified using the **`FLENGTH`** parameter, which you can omit if the compiler is able to determine the size of the data area being inserted into the container.

Inserting data as character (CHAR) or binary (BIT) has an important difference. Character data is susceptible to code page conversion and binary data is not. Data inserted as BIT means that it remains untouched during its life in the container. Alternatively, CHAR data might get converted to another code page or coded character set identifier (CCSID).

Put container CHAR

Example 3-2 shows character data being put into a container with the Extended Binary Coded Decimal Interchange Code (EBCDIC) code page specified with the **FROMCCSID(037)** parameter. The CCSID value is stored with the data, such that, if it is extracted with an alternative CCSID specification, the data converts to the new code page before being given to the application.

Example 3-2 Putting character data into a container with CCSID 037

```
EXEC CICS PUT CONTAINER('USER-ID') CHANNEL('ACCOUNT-DATA')
FROM(USER-ID) FLENGTH(LENGTH OF USER-ID) CHAR FROMCCSID(037) END-EXEC.
```

In Example 3-2, you could use DATATYPE(DFHVALUE(CHAR)) rather than CHAR. After you insert a character data into a container, its CCSID value is unchangeable. Therefore, in a subsequent **PUT** to the same container, if you specify a different **FROMCCSID** value, the data is converted to that of the original data's CCSID, and the old data is overwritten.

Important: If CHAR is specified and **FROMCCSID** is not, the **put** command uses the value of the **LOCALCCSID SIT** parameter for the data's code page. If **LOCALCCSID** is not set, the value 037 (US EBCDIC) is used.

Put container BIT

The **FROMCCSID** and **FROMCODEPAGE** parameters are not applicable to containers of BIT type, and an INVREQ response is returned if you attempt to use it.
Example 3-3 shows how you can insert binary data into a container.

Example 3-3 Putting binary data into a container

```
EXEC CICS PUT CONTAINER('ACCOUNT-NO') CHANNEL('ACCOUNT-DATA')
FROM(ACCOUNT-NO) FLENGTH(LENGTH OF ACCOUNT-NO) BIT END-EXEC.
```

In Example 3-3, you could use DATATYPE(DFHVALUE(BIT)) rather than BIT.

Note: If CHAR or BIT is not specified on a **put container** command, the default behavior is to insert it as BIT unless you specify the **FROMCCSID** or **FROMCODEPAGE**.

You can insert character data into a container as `BIT`, *if* the region that the data is extracted from is running on the same code page as that which inserted the data. If the region has an alternative code page, the character data might be unrecognizable when it is extracted.

Requirement: You must always insert character data as `CHAR` and binary data as `BIT`. This increases the portability of your applications and enables you to deploy them on different regions with different code pages.

Put container APPEND

If you specify the `APPEND` option, the data passed to the container is appended to the existing data in the container, as shown in Example 3-4. If this option is not stated, the existing data in the container is overwritten by the data passed to the container.

Example 3-4 Appending data into a channel with existing data

```
EXEC CICS PUT CONTAINER(‘LARGE-CONTAINER’) CHANNEL(‘MYCHANNEL’)
FROM(SOME-DATA) FLENGTH(LENGTH OF SOME-DATA-AREA) END-EXEC.
:
EXEC CICS PUT CONTAINER(‘LARGE-CONTAINER’) CHANNEL(‘MYCHANNEL’)
FROM(SOME-DATA) FLENGTH(LENGTH OF SOME-DATA) APPEND END-EXEC.
```

Example 3-4 shows that data is placed in the LARGE-CONTAINER container in the MYCHANNEL channel from the SOME-DATA data area with a **`put container`** command.

Then, although the example does not show this, more data is placed into DATA-AREA, which is placed into LARGE-CONTAINER by the second **`put container`** command. Because we have specified the `APPEND` option, the data from SOME-DATA is appended to the end of the existing data created in LARGE-CONTAINER by the first **`put container`** command.

3.1.3 Passing a channel to another program or task

Similar to the process that you use to pass communication areas (COMMAREAs) to programs or tasks, the facility to pass a channel also exists. This is applicable to programs that run either locally or remotely. When passing channels to other programs or tasks, it is useful to understand the scope of channels as they move around. We describe this in detail in Chapter 2, “Application design and implementation” on page 27.

Link channel

You can pass a channel on a `link` request using the same process you use to pass a COMMAREA. If the new program alters any content in the containers of the channel, the resulting changes are reflected in the channel when it is returned to the controlling program. The following example shows the linking of a program with a channel:

```
EXEC CICS LINK PROGRAM(‘MY-PROG’) CHANNEL(‘ACCOUNT-DATA’) END-EXEC.
```

Xctl channel

When transferring control to another program, you can pass a channel to the new program. Because control is never given back to the calling program, any channel that it had access to goes out of scope and the storage is freed. The following example shows the passing of a channel on an `xctl` command:

```
EXEC CICS XCTL PROGRAM(‘MY-PROG’) CHANNEL(‘ACCOUNT-DATA’) END-EXEC.
```

Return channel

For pseudo-conversational programming, you can pass a channel to the next transaction to be run, which is similar to that of a COMMAREA. The following example shows the passing of a channel in a pseudo-conversational program:

```
EXEC CICS RETURN TRANSID(‘ABCD’) CHANNEL(‘ACCOUNT-DATA’) END-EXEC.
```

Start channel

When starting a new task, you can pass a channel to the newly running program. The channel being passed is copied along with its containers and placed into another channel with the same name given to the program. Therefore, any changes that are made to this new channel are not reflected back in the original channel, because it goes out of scope of the original program.

Also, although the `start` command might *not* start the new transaction until after the current transaction finishes, any subsequent changes to containers are *not* reflected in the channel copy. The channel is a copy *made at the time that you issue* the `start` command. The example shows a channel passing on a start call:

```
EXEC CICS START TRANSID(‘ABCD’) CHANNEL(‘ACCOUNT-DATA’) END-EXEC.
```

Information: Some key points about passing channels to programs or tasks are described in the following list:

- COMMAREAs and channels are mutually exclusive. You can pass one or the other, but not both.
- If the channel specified on a **`link`**, **`xctl`**, **`return`**, or **`start`** command does not already exist, an empty one is created and passed to the program.

3.1.4 Receiving the current channel

When a channel is passed to a program, it is called the *current channel*. The application owns this channel, and can use it without explicitly referencing its name. Example 3-5 shows data being put into a container in the current channel, without the **`CHANNEL`** parameter. If the current channel did not exist, the command would fail with an `INVREQ` response.

Example 3-5 Adding a container to the current channel

```
EXEC CICS PUT CONTAINER('ACCOUNT-NO') FROM(ACCOUNT-NO)
FLENGTH(LENGTH OF ACCOUNT-NO) END-EXEC.
```

If you require the name of the current channel for use on the commands previously described in 3.1.3, “Passing a channel to another program or task” on page 69, you can use the **`assign channel`** command:

```
EXEC CICS ASSIGN CHANNEL(CHANNEL-NAME) END-EXEC.
```

The value is set to blanks if no current channel exists.

Tip: You can reference the current channel explicitly, if you specify its name in the **`CHANNEL`** parameter on any channel-supported command.

Observe: When using any of the commands in 3.1.3, “Passing a channel to another program or task” on page 69, omitting the channel option does *not* cause the program or task to pass to the current channel. Rather, it passes nothing at all.

3.1.5 Getting data from a container

The **get container** command is used to retrieve data from a container. You can place the data into an existing piece of storage, have the container code allocate storage for you and get it to return the address, or simply query the size of the container. This is all done through slight variations of the **get container** command.

Get container NODATA

As the parameter name suggests, **NODATA** does not return the container contents in the **get container** command. Rather, it sets the field specified in **FLENGTH** to the length of the container data that you can retrieve.

If the container has character data on a different code page in comparison to that with which it is being extracted, the data first converts to the new code page and then the converted data length is returned. This enables you to check if the size of the data area that you have is sufficient for the converted data.

Example 3-6 shows a **get container NODATA** call on a container holding character data. Because a channel is not specified, you can use the current channel.

Example 3-6 Using NODATA to get the size of character data in a container

```
EXEC CICS GET CONTAINER(‘USER-ID’) NODATA FLENGTH(DATA-LEN)
INTOCCSID(037) END-EXEC.
```

Note: Similar to **put container**, if **INTOCCSID** or **INTOCODEPAGE** is not used when extracting character data, the value specified in the **LOCALCCSID SIT** parameter is used. If **LOCALCCSID** is not set, the value 037 (US EBCDIC) is used.

Get container INTO

The **INTO** parameter is used to retrieve the container contents and place it in an existing data area. With this approach, the **FLENGTH** parameter has the following two purposes:

- On input, it specifies the maximum size of the data that can be put into the supplied data area. If BYTEOFFSET is specified, the data returned begins at the offset specified (from the start of the container).
- On output, it holds the size of the data that was returned in the request.

If the available data in the container is greater than the value specified in **FLENGTH**, a LENGERR response is given to signify that the retrieved data was truncated. **FLENGTH** is optional, depending on the compiler's ability to determine the length of the field specified in the **INTO** parameter. Example 3-7 shows a **get container INTO** command to extract binary data. The **INTOCCSID** option is not specified, because it is not applicable to binary data.

Example 3-7 Getting the contents INTO an existing storage area

```
EXEC CICS GET CONTAINER('ACCOUNT-NO') INTO(DATA-AREA) FLENGTH(DATA-LEN)
END-EXEC.
```

Get container SET

For greater flexibility while working with varying length data sizes, the **get container SET** command dynamically allocates a piece of storage to place the container's contents. The address of the allocated storage is returned in the field specified on the **SET** parameter. The returned storage is internally managed, and therefore you cannot depend on it to exist indefinitely.

If BYTEOFFSET is specified, the data returned begins at the offset specified (from the start of the container).

Information: Each container has its own **SET** storage. This remains until the channel goes out of scope, or until you issue a command against that channel, which could change the contents of the channel. For example, you might issue a **MOVE**, **PUT**, **DELETE**, or some forms of **GET**. Large character containers that require CCSID conversion can use a large amount of **SET** storage.

The following situations are some in which the data can no longer be guaranteed to exist:

- When any program that can access this storage issues a subsequent **get container** command with the **SET** option, for the same container in the same channel
- When a **delete container** command deletes the container
- When a **move container** command moves the container
- When the channel goes out of program scope

The **FLENGTH** is an output-only parameter with the **SET** command. The length of the data returned is stored in the field given to it.

Example 3-8 shows a **`get container set`** command using a pointer to hold the address of the extracted data.

Example 3-8 Obtaining the address of storage representing the container contents

```
EXEC CICS GET CONTAINER('USER-ID') SET(DATA-PTR) FLENGTH(DATA-LEN)
END-EXEC.
```

The following points are important to note when using the **`SET`** option:

- Do not issue a **`FREEMAIN`** command to release this storage, because storage management is handled internally.
- If your application needs to store the data, it must move it into its own storage, because there is no guarantee that it exists for any prolonged period of time.
- Be careful if your program links to another program after using the **`SET`** option. If another command is issued for this container, it might delete the **`SET`** pointer, which could cause a storage overwrite.

> **Important:** Data is *always* inserted and retrieved from containers by copy, even with a **`SET`** command. Therefore, if you want to update a container's contents, you must perform a **`put container`** after the original **`get container`**.

3.1.6 Browsing the current channel

In situations where you get a channel, but do not know what containers it has or even how many of them there are, you can perform a browse on the channel to get back the name of each container.

STARTBROWSE container

You can initiate a browse on a channel using the **`STARTBROWSE CONTAINER`** command. A browse token is returned in the field specified on the **`BROWSETOKEN`** parameter. Use this parameter to perform the browse on the channel. If the **`CHANNEL`** parameter is not specified, an attempt is made to browse the current channel. An `ACTIVITYERR` response is returned if no current channel exists. Example 3-9 shows you how to obtain the browse token for browsing a channel.

Example 3-9 Initiating a browse on a channel

```
EXEC CICS STARTBROWSE CONTAINER CHANNEL('ACCOUNT-DATA')
BROWSETOKEN(TOKEN) END-EXEC.
```

GETNEXT container

After receiving the browse token, it is used to move through the list of containers in the channel. Each **GET NEXT CONTAINER** call inserts the name of a container into the field specified on the **CONTAINER** parameter. Each container name is returned only once, and there is no guarantee of the order that they are in. After you have a container name, you can use it with a **get container** command to retrieve the container's contents.

Example 3-10 shows a **GET NEXT CONTAINER** call using the browse token. You do not need to specify the channel name on the command, because the browse token signifies the particular channel that you are browsing. When all container names have been returned, an `END` response condition is given.

Example 3-10 Getting the next container name in a channel

```
EXEC CICS GETNEXT CONTAINER(DATA-AREA) BROWSETOKEN(TOKEN) RESP(WS-RESP)
END-EXEC.
```

ENDBROWSE container

To signify that the browse has finished, the browse token is passed to an **ENDBROWSE CONTAINER** call, as the following example shows:

```
EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(TOKEN) END-EXEC.
```

Combining all of the previous commands together, and wrapping the **GETNEXT CONTAINER** call in a loop, iterates all of the container names. The result gives you something like that shown in Example 3-11.

Example 3-11 Browsing the containers in a channel

```
EXEC CICS STARTBROWSE CONTAINER CHANNEL('ACCOUNT-DATA')
BROWSETOKEN(TOKEN) END-EXEC.

PERFORM WITH TEST AFTER
  UNTIL WS-RESP NOT EQUAL DFHRESP(NORMAL)

  EXEC CICS GETNEXT CONTAINER(DATA-AREA) BROWSETOKEN(TOKEN)
RESP(WS-RESP) END-EXEC

  DISPLAY 'Container name: ' DATA-AREA

END-PERFORM.

EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(TOKEN) END-EXEC.
```

3.1.7 Deleting a container

You can use the **`delete container`** command to permanently remove a container from a channel. When you delete a container, its storage is released and its name is removed from the list of containers that the channel holds. There is absolutely no way to restore a container after you delete it. It is impossible to delete a channel, because this is done implicitly whenever it goes out of program scope. Example 3-12 shows a container being deleted from a channel.

Example 3-12 Deleting a container from a channel

```
EXEC CICS DELETE CONTAINER('ACCOUNT-NO') CHANNEL('ACCOUNT-DATA')
END-EXEC.
```

Remember: There is no requirement to delete a container when it returns from a remote program that it has been linked to, because an internal optimization facilitates only the particular data that changes in the return call.

3.1.8 Moving containers between channels

You can use the **`move container`** command to pass containers between channels. Moving a container from one channel to another is destructive, because the original container no longer exists. If you attempt to move a container within the same channel, it effectively renames the container.

The following list describes some key points when using **`move container`**:

- If the source container does not exist, an error occurs.
- If the target channel does not exist, it is created.
- If the target container does not exist, it is created. If it does exist, it is deleted. This enables a container of a different type to be replaced.
- If you attempt to overwrite a container with itself, nothing happens, and no error condition is raised.

When linking to a program, we suggest that you pass only the necessary containers that the program requires in a channel, so that you can minimize the amount of data being transferred. Example 3-13 shows a container moving to a temporary channel that you can later use for the program link. Note that as part of the move, the container is being renamed to *ACC-NO*.

Example 3-13 Moving a container from one channel to another

```
EXEC CICS MOVE CONTAINER('ACCOUNT-NO') AS('ACC-NO')
CHANNEL('ACCOUNT-DATA') TOCHANNEL('TMP-CHANNEL') END-EXEC.
```

Fast path: You can use **`move container`**, rather than **`get container`** and **`put container`**, as a more efficient way of transferring data between channels.

3.2 CICS Java

You can use the Java based JCICS API to perform all of the functions that have been previously explained using the more traditional style of a channels and containers API.

Information: For detailed information about the JCICS API, see *Java Development using JCICS* in the CICS Transaction Server 5.2 Knowledge Center on the following website:

http://www-01.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.java.doc/topics/dfhpjlp.html?lang=en

As with any JCICS implementation, the interface has been developed in an object-oriented perspective.

Important: When using JCICS, make sure that the `dfjcics.jar` file is in your class path. In addition, add the following import statement to the top of your Java class:

```
import com.ibm.cics.server.*;
```

3.2.1 Creating channels and containers in JCICS

To create a channel, use the **`createChannel()`** method of the `Task` class. The string supplied to the **`createChannel()`** method is the name that CICS uses to recognize the channel. To create a container in the channel, call the **`createContainer()`** method on the channel object.

If a container of the same name already exists in the channel, a `ContainerErrorException` message is displayed. The channel and container names are limited to 16 characters. Example 3-14 shows how to create a channel object, and then use it to create a container object.

Example 3-14 Creating a channel and container

```
Task task = Task.getTask();
Channel myChannel = task.createChannel("ACCOUNT-DATA");
Container myContainer = myChannel.createContainer("ACCOUNT-NO");
```

3.2.2 Placing data in a container

To add data to a container use the **put()** method on the container class. The API supports the addition of data as either a byte array or a string.

Important: When using channels and containers in JCICS, data is *always* inserted into containers as type BIT. It is not possible to create a CHAR container in JCICS. However, they can be received in the current channel.

Example 3-15 shows data insertion into a container.

Example 3-15 Adding data to a container

```
String custNo = "00054321";
byte[] tmpBytes = custNo.getBytes();
myContainer.put(tmpBytes);

// Or alternatively
myContainer.put("00054321");
```

Note the following points based on Example 3-15:

- The **Container.put(String)** method causes an implicit **String.getBytes()** call, which means it performs the same action as an explicit **String.getBytes()** call followed by a **put(byte[])**.
- JCICS always puts data into containers as type BIT, because if it is character data then it becomes a constant and remains the same from when it is inserted. The **String.getBytes()** call encodes the string into a sequence of bytes using the platform's default code page, which in this case is most likely EBCDIC.

3.2.3 Passing a channel to another program or task

The facility to pass a channel to programs or tasks is similar to the process of passing COMMAREAs.

Link and xctl

To pass a channel on a program-link or transfer program control call, **xctl**, use the **link()** and **xctl()** methods of the program class.

Example 3-16 shows a link with a channel.

Example 3-16 Passing a channel on a link()

```
Program program1 = new Program();
program1.setName("PROG1");
program1.link(myChannel);
```

Example 3-17 shows an **`xctl`** with a channel.

Example 3-17 Passing a channel on an xctl()

```
Program program2 = new Program();
program2.setName("PROG2");
program2.xctl(myChannel);
```

Return transid

To set the next channel in a pseudo-conversational program, use the **`setNextChannel()`** method of the `TerminalPrincipalFacility` class. Example 3-18 shows a channel being passed in a pseudo-conversational transaction.

Example 3-18 Setting the next channel on a return transid call

```
TerminalPrincipalFacility terminalPF =
    (TerminalPrincipalFacility) Task.getTask().getPrincipalFacility();
terminalPF.setNextTransaction("ABCD");
terminalPF.setNextChannel(myChannel);
```

Start transid

To pass a channel on a start request, use the **`issue()`** method of the `StartRequest` class. Example 3-19 shows a channel being passed on a start request.

Example 3-19 Passing a channel on a start transid call

```
StartRequest startRequest = new StartRequest();
startRequest.setTransId("ABCD");
startRequest.issue(myChannel);
```

3.2.4 Receiving the current channel

If a program links to the Java program and passes a channel during the link, it becomes the current channel for the program. To instantiate an instance of the current channel, you can use the **getCurrentChannel()** method on the Task class. If no current channel exists, null is returned from the call.

Example 3-20 shows how to reference the current channel.

Example 3-20 Getting an instance of the current channel

```
// Current channel example
Task task = Task.getTask();
Channel currentChannel = task.getCurrentChannel();

if (currentChannel != null) {
   Container myContainer = currentChannel.getContainer("ACCOUNT-NO");
}
else {
   System.out.println("There is no current channel");
}
```

3.2.5 Getting data from a container

To retrieve data from a container, use the **get()** method of the container class. This method always returns its data as a byte array, irrespective of whether it is of character data or binary data. To reinstantiate the data as character data, the byte array can be passed to the constructor of the String class as Example 3-21 shows.

Example 3-21 Getting the data from a container

```
byte[] custInfo = myContainer.get();
String custString = new String(custInfo);
```

Example 3-21 shows how **new String(byte[] bytes)** decodes the byte array data using the system's default code page, which in this case would be EBCDIC. If you know that the data is in a different code page, you must use **new String(byte[] bytes, String charsetName)**, passing the name of that code page.

Note: If the container is of type CHAR, its data is converted to the default code page of the system before being returned to the Java application. So if the container's original code page has characters that the system's default code page does not support, which here would be a version of EBCDIC, character loss can occur during the conversion. A way to prevent this is to pass the data in BIT containers, and instantiate it using **new String(byte[] bytes, String charsetName)**.

3.2.6 Browsing the current channel

A channel that a JCICS program passes can access all of the container objects without receiving the channel explicitly. To do this, it uses a **ContainerIterator** object, which implements the java.util.Iterator interface. The **containerIterator()** method on the Task object returns an Iterator for the current channel, or a null value if there is no current channel. Example 3-22 shows how to browse the current channel.

Example 3-22 Browsing the containers in the current channel

```
Task task = Task.getTask();
ContainerIterator iterator = task.containerIterator();

while (iterator.hasNext()) {
    Container myContainer = (Container) iterator.next();
    // Process the container...
}
```

3.2.7 Browsing a name channel

It is also possible to perform a browse on an instance of a channel. You can perform this function using the **containerIterator()** method on a channel object. The code is almost identical to that of browsing the current channel, Example 3-23 shows how to perform a browse on a channel instance.

Example 3-23 Browsing the containers in a channel instance

```
ContainerIterator iterator = myChannel.containerIterator();

while (iterator.hasNext()) {
    Container myContainer = (Container) iterator.next();
    // Process the container...
}
```

3.2.8 Deleting a container

You can delete a container in either of the following two ways:

- If you have an instance of a container, you can call the **`delete()`** method to delete it.
- Alternatively, you can call the **`deleteContainer()`** method on the channel class.

Example 3-24 shows the process to delete containers using both methods.

Example 3-24 Deleting containers in a channel

```
Container cont1 = myChannel.createContainer("CONT1");
Container cont2 = myChannel.createContainer("CONT2");

// Add some data
cont1.put("Some data");
cont2.put("Some more data");

// Deleting a container using a channel instance
myChannel.deleteContainer("CONT1");

// Deleting a container using its container instance
cont2.delete();
```

Observation: In Example 3-24, no channel or container is created in CICS until you perform the **`put()`** command on the Java container instance. It is the **`put()`** command that causes an **`EXEC CICS PUT CONTAINER`** in CICS to create the channel and container, and then insert the data.

3.2.9 Code page considerations

Both Java and CICS work in two different code pages. Java represents String literals using the Unicode variable length character encoding, Unicode Transformation Format-8 (UTF-8). CICS uses EBCDIC for character encodings. A character in Java is represented by a different code point (a number) internally than that of CICS. Therefore, when passing character data between CICS and Java programs, you need to be aware of these code page considerations.

Because the main input and output type when using containers is the byte array, the easiest way to convert to and from String data is to use utility methods on the String class itself. When using **`String.getBytes()`** and **`new String(byte[])`**, Java encodes and decodes the bytes using the platform's default code page.

In CICS, this is one of the EBCDIC code pages. Therefore, Java handles most of the conversion between UTF-8 and EBCDIC using these two methods. This helps with linking between Java and non-Java technology programs, because the majority of these running in CICS use the EBCDIC code page.

Attention: Putting character data into a container using JCICS, encodes it in the system's local code page and places it in a `BIT` container. Linking to a CICS program in a remote system that runs an alternative code page means that when the data is extracted, it is in the code page of the original system. This is because `BIT` containers are not applicable to code page conversion. Therefore, the character data might appear corrupted in the new application.

A solution to this is if you insert the data into a temporary `CHAR` container in the new program, while specifying the original CCSID, it can be extracted in the local CCSID of that system.

3.3 Business transaction services

The **`put`**, **`get`**, **`move`**, and **`delete container`** commands that you use to build and interact with a channel are similar to those that you use in CICS BTS applications. Therefore, programmers with experience using BTS find it easy to use containers in non-BTS applications. Furthermore, you can call server programs that use containers from both channel and BTS applications.

Figure 3-1 shows the bilingual application scenario that we use to call a standard CICS server program from both a BTS wrapper activity and a non-BTS client.

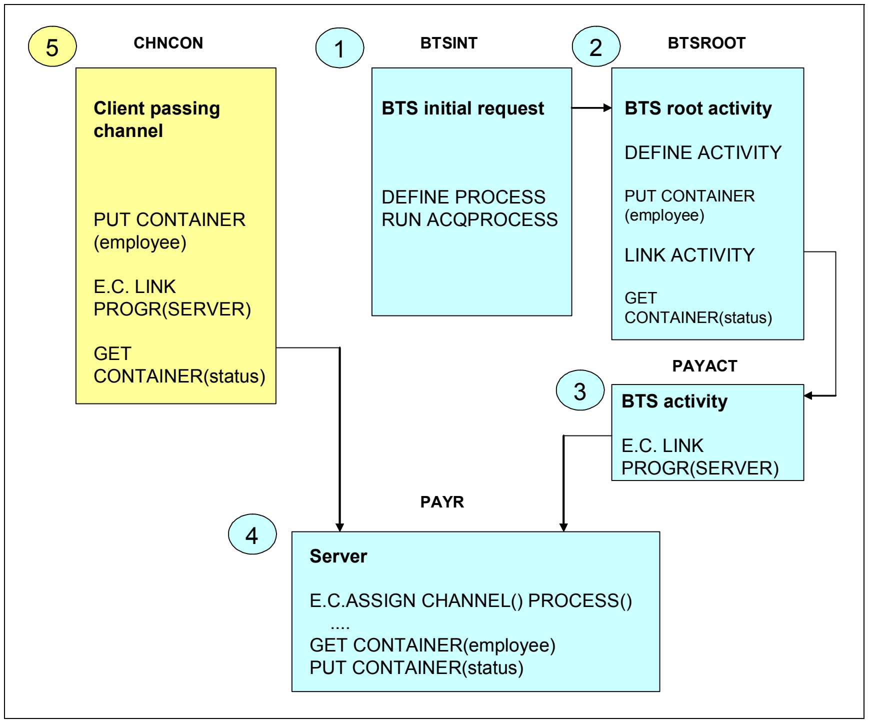


Figure 3-1 Bilingual channel or BTS application environment

This section explains the process of developing a bilingual channel or BTS application. The application uses a BTS activity to link to a standard CICS server program that is using containers. We also develop a non-BTS client program that creates a channel and links to the same server.

3.3.1 Application components

The application shown in Figure 3-1 consists of the components that we describe in the following sections.

Business transaction services initial request

This is the initial request to start the business transaction. The BTSINIT program handles this request. The module shown in Example 3-25 performs two steps:

- It creates the business transaction. To create an instance of the BTSX business transaction, BTSINIT issues a **DEFINE PROCESS** command. The PROGRAM option of **DEFINE PROCESS** defines a program to run under the control of BTS.

  It is the root activity program that typically manages the ordering and execution of the child activities that make up a business transaction. In this case, the program is BTSROOT, which is the root activity program for the BTSX business transaction.
- On return from the **DEFINE PROCESS** command, BTSINIT issues a **RUN ACQPROCESS** command to start an instance of the business transaction.

See the module BTSINIT in Example 3-25.

Example 3-25 Module BTSINIT

```
//*                                                    /*JCTRL*/
//         EXEC PROC=TCKEITAL,OUTC=K,
//         INDEX='CICSTS31.CICS'
//TRN.STEPLIB  DD
//SYSPRINT DD SYSOUT=*
//TRN.SYSIN DD  *
DFHEISTG DSECT
BTSINIT  CSECT
BEGIN    DS    0H
         EXEC CICS DEFINE PROCESS ('INITREQ0002')
X
               PROCESSTYPE('INITREQ')
X
               TRANSID('BTSX')
X
               PROGRAM('BTSROOT')
         EXEC CICS RUN ACQPROCESS ASYNCHRONOUS
RETURN   DS    0H
         EXEC CICS RETURN
         END
//LKED.SYSIN  DD  *
    MODE RMODE(ANY),AMODE(31)
              NAME  BTSINIT(R)
/*
//
```

Business transaction services root activity

The BTSINIT program starts a new instance of the BTSX business transaction by starting the BTSROOT program, running under the transid BTSX. BTSROOT implements a root activity that manages the inter-relationship, ordering, and execution of the child activities that make up the BTSX business transaction.

A root activity program, such as BTSROOT, is designed so that BTS can reattach it, when the running transactions trigger the events that interest it. The activity program determines the possible events that cause BTS to attach it and what to do as a result.

BTSROOT defines one child activity that is later used as a wrapper to the standard CICS server program. When the child activity completes, it triggers an event indicating that the activity is complete. Using the **DEFINE ACTIVITY** command to create the payroll-2004 activity, we specified PAYACT on the PROGRAM option, BTS1 on the TRANSID option, and SERVERFINISHED on the EVENT option.

Therefore, the SERVERFINISHED event triggers the root activity to reattach when the child activity completes. The process we used is explained in the following steps showing the logic flow of BTSROOT:

1. BTSINIT starts the root activity. Issue a **RETRIEVE REATTACH EVENT** command to determine the event.
2. The first event is always DFHINITIAL. If the event is DFHINITIAL, run the code on the initial label. Define the child activity.
3. After this, issue two **put container** commands.
4. Then, issue a **link activity** command that calls the PAYACT program.
5. On return, issue a **get container** command to get the status from the server program.
6. Issue an **EXEC CICS RETURN** command. BTSROOT is sleeping.
7. The SERVERFINISHED event drives the root activity, BTSROOT, again.
8. Issue the **retrieve reattach event** command to determine the event.
9. If the event is SERVERFINSHED, issue a **check activity** command. If the child activity returns successfully, issue an **EXEC CICS RETURN ENDACTIVITY** command.
10. The business transaction ends.

Example 3-26 shows a module BTSROOT and the steps explained previously.

Example 3-26 Module BTSROOT

```
//*                                                    /*JCTRL*/
//         EXEC PROC=TCKEITAL,OUTC=K,
//         INDEX='CICSTS31.CICS'
//TRN.STEPLIB  DD
//SYSPRINT DD SYSOUT=*
//TRN.SYSIN DD  *
DFHEISTG DSECT
STAT     DS    CL2
EVENT    DS    CL16
         DS    CL16
CS       DS    XL4
RESP     DS    XL4
RESP2    DS    XL4
BTSROOT  CSECT
BEGIN    DS    OH
*******************************************************
***  CHECK POSSIBLE EVENTS - START WITH DFHINITIAL  ***
*******************************************************
         EXEC CICS RETRIEVE REATTACH EVENT(EVENT)
         CLC   EVENT(16),=CL16'DFHINITIAL      '
         BNE   NXTEVENT
         BAL   8,INITIAL
         B     RETURN
*******************************************************
***  ROOT DRIVEN BY SERVERFINISHED EVENT?           ***
*******************************************************
NXTEVENT DS    OH
         CLC   EVENT(16),=CL16'SERVERFINISHED  '
         BE    CHKACT
         B     RETURN
*******************************************************
***  RETURN WITH OR WITHOUT ENDACTIVITY OPTION      ***
*******************************************************
RETURN   DS    OH
         EXEC CICS RETURN
RETURN1  DS    OH
         EXEC CICS RETURN ENDACTIVITY
*******************************************************
***  IF SERVERFINISHED EVENT DO CHECK ACTIVITY ...  ***
***  AND RETURN USING ENDACTIVITY OPTION            ***
*******************************************************
```

```
CHKACT   DS    0H
         EXEC CICS CHECK ACTIVITY('PAYROLL-2004')
X
               COMPSTATUS(CS)
X
               RESP(RESP)
X
               RESP2(RESP2)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   RETURN
         B     RETURN1
*******************************************************
***  INITIAL PROCESS - DEFINE AND LINK ACTIVITY     ***
*******************************************************
INITIAL  DS    0H
         EXEC CICS DEFINE ACTIVITY('PAYROLL-2004')
X
               PROGRAM('PAYACT')
X
               TRANSID('BTS1')
X
               EVENT('SERVERFINISHED')
         EXEC CICS PUT CONTAINER('EMPLOYEE')
X
               ACTIVITY('PAYROLL-2004')
X
               FROM('JOHN DOE')
         EXEC CICS PUT CONTAINER('WAGE')
X
               ACTIVITY('PAYROLL-2004')
X
               FROM('100')
         EXEC CICS LINK ACTIVITY('PAYROLL-2004')
         EXEC CICS GET CONTAINER('STATUS')
X
               ACTIVITY('PAYROLL-2004')
X
               INTO(STAT)
         BR    8
**********************************************************************
```

```
         END
//LKED.SYSIN  DD  *
    MODE RMODE(ANY),AMODE(31)
              NAME  BTSROOT(R)
/*
//
```

Business transaction services activity

The PAYACT child activity program issues the **EXEC CICS LINK** command that calls the CICS standard non-BTS server program. Specify PAYR on the program option. Do *not* determine the event that triggered the activity, to keep the sample as short as possible. The event is always DFHINITIAL.

Example 3-27 shows a PAYACT module.

Example 3-27 Module PAYACT

```
//*                                                    /*JCTRL*/
//         EXEC PROC=TCKEITAL,OUTC=K,
//         INDEX='CICSTS31.CICS'
//TRN.STEPLIB  DD
//SYSPRINT DD SYSOUT=*
//TRN.SYSIN DD  *
DFHEISTG DSECT
EVENT    DS    CL16
         DS    CL16
PAYACT   CSECT
BEGIN    DS    OH
         EXEC CICS RETRIEVE REATTACH EVENT(EVENT)
         EXEC CICS LINK PROGRAM('PAYR')
RETURN   DS    OH
         EXEC CICS RETURN
         END
//LKED.SYSIN  DD  *
    MODE RMODE(ANY),AMODE(31)
              NAME  PAYACT(R)
/*
//
```

Server

The PAYR server program is a standard CICS server program using channels and containers. The program can be called from both BTS and non-BTS clients. Therefore, note the following steps:

1. Issue an **assign channel() process()** command to determine if the caller is BTS or non-BTS.
2. After determining the caller identification, set up an informational message and issue a **writeq td** command.
3. Issue a **get container** command followed by the **put container** command to set the status to OK.

Example 3-28 shows the server module PAYR.

Example 3-28 Server module PAYR

```
//*                                                         /*JCTRL*/
//         EXEC PROC=TCKEITAL,OUTC=K,
//         INDEX='CICSTS31.CICS'
//TRN.STEPLIB  DD
//SYSPRINT DD SYSOUT=*
//TRN.SYSIN DD  *
DFHEISTG DSECT
CHN      DS    CL16
PROCESS  DS    CL36
STAT     DS    CL16
EMP      DS    CL16
EMPDAT   DS    CL8
STATDAT  DS    CL2
MSGAREA  DS    CL35
RESP     DS    F
RESP2    DS    F
PAYR     CSECT
BEGIN    DS    OH
         MVC   STAT,=CL16'STATUS          '
         MVC   EMP,=CL16'EMPLOYEE        '
         MVC   STATDAT,=CL2'OK'
         EXEC CICS ASSIGN CHANNEL(CHN) PROCESS(PROCESS)
X
               RESP(RESP) RESP2(RESP2)
         CLC   CHN(7),=CL7'PAYROLL'
         BNE   NEXT
         MVC   MSGAREA(35),=CL35'PAYR WAS CALLED BY NON BTS CLIENT  '
         B     START
NEXT     DS    OH
```

```
         CLC   PROCESS(7),=CL7'INITREQ'
         BNE   RETURN
         MVC   MSGAREA(35),=CL35'PAYR WAS CALLED BY A BTS CLIENT    '
START    DS    OH
         EXEC CICS GET CONTAINER(EMP) INTO(EMPDAT)
         EXEC CICS PUT CONTAINER(STAT) FROM (STATDAT) FLENGTH(STATLEN)
         EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(MSGAREA) LENGTH(LEN)
RETURN   DS    OH
         EXEC CICS RETURN
LEN      DC    H'0035'
STATLEN  DC    F'2'
         END
//LKED.SYSIN  DD  *
    MODE RMODE(ANY),AMODE(31)
              NAME  PAYR(R)
/*
//
```

Client

To create a simple non-BTS client that links to the server passing a channel, with the program name of the client as CHNCON, complete the following steps:

1. Create a channel called PAYROLL.
2. Issue two **put container** commands.
3. Issue an **EXEC CICS LINK PROGRAM(PAYR)** command.
4. Specify PAYROLL on the channel option.
5. On return, issue a **get container** command to get the status.

See Example 3-29, which shows the CHNCON client module.

Example 3-29 Client module CHNCON

```
//*                                                        /*JCTRL*/
//         EXEC PROC=TCKEITAL,OUTC=K,
//         INDEX='CICSTS31.CICS'
//TRN.STEPLIB  DD
//SYSPRINT DD SYSOUT=*
//TRN.SYSIN DD  *
DFHEISTG DSECT
CHN      DS    CL16
EMP      DS    CL16
WGE      DS    CL16
STAT     DS    CL16
STATDAT  DS    CL2
CHNCON   CSECT
```

```
BEGIN    DS    OH
         MVC   CHN,=CL16'PAYROLL         '
         MVC   EMP,=CL16'EMPLOYEE        '
         MVC   WGE,=CL16'WAGE            '
         MVC   STAT,=CL16'STATUS          '
         EXEC CICS PUT CONTAINER(EMP) CHANNEL(CHN) FROM('JOHN DOE')
X
               FLENGTH(LEN)
         EXEC CICS PUT CONTAINER(WGE) CHANNEL(CHN) FROM('100')
X
               FLENGTH(LEN1)
         EXEC CICS LINK PROGRAM('PAYR') CHANNEL(CHN)
         EXEC CICS GET CONTAINER(STAT) CHANNEL(CHN) INTO(STATDAT)
RETURN   DS    OH
         EXEC CICS RETURN
LEN      DC    F'8'
LEN1     DC    F'3'
         END
//LKED.SYSIN  DD  *
    MODE RMODE(ANY),AMODE(31)
              NAME  CHNCON(R)
/*
//
```

3.3.2 Channel and container options

As the sample application previously described shows, you can use a program that issues container commands, without change and as part of a channel application, or as part of a BTS activity.

To use a program in both a channel and a BTS context, the container commands that it issues must not specify any options that identify them as either channel or BTS commands.

You can avoid the following options on each of the container commands:

- **Delete container**
  - ACQACTIVITY (BTS-specific)
  - ACQPROCESS (BTS-specific)
  - ACTIVITY (BTS-specific)
  - CHANNEL (channel-specific)
  - PROCESS (BTS-specific)
- **Get container**
  - ACQACTIVITY (BTS-specific)
  - ACQPROCESS (BTS-specific)
  - ACTIVITY (BTS-specific)
  - CHANNEL (channel-specific)
  - INTOCCSID (channel-specific)
  - PROCESS (BTS-specific)
- **Move container**
  - FROMACTIVITY (BTS-specific)
  - CHANNEL (channel-specific)
  - FROMPROCESS (BTS-specific)
  - TOACTIVITY (BTS-specific)
  - TOCHANNEL (channel-specific)
  - TOPROCESS (BTS-specific)
- **Put container**
  - ACQACTIVITY (BTS-specific)
  - ACQPROCESS (BTS-specific)
  - ACTIVITY (BTS-specific)
  - CHANNEL (channel-specific)
  - DATATYPE (channel-specific)
  - FROMCCSID (channel-specific)
  - PROCESS (BTS-specific)

When you run a container command, CICS analyzes the context (channel, BTS, or neither) in which it occurs, to determine how to process the command. To determine the context, CICS uses the following sequence of tests:

Channel	Does the program have a current channel?
BTS	Is the program part of a BTS activity?
None	The program has no current channel, and is not part of a BTS activity. Therefore, it has no context in which to run container commands. The command is rejected with an INVREQ condition and a RESP2 value of 4.

3.4 Web services

As previously mentioned, one of the reasons channels and containers were introduced was to alleviate the 32 KB limitations when working with web services. Channels and containers compliment core web services interacting through SOAP messages.

The increase in usability with SOAP means that the size of these messages is larger than that of a compact binary message. However, the binary format is almost impossible to read with the human eye. The increase in usability is proportional to the increase in storage use, which means that channels and containers are now an integral part of web services in CICS.

3.4.1 Using channels and containers in CICS web services

You can use channels and containers to transfer SOAP message data to a service provider application. You can also use channels and containers to work with arrays of varying amounts of elements.

Using channels and containers in a service provider application

You can decide to use channels and containers or a COMMAREA in your service provider application during the creation of the WSBind file with the Web Services Assistant. You can give the **`PGMINT`** parameter either of the following two values:

- Channel. CICS uses a channel interface to pass data to the target application.
- COMMAREA. CICS uses a COMMAREA to pass data to the target application.

If CICS uses a channel, you can specify the optional **`CONTID`** parameter of the job control language (JCL) procedure (DFHWS2LS or DFHLS2WS) as the name of the container. In this way, it can hold the top-level data structure that you use to represent the SOAP message.

If you do not specify **`CONTID`**, the container name defaults to DFHWS-DATA. The channel interface supports arrays with varying numbers of elements, which are represented as series of connected data structures in a series of containers. These containers are also present.

Variable arrays of elements

When using DFHWS2LS to generate language structure with array elements, if the array is a fixed length, DFHWS2LS can easily represent this array in the resulting language structure. However, if the number of elements in an array is allowed to vary at run time, this process might be too resource-intensive. In this case, generate an array type in the local system (LS), which is large enough to hold the maximum number of elements that can be received.

What would happen if the Web Services Description Language (WSDL) specifies that the application can receive an unbounded number of elements? In this scenario, channels and containers are an easy and reliable option.

Example 3-30 shows a message that can contain zero - eight string elements.

Example 3-30 Sample WSDL

```
<xsd:element name="component" minOccurs="0" maxOccurs="8">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

In this case, the main data structure does not contain a declaration of an array. Instead, it contains a declaration of two fields:

- `05 component-num PIC S9(9) COMP-4`
- `05 component-cont PIC X(16)`

At run time, the first field (`component-num`) contains the number of times, zero - eight, that the element is displayed in the SOAP message. The second field (`component-cont`) contains the name of a container.

A second data structure contains the declaration of the element itself:

```
01 DFHWS-component
  02 component PIC X(8)
```

Therefore, to process the data structure in your application program, you must examine the value of `component-num`. If it is zero, there is no component element in the message, and the contents of `component-cont` are undefined.

If the value is not zero, the component element is in the container that is named *in* `component-cont`. The container holds an array of elements, and the DFHWS-component data structure maps each of these elements in the container.

We advise that you start considering how you can represent nested arrays. You can see further information about arrays of elements in the CICS Transaction Server version 5.2 Knowledge Center on the following website:

http://www-01.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.applicationprogramming.doc/datamapping/dfhws_variablearrays.html?cp=SSGMCP_5.2.0%2F8-9-0-0-0-4

Channel description document

If your service provider application uses channels and many containers, you should create a channel description document that describes the interface with Extensible Markup Language (XML). The channel description document in a suitable directory on IBM z/OS and CICS uses this document to construct and deconstruct a SOAP message from the containers in the channel. If you use only one container in a channel, you do not need a channel descriptor document.

The schema for the channel description document is called `channel.xsd` and is in the `/usr/lpp/cicsts/cicsts52/schemas/channel` directory (where `/usr/lpp/cicsts/cicsts52` is the default install directory for CICS files).

Procedure

To create a channel descriptor document, use the following procedure:

1. Create an XML document with a `<channel>` element and the CICS channel namespace. Example 3-31 shows a sample `<channel>` element.

   Example 3-31 Sample <channel> element

   ```
   <channel name="myChannel"
   xmlns="http://www.ibm.com/xmlns/prod/CICS/channel">
   </channel>
   ```

2. Add a `<container>` element for every container that the API uses on the channel. You must use `name`, `type`, and `use` attributes to describe each container. Example 3-32 shows six containers with different attribute values. The `<structure>` element indicates that the content is defined in a language structure in a partitioned data set member.

   Example 3-32 Six containers with different attribute values

   ```
   <container name="cont1" type="char" use="required"/>
   <container name="cont2" type="char" use="optional"/>
   <container name="cont3" type="bit" use="required"/>
   <container name="cont4" type="bit" use="optional"/>
   <container name="cont5" type="bit" use="required">
       <structure location="//HLQ.PDSNAME(MEMBER)"/>
   </container>
   ```

```
<container name="cont6" type="bit" use="optional">
    <structure location="//HLQ.PDSNAME(MEMBER2)"/>
</container>
```

3. Save the XML document in z/OS UNIX.

Channel schema

Example 3-33 shows the schema format to which the channel description document should conform:

Example 3-33 Sample Channel schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/xmlns/prod/CICS/channel"
   xmlns:tns="http://www.ibm.com/xmlns/prod/CICS/channel"
elementFormDefault="qualified">
   <element name="channel"> 1
      <complexType>
         <sequence>
            <element name="container" maxOccurs="unbounded" "unbounded"
minOccurs="0"> 2
               <complexType>
                  <sequence>
                     <element name="structure" minOccurs="0"> 3
                       <complexType>
                           <attribute name="location" type="string" use="required"/>
                           <attribute name="structure" type="string" use="optional"/>
                       </complexType>
                     </element>
                  </sequence>
                  <attribute name="name" type="tns:name16Type" use="required"/>
                  <attribute name="type" type="tns:typeType" use="required"/>
                  <attribute name="use" type="tns:useType" use="required"/>
               </complexType>
             </element>
          </sequence>
          <attribute name="name" type="tns:name16Type" use="optional" />
       </complexType>
   </element>
  <simpleType name="name16Type">
      <restriction base="string">
         <maxLength value="16"/>
      </restriction>
  </simpleType>
  <simpleType name="typeType">
```

```
      <restriction base="string">
        <enumeration value="char"/>
        <enumeration value="bit"/>
      </restriction>
  </simpleType>
  <simpleType name="useType">
     <restriction base="string">
        <enumeration value="required"/>
        <enumeration value="optional"/>
     </restriction>
  </simpleType>
</schema>
```

In Example 3-33 on page 96, the numbered entries describe the following elements:

1. This element represents a CICS channel.
2. This element represents a CICS container with the channel.
3. A structure can only be used with `bit` mode containers. The `location` attribute indicates the location of a file that maps the contents of the container. The `structure` attribute can be used in C and C++ to indicate the name of the structure.

Creating mappings and WSDL document

Run DFHLS2WS to create the mappings and WSDL document for the Web Service Provider application. DFHLS2WS puts the mappings for the channel in the WSDL document in the order that the containers are specified in the channel description document.

3.5 CICS-WebSphere MQ bridge

An application that is not CICS can communicate with a CICS program or transaction by sending and receiving IBM WebSphere MQ messages over the CICS-WebSphere MQ bridge. The data required by the CICS application is included in request messages, and the CICS-WebSphere MQ bridge uses reply messages to return the data provided by the CICS application.

CICS programs that are called using the **`EXEC CICS LINK`** command, known as distributed program link (DPL) programs, have traditionally accepted input from the CICS-WebSphere MQ bridge process in a COMMAREA. However, now the CICS-WebSphere MQ bridge also supports data being passed to the DPL program using a channel with request and reply containers.

3.5.1 Channels and containers and the CICS-WebSphere MQ bridge

You can use channels and containers to transfer data to and from programs started by the CICS-WebSphere MQ bridge, by ensuring that your DPL message structure is set up to do so.

DPL message structure for CICS-WebSphere MQ bridge

If you want to pass data through channels and containers to your DPL program, the program should be set up to receive a container named DFHREQUEST, and to place output data in a container called DFHRESPONSE.

Example 3-34 shows the message structure to use when your non-CICS application runs one or more DPL programs in a unit of work (UOW), sends DFHREQUEST container data, and receives DFHRESPONSE container data. `ProgName` is the name of the DPL program.

Example 3-34 Message structure

```
············································
· MQMD · MQCIH · ProgName · DPL program data ·
············································
```

MQCIH fields for channels and containers

The WebSphere MQ Bridge transaction that processes channels and containers input and output data is called `CC`, and it runs the `DFHMQBR3` program. If you want to use this transaction, there are certain fields in the WebSphere MQ Control Information Header (CIH) you must be aware of:

MQCIH.TransactionId	Specifies the transaction code that you want to run the CICS DPL bridge program under. In this case, specify `CC`.
MQCIH.OutputDataLength	This field sets the length of data returned by the program, and is ignored for DPL requests that use the channel and container interface. For these requests, the output (the response) length is the size of the DFHRESPONSE container
MQCIH.ReplyToFormat	This field specifies the CCSID and encoding format of the data returned, and is ignored for DPL requests that use the channel and container interface. For these requests, the reply-to format is set based on the content of the DFHRESPONSE container. If the content is character data, the reply-to format is `MQFMT_STRING`. If the content is binary data, the reply-to format is `MQFMT_NONE`.

4

Systems management and configuration

This chapter provides information about the systems management and configuration tasks and techniques that you must consider when implementing a channels and containers solution.

Although these are tasks that you would normally expect the systems programmer to undertake, you can see that, in a channels and containers implementation, some of these tasks are no longer the responsibility of the systems programmer role.

Because successful systems management requires a well-tuned software implementation, this chapter also looks at the monitoring groups and statistics data related to channels and containers. It also describes the options available to the systems programmer for problem determination.

4.1 Storage

When the application programmer creates a container, the IBM Customer Information Control System (CICS) does not enforce any size limitation regarding how large the container can be. The only upper limit for size is the amount of available user storage in the CICS address space.

CICS deletes channels when they are no longer in scope, but you should still encourage the application programmer to limit the amount of storage that they use for a channel at any given time. For example, you must use the **`delete container`** command whenever it is appropriate, such as after calling a program with the container as input.

Also, to ensure that multiple copies of the same data are not processed, encourage the use of the **`EXEC CICS MOVE CONTAINER`** command to move data from one channel to another.

4.2 The DFHROUTE container

If you use a channels and containers solution in an application that employs dynamic transaction routing, the DFHDYPDS communication area (COMMAREA) still passes to the dynamic routing program, or your version of it. This is an important change made to the DFHDYPDS to remove what was previously a restriction.

How the routing program uses DFHROUTE

When you port a dynamically-routed **`EXEC CICS LINK`** or **`EXEC CICS START`** command to use a channel rather than a COMMAREA, the `DFHDYP` routing program is passed the name of the channel in the `DYRCHANL` field of DFHDYPDS.

Because it is the name of the channel passed, rather than the name of the channel address, if the routing program attempts to use the `DYRCHANL` as a parameter on an **`EXEC CICS GET CONTAINER CHANNEL`** command, it fails with an `INVREQ` message appearing.

The routing program cannot inspect or modify the contents of the channel's containers that are passed to it. To circumvent this problem, and to ensure that the routing program has the same kind of functionality with channels that it had with COMMAREAs, a special container named DFHROUTE is available to the application programmer in the channel to be passed to the routing program.

This container can hold the data that the routing program requires to access the channel.

When the application issues a link or terminal-related start (but not a non-terminal related start request) to be dynamically routed, the dynamic routing program is passed the address of the DFHROUTE container in the `DYRACMAA` field of DFHDYPDS. Therefore, it can now inspect and change its contents if required.

If you are porting a program to pass a channel rather than a COMMAREA, consider using the program's existing COMMAREA structure to map DFHROUTE. This is especially important if you rely on the contents of all or part of your COMMAREA to influence the decision that the dynamic routing program makes regarding where to route work.

Note: If you are using IBM CICSPlex System Manager and the `EYU9XLOP` routing program, you can use the DFHROUTE container in the user-replaceable `EYU9WRAM` module, using the same process described previously in this chapter.

4.3 Code page conversion

Traditionally, it was the responsibility of the systems programmer to handle the code page conversion using the `DFHCNV` conversion table, the `DFHCCNV` conversion program, and the `DFHUCNV` user-replaceable conversion program, which is an optional choice. The systems programmer handled data conversion when it was required, because the process was complex.

For example, an IBM CICS Transaction Server program, using Extended Binary Coded Decimal Interchange Code (EBCDIC) encoding is interacting through a COMMAREA with a Java program using Unicode Transformation Format-8 (UTF-8) encoding. However, when a channels and containers solution replaces the COMMAREA, the task of data conversion simplifies such that the application programmer can perform the operation as part of the application code.

4.3.1 Simple code page conversion

If you use a combination of the **`EXEC CICS PUT CONTAINER`** command with the **`DATATYPE`** parameter, the **`FROMCCSID`** parameter, or both parameters, and the **`EXEC CICS GET CONTAINER`** command with the **`INTOCCSID`** parameter, the application programmer can now perform a simple code page conversion. This is described in more detail in Chapter 2, "Application design and implementation" on page 27.

However, there is still a requirement for the systems programmer to ensure that the CICS region is using the correct coded character set identifier (CCSID). By default this is CCSID 37. If you need to change this, you can specify the relevant code page in the `LOCALCSSID SIT` parameter, as the following example shows:

```
LOCALCSSID={037|CCSID}
```

4.3.2 z/OS Unicode conversion services

Code page conversion is performed using IBM z/OS Unicode conversion services.

Important: You must activate support for Unicode before you start CICS.

Two messages are issued resulting from failures in starting the z/OS conversion services:

- The following console message is issued during CICS initialization to indicate that this CICS region does not support Unicode conversion, because the services are not enabled:

  ```
  DFHAP0801I applid z/OS Conversion Services are not available
  ```

- The following message is issued to report that this system does not support a particular conversion between two specific CCSIDs:

  ```
  DFHAP0802 applid Data conversion using CCSID ccsid and CCSID ccsid
  is not supported by this system
  ```

  Note: If you attempt to use a code page in CICS, which Unicode supports but has not yet enabled, z/OS still attempts to enable or install that conversion, anyway.

  It is not necessary for the system programmers to configure all possible pairs of CCSIDs that application programs require. CICS uses its internal tables for pairs of CCSIDs that the DFHCNV mechanism currently supports.

4.4 Performance considerations

There are several performance-related options during the design phase of a channels and containers implementation that you must encourage the application programmer to consider.

4.4.1 Configuration

There are two aspects of configuration that are described in the following sections:

- ▸ Data separation
- ▸ Error data handling

Data separation

If you are converting an existing application, it might look like an obvious choice to use a channel with a single container to replace your current COMMAREA. However, from a performance perspective, it might be better to separate the input and output fields in your COMMAREA into different containers. The main performance benefit of doing this is that the amount of unnecessary data transmission is significantly reduced.

For example, at the end of a distributed program link (DPL) call, input containers whose contents the server program has not changed do not return to the client. Alternatively, input containers whose contents the server program has changed, and also those containers that the server program creates, are returned to the client.

To demonstrate the previous point, we present a scenario.

Assume that an existing COMMAREA-based application, in which a doctor collates a large amount of data in the form of a patient's medical history, passes on to a server program. The server program provides, by return, a small amount of data, which is the patient's current diagnosis.

Figure 4-1 on page 104 shows that the COMMAREA consists of one contiguous block of data containing three separate fields for the patient, which includes personal details, medical history, and current diagnosis. The doctor only wants to receive the current diagnosis back from the server. However, the server *also* retransmits the medical history field, which could potentially be very large, back to the doctor, even though it has probably not been altered.

Figure 4-1 illustrates this simple COMMAREA scenario.

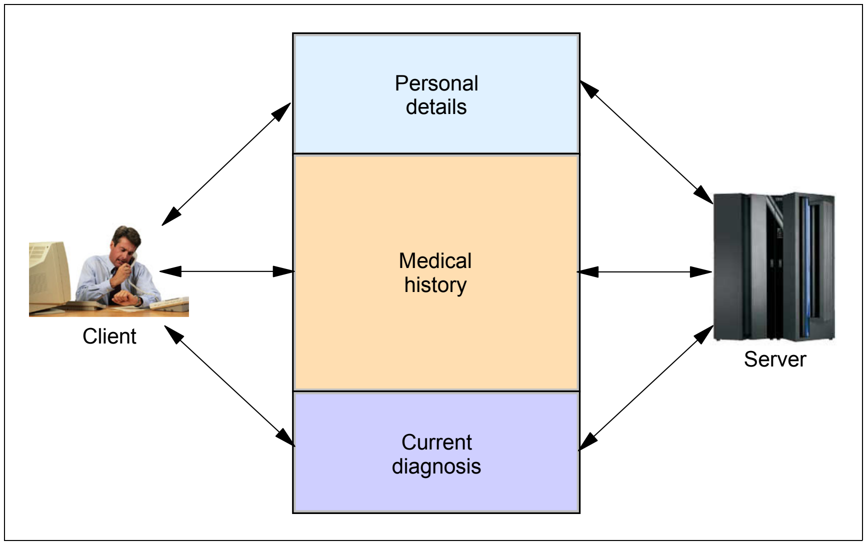


Figure 4-1 Simple COMMAREA scenario

Figure 4-2 on page 105 shows the same scenario, but with separate containers for the personal details, medical history, and current diagnosis data fields.

In this case, the doctor passes the personal details and medical history containers, PATIENT_DETAILS and MEDICAL_HISTORY, to the diagnosis server program and receives the patient diagnosis data back in another separate container, CURR_DIAGNOSIS.

Note that because the data in the PATIENT_DETAILS container and the MEDICAL_HISTORY container were not modified on the call, the server does not return these data. All that the doctor receives back is the CURR_DIAGNOSIS container in which the diagnosis data is held.

See Figure 4-2 that illustrates this simple containers scenario.

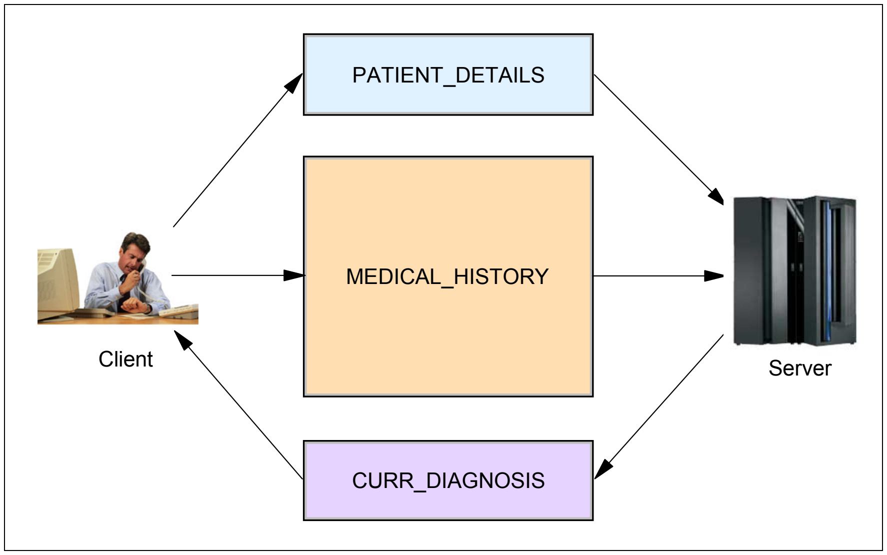


Figure 4-2 Simple containers scenario

Error data handling

Channels and containers offer a different option for handling error data in an application. Using a single, dedicated container for error information can lead to improved performance, and can also simplify application logic.

If you include an **`EXEC CICS GET CONTAINER`** command for the error container in your code, a `NOTFOUND` condition would indicate no error. Therefore, if an error is found, you could interrogate the container for the error information.

Additionally, the application benefits from improved efficiency between CICS regions, because the server only needs to transmit the error container when an error occurs. We list here the monitoring groups and statistics domain entries for channels and containers support.

4.5 Monitoring and statistics

This section describes monitoring and statistics activities.

4.5.1 Monitoring groups

There are three monitoring groups that contain entries relating to channels and containers. The following sections describe the relevant entries in those groups.

DFHCHNL

DFHCHNL is a monitoring group that contains monitor data for channel and container usage. Table 4-1 shows the fields in the DFHCHNL group.

Table 4-1 DFHCHNL group entries

Field name	Field ID	Description
PGTOTCCT	321	Total number of CICS requests for channel containers for the task
PGBRWCCT	322	Number of browse requests for channel containers for the task
PGGETCT	323	Number of **GET CONTAINER** and **GET64 CONTAINER** requests for the task
PGPUTCT	324	Number of **PUT CONTAINER** and **PUT64 CONTAINER** requests for the task
PGMOVCT	325	Number of **MOVE CONTAINER** requests for the task
PGGETCDL	326	Total length, in bytes, of all of the **GET CONTAINER** and **GET64 CONTAINER** data returned
PGPUTCDL	327	Total length, in bytes, of all of the **PUT CONTAINER** and **PUT64 CONTAINER** data returned
PGCRECCT	328	Number of containers created by **MOVE**, **PUT**, and **PUT64 CONTAINER** requests by the task
PGCSTHWM	329	Maximum amount, in bytes, of container storage allocated to the task

DFHPROG

Table 4-2 shows the channels and containers fields in the DFHPROG group.

Table 4-2 DFHPROG group channels and containers entries

Field name	Field ID	Description
PCDLCSDL	286	Total length, in bytes, of the container data for a DPL
PCDLCRDL	287	Total length, in bytes, of all of the container data returned from a DPL
PCLNKCCT	306	Number of **LINK** and **START APPLICATION** requests issued with the `CHANNEL` option for this task
PCXCLCCT	307	Number of **XCTL** requests issued with the `CHANNEL` option for this task
PCDPLCCT	308	Number of DPL requests issued with the `CHANNEL` option by the user task
PCRTNCCT	309	Number of **RETURN** requests issued with the `CHANNEL` option for this task
PCRTNCDL	310	Total length, in bytes, of the container data return

DFHTASK

Table 4-3 shows the channels and containers fields in the DFHTASK group.

Table 4-3 DFHTASK group channels and containers entries

Field name	Field ID	Description
ICSTACCT	65	Number of start requests issued with the channel option
ICSTACDL	345	Length of the data in the containers of all the locally run start channel requests
ICSTRCCT	346	Number of interval control start channel requests to be run on remote systems
ICSTRCDL	347	Total length of the data in the containers of all the remotely run start channel requests

4.5.2 Statistics domain

There are several fields in the intersystem communication (ISC) or interregion communication (IRC) system entries of the statistics report that relate to channels and containers:

- Terminal sharing
  - Number of terminal-sharing channel requests
  - Number of bytes sent on terminal-sharing channel requests
  - Number of bytes received on terminal-sharing channel requests

- Program control
  - Number of program control link requests, with channels, for function shipping
  - Number of bytes sent on link channel requests
  - Number of bytes received on link channel requests
- Interval control
  - Number of interval control start requests, with channels, for function shipping
  - Number of bytes sent on start channel requests
  - Number of bytes received on start channel requests

Figure 4-3 shows a sample statistic report produced for channels and containers usage.

```
Transaction Routing Requests               Max Queue Time - Allocate Purge ....: 0

Transaction Routing - Total...:     0      Allocates Purged - Max Queue Time ..: 0
 Transaction Routing - Channel.:     0
                                           Allocates Rejected - XZIQUE ........: 0
Function Shipping Requests                 XZIQUE - Allocate Purge ............: 0

  File Control................:     0      Allocates Purged - XZIQUE ..........: 0
 Interval Control - Total......:     0
  Interval Control - Channel..:     0
 Transient Data..............:     0
 Temporary Storage ...........:    0
 Program Control - Total .....:     0
  Program Control - Channel..:     0

Total ......................:     0
Bytes Sent by Transaction Routing requests:0Average Bytes Sent by Routing request:0
Bytes Received by Transaction Routing requests:0
Bytes Sent by Program Channel requests:0    Average Bytes Sent by Channel request:0
Bytes Received by Program Channel requests:0
Bytes Sent by Interval Channel requests:0   Average Bytes Sent by Channel request:0
Bytes Received by Interval Channel requests:0
```

Figure 4-3 Statistics output

4.6 Problem determination

This section details the various abnormal end of task (abend) codes and trace points associated with channels and containers to help with problem determination.

4.6.1 Channels and containers abend codes

There is a set of abend codes, which relate to CICS regions in which channels and containers have been implemented. These are depicted in the following sections.

AEYF

An AEYF abend occurs when an invalid storage area passes to CICS on a **put container**, **put64 container**, **get container**, or **get64 container** command. The error can occur when one of the following events takes place:

- Either the FROM or INTO address is specified incorrectly.
- The **FLENGTH** value specifies a value large enough to cause the area to include storage that the transaction cannot access.

A common cause of this error is specifying the address of a halfword area in the **FLENGTH** parameter, which expects a fullword area. This error can arise when a program that previously used COMMAREAs, which have halfword lengths, has been modified to use containers, which have fullword lengths.

AITI

An AITI abend occurs when a mirror transaction that is processing a **start channel** or a **link channel** request, fails while trying to receive data from, or send data to, a connected CICS system. Because a channel can include a considerable amount of data, it might require many calls to terminal control to transmit channel data.

DFHMIRS calls the DFHAPCR program to perform all of the inter-system transmission of channel data. Terminal control has detected an error in one of these calls. The error could be a read timeout, or a more serious error in the flows, that prevented CICS from correctly processing the data.

AXGA

An AXGA abend occurs when the `DFHAPCR` program returns an unexpected response on a function shipping request.

`DFHAPCR` performs the following functions:

- Extracts the contents of all of the containers making up a channel, and transmits them to a remote system.
- Re-creates the channel and containers from inbound data received from a remote system.

AXTS

An AXTS abend indicates that an attempt was made to pass channel and container data between the transactions in a pseudo-conversation, but the next transaction in the pseudo-conversation is in a CICS region that does not support channels and containers.

AXTU

An AXTU abend occurs when the `DFHAPCR` program returns an unexpected response. `DFHAPCR` performs the following functions:

- Extracts the contents of all containers making up a channel and transmits them to a remote system.
- Re-creates the channel and containers from inbound data received from a remote system.

DFHAPCR has either detected an error in inbound data or has received an

4.6.2 Channels and containers trace entries

There are several trace entries pertaining to channels and containers. These are shown in Table 4-4.

Table 4-4 Container data transformation trace points

Point ID	Module	Lvl	Type	Data
AP 4E00	DFHAPCR	AP 1	Entry	1 APCR parameter list
AP 4E01	DFHACPR	AP 1	Exit	1 APCR parameter list
AP 4E02	DFHACPR	AP Exc	Invalid format	1 APCR parameter list

Point ID	Module	Lvl	Type	Data
AP 4E03	DFHACPR	AP Exc	Invalid function	1 ACPR parameter list
AP 4E04	DFHACPR	AP Exc	Recovery	1 APCR parameter list 2 Kernel error data
AP 4E05	DFHAPCR	AP Exc	Delete container failed	1 APCR parameter list 2 PGCR parameter list
AP 4E06	DFHACPR	AP Exc	Put container failed	1 ACPR parameter list 2 PGCR parameter list
AP 4E07	DFHACPR	AP 2	Receive terminal input/output area (TIOA)	1 TIOA 2 03307400
AP 4E08	DFHACPR	AP Exc	Create channel failed	1 APCR parameter list 2 PGCR parameter list 3 03307900
AP 4E09	DFHACPR	AP Exc	No room for channel header	1 APCR parameter list 2 03308400
AP 4E0A	DFHACPR	AP Exc	Getmain failed	1 ACPR parameter list 2 SMGF parameter list 3 03308900

Point ID	Module	Lvl	Type	Data
AP 4E0B	DFHACPR	AP Exc	DFHtc error	1 Response 2 Abend code 3 Sense code 4 03309400
AP 4E0C	DFHACPR	AP Exc	Extract total length	1 ACPR parameter list 2 Buffer left 3 03309900
AP 4E0D	DFHACPR	AP Exc	Extract channel header	1 APCR parameter list 2 Buffer left 3 03310900
AP 4E0E	DFHACPR	AP Exc	Extract container header	1 APCR parameter list 2 Buffer left 3 03310900
AP 4E0F	DFHACPR	AP Exc	Premature end of data	1 APCR parameter list 2 Buffer left 3 03311400
AP 4E10	DFHAPCR	AP Exc	More data expected	1 APCR parameter list 2 Buffer left 3 03311900

Point ID	Module	Lvl	Type	Data
AP 4E11	DFHACPR	AP Exc	Extract container length	1 APCR parameter list 2 Buffer left 3 03312100
AP 4E12	DFHACPR	AP Exc	Bad channel eye-catcher	1 APCR parameter list 2 Channel header 3 03312200
AP 4E13	DFHACPR	AP Exc	Bad container eye-catcher	1 ACPR parameter list 2 Container header 3 03312300
AP 4E14	DFHACPR	AP 2	Extract channel length	1 Channel length 2 03312400
AP 4E15	DFHACPR	AP 2	Extract channel header	1 Channel header 2 03312500
AP 4E16	DFHACPR	AP 2	Extract container header	1 Container header 2 03312600
AP 4E17	DFHAPCR	AP 2	Extract container length	1 Container length 2 03312800
AP 4E18	DFHACPR	AP Exc	Extract container data	1 Container data 2 03313000

Point ID	Module	Lvl	Type	Data
AP 4E19	DFHACPR	AP Exc	Bad response to domain call	1 APCR parameter list 2 DOMAIN PLIST Parameter list
AP 4E20	DFHACPR	AP 2	Container added	1 Container name 2 Owner 3 03313400
AP 4E21	DFHACPR	AP 2	Container changed	1 Container name 2 Owner 3 iseq 4 cseq 5 origi03313500
AP 4E22	DFHACPR	AP 2	Container deleted	1 Container name 2 Owner 3 03313600

There is also a trace point entry for channels and containers in the AP domain recovery set, as shown in Table 4-5.

Table 4-5 P domain recovery trace point for channels and containers AP domain rEvent

Point ID	Module	Lvl	Type	Data
AP 0785	DFHSRP	Exc	Abend AEYF	1 Application program name 2 Parameter address 3 ARG0

Event processing has some channels and containers-related trace points, which are shown in Table 4-6.

Table 4-6 Event processing trace points for channels and containers

Point ID	Module	Lvl	Type	Data
EP 070A	DFHEPRL	Exc	Create Channel	1 EPRL parameter list 2 EP adapter name 3 PGCH parameter list
EP 070E	DFHEPRL	Exc	Get container failed	1 EPRL parameter list 2 EP adapter name 3 PGCR parameter list
EP 070F	DFHEPRL	Exc	Inquire channel failed	1 EPRL parameter list 2 EP adapter name 3 PGCH parameter list
EP 0A05	DFHEPXM	Exc	Bind channel error	1 XMAC parameter list 2 PGCH parameter list
EP 042F	DFHEPEV	Exc	ADAPTER container put failed	1 EPEV parameter list 2 PGCH or PGCR parameter list
EP 0606	DFHEPAS	Exc	Container error	1 EPAS parameter list 2 PGCR parameter list
EP 0607	DFHEPAS	Exc	Container length error	1 EPAS parameter list 2 PGCR parameter list 3 Container data

4.6.3 Tracing channels and containers applications

As with most complex problems in CICS, a problem arising from using channels and containers is probably best solved by obtaining a dump and inspecting it using Interactive Problem Control System (IPCS).

The channel and containers trace points have been described previously. The trace table in the memory dump is, more often than not, the best place to begin debugging, looking first for the presence of any of the exception entries listed in these tables previously mentioned.

This is usually a good indicator of why any given problem might have occurred, and the trace entries, immediately before the exception entry, can provide vital clues as to what led to the problem occurring in the first place.

Note: Trace points related to channels and containers are part of the standard set of trace points in the application program domain, and you do not need to turn on special tracing in CICS Trace Control Facility (CETR) to obtain them.

4.6.4 Sample application trace flow

The following sections depict the trace output of a sample application that we ran, which includes some of the trace points listed previously.

The environment

We defined a transaction (BIGL), which uses an assembler program called BIGLOC, as shown in Figure 4-4. This program allocates a 10 megabyte (MB) user area containing `x'FF'`, runs a **`put container`** command specifying a channel (CHN), and then starts a second transaction (BIG).

```
PTR1     DSECT
         DS    CL16
DFHEISTG DSECT
CHN      DS    CL16
BIGONE   DS    CL16
BIGLOC   CSECT
BEGIN    DS    OH
        MVC   BIGONE,=CL16'BIGONE         '
        MVC   CHN,=CL16'CHN            '
       EXEC CICS GETMAIN FLENGTH(10000000) INITIMG(FFS) SET(PTR)
        USING PTR1,6
        EXEC CICS PUT CONTAINER(BIGONE) CHANNEL(CHN) FROM(PTR1)
X
              FLENGTH(LEN)
        EXEC CICS START TRANSID('BIG') CHANNEL(CHN)
RETURN   DS    OH
        EXEC CICS RETURN
FFS      DC    XL1'FF'
PTR      EQU   6
LEN      DC    XL4'989680'
          END
```

Figure 4-4 The BIGLOC program

The BIG transaction runs assembler program BIGREM, which runs a **get container** command to retrieve the 10 MB of x'FF' that BIGLOC creates, and then returns. Figure 4-5 provides the source code for the BIGREM program.

```
PTR1     DSECT
         DS    CL16
DFHEISTG DSECT
CHN      DS    CL16
BIGONE   DS    CL16
BIGREM   CSECT
BEGIN    DS    OH
         MVC   BIGONE,=CL16'BIGONE         '
         MVC   CHN,=CL16'CHN            '
        EXEC CICS GETMAIN FLENGTH(10000000) INITIMG(FFS) SET(PTR)
         USING PTR1,6
S        EXEC CICS GET CONTAINER(BIGONE) CHANNEL(CHN) INTO(PTR1)
X
               FLENGTH(LEN)
RETURN   DS    OH
         EXEC CICS RETURN
FFS      DC    XL1'FF'
PTR      EQU   6
LEN      DC    XL4'989680'
          END
```

Figure 4-5   The BIGREM program

This application was started in three different modes:

- Specifying both the transactions, BIGL and BIG, as local. Therefore, they both ran in the same CICS region, SCSCPTA1.
- Specifying the BIG transaction as remote, having the name BIG2, and running on another CICS region (SCSCPAA1) in the same z/OS image (therefore connecting to SCSCPTA1 with a multiregion operation (MRO) link).
- Specifying transaction BIG as remote, having the name BIG2, and running on CICS region SCSCPAA4 in a different IBM MVS™ image (therefore using an ISC LU6.2 connection between the two regions).

The application ran with an 8 MB internal trace active on all CICS regions, and took SNAP dumps after each test.

Transactions running locally

The first test was run inside SSYKZCCQ, where the BIGL transaction (`BIGLOC` program) runs a **`put container`** command and starts the BIG transaction (`BIGREM` program) that runs a **`get container`**. The size of the area passed is 10 MB, and it contains `x'FF'`.

Figure 4-6 shows the first part of the trace output of this test, which produces the following results:

1. After the BIGL transaction was started, with transaction number 00202, the initial link to the `BIGLOC` program completes, trace entry 0001.
2. The `BIGLOC` program requests a 10 MB area (`x'989680'`). Therefore, we run the **`GETMAIN`** function to allocate this request into extended user dynamic storage area (EUDSA) and associate it to the task, as trace entries 0002 and 0003 show.
3. The `BIGLOC` program runs the **`put container`** command to move the data from the EUDSA to the container, trace entry 0004.
4. This leads to a check, to see whether the CHN channel already exists. Because it does not exist, a `CHANNEL_NOT_FOUND` condition is received, as shown in trace entries 0005 and 0006.

```
PG 0901 PGPG  ENTRY - FUNCTION(INITIAL_LINK) PROGRAM_NAME(BIGLOC)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-A036878E TIME-14:25:47.7875687 =0001

SM 0C01 SMMG  ENTRY - FUNCTION(GETMAIN) GET_LENGTH(989680) SUSPEND(YES)
INITIAL_IMAGE(FF) STORAGE_CLASS(USER) CALLER(EXEC)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-207A4BA3 TIME-14:25:48.9199094 =0002

SM 0C02 SMMG  EXIT  - FUNCTION(GETMAIN) RESPONSE(OK) ADDRESS(22100008)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-207A4BA3 TIME-14:25:48.9214815 =0003

AP F801 EIBAM  ENTRY - PUT_CONTAINER
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-80082DEA TIME-14:25:48.9214852 =0004

PG 1700 PGCH ENTRY - FUNCTION(INQUIRE_CHANNEL) CHANNEL_NAME(CHN)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-20727757 TIME-14:25:48.9214868 =0005

PG 1701 PGCH EXIT  - FUNCTION(INQUIRE_CHANNEL) RESPONSE(EXCEPTION)
REASON(CHANNEL_NOT_FOUND) CONTAINER_POOL_TOKEN(00000000)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-20727757 TIME-14:25:48.9214874 =0006
```

Figure 4-6 Inquire channel

Figure 4-7 shows the next couple of steps:

1. The channel is now created with the name CHN, as shown in trace entries 0007 and 0010. It is allocated into the subpool (PGCHCB) in extended CICS DSA (ECDSA).
2. The container pool is created too, as shown in trace entries 0008 and 0009. It is allocated into the PGCPCB subpool in ECDSA.

```
PG 1700 PGCH ENTRY - FUNCTION(CREATE_CHANNEL) CHANNEL_NAME(CHN) LINK_LEVEL(CURRENT)
CURRENT_CHANNEL(NO)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-207275E9 TIME-14:25:48.9214887 =0007

PG 1800 PGCP ENTRY - FUNCTION(CREATE_CONTAINER_POOL) CCSID(25) IMPORTED(NO)
CHANNEL_RELATED(YES)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-9FE8F40C TIME-14:25:48.9214930 =0008

PG 1801 PGCP EXIT  - FUNCTION(CREATE_CONTAINER_POOL) RESPONSE(OK)
POOL_TOKEN(20CC1060)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-9FE8F40C TIME-14:25:48.9214950 =0009

PG 1701 PGCH EXIT  - FUNCTION(CREATE_CHANNEL) RESPONSE(OK) CHANNEL_TOKEN(20CC0070)
CONTAINER_POOL_TOKEN(20CC1060)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-207275E9 TIME-14:25:48.9214951 =0010
```

Figure 4-7 Create channel

Figure 4-8 shows the **`put container`** process:

1. After the setup of the environment for container creation is complete, the **`put container`** command was run, as shown in trace entry 0011.
2. The **`GETMAIN`** function is used for the container, to copy the data from user storage. The **`GETMAIN`** function is used in GSDSA subpool PGCSDB, for a length of 10 MB (`x'989680'`), as shown in trace entry 0012.
3. Now the container is created and filled with data, as shown in trace entries 0013 and 0014.

```
PG 1900 PGCR ENTRY - FUNCTION(PUT_CONTAINER) POOL_TOKEN(20CC1060)
CONTAINER_NAME(BIGONE) CALLER(EXEC) ITEM_DATA(22100008 , 00989680)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-20722277 TIME-14:25:48.9219446 =0011

SM 4201 S2GF  ENTRY - FUNCTION(GETMAIN) SUBPOOL_TOKEN(00000048_40704584 ,
00000000_0000006F) GET_LENGTH(98A000) SUSPEND(YES) REMARK(CSDB) LOCK_POOL(YES)
LMLM_ADDRESS(20C19FD0)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-1FE40937 TIME-14:25:48.9219524 =0012

SM 4202 S2GF  EXIT  - FUNCTION(GETMAIN) RESPONSE(OK) ADDRESS(00000048_41600000)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-1FE40937 TIME-14:25:48.9219596 =0013

PG 1901 PGCR EXIT  - FUNCTION(PUT_CONTAINER) RESPONSE(OK)
CONTAINER_TOKEN_OUT(20B73110) GENERATION_NUMBER(1) INITIAL_GENERATION(1)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-20722277 TIME-14:25:48.9283619 =0014

AP F802 EIBAM  EXIT  - PUT_CONTAINER RESP=0 RESP2=0
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-80082DEA TIME-14:25:48.9283635 =0015
```

Figure 4-8 Put container process

4. The `BIGLOC` program next performs a start of the BIG transaction.
5. This causes an **`inquire_channel`** command to be issued. This time, a positive response is received, because channel CHN now exists.

A copy of the channel, the container pool, and the previously created container is created. Figure 4-9 and Figure 4-10 on page 123 show the process:

1. Trace entries 0016 and 0017 show the **`inquire_channel`** succeeding.
2. The copy channel, container pool, and container process were started, as shown in trace entries 0018 - 0026.
3. Again, the **`GETMAIN`** function is used in GSDSA subpool PGCSDB, for a length of 10 MB (`x'989680'`).
4. Trace entries 0027 - 0029 show the positive end of the process.

```
PG 1700 PGCH ENTRY - FUNCTION(INQUIRE_CHANNEL) CHANNEL_NAME(CHN)
LINK_LEVEL(CURRENT)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-A079914C TIME-14:25:48.9283809 =0016

PG 1701 PGCH EXIT  - FUNCTION(INQUIRE_CHANNEL) RESPONSE(OK) CHANNEL_TOKEN(20CC0070)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-A079914C TIME-14:25:48.9284031 =0017

PG 1700 PGCH ENTRY - FUNCTION(COPY_CHANNEL) CHANNEL_TOKEN(20CC0070)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-A0799228 TIME-14:25:48.9284038 =0018

SM 0301 SMGF  ENTRY - FUNCTION(GETMAIN) SUBPOOL_TOKEN(1FAFD41C , 0000006A)
GET_LENGTH(40) SUSPEND(YES) INITIAL_IMAGE(00) REMARK (CHCB) LOCK_POOL(YES)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8F6A4 TIME-14:25:48.9284046 =0019

SM 0302 SMGF  EXIT  - FUNCTION(GETMAIN) RESPONSE(OK) ADDRESS(20CC0030)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8F6A4 TIME-14:25:48.9284055 =0020

PG 1800 PGCP ENTRY - FUNCTION(COPY_CONTAINER_POOL) POOL_TOKEN(20CC1060)
CHANNEL_RELATED(YES)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8F760 TIME-14:25:48.9284062 =0021

SM 0301 SMGF  ENTRY - FUNCTION(GETMAIN) SUBPOOL_TOKEN(1FAFD4E8 , 0000006B)
SUSPEND(YES) INITIAL_IMAGE(00) REMARK(CPCB) LOCK_POOL
                   (YES)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE93190 TIME-14:25:48.9284069 =0022

SM 0302 SMGF  EXIT  - FUNCTION(GETMAIN) RESPONSE(OK) ADDRESS(20CC1030)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE93190 TIME-14:25:48.9284070 =0023

PG 1900 PGCR ENTRY - FUNCTION(COPY_CONTAINER) CONTAINER_TOKEN(20B73110)
TO_POOL_TOKEN(20CC1030)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE92F68 TIME-14:25:48.9284081 =0024
```

Figure 4-9 Copy channel and container process (1)

Figure 4-10 is a continuation of the previous Figure 4-9 on page 122.

```
SM 4201 S2GF  ENTRY - FUNCTION(GETMAIN) SUBPOOL_TOKEN(00000048_40704584 ,
00000000_0000006F) GET_LENGTH(98A000) SUSPEND(YES) REMARK (CSDB)
LOCK_POOL(YES) LMLM_ADDRESS(20C1B970)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-1FE40937 TIME-14:25:48.9284117 =0025

SM 4202 S2GF  EXIT  - FUNCTION(GETMAIN) RESPONSE(OK) ADDRESS(00000048_41F8A000)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-1FE40937 TIME-14:25:48.9284142 =0026
..................................................................................
PG 1901 PGCR EXIT  - FUNCTION(COPY_CONTAINER) RESPONSE(OK)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE92F68 TIME-14:25:48.9316879 =0027

PG 1801 PGCP EXIT  - FUNCTION(COPY_CONTAINER_POOL) RESPONSE(OK)
COPIED_POOL_TOKEN(20CC1030)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8F760 TIME-14:25:48.9316891 =0028

PG 1701 PGCH EXIT  - FUNCTION(COPY_CHANNEL) RESPONSE(OK)
COPIED_CHANNEL_TOKEN(20CC0030)
TASK-00202 KE_NUM-0042 TCB-QR   /008F8680 RET-A0799228 TIME-14:25:48.9316895 =0029
```

Figure 4-10 Copy channel and container process (2)

Figure 4-11 shows the next stage of the process:

1. We start the BIG transaction locally in the same CICS. Therefore, the `BIGLOC` program issues a return, as trace entry 0030 shows.
2. The delete functions for channel, container pool, and container start, as trace entries 0031 - 0033 show.
3. The owned channel, container pool, and container were freed, other than the copied container.
4. Also, the 10 MB user storage associated to the transaction was freed, as shown in trace entry 0037.

```
AP 00E1 EIP ENTRY RETURN                                    REQ(0004) FIELD-A(20D00A80
.}..) FIELD-B(08000E08 ....) BOUNDARY(0200)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-A2010172 TIME-14:25:48.9325179 =0030

PG 1700 PGCH ENTRY - FUNCTION(DELETE_OWNED_CHANNELS) SCOPE(TRANSACTION)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-9FE3697A TIME-14:25:48.9328367 =0031

PG 1800 PGCP ENTRY - FUNCTION(DELETE_CONTAINER_POOL) POOL_TOKEN(20CC1060)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-9FE8FE42 TIME-14:25:48.9328378 =0032

PG 1900 PGCR ENTRY - FUNCTION(DELETE_CONTAINER) CONTAINER_TOKEN(20B73110)
CALLER(SYSTEM)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-9FE93440 TIME-14:25:48.9328386 =0033
...............................................................................
PG 1901 PGCR EXIT  - FUNCTION(DELETE_CONTAINER) RESPONSE(OK)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-9FE93440 TIME-14:25:48.9328468 =0034

PG 1801 PGCP EXIT  - FUNCTION(DELETE_CONTAINER_POOL) RESPONSE(OK)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-9FE8FE42 TIME-14:25:48.9328486 =0035

PG 1701 PGCH EXIT  - FUNCTION(DELETE_OWNED_CHANNELS) RESPONSE(OK)
TASK-00202 KE_NUM-0042 TCB-L8003/008A02C0 RET-9FE3697A TIME-14:25:48.9328495 =0036

SM 0F0D SMAR  EVENT - Storage_released - USER storage at 22100008
TASK-XM    KE_NUM-0042 TCB-QR   /008F8680 RET-9F7599BC TIME-14:25:48.9359276 =0037
1-0000  22100008
2-0000  11
3-0000  FFFFFFFF FFFFFFFF
```

Figure 4-11 Delete channel and container

The BIGL transaction ended, and control was passed to the BIG transaction, as shown in task number 00204. Figure 4-12 shows the process flow:

1. The transaction environment was set up, and a bind for the channel was performed, as shown in trace entries 0038 - 0039.

   Information: The channel token is the same as the one assigned to the copy of the channel that Figure 4-9 on page 122 shows in trace entry 0020.

2. The channel found was set as the current channel, as shown in trace entries 0040 - 0041.
3. The `BIGREM` program was started, as shown in trace entry 0042.
4. The user storage of 10 MB for the transaction was allocated by the **`GETMAIN`** function, as shown in trace entries 0043 - 0044.

```
PG 1700 PGCH ENTRY - FUNCTION(BIND_CHANNEL) CHANNEL_TOKEN(20CC0030)
TASK-00204 KE_NUM-0040 TCB-QR   /008F8680 RET-A091AFEC TIME-14:25:48.9339104 =0038

PG 1701 PGCH EXIT  - FUNCTION(BIND_CHANNEL) RESPONSE(OK)
TASK-00204 KE_NUM-0040 TCB-QR   /008F8680 RET-A091AFEC TIME-14:25:48.9339110 =0039

PG 1700 PGCH ENTRY - FUNCTION(SET_CURRENT_CHANNEL) CHANNEL_TOKEN(20CC0030)
OWNER(YES)
TASK-00204 KE_NUM-0040 TCB-QR   /008F8680 RET-9FE36A18 TIME-14:25:48.9339239 =0040

PG 1701 PGCH EXIT  - FUNCTION(SET_CURRENT_CHANNEL) RESPONSE(OK)
TASK-00204 KE_NUM-0040 TCB-QR   /008F8680 RET-9FE36A18 TIME-14:25:48.9339246 =0041

AP 1940 APLI  ENTRY - FUNCTION(START_PROGRAM) PROGRAM(BIGREM) CEDF_STATUS(CEDF)
EXECUTION_SET(FULLAPI) ENVIRONMENT_TYPE(EXEC) SYNCONRETURN(NO)
LANGUAGE_BLOCK(20E0D5D4) COMMAREA(00000000 , 00000000) LINK_LEVEL(1)
SYSEIB_REQUEST(NO)
TASK-00204 KE_NUM-0040 TCB-QR   /008F8680 RET-9FE32EB4 TIME-14:25:48.9339258 =0042

SM 0C01 SMMG  ENTRY - FUNCTION(GETMAIN) GET_LENGTH(989680) SUSPEND(YES)
INITIAL_IMAGE(FF) STORAGE_CLASS(USER) CALLER(EXEC)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-207A4BA3 TIME-14:25:48.9342775 =0043

SM 0C02 SMMG  EXIT  - FUNCTION(GETMAIN) RESPONSE(OK) ADDRESS(22B00008)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-207A4BA3 TIME-14:25:48.9358818 =0044
```

Figure 4-12 BIG transaction started locally

5. The **get container** command was run, moving the data from the container to the user storage that trace entries 0045 - 0050 show.

   Notice the item buffer on trace entry 0048, which contains the address and length of the area into which the container was copied. The address and the length are those provided by the **GETMAIN** function in the previous trace entry 0044, which is shown in Figure 4-12 on page 125.

   Figure 4-13 shows the **get container** process.

```
AP F801 EIBAM  ENTRY - GET_CONTAINER
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-80082DEA TIME-14:25:48.9358859 =0045

PG 1700 PGCH ENTRY - FUNCTION(INQUIRE_CHANNEL) CHANNEL_NAME(CHN)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-20727757 TIME-14:25:48.9358869 =0046

PG 1701 PGCH EXIT  - FUNCTION(INQUIRE_CHANNEL) RESPONSE(OK)
CONTAINER_POOL_TOKEN(20CC1030)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-20727757 TIME-14:25:48.9358875 =0047

PG 1900 PGCR ENTRY - FUNCTION(GET_CONTAINER_INTO) POOL_TOKEN(20CC1030)
CONTAINER_NAME(BIGONE) CALLER(EXEC) ITEM_BUFFER(22B00008 , 00000000 , 00989680)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-20727A05 TIME-14:25:48.9363466 =0048

PG 1901 PGCR EXIT  - FUNCTION(GET_CONTAINER_INTO) RESPONSE(OK) USERACCESS(ANY)
DATATYPE(BIT) ITEM_BUFFER(22B00008 , 00989680 , 00989680) GENERATION_NUMBER(1)
INITIAL_GENERATION(1) CONTAINER_CCSID(25)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-20727A05 TIME-14:25:48.9395862 =0049

AP F802 EIBAM  EXIT  - GET_CONTAINER RESP=0 RESP2=0
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-80082DEA TIME-14:25:48.9395877 =0050
```

Figure 4-13 Get container into storage

The process completes. After this, the BIGREM program runs the **return** command, and the copies of the channel, container pool, and container areas are freed. Figure 4-14 on page 127 shows trace entries for this phase, which the following steps explain:

1. The BIGREM program issues the **EXEC CICS RETURN** command, as trace entry 0051 shows.
2. This leads to program control deleting the owned channel, container pool, and container areas, as trace entries 0052 - 0054 show.
3. **FREEMAIN** functions are issued for the subpools owning those areas, as trace entry 0055 shows.

Note: The address of the container area that **FREEMAIN** was used for is the same as Figure 4-10 on page 123 shows. This was the address of the area that the BIGL transaction allocated to copy the container.

4. The BIG transaction completes and the user storage is freed, as trace entry 0059 shows.

```
AP 00E1 EIP ENTRY RETURN                             REQ(0004) FIELD-A(20D10A80
.J..) FIELD-B(08000E08 ....) BOUNDARY(0440)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-A2010332 TIME-14:25:48.9395889 =0051

PG 1700 PGCH ENTRY - FUNCTION(DELETE_OWNED_CHANNELS) SCOPE(TRANSACTION)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-9FE3697A TIME-14:25:48.9395966 =0052

PG 1800 PGCP ENTRY - FUNCTION(DELETE_CONTAINER_POOL) POOL_TOKEN(20CC1030)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-9FE8FE42 TIME-14:25:48.9395971 =0053

PG 1900 PGCR ENTRY - FUNCTION(DELETE_CONTAINER) CONTAINER_TOKEN(20B730A0)
CALLER(SYSTEM)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-9FE93440 TIME-14:25:48.9395976 =0054

SM 4201 S2GF  ENTRY - FUNCTION(FREEMAIN) SUBPOOL_TOKEN(00000048_40704584 ,
00000000_0000006F) ADDRESS(00000048_41F8A000) FREE_LENGTH (98A000) REMARK(CSDB)
LOCK_POOL(YES) LMLM_ADDRESS(20C0ABC0)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-1FE40A7F TIME-14:25:48.9395986 =0055
.................................................................................
PG 1901 PGCR EXIT  - FUNCTION(DELETE_CONTAINER) RESPONSE(OK)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-9FE93440 TIME-14:25:48.9396026 =0056

PG 1801 PGCP EXIT  - FUNCTION(DELETE_CONTAINER_POOL) RESPONSE(OK)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-9FE8FE42 TIME-14:25:48.9396040 =0057

PG 1701 PGCH EXIT  - FUNCTION(DELETE_OWNED_CHANNELS) RESPONSE(OK)
TASK-00204 KE_NUM-0040 TCB-L8002/008A7C88 RET-9FE3697A TIME-14:25:48.9396047 =0058
.................................................................................
SM 0F0D SMAR  EVENT - Storage_released - USER storage at 22B00008
TASK-XM    KE_NUM-0040 TCB-QR   /008F8680 RET-9F7599BC TIME-14:25:48.9403285 =0059
1-0000  22B00008
2-0000  11
3-0000  FFFFFFFF FFFFFFFF
```

Figure 4-14 Ending the transaction: Delete channel

4.6.5 Multiregion operation flow

In this example, we define a transaction named BIG1 in CICS region SSYKZCCQ (SYSA). The related COBOL program is `BIGCN1` performing the same **`EXEC CICS PUT CONTAINER`** command as `BIGLOC` did in the previous section to pass a 10 MB user area to the started transaction (BIG2).

Here, the BIG2 transaction is defined as remote and running on region SSYKZCCR (SYSB). SSYKZCCQ and SSYKZCCR were on the same MVS image and connected by an MRO. The BIG2 transaction starts program `BIGREM`, which performed the **`EXEC CICS GET CONTAINER`** command.

Tracing the terminal-owning region

The initial part of this is the same as described in Chapter 3, “Programming” on page 65:

1. Transaction BIG1 was started and program `BIGCN1` got control. It used **`GETMAIN`** for the 10 MB user area and performed an **`EXEC CICS PUT CONTAINER`** command. See Figure 4-6 on page 119 and Figure 4-7 on page 120.
2. The CHN channel, container pool, and BIGONE container are allocated in the proper subpools. See Figure 4-8 on page 121.
3. The start was issued for transaction BIG2, which caused a copy of the channel, container pool, and container areas to be created in the same subpools. See Figure 4-9 on page 122 and Figure 4-10 on page 123.

The environment to start the BIG transaction is now ready.

In Example 4-1, you can see the following items:

1. The BIG2 transaction is located, as shown in trace entries 0001 - 0004.
2. It is found that it had to run on CICS SYSB, therefore, the ISP CONVERSE is issued, as shown in trace entry 0005.
3. The TCTSE for CICS SYSB is found, as shown in trace entries 0006 - 0009.
4. The transformer 1 program is called, as shown in trace entry 0010.
5. An estimation of the amount of data to send is performed, and the result is `x'9896D4'`, as shown in trace entries 0011 - 0012.

Example 4-1 Transmitting the channel: Part 1

```
XM 0401 XMLD  ENTRY - FUNCTION(LOCATE_AND_LOCK_TRANDEF) TRANSACTION_ID(BIG2)
BUNDLE_PROTECT(YES)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A078FD28 TIME-12:01:33.6650073 =0001
```

```
DD 0301 DDLO  ENTRY - FUNCTION(LOCATE) DIRECTORY_TOKEN(1FB00060) ENTRY_NAME(20C11E4C)
DIRECTORY_NAME(TXD ) NAME(BIG2)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-9F84D44E TIME-12:01:33.6650087 =0002=

DD 0302 DDLO  EXIT  - FUNCTION(LOCATE) RESPONSE(OK) DATA_TOKEN(20EF1C90 , D7000000)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-9F84D44E TIME-12:01:33.6650095 =0003

XM 0402 XMLD  EXIT  - FUNCTION(LOCATE_AND_LOCK_TRANDEF) RESPONSE(OK)
TRANDEF_TOKEN(20EF7810 , 000002D4)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A078FD28 TIME-12:01:33.6650106 =0004

AP 00DF ISP ENTRY CONVERSE                            REQ(0003) FIELD-A(04000000
....) FIELD-B(E2E8E2C2 SYSB)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A079291C TIME-12:01:33.6650196 =0005

AP FD02 ZLOC ENTRY LOCATE ID(SYSB) LOC_REQ ID_LOCAL
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03E6864 TIME-12:01:33.6650206 =0006

AP EA01 TMP EXIT FUNCTION(LOCATE) TABLE(TCTS) KEY(SYSB) ENTRY_ADDRESS(20CB11E0)
RESPONSE(NORMAL)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A048A94A TIME-12:01:33.6650270 =0007

AP FD0B ZISP ENTRY FACILITY_REQ TCTTE(20CB11E0) ALLOCATE FREESYNC FREEREST UNPROT Q
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03E6B14 TIME-12:01:33.6650323 =0008

AP FD8B ZISP EXIT  FACILITY_REQ
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03E6B14 TIME-12:01:33.6653640 =0009

AP D902 XFX   ENTRY - TRANSFORMER_1 PLIST_ADDR(20C11BD4) FUNCTION(1008)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03E6E12 TIME-12:01:33.6653658 =0010

AP 4E00 APCR ENTRY - FUNCTION(ESTIMATE_ALL) CHANNEL_TOKEN(20ECD070)
COMMAND(START_MRO)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A073D310 TIME-12:01:33.6653865 =0011

AP 4E01 APCR EXIT  - FUNCTION(ESTIMATE_ALL) RESPONSE(OK) BYTES_NEEDED(9896D4)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A073D310 TIME-12:01:33.6653905 =0012
```

Now the container data can start to be sent over an MRO link.

Example 4-2 shows the process of sending the buffer of data to SSYKZCCR:

1. The 32 KB buffer is set with the **GETMAIN** function, with a length x'8210'. The **IOAREALEN** parameter cannot alter this length, as shown in trace entries 0013 - 0014.
2. The **export_all** function over an MRO link is started, as shown in trace entry 0015.
3. An attempt to get the whole container into the acquired area is made, as shown in trace entry 0016.
4. The response more_data is received because the container is 10 MB long, as shown in trace entry 0017.
5. Therefore, the first 32 KB buffer is sent over an MRO link, as trace entries 0018 - 0020 show.

This process is repeated using 32 KB buffers until the whole container transmits to CICS SYSB.

Example 4-2 Transmitting the channel: Part 2

```
SM 0C01 SMMG  ENTRY - FUNCTION(GETMAIN) GET_LENGTH(8210) TCTTE_ADDRESS(21005030)
SUSPEND(YES) INITIAL_IMAGE(00) STORAGE_CLASS (TERMINAL)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A073A67C TIME-12:01:33.6654019 =0013

SM 0C02 SMMG  EXIT  - FUNCTION(GETMAIN) RESPONSE(OK) ADDRESS(2108A000)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A073A67C TIME-12:01:33.6654119 =0014

AP 4E00 APCR ENTRY - FUNCTION(EXPORT_ALL) TERMINAL_TOKEN(21005030)
CHANNEL_TOKEN(20ECD070) COMMAND(START_MRO)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A073D97A TIME-12:01:33.6654146 =0015

PG 1900 PGCR ENTRY - FUNCTION(GET_CONTAINER_INTO) CONTAINER_TOKEN(20AFF110)
CALLER(SYSTEM) CONVERT(NO) ITEM_BUFFER(2108A2B3 , 00000000 , 00007D58)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D7552 TIME-12:01:33.6654181 =0016

PG 1901 PGCR EXIT  - FUNCTION(GET_CONTAINER_INTO) RESPONSE(EXCEPTION)
REASON(MORE_DATA) ITEM_BUFFER(2108A2B3 , 00007D58 , 00007D58)
DATA_TOKEN_OUT(20C26080 , 00007D58)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D7552 TIME-12:01:33.6654278 =0017
..................................................................................
AP DD21 ZIS2  EVENT - IRC SWITCH FIRST TO SYSTEM (IYCKZCCR) - RETURN CODE WAS
00000000
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03BE638 TIME-12:01:33.6658764 =0018
```

```
AP DD22 ZIS2  EVENT - IRC OUTBOUND REQUEST HEADER: FMH RQE BB  - PLUS LU62 FMH5,
SEQNUM(928)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03BE638 TIME-12:01:33.6658769 =0019

AP FC01 ZARQ  EVENT MRO/LU6.1 STATE SETTING TO SEND
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D6FAC TIME-12:01:33.6670094 =0020
```

The final part of the transmission phase is shown in Example 4-3:

1. Trace entries 0021 and 0022 show that only 1847 bytes remain. Therefore, this time the **`get_container_into`** received a response `OK` rather than `more_data` as before.
2. The **`export_all`** function is exited, as shown by trace entry 0023. The last buffer is sent, as trace entries 0024 - 0025 show.
3. At the end of the process, transformer 4 got control, and the ISP converse and the EIP start are exited, as shown in trace entries 0026 - 0029.

Example 4-3 Transmitting the channel: Part 3

```
PG 1900 PGCR ENTRY - FUNCTION(GET_CONTAINER_INTO) CALLER(SYSTEM) CONVERT(NO)
ITEM_BUFFER(2108A00C , 00000000 , 00007FFF) DATA_TOKEN_IN(20C26080 , 00987C28)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D7674 TIME-12:01:33.7319224 =0021

PG 1901 PGCR EXIT  - FUNCTION(GET_CONTAINER_INTO) RESPONSE(OK) ITEM_BUFFER(2108A00C ,
00001A58 , 00007FFF) DATA_TOKEN_OUT(00000000 , 00000000)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D7674 TIME-12:01:33.7319254 =0022

AP 4E01 APCR EXIT  - FUNCTION(EXPORT_ALL) RESPONSE(OK) TC_RESPONSE(0) TC_ABEND()
TC_SENSE(00000000)
TASK-00121KE_NUM-0042TCB-QR /008F8680RET-A073D97ATIME-12:01:33.7319276=0023

AP DD21 ZIS2  EVENT - IRC SWITCH SUBSEQUENT TO SYSTEM (IYCKZCCR) - RETURN CODE WAS
00000000
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03BE638 TIME-12:01:33.7319338 =0024

AP DD22 ZIS2  EVENT - IRC OUTBOUND REQUEST HEADER: RQE CD , SEQNUM(1233)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03BE638 TIME-12:01:33.7319340 =0025
..............................................................................
AP D902 XFX   ENTRY - TRANSFORMER_4 PLIST_ADDR(20C11BD4) FUNCTION(1008)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03E72D8 TIME-12:01:33.7496153 =0026

AP D903 XFX   EXIT  - TRANSFORMER_4 PLIST_ADDR(20C11BD4) FUNCTION(1008)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A03E72D8 TIME-12:01:33.7496158 =0027
```

```
AP 00DF ISP EXIT CONVERSE REQ(0005) FIELD-A(04000000 ....) FIELD-B(E2E8E2C2 SYSB)
TASK-00121 KE_NUM-0042 TCB-QR   /008F8680 RET-A079291C TIME-12:01:33.7496254 =0028

AP 00E1 EIP EXIT START OK REQ(00F4) FIELD-A(00000000 ....) FIELD-B(00001008 ....)
BOUNDARY(0200)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-A201015C TIME-12:01:34.3823071 =0029
```

The BIG1 transaction now ends:

1. Example 4-4 shows that after the return and the exit of the start function that trace entries 0030 - 0031 show, the channel and container environment for transaction BIG1 is freed, as the trace entries 0032 - 0041 show.
2. The **FREEMAIN** for PGCSDB is shown in trace entries 0035 - 0036.
3. Finally, the normal termination for transaction BIG1 occurs, and the 10 MB user storage is freed.

Example 4-4 End of transaction on terminal-owning region

```
AP 00E1 EIP ENTRY RETURN REQ(0004) FIELD-A(20D00A80 .}..) FIELD-B(08000E08 ....)
BOUNDARY(0200)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-A2010172 TIME-12:01:34.3823083 =0030

AP 1941 APLI  EXIT  - FUNCTION(START_PROGRAM) RESPONSE(OK) ABEND_CODE()
IGNORE_PENDING_XCTL(NO)  Program_name(BIGCN1)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-9FE32EB4 TIME-12:01:36.9055516 =0031

PG 1700 PGCH ENTRY - FUNCTION(DELETE_OWNED_CHANNELS) SCOPE(TRANSACTION)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-9FE3697A TIME-12:01:36.9057106 =0032

PG 1800 PGCP ENTRY - FUNCTION(DELETE_CONTAINER_POOL) POOL_TOKEN(20EF8030)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-9FE8FE42 TIME-12:01:36.9057120 =0033

PG 1900 PGCR ENTRY - FUNCTION(DELETE_CONTAINER) CONTAINER_TOKEN(20AFF0A0)
CALLER(SYSTEM)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-9FE93440 TIME-12:01:36.9057140 =0034

SM 4201 S2GF  ENTRY - FUNCTION(FREEMAIN) SUBPOOL_TOKEN(00000048_40704584 ,
00000000_0000006F) ADDRESS(00000048_41600000) FREE_LENGTH (98A000) REMARK(CSDB)
LOCK_POOL(YES) LMLM_ADDRESS(20C11BC0)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-1FE40A7F TIME-12:01:36.9057190 =0035

SM 4202 S2GF  EXIT  - FUNCTION(FREEMAIN) RESPONSE(OK)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-1FE40A7F TIME-12:01:36.9057275 =0036
```

```
PG 1901 PGCR EXIT  - FUNCTION(DELETE_CONTAINER) RESPONSE(OK)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-9FE93440 TIME-12:01:36.9057296 =0037

PG 1801 PGCP EXIT  - FUNCTION(DELETE_CONTAINER_POOL) RESPONSE(OK)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-9FE8FE42 TIME-12:01:36.9057313 =0038

PG 1701 PGCH EXIT  - FUNCTION(DELETE_OWNED_CHANNELS) RESPONSE(OK)
TASK-00121 KE_NUM-0042 TCB-L8005/008AAC88 RET-9FE3697A TIME-12:01:36.90573212 =0039
```

Tracing the application-owning region

This section describes the SSYKZCCR region, and explains the processes that occurred, as shown in Example 4-5:

1. Input is received from SSYKZCCQ, trace entries 0040 - 0043. The *seqnum* is 928. Therefore, it is the first buffer sent, as trace entry 0019 in Example 4-2 on page 130 shows.
2. A CICS mirror transaction, number 00083, is attached to manage the receiving of the data, as shown in trace entry 0045.
3. After CICS mirror transaction starts, the **`import_all`** function is performed, as shown in trace entry 0048.

The CICS mirror transaction creates the channel, container pool, and container for the incoming data, as trace entries 0047 - 0049 show.

Example 4-5 CICS mirror transaction creating the channel

```
AP DD18 CRNP  EVENT - DEQUEUE WORK ELEMENT TYPE (INITIAL INPUT RECEIVED) TIMESTAMP
(CDF39635C866430D)  SCCB AT 7F5DDE70 TCTTE AT 21006930 SESSION NAME AR1  SYSTEM
IYCKZCCQ
TASK-00023 KE_NUM-0008 TCB-QR   /008F8680 RET-0008B614 TIME-12:01:33.6659271 =0040

AP FD0D ZIS2 ENTRY IRC TCTTE(21006930) GETDATA
TASK-00023 KE_NUM-0008 TCB-QR   /008F8680 RET-A00F9B98 TIME-12:01:33.6659301 =0041

AP DD20 ZIS2  EVENT - IRC PULL - DATA FROM SYSTEM (IYCKZCCQ) - RETURN CODE WAS
00000000
TASK-00023 KE_NUM-0008 TCB-QR   /008F8680 RET-A00F9B98 TIME-12:01:33.6660216 =0042

AP DD15 CRNP  EVENT - IRC INBOUND REQUEST HEADER: FMH RQE BB , SEQNUM(928)
TASK-00023 KE_NUM-0008 TCB-QR   /008F8680 RET-0008B614 TIME-12:01:33.6660231 =0043

AP FD11 ZATT ENTRY ATTACH ID(AR1 )
TASK-00023 KE_NUM-0008 TCB-QR   /008F8680 RET-A00F92C6 TIME-12:01:33.6660253 =0044
```

```
XM 1101 XMAT  ENTRY - FUNCTION(ATTACH) TRANSACTION_ID(CSMI) EXTERNAL_UOW_ID(1F934DF4
,
0000001B) PRIORITY(0) START_CODE(T) RETURN_NOT_FOUND(NO) USE_DTRTRAN(YES)
PRIMARY_CLIENT_TYPE(MRO_SESSION) PRIMARY_CLIENT_REQ_BLOCK(21006930 , 02130000)
TRANSACTION_GROUP(SAME) WLM_SRC_TOKEN(055D8000) TRANSACTION_GROUP_ID(1F934E78 ,
0000001C) ORIGIN_DATA (21086083 , 000001EC) ADAPTER_FIELDS(00000000 , 00000000)
INITIAL_IS_CURRENT_CTX(NO)
TASK-00023 KE_NUM-0008 TCB-QR   /008F8680 RET-A03B8036 TIME-12:01:33.6660301 =0045
................................................................................
AP 4E00 APCR ENTRY - FUNCTION(IMPORT_ALL) TERMINAL_TOKEN(21006930) DATA_START(253)
COMMAND(START_MRO)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A0735576 TIME-12:01:33.6669420 =0046

PG 1700 PGCH ENTRY - FUNCTION(CREATE_CHANNEL) CHANNEL_NAME(CHN) CCSID(25)
LINK_LEVEL(CURRENT) CURRENT_CHANNEL(NO) IMPORTED(NO)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D6C20 TIME-12:01:33.6669465 =0047

PG 1800 PGCP ENTRY - FUNCTION(CREATE_CONTAINER_POOL) CCSID(25) IMPORTED(NO)
CHANNEL_RELATED(YES)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8F40C TIME-12:01:33.6669514 =0048

PG 1801 PGCP EXIT  - FUNCTION(CREATE_CONTAINER_POOL) RESPONSE(OK)
POOL_TOKEN(20EFF060)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8F40C TIME-12:01:33.6669538 =0049
```

Figure 4-15 on page 135 shows the put container process:

1. A **put container** command is issued to receive the first 32 KB buffer of data, as shown in trace entry 0050.
2. This leads to a **GETMAIN** function in subpool PGCSDB to acquire the container area for the 32 KB buffer of data, as shown in trace entry 0051.
3. After the first 32 KB buffer is received into the container, the **put container** exits, as trace entry 0053 shows.
4. When the next 32 KB buffer of data is available, another **put container** is issued and the cycle repeats, until all the 10 MB of data are put into the container. In this case, it is repeated 132 times, as you can see from the **generation_number** in trace entry 0054.

```
PG 1900 PGCR ENTRY - FUNCTION(PUT_CONTAINER) POOL_TOKEN(20EFF060)
CONTAINER_NAME(BIGONE) CALLER(IMPORTED) TYPE(USER) USERACCESS(ANY)
DATATYPE(BIT) CONVERT(NO) PUT_TYPE(REPLACE) ITEM_DATA(210862B3 ,
00007D58)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D7598 TIME-12:01:33.6669573 =0050

SM 4201 S2GF  ENTRY - FUNCTION(GETMAIN) SUBPOOL_TOKEN(00000048_40704584 ,
00000000_0000006F) GET_LENGTH(8000) SUSPEND(YES) REMARK (CSDB)
LOCK_POOL(YES) LMLM_ADDRESS(20C19840)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-1FE40937 TIME-12:01:33.6669651 =0051

SM 4202 S2GF  EXIT  - FUNCTION(GETMAIN) RESPONSE(OK) ADDRESS(00000048_40C01000)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-1FE40937 TIME-12:01:33.6669680 =0052
.................................................................................
PG 1901 PGCR EXIT  - FUNCTION(PUT_CONTAINER) RESPONSE(OK) GENERATION_NUMBER(1)
INITIAL_GENERATION(1)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D7598 TIME-12:01:33.6669810 =0053
.................................................................................
PG 1901 PGCR EXIT  - FUNCTION(PUT_CONTAINER) RESPONSE(OK) GENERATION_NUMBER(132)
INITIAL_GENERATION(1)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A01D7598 TIME-12:01:33.7319465 =0054
```

Figure 4-15 Put container cycle

Figure 4-16 on page 136 shows the COPY channel process

1. Receiving the complete data into the container ends the **`import_all`** function, as shown in trace entry 0055.
2. The request to start transaction BIG2 is received, as shown in trace entries 0056-0058.
3. This leads to a copy of the channel, container pool, and container data being created. Therefore, the amount of storage allocated is duplicated in the same subpools (PGCPCB, PGCHCB, and PGCSCB), as shown in trace entries 0059-0061.
4. The copy of the channel and container is made available to the BIG2 transaction, as shown in trace entry 0062.
5. After this, it detaches itself from the channel, as trace entry 0063 shows, and deletes the container, as shown in trace entries 0064 - 0067. The subpool storage related to the original container is freed, but not the copied one, because this is passed to BIG2.
6. Finally, the CICS mirror transaction ends and BIG2 gets control, with transaction number 0068.

Figure 4-16 shows the COPY channel process trace entries.

```
AP 4E01 APCR EXIT  - FUNCTION(IMPORT_ALL) RESPONSE(OK) CHANNEL_TOKEN(20EEA070)
CHANNEL_NAME(CHN) TC_RESPONSE(0) TC_ABEND() TC_SENSE (00000000) SIZE(9896D4)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A0735576 TIME-12:01:33.7319513 =0055

AP 00E1 EIP ENTRY START REQ(0004) FIELD-A(20C174E8 .A.Y) FIELD-B(08001008 ....)
BOUNDARY(0200)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A13D5034 TIME-12:01:33.7319582 =0056

PG 1700 PGCH ENTRY - FUNCTION(INQUIRE_CHANNEL) CHANNEL_NAME(CHN) LINK_LEVEL(CURRENT)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A079F64C TIME-12:01:33.7319664 =0057

PG 1701 PGCH EXIT  - FUNCTION(INQUIRE_CHANNEL) RESPONSE(OK) CHANNEL_TOKEN(20EEA070)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A079F64C TIME-12:01:33.7319670 =0058

PG 1700 PGCH ENTRY - FUNCTION(COPY_CHANNEL) CHANNEL_TOKEN(20EEA070)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A079F728 TIME-12:01:33.7319675 =0059

PG 1800 PGCP ENTRY - FUNCTION(COPY_CONTAINER_POOL) POOL_TOKEN(20EFF060)
CHANNEL_RELATED(YES)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8F760 TIME-12:01:33.7319702 =0060

PG 1900 PGCR ENTRY - FUNCTION(COPY_CONTAINER) CONTAIbNER_TOKEN(20AFF110)
TO_POOL_TOKEN(20EFF030)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE92F68 TIME-12:01:33.7322205 =0061
.................................................................................
AP 00E1 EIP EXIT START OK REQ(00F4) FIELD-A(00000000 ....) FIELD-B(00001008 ....)
BOUNDARY(0200)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A13D5034 TIME-12:01:33.7411030 =0062

PG 1700 PGCH ENTRY - FUNCTION(DETACH_CHANNEL) CHANNEL_TOKEN(20EEA070) DELETE(YES)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A07356A6 TIME-12:01:33.7411093 =0063

PG 1800 PGCP ENTRY - FUNCTION(DELETE_CONTAINER_POOL) POOL_TOKEN(20EFF060)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8FA02 TIME-12:01:33.7411101 =0064

PG 1700 PGCH ENTRY - FUNCTION(DETACH_CHANNEL) CHANNEL_TOKEN(20EEA070) DELETE(YES)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-A07356A6 TIME-12:01:33.7411093 =0065

PG 1800 PGCP ENTRY - FUNCTION(DELETE_CONTAINER_POOL) POOL_TOKEN(20EFF060)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE8FA02 TIME-12:01:33.7411101 =0066

PG 1900 PGCR ENTRY - FUNCTION(DELETE_CONTAINER) CONTAINER_TOKEN(20AFF110)
CALLER(SYSTEM)
TASK-00083 KE_NUM-0042 TCB-QR   /008F8680 RET-9FE93440 TIME-12:01:33.7411107 =0067
```

Figure 4-16 Copy the channel

Figure 4-17 shows the start of the get container process:

1. The BIG2 transaction performs the initial link to the `BIGREM` program, as trace entry 0068 shows, and the current channel is set, as shown in trace entries 0069 - 0070.
2. The `BIGREM` program uses the **`GETMAIN`** function to allocate the 10 MB user storage required, as shown in trace entries 0071 - 0072.
3. The **`get_container`** command is issued, as shown in trace number 0073.

```
PG 0901 PGPG  ENTRY - FUNCTION(INITIAL_LINK) PROGRAM_NAME(BIGREM)
TASK-00084 KE_NUM-0043 TCB-QR   /008F8680 RET-9F758DD2 TIME-12:01:33.7513875 =0068

PG 1700 PGCH ENTRY - FUNCTION(SET_CURRENT_CHANNEL) CHANNEL_TOKEN(20EEA030)
OWNER(YES)
TASK-00084 KE_NUM-0043 TCB-QR   /008F8680 RET-9FE36A18 TIME-12:01:33.7518134 =0069

PG 1701 PGCH EXIT  - FUNCTION(SET_CURRENT_CHANNEL) RESPONSE(OK)
TASK-00084 KE_NUM-0043 TCB-QR   /008F8680 RET-9FE36A18 TIME-12:01:33.7518142 =0070

AP E123 EISC  ENTRY - GETMAIN
TASK-00084 KE_NUM-0043 TCB-L8001/008AAC88 RET-80082DEA TIME-12:01:33.7518648 =0071

SM 0C01 SMMG  ENTRY - FUNCTION(GETMAIN) GET_LENGTH(989680) SUSPEND(YES)
INITIAL_IMAGE(FF) STORAGE_CLASS(USER) CALLER(EXEC)
TASK-00084 KE_NUM-0043 TCB-L8001/008AAC88 RET-207C27A3 TIME-12:01:33.7518691 =0072

AP F801 EIBAM  ENTRY - GET_CONTAINER
TASK-00084 KE_NUM-0043 TCB-L8001/008AAC88 RET-80082DEA TIME-12:01:33.7539396 =0073
```

Figure 4-17 Get container start

To conclude the get container process, the following steps are performed as Figure 4-18 shows:

1. An inquiry for channel CHN is performed, resulting in the find of the proper container, as shown in trace entries 0074 - 0075.
2. Container BIGONE is copied into the user storage allocated by the GETMAIN function, as trace entries 0076 - 0078 show.

```
PG 1700 PGCH ENTRY - FUNCTION(INQUIRE_CHANNEL) CHANNEL_NAME(CHN)
TASK-00084 KE_NUM-0043 TCB-L8001/008AAC88 RET-2073D057 TIME-12:01:33.7539412 =0074

PG 1701 PGCH EXIT  - FUNCTION(INQUIRE_CHANNEL) RESPONSE(OK)
CONTAINER_POOL_TOKEN(20EFF030)
TASK-00084 KE_NUM-0043 TCB-L8001/008AAC88 RET-2073D057 TIME-12:01:33.7539417 =0075

PG 1900 PGCR ENTRY - FUNCTION(GET_CONTAINER_INTO) POOL_TOKEN(20EFF030)
CONTAINER_NAME(BIGONE) CALLER(EXEC) ITEM_BUFFER(22100008 , 00000000 , 00989680)
TASK-00084 KE_NUM-0043 TCB-L8001/008AAC88 RET-2073D305 TIME-12:01:33.7544125 =0076

PG 1901 PGCR EXIT  - FUNCTION(GET_CONTAINER_INTO) RESPONSE(OK) USERACCESS(ANY)
DATATYPE(BIT) ITEM_BUFFER(22100008 , 00989680 , 00989680) GENERATION_NUMBER(1)
INITIAL_GENERATION(1) CONTAINER_CCSID(25)
TASK-00084 KE_NUM-0043 TCB-L8001/008AAC88 RET-2073D305 TIME-12:01:33.7588918 =0077

AP F802 EIBAM  EXIT  - GET_CONTAINER RESP=0 RESP2=0
TASK-00084 KE_NUM-0043 TCB-L8001/008AAC88 RET-80082DEA TIME-12:01:33.7588960 =0078
```

Figure 4-18 Get container end

The `BIGREM` program runs an **`EXEC CICS RETURN`** command to end the process. This leads to the remaining copied channel and container storage allocated in the GCDSA subpools being freed. However, the trace entries for this process are not illustrated.

4.6.6 Intersystem communication flow

The trace activated on the ISC connection between CICSPTA1 and CICSPAA4 shows exactly the same behavior as the MRO tracing from a channel and container point of view. However, the communication protocol is different, as are the modules involved. Nevertheless, the management of channel and container storage inside the terminal-owning region (TOR) and application-owning region (AOR) is quite similar to the MRO process described previously.

5

Sample application

This chapter describes the process you can use to port an existing IBM Customer Information Control System (CICS) application to use channels and containers rather than using communication areas (COMMAREAs). To show the porting process, this chapter uses the CICS catalog manager example application.

In this chapter, we also demonstrate how to use the Java CICS (JCICS) channel and container application programming interface (API) commands within a Liberty profile servlet. We show in detail how to use a Liberty profile servlet to access the existing catalog manager business logic using JCICS channel and container commands.

5.1 Implementation scenario

The original CICS catalog example application is designed to demonstrate how to extend an IBM 3270 CICS Common Business Oriented Language (COBOL) application to use web service support. The web service support extension for the base application provides a web client front end and a web service endpoint. Therefore, either the web client front end or the IBM 3270 interface of the catalog manager application can drive the business logic of the example application.

Note: Based on the previously described starting position, we decided to provide a Liberty profile servlet that can also be used to access the catalog business logic. The servlet can use JCICS API services, such as channel, container, and program link commands.

The CICS catalog example program has a modular structure with well-defined interfaces. The `EXEC CICS LINK` command calls these various components using a COMMAREA.

What is the requirement to change the design of such an approved existing CICS application? For an existing CICS application, the COMMAREA is equivalent to the functions that channels and containers provide. If you do not have any COMMAREA constraints, and if you have no plans to extend your application design (which eventually would exceed the COMMAREA limit), there is no reason to port the application to the new function.

COMMAREA-based CICS application programs can be extended or modernized in many ways using the capabilities of web services or CICS Liberty profile servlets. If you use an Extensible Markup Language (XML)-based or servlet-based front end to call your existing business logic, you might have additional requirements that can exceed the limit of your COMMAREAs.

To demonstrate how to port an existing COMMAREA-based program, we use the CICS catalog example application to show the function of channels and containers. This section covers the following topics:

- The CICS catalog manager example application
- The base application
- Porting steps: CICS back end
- Stage 1: Porting to the new function
- Using channels and containers: A first approach
- Best practice approach: Replacing the COMMAREA
- Stage 2: Place an order using a Liberty profile servlet
- Running the application

5.1.1 The CICS catalog manager example application

We use a Liberty profile servlet to demonstrate how to access the back-end CICS Cobol catalog manager using JCICS link with the channels and containers API. The CICS servlet uses the JCICS API to access CICS directly, so we developed the catalog servlet. The catalog servlet demonstrates an easy and efficient way to access the catalog example application.

The catalog manager example application accesses an order catalog that a Virtual Storage Access Method (VSAM) file stores. The example application is a catalog management, purchase-order style application. It is a simple application that provides the functions to list details of an item in the catalog and then selects a quantity of that item to order.

Then, the application updates the catalog to reflect the new stock levels. If the catalog servlet is deployed, it provides the functions to deal with the catalog from a browser session. The following section describes the CICS implementation of this process.

Figure 5-1 shows the application porting environment at the Stage 0 level. There are two stages in performing the port, which this chapter describes in its latter half.

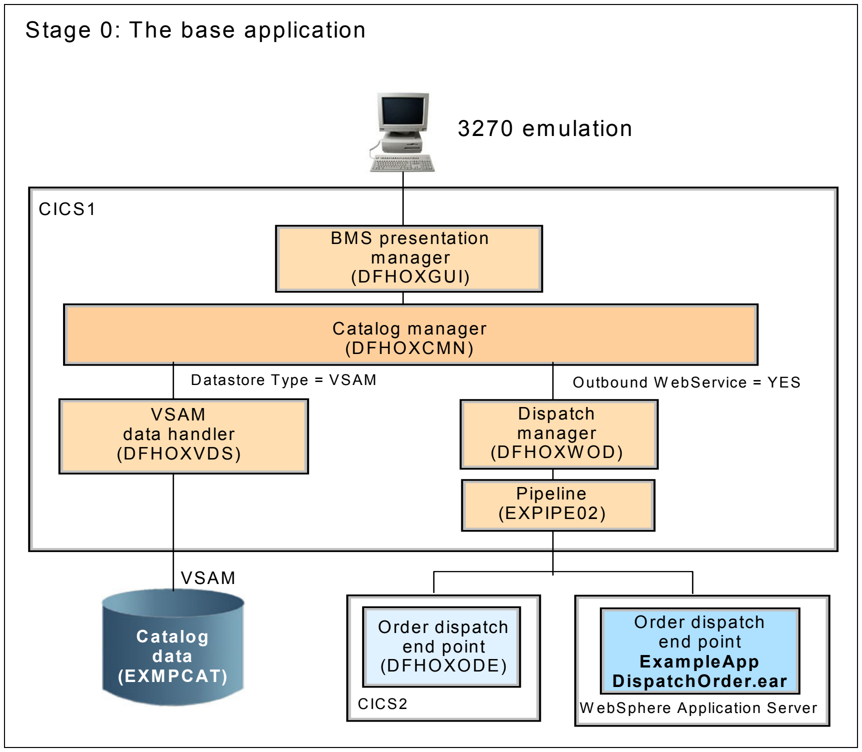

Stage 0: The base application
3270 emulation
CICS1
BMS presentation manager (DFHOXGUI)
Catalog manager (DFHOXCMN)
Datastore Type = VSAM
Outbound WebService = YES
VSAM data handler (DFHOXVDS)
Dispatch manager (DFHOXWOD)
Pipeline (EXPIPE02)
VSAM
Catalog data (EXMPCAT)
Order dispatch end point (DFHOXODE)
CICS2
Order dispatch end point ExampleApp DispatchOrder.ear
WebSphere Application Server

Figure 5-1 CICS catalog manager example application

5.1.2 The base application

The base application, with its 3270 user interface (UI), provides functions with which you can perform the following tasks:

- List the contents of a stored catalog: Inquire catalog
- Select an item from the list: Inquire single
- Enter a quantity to order: Place order

The application has a modular design, which makes it easier to extend the application to support newer technology, such as web services.

Components of the base application

When you call the CICS catalog manager example using the basic mapping support (BMS) 3270 interface, there are three modules involved in the process. Figure 5-2 shows that there are two parts that separate the application.

Module DFH0XGUI provides the presentation logic, which exclusively manages the process of sending and receiving the required BMS maps. DFH0XGUI links to the DFH0XCMN catalog manager program to perform the supported functions against the catalog. DFH0XGUI uses an **`EXEC CICS LINK`** command that passes the COMMAREA structure that Example 5-1 on page 150 shows.

The catalog manager module DFH0XCMN provides the business logic, which in turn links to the VSAM data handler stub DFH0XVDS. The catalog manager program has other functions that it also implements, such as the outbound web service function. They do not play a role in the channels and containers migration project, so they are not described in this section.

The catalog manager does not require any considerations to be able to get called by the catalog servlet. The channel and container migration is unaffected in that situation.

As previously mentioned, the 3270 user interface and the provided web client front end can call the catalog manager example business logic. We provide a new web browser front end that can be used to access the capabilities of JavaServer Pages (JSP) and servlets to display the catalog manager results.

We demonstrate how to access the business logic of the catalog manager shown in Figure 5-2, and also describe the opportunities to extend the look of web-based presentation logic.

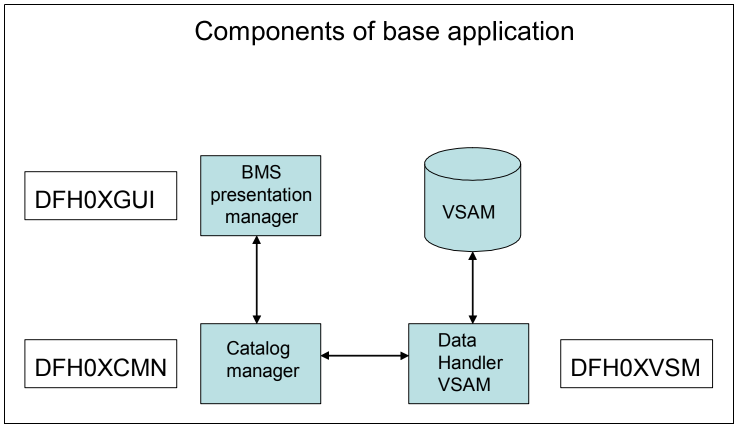


Figure 5-2 Components of the base application

Migration considerations

You do not want to port the presentation logic DFH0XGUI to the new function. The presentation logic is using the COMMAREA, and it is inconceivable that you get any constraints with it in the future. Therefore, the presentation logic remains unaffected.

However, the catalog manager must port. We provide a new Liberty profile-based browser front end to it that can additionally display the corresponding item images. Because of their sizes, the relevant `.gif` images cannot pass between the wrapper program and the catalog manager using COMMAREAs.

The `.gif` images that this chapter uses to demonstrate how to display catalog item images have a size of approximately 60 kilobytes (KB), which exceeds the COMMAREA limit. Therefore, you need to port the catalog manager module DFH0XCMN to the channels and container functionality.

Currently, the presentation logic uses an **`EXEC CICS LINK`** command that passes a COMMAREA to call the catalog manager module DFH0XCMN. However, you want to use a wrapper program for the web client front end that uses an **`EXEC CICS LINK`** command to pass a channel to link to the catalog manager program.

There is an alternative solution to avoid porting the presentation logic of the catalog manager example program. You can create an additional routine that the presentation logic DFH0XGUI can call to separate the COMMAREA structure to individual containers.

Figure 5-3 on page 145 shows the new structure of the base application. The presentation logic of the application calls the data separation logic first. The data separation logic then calls the business logic using an **`EXEC CICS LINK`** command that passes a channel rather than a COMMAREA. The latter half of this chapter describes the structure of the data separation logic.

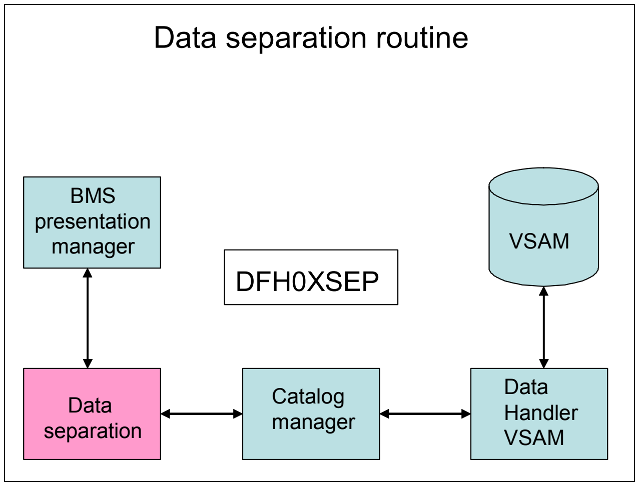


Figure 5-3 Data separation module DFH0XSEP

5.1.3 Porting steps: CICS back end

The previous sections describe the basic structure of the CICS catalog manager example program. They also provide information about the addition of a new function to the catalog manager that enables you to retrieve the corresponding image of a catalog item before any order of the item takes place.

To port the back end of the CICS catalog manager example program to the new function, you must perform the following steps:

1. Separate the COMMAREA structure into containers.

   You do not want to port the presentation logic DFH0XGUI, but you must change the module to link to the data separation module DFH0XSEP. You must separate the COMMAREA structure into individual containers first. Module DFH0XSEP does not exist yet, so you must create it.

2. Create the data separation module DFH0XSEP.

   The module takes the COMMAREA structure from DFH0XGUI and creates suitable containers for the catalog manager DFH0XCMN. When the catalog manager returns, you receive the information from the relevant containers back to the COMMAREA.

3. Port the catalog manager module DFH0XCMN, Stage 1.

   The catalog manager ports in two stages. During Stage 1, you change the logic of the catalog manager module to use channels and containers. After this, the logic of DFH0XCMN manages the channels that get passed from either the web client front end or from the presentation logic.

4. Verify the basic functions.

   When the Stage1 migration process has completed, you can perform a function test of the base application using the 3270 interface. You must make sure that the basic functions (inquire catalog, inquire single, and place order) still work fine after porting the catalog manager application.

5. Port the catalog manager DFH0XCMN, Stage2.

   During the Stage 2 migration step, you can add the display catalog item support to the catalog manager module. Store the item images to a hierarchical file system (HFS) data set. The catalog manager module DFH0XCMN is modified to retrieve the item image from the HFS data set. This process is described later in this chapter.

6. Create a new module that reads the catalog item image from HFS.

   The new routine gets control from the catalog manager DFH0XCMN. After that, it gets the item image from the HFS data set and puts it into a container that the current channel of the catalog manager program owns. Therefore, the item image is available when you return to the catalog manager.

7. Verify the extended functionality of the base application.

   When you have completed step 5 and step 6, make sure that the basic functions of the application still work properly. Also, verify that the item image is available in a container when the catalog manager returns.

5.1.4 Stage 1: Porting to channels and containers

This section describes the process of porting the CICS catalog manager example program to use the functionality of channels and containers. As mentioned previously, the beginning of Stage 1 is to create the data separation module. Before you start, you must consider the way to separate the COMMAREA to individual containers.

It is possible to use a channel with a single container to replace the COMMAREA that passes between the presentation logic and the catalog manager. This is the simplest and quickest way to port the catalog manager application. However, we do *not* recommend replacing the COMMAREA with just one single container as a leading practice. We advise that the process to replace the existing COMMAREA structure complies with the following statements:

- Use separate containers for input and output.
- Use a dedicated container for error information.
- Use separate containers for each structure.
- Use a copybook that records the name of the channel, records the names of the containers used, and defines the data fields that map to the containers.
- Include the copybook in both the client and the server program.

See Figure 5-4 illustrates the suggested separation of data.

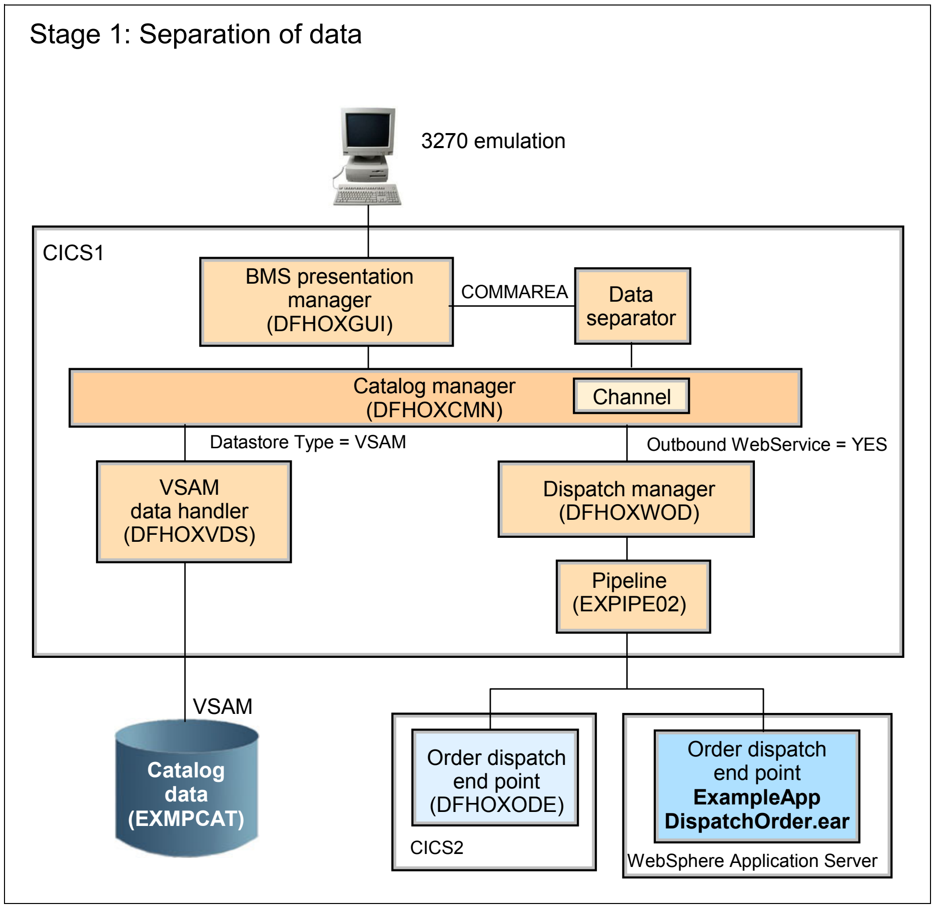

Stage 1: Separation of data
3270 emulation
CICS1
BMS presentation manager (DFHOXGUI)
COMMAREA
Data separator
Catalog manager (DFHOXCMN)
Channel
Datastore Type = VSAM
Outbound WebService = YES
VSAM data handler (DFHOXVDS)
Dispatch manager (DFHOXWOD)
Pipeline (EXPIPE02)
VSAM
Catalog data (EXMPCAT)
Order dispatch end point (DFHOXODE)
CICS2
Order dispatch end point ExampleApp DispatchOrder.ear
WebSphere Application Server

Figure 5-4 Catalog manager application at Stage 1 level

There are two migration scenarios that you can perform:

- A first approach is where you do not split the COMMAREA structure into separate containers. The following section describes the disadvantages of this approach.
- The leading practice approach is where you split the COMMAREA structure of the application into separate containers according to the statements mentioned previously.

Using channels and containers: A first approach

The simplest approach to convert an existing CICS application to take advantage of containers and channels might seem to be replacing an existing COMMAREA implementation by creating a new channel with a single container that holds the existing COMMAREA structure. However, this approach does not enable you to separate a monolithic parameter block into its separate component parts.

Consequently, the application suite cannot take complete advantage of the benefits of the containers and channels approach.

We provide a data separation module DFH0XSEP, and also a ported version of catalog manager module DFH0XCMN, to demonstrate the simplest approach to converting existing CICS applications to use channels and containers. A detailed description of the leading practice approach to the new function is presented in the next section.

Leading practice approach: Replacing the COMMAREA

This section describes the process to port the CICS catalog manager example to the new function using the leading practice approach. Create a new module DFH0XSEP that runs the structure that Figure 5-5 illustrates.

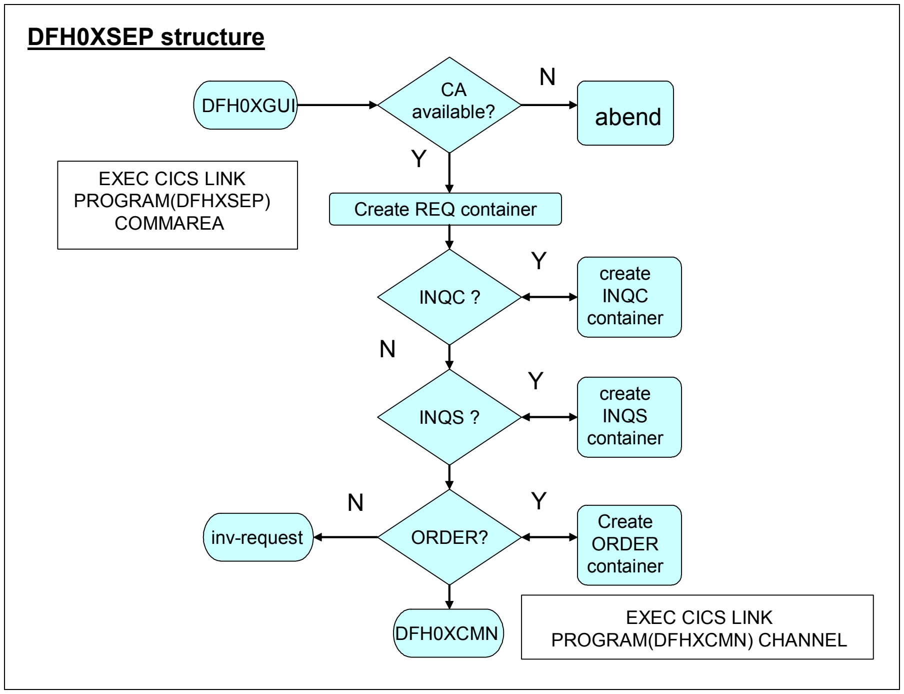


Figure 5-5 DFH0XSEP structure

The following steps describe the logic of the structure:

1. DFH0XSEP gets control from DFH0XGUI through the **`EXEC CICS LINK`** command. DFH0XGUI passes a COMMAREA.
2. If there is no COMMAREA available, write an error message and issue an abnormal end (abend) called EXCA using an **`EXEC CICS ABEND`** command.
3. If you have a COMMAREA available, create a separate input container for the required request, which can be inquire catalog, inquire single, or place order.
4. Do not create a separate input container for return codes and messages at this time. The catalog manager module creates this later.
5. After that, evaluate the request type and create another input container according to the relevant request.

When the evaluation of the request type is complete, call the catalog manager using an **`EXEC CICS LINK`** command passing the channel holding the containers. Do not create any output containers in the data separation logic. We suggest that you create the output containers in the server program, which in our case is the catalog manager module DFH0XCMN.

The following section provides a more practical description about how you can separate the COMMAREA structure.

COMMAREA structure

Example 5-1 on page 150 shows the COMMAREA structure used by the CICS catalog manager example program. The structure consists of the following parts:

- A general part

  This part contains the request type and structure items for the return code and the response message. The general part of the structure also contains a field for the result of the different requests, CA-REQUEST-SPECIFIC.

- Fields used in inquire catalog

  This section of the COMMAREA structure is used for the inquire catalog function. It redefines CA-REQUEST-SPECIFIC and uses a table to store 15 catalog items.

- Fields used in inquire single

  This section of the COMMAREA structure is used to describe the fields used for the inquire single function. It also redefines CA-REQUEST-SPECIFIC.

- Fields used in place order

  This section of the COMMAREA structure is used for the place order function. It also redefines CA-REQUEST-SPECIFIC.

Example 5-1 shows the COMMAREA structure.

Example 5-1 COMMAREA structure of catalog manager example

```
*    Catalogue COMMAREA structure
     03 CA-REQUEST-ID            PIC X(6).
     03 CA-RETURN-CODE           PIC 9(2).
     03 CA-RESPONSE-MESSAGE      PIC X(79).
     03 CA-REQUEST-SPECIFIC      PIC X(911).
*    Fields used in Inquire Catalog
     03 CA-INQUIRE-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
         05 CA-LIST-START-REF        PIC 9(4).
         05 CA-LAST-ITEM-REF         PIC 9(4).
         05 CA-ITEM-COUNT            PIC 9(3).
         05 CA-INQUIRY-RESPONSE-DATA PIC X(900).
         05 CA-CAT-ITEM  REDEFINES CA-INQUIRY-RESPONSE-DATA
                         OCCURS 15 TIMES.
             07 CA-ITEM-REF          PIC 9(4).
             07 CA-DESCRIPTION       PIC X(40).
             07 CA-DEPARTMENT        PIC 9(3).
             07 CA-COST              PIC X(6).
             07 IN-STOCK             PIC 9(4).
             07 ON-ORDER             PIC 9(3).
*    Fields used in Inquire Single
     03 CA-INQUIRE-SINGLE REDEFINES CA-REQUEST-SPECIFIC.
         05 CA-ITEM-REF-REQ          PIC 9(4).
         05 FILLER                   PIC 9(4).
         05 FILLER                   PIC 9(3).
         05 CA-SINGLE-ITEM.
             07 CA-SNGL-ITEM-REF     PIC 9(4).
             07 CA-SNGL-DESCRIPTION  PIC X(40).
             07 CA-SNGL-DEPARTMENT   PIC 9(3).
           07 CA-SNGL-COST         PIC X(6).
             07 IN-SNGL-STOCK        PIC 9(4).
             07 ON-SNGL-ORDER        PIC 9(3).
         05 FILLER                   PIC X(840).
*    Fields used in Place Order
     03 CA-ORDER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
         05 CA-USERID                PIC X(8).
         05 CA-CHARGE-DEPT           PIC X(8).
         05 CA-ITEM-REF-NUMBER       PIC 9(4).
         05 CA-QUANTITY-REQ          PIC 9(3).
         05 FILLER                   PIC X(888).
```

Data separation of the structure

According to the leading practice approach to implement channel and containers, you can separate the COMMAREA structure into the following functional parts:

- The request ID. CA-REQUEST-ID is part of the general section, and you must place it in a single input container.
- Return code and response message. We advise that you place return codes and response messages in a separate output-only container.
- Keep the CA-INQUIRE-REQUEST structure in a separate container. You can separate this structure further. However, we decided to keep the structures for each function in a single container rather than using further containers.
- Keep the CA-INQUIRE-SINGLE structure in a single container.
- The CA-ORDER-REQUEST structure also goes in a single container.
- The catalog manager module creates the result container. Copy the result information for each function to a separate container. It is a good practice to use different containers for input and output processing.

Figure 5-6 shows the COMMAREA structure separated into different containers.

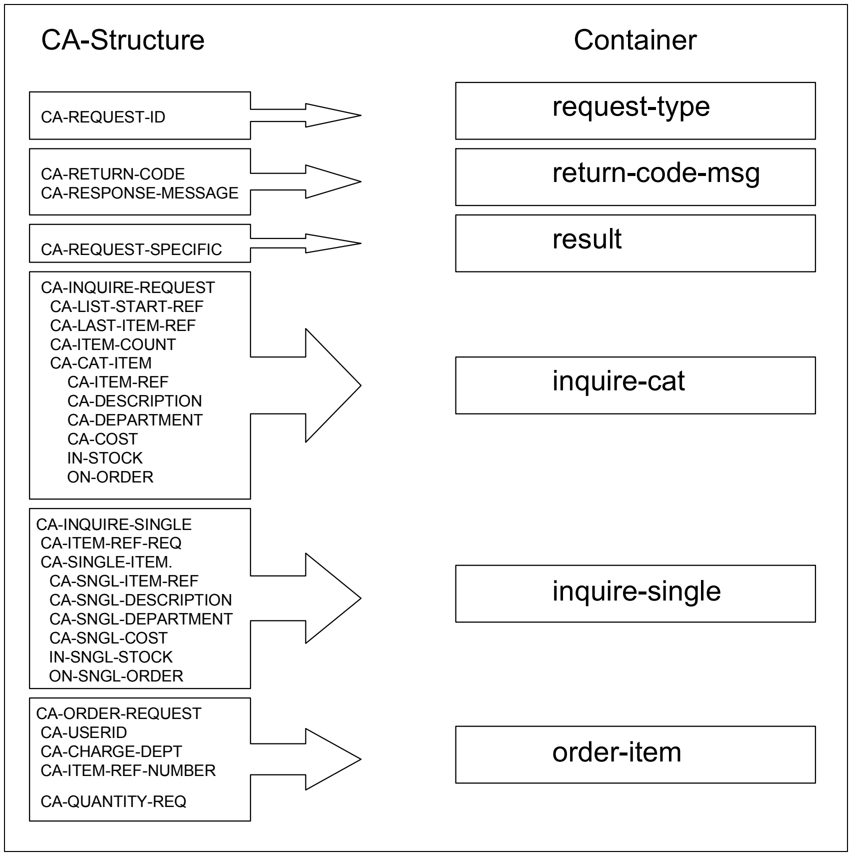


Figure 5-6 Data separation of the COMMAREA structure

After separating the COMMAREA structure to port the application to the new function according to leading practices, verify that you have completed the following actions:

☐ Used different containers for input and output operations. This simplifies the copybook structure and makes the program easier to understand.

☐ Separated the structures of the individual functions into their own containers. Input containers that the server program has not changed are not returned to the caller.

☐ Used a dedicated container for return code and response messages.

DFH0X02 copybook

It is a leading practice to create a copybook that records the name of the channel, and the names of the containers that you use. The copybook must define the data fields that map to the containers.

Example 5-2 shows the copybook that our example uses for the data separation routine, and for the catalog manager module DFH0XCMN.

The first part of the structure defines the use of the channel name and the container names to port the application. The data separation routine DFH0XSEP creates the following containers:

- request-type
- return-code-msg
- inquire-cat
- inquire single
- order-item

The catalog manager module creates the result container. Use the other containers in the structure during Stage 2 of the migration, when you add the item image support to the catalog manager. This process is described later in this chapter.

Example 5-2 illustrates the creation of copybook DFH0X02.

Example 5-2 Copybook DFH0X02

```
*  Channel name
   01 CMN-CHANNEL  PIC X(16) VALUE 'cmn-channel'.

*  Container names
   01 REQ                PIC X(16) VALUE 'request-type'.
   01 RC-MSG             PIC X(16) VALUE 'return-code-msg'.
   01 INQC               PIC X(16) VALUE 'inquire-cat'.
   01 INQS               PIC X(16) VALUE 'inquire-single'.
   01 ORDR               PIC X(16) VALUE 'order-item'.
   01 RESULT             PIC X(16) VALUE 'result'.
   01 IMAGE-REF          PIC X(16) VALUE 'ref-no'.
   01 IMG-CONTAINER      PIC X(16) VALUE 'gif-data'.
   01 IMG-DIR            PIC X(16) VALUE 'img-dir'.
   01 GET-IMG-ERROR      PIC X(16) VALUE 'error-msg'.
   01 SINGLE-CNT         PIC X(16) VALUE 'single'.
```

```
*  Define the data fields used by the program
   01 XCMNPROG          PIC X(8) VALUE 'EFH2XCMN'.
   01 CATALOG-SERVER    PIC X(8) VALUE 'CTLGSERV'.
   01 IMG-DIR-CONTENTS  PIC X(18) VALUE '/u/cicsrs9/images/'.
   01 CHN-NAME          PIC X(16).
   01 IMG-REF-NUMBER    PIC 9(2) VALUE 10.

* Request-ID container structure
 01  REQUEST-ID                 PIC X(6).

* Return code & MSG container structure
 01  RETCODE-MSG.
     03 RC                      PIC 9(2).
     03 RESPONSE-MESSAGE        PIC X(79).

* Inquire single container structure
 01  INQUIRE-SINGLE.
     03 ITEM-REF-REQ            PIC 9(4).
     03 FILLER                  PIC 9(4).
     03 FILLER                  PIC 9(3).

* Inquire catalog container structure
 01  INQUIRE-CAT.
     03 LIST-START-REF          PIC 9(4).
     03 LAST-ITEM-REF           PIC 9(4).
     03 ITEM-COUNT              PIC 9(3).

* Order request catalog container structure
 01  ORDER-REQUEST.
     03 USERID                  PIC X(8).
     03 CHARGE-DEPT             PIC X(8).
     03 ITEM-REF-NUMBER         PIC 9(4).
     03 QUANTITY-REQ            PIC 9(3).
```

Creating DFH0XSEP

This section describes steps that we used in our example to create the data separation module DFH0XSEP. You can also complete the following major tasks:

1. Separate the existing COMMAREA structure of the catalog manager example program into different containers.
2. Copy the structure of copybook DFH0X02 to module DFH0XSEP.
3. Check in the mainline section of the program to see if you were passed a valid COMMAREA from DFH0XGUI. Prepare a suitable error message and issue an EXEC CICS ABEND if there is no COMMAREA available.

4. If the COMMAREA is passed to DFH0XSEP, create the first input channel.
5. The first channel takes only the request type. Therefore, move CA-REQUEST to the structure that maps the request ID container.
6. Example 5-3 also shows how to create the container that takes the return code and response messages. The name of the container is return-code-msg. Move the return code and response message to the structure that maps the container.
7. Set up the container with the appropriate request type structure. Do not pass data structures superfluously.
8. Evaluate the request type and set up the appropriate container that corresponds to the request.

Complete the following steps to create the data separation module:

1. Create the request type container, as shown in the code snippet in Example 5-3.

Example 5-3 Set up request ID container

```
* Set up request-id container

     MOVE CA-REQUEST-ID TO REQUEST-ID.

* Create request container

     EXEC CICS PUT CONTAINER(REQ) CHANNEL(CMN-CHANNEL)
          FROM(REQUEST-ID) FLENGTH(LENGTH OF REQUEST-ID)
               END-EXEC.

* Set up return code & Msg container

     MOVE CA-RETURN-CODE TO RC.
     MOVE CA-RESPONSE-MESSAGE TO RESPONSE-MESSAGE.

* Create return code and message container

     EXEC CICS PUT CONTAINER(RC-MSG) CHANNEL(CMN-CHANNEL)
          FROM(RETCODE-MSG) FLENGTH(LENGTH OF RETCODE-MSG)
               END-EXEC.
```

2. Evaluate the request type to create the corresponding container, as shown in Example 5-4.

   This example uses three individual paragraphs to set up the containers:

   – CATALOG-INQUIRE creates the container for the inquire catalog function.
   – CATALOG-SIN creates the container for the inquire single function.
   – PLACE-ORDER creates the container for the order function.

Example 5-4 Evaluate request type structure

```
EVALUATE CA-REQUEST-ID
           WHEN '01INQC'
  *        prepare request container for catalog inquiry
               PERFORM CATALOG-INQUIRE

           WHEN '01INQS'
  *        prepare request container for inquire single
               PERFORM INQUIRE-SIN

           WHEN '01ORDR'
  *        prepare reuest container for place order request
               PERFORM PLACE-ORDER

           WHEN OTHER
```

The following three examples show the paragraphs that our example uses to set up the structures that map to the data fields of the containers.

3. Example 5-5 shows the creation of the container that takes the structure for the inquire catalog request.

   Move two parameters from the COMMAREA to the INQUIRE-CAT structure. The parameters are LIST-START-REF and LAST-ITEM-REF. They define the start of the item list, and are a reference of the displayed item.

   Example 5-5 Create inquire catalog container

```
*===========================================================*
* Catalog inquire routine                                   *
*===========================================================*
CATALOG-INQUIRE.
* Set up inquire catalog container structure
          MOVE CA-LIST-START-REF TO LIST-START-REF.
          MOVE CA-LAST-ITEM-REF TO LAST-ITEM-REF.

* Create inquire catalog container
         EXEC CICS PUT CONTAINER(INQC) CHANNEL(CMN-CHANNEL)
              FROM(INQUIRE-CAT) FLENGTH(LENGTH OF INQUIRE-CAT)
                       END-EXEC.
          EXIT.
```

4. Example 5-6 shows the paragraph that our example uses to create the container for the inquire single request.

   To run the inquire single request, you must pass one parameter. Move CA-ITEM-REF-REQ from the COMMAREA to the INQUIRE-SINGLE structure, which maps the data fields of the container.

   Example 5-6 Create inquire single container

```
*=================================================================*
* Catalog inquire single routine                                  *
*=================================================================*
INQUIRE-SIN.
* Set up inquire single container structure

          MOVE CA-ITEM-REF-REQ TO ITEM-REF-REQ.

* Create container for inquire single
         EXEC CICS PUT CONTAINER(INQS) CHANNEL(CMN-CHANNEL)
                FROM(INQUIRE-SINGLE) FLENGTH(LENGTH OF
INQUIRE-SINGLE)
                       END-EXEC.
           EXIT.
```

5. Example 5-7 shows the PLACE-ORDER paragraph that our example uses to create the container for the order request.

   Move four fields from the COMMAREA to the ORDER-REQUEST structure that was created in copybook DFH0S02. The following parameters are the four that our example uses for the order request container:

   – USERID: The name of the person that places the order.
   – CHARGE-DEPT: The name of the department.
   – ITEM-REF-NUMBER: The reference number of the catalog item.
   – QUANTITY-REQ: The number of items to order.

Example 5-7 Create place order container

```
*=================================================================*
* Place order routine                                             *
*                                                                 *
*=================================================================*
PLACE-ORDER.
* Set up request and container name
          MOVE CA-USERID TO USERID.
          MOVE CA-CHARGE-DEPT TO CHARGE-DEPT.
          MOVE CA-ITEM-REF-NUMBER TO ITEM-REF-NUMBER.
          MOVE CA-QUANTITY-REQ TO QUANTITY-REQ.

* Create container for place order
         EXEC CICS PUT CONTAINER(ORDR) CHANNEL(CMN-CHANNEL)
                 FROM(ORDER-REQUEST) FLENGTH(LENGTH OF
ORDER-REQUEST)
                        END-EXEC.
           EXIT.
```

6. Example 5-8 on page 159 shows how the example links to the catalog manager. Issue an **`EXEC CICS LINK`** command passing the channel that owns the following containers:

   – request-type
   – return-code-msg
   – inquire-cat
   – inquire-single
   – order-item
   – result

   The catalog manager DFH0XCMN creates the result container.

Example 5-8 shows the process of linking to the catalog manager module.

Example 5-8 Link to catalog manager module

```
*===========================================================*
* Link to the catalog manager                               *
*===========================================================*
       EXEC CICS LINK PROGRAM(XCMNPROG)
                      CHANNEL(CMN-CHANNEL)
       END-EXEC.
       EXEC CICS GET CONTAINER(RC-MSG) CHANNEL(CMN-CHANNEL)
            INTO(RETCODE-MSG)
       END-EXEC

       MOVE RC TO CA-RETURN-CODE.
       MOVE RESPONSE-MESSAGE TO CA-RESPONSE-MESSAGE.
```

7. After the return from catalog manager DFH0XCMN, perform an EXEC CICS GET CONTAINER command to get back the return code and response message to the COMMAREA. The data separation routine converts the COMMAREA structure to separate containers to link to the catalog manager.

   When the catalog manager returns, the logic in DFH0XSEP receives the information from the outbound containers to the COMMAREA. After that, DFH0XSEP returns to DFH0XGUI using the updated COMMAREA, as shown in Example 5-9.

Example 5-9 Get container commands on return

```
EVALUATE CA-REQUEST-ID

               WHEN '01INQC'
         EXEC CICS GET CONTAINER(RESULT) CHANNEL(CMN-CHANNEL)
                INTO(CA-INQUIRE-REQUEST)
         END-EXEC

               WHEN '01INQS'
         EXEC CICS GET CONTAINER(RESULT) CHANNEL(CMN-CHANNEL)
                INTO(CA-INQUIRE-SINGLE)
         END-EXEC

              WHEN '01ORDR'
         EXEC CICS GET CONTAINER(RESULT) CHANNEL(CMN-CHANNEL)
                INTO(CA-ORDER-REQUEST)
         END-EXEC
```

8. When the catalog manager DFH0XCMN returns, DFH0XSEP expects the result of the relevant request in the result container. Therefore, evaluate the request ID to move the data fields of the result container to the corresponding structure in the COMMAREA.

   Example 5-9 on page 159 shows the evaluation of the request ID and the corresponding EXEC CICS GET CONTAINER commands.

   The next section describes the process used to port the catalog manager DFH0XCMN.

Extending the catalog manager module

This section describes the migration of the catalog manager module to use channels and containers. So far, you have developed the data separation module that you use as a converter from COMMAREA to channels and container. The presentation logic DFH0XGUI uses the data separation module to convert its COMMAREA structure to the channel and container design that the catalog manager module DFH0XCMN accepts.

Consider the following points, before the description of the porting steps:

- The original catalog manager module takes the COMMAREA from the presentation logic DFH0XGUI.
- The catalog manager module links to a VSAM data handler module DFH0XVDS to perform one of the three request types, which are inquire catalog, inquire single, or place order.
- Use an **`EXEC CICS LINK`** command passing a COMMAREA to call the VSAM data handler. You must not port the VSAM data handler to the function, because it is not possible to get any COMMAREA constraints with it in the future.

Therefore, after you complete the previously mentioned checks, you must port the catalog manager module so that it is able to perform the following actions:

- When the data separation module DFH0XSEP or the web client front end calls the catalog manager, it expects a channel.
- The catalog manager module calls the VSAM data handler DFH0XVDS using an **`EXEC CICS LINK`** command, over a COMMAREA.
- Therefore, port the catalog manager to run in dual mode. When it calls the VSAM data handler module DFH0XVDS, it uses channels and containers and also the COMMAREA.

To extend the catalog manager module, perform the following steps:

1. Example 5-10 shows the start of the mainline section within the original catalog manager module. The initial action is to perform a check against EIBCALEN. If the length of the COMMAREA is zero, you can set up a response message and issue an **EXEC CICS ABEND**.

Example 5-10 Before migration: check for a valid COMMAREA

```
*----------------------------------------------------------------*
* Check commarea and obtain required details                     *
*----------------------------------------------------------------*
* If NO COMMAREA received issue an ABEND
           IF EIBCALEN IS EQUAL TO ZERO
              MOVE ' NO COMMAREA RECEIVED' TO EM-DETAIL
               PERFORM WRITE-ERROR-MESSAGE
              EXEC CICS ABEND ABCODE('EXCA') NODUMP END-EXEC
           END-IF

* Initalize COMMAREA return code to zero
           MOVE '00' TO CA-RETURN-CODE.
           MOVE EIBCALEN TO WS-CALEN.
```

2. Replace the code that examines the COMMAREA. A channel, rather than a COMMAREA, calls the ported version of the catalog manager. Therefore, the first thing you must do is to check if you were passed a channel.
3. Use an **EXEC CICS ASSIGN CHANNEL** command to make sure that you received the expected channel. The name of the channel must be cmn-channel.

   If there is no channel available, set up an error message and issue an **EXEC CICS ABEND** command. See Example 5-11.

Example 5-11 Assign channel command

```
*----------------------------------------------------------------*
* Check if we were passed a channel                              *
* If NO channel available issue an ABEND                         *
*----------------------------------------------------------------*
           EXEC CICS ASSIGN CHANNEL(CHN-NAME)
           END-EXEC
           IF CHN-NAME NOT = 'cmn-channel     '
              MOVE ' NO CHANNEL RECEIVED ' TO EM-DETAIL
               PERFORM WRITE-ERROR-MESSAGE
              EXEC CICS ABEND ABCODE('EXCH') NODUMP END-EXEC
            END-IF
```

4. If you have received the expected channel available, you can issue an **EXEC CICS GET CONTAINER(REQ)** command to retrieve the required request ID.

   Our example specifies CA-REQUEST-ID on the **INTO** parameter, which sets up the COMMAREA structure used to call the VSAM data handler later on. See Example 5-12.

   Example 5-12 Get request ID

   ```
   EXEC CICS GET CONTAINER(REQ)
             CHANNEL(CMN-CHANNEL)
             INTO(CA-REQUEST-ID)
   END-EXEC
   ```

   In the next step of the catalog manager migration, perform an evaluation of the request ID.

5. In this example, arrange the three paragraphs that issue the **EXEC CICS GET CONTAINER** commands to retrieve the necessary data structures required to perform the corresponding functions:

   – The inquire catalog request 01INQC uses paragraph CATALOG-INQUIRE.
   – Inquire single request 01INQS uses paragraph CATALOG-INQUIRE-S.
   – Place order request 01ORDR uses paragraph PLACE-ORDER.

   Each of the paragraphs issue the relevant **EXEC CICS GET CONTAINER** command and an **EXEC CICS LINK** command to the VSAM data handler module passing a COMMAREA.

   Example 5-13 shows how to evaluate the request ID to call the corresponding paragraph.

   Example 5-13 Evaluate request ID

   ```
   *----------------------------------------------------------------*
   * Check which operation is being requested
   *----------------------------------------------------------------*
   * Uppercase the value passed in the Request Id field
            MOVE FUNCTION UPPER-CASE(CA-REQUEST-ID) TO CA-REQUEST-ID

             EVALUATE CA-REQUEST-ID
                 WHEN '01INQC'
        *       Call routine to perform for inquire
                    PERFORM CATALOG-INQUIRE

                 WHEN '01INQS'
        *       Call routine to perform for inquire for single item
                    PERFORM CATALOG-INQUIRE-S
   ```

```
             WHEN '01ORDR'
      *      Call routine to place order
                PERFORM PLACE-ORDER

             WHEN OTHER
      *      Request is not recognised or supported
                PERFORM REQUEST-NOT-RECOGNISED

          END-EVALUATE
```

6. Example 5-14 shows the CATALOG-INQUIRE paragraph that our example uses to get the inquire catalog request parameter from the INQC container. Specify INQUIRE-CAT on the **INTO** parameter. INQUIRE-CAT is the structure defined in the DFH0S02 copybook to map the data fields of the INQC container.
7. Then, set up the COMMAREA for the VSAM data handler. Move the three parameters from the INQC input container structure to the corresponding fields in the COMMAREA.
8. After this, call the VSAM data handler using an **EXEC CICS LINK** command passing the COMMAREA.

   See Example 5-14 for details of these steps.

Example 5-14 Paragraph catalog-inquire

```
*================================================================*
* Procedure to link to Datastore program to inquire             *
*   on the catalog data                                          *
*================================================================*
CATALOG-INQUIRE.
           MOVE 'EXCATMAN: CATALOG-INQUIRE' TO CA-RESPONSE-MESSAGE.
          EXEC CICS GET CONTAINER(INQC)
                        CHANNEL(CMN-CHANNEL)
                        INTO(INQUIRE-CAT)
          END-EXEC

          MOVE LIST-START-REF TO CA-LIST-START-REF.
          MOVE LAST-ITEM-REF  TO CA-LAST-ITEM-REF.
          MOVE ITEM-COUNT     TO CA-ITEM-COUNT.

          EXEC CICS LINK   PROGRAM(WS-DATASTORE-PROG)
                          COMMAREA(WS-CHN-DATA)
           END-EXEC.
```

9. If the request type is inquire single, use paragraph CATALOG-INQUIRE-SINGLE to get the inquire single parameter from the INQS container.

   Example 5-15 shows an **EXEC CICS GET CONTAINER(INQS)** command that you can use to store the parameter in the INQUIRE-SINGLE structure.

10. The inquire single request only takes one parameter. Therefore, move ITEM-REF-REQ to the relevant COMMAREA field, CA-ITEM-REF-REQ.

Example 5-15 Paragraph catalog-inquire-s

```
*=================================================================*
* Procedure to link to Datastore program to inquire a single item*
*   on the catalog data                                           *
*=================================================================*
CATALOG-INQUIRE-S.
         MOVE 'EXCATMAN: CATALOG-INQUIRE' TO CA-RESPONSE-MESSAGE
          EXEC CICS GET CONTAINER(INQS)
                        CHANNEL(CMN-CHANNEL)
                        INTO(INQUIRE-SINGLE)
          END-EXEC

          MOVE ITEM-REF-REQ TO CA-ITEM-REF-REQ.

          EXEC CICS LINK   PROGRAM(WS-DATASTORE-PROG)
                          COMMAREA(WS-CHN-DATA)
           END-EXEC
```

11. Use the third paragraph for the place order request type. Example 5-16 on page 165 shows the **EXEC CICS GET CONTAINER(ORDR)** command that maps the required input parameters to the ORDER-REQUEST structure, which is specified on the **INTO** parameter.

    Our example uses four parameters for the place order request type. You can move them from the ORDER-REQUEST structure to the relevant fields in the COMMAREA.

    You can move the following parameters to the COMMAREA:

    – USERID
    – CHARGE-DEPT
    – ITEM-REF
    – QUANTITY-REF

12. When the parameters are set, issue an **EXEC CICS LINK** with **COMMAREA** command to link to the VSAM data handler to perform the request.

See Example 5-16, which shows the paragraph place-order.

Example 5-16 Paragraph place-order

```
*=================================================================*
* Procedure to link to Datastore program to place order,          *
*   send request to dispatcher and notify stock manager           *
*   an order has been placed                                      *
*=================================================================*
PLACE-ORDER.
           MOVE 'EXCATMAN: PLACE-ORDER' TO CA-RESPONSE-MESSAGE.
          EXEC CICS GET CONTAINER(ORDR)
                        CHANNEL(CMN-CHANNEL)
                        INTO(ORDER-REQUEST)
          END-EXEC

          MOVE USERID TO CA-USERID.
          MOVE CHARGE-DEPT TO CA-CHARGE-DEPT.
          MOVE ITEM-REF-NUMBER TO CA-ITEM-REF-NUMBER.
          MOVE QUANTITY-REQ TO CA-QUANTITY-REQ.

          EXEC CICS LINK PROGRAM(WS-DATASTORE-PROG)
                        COMMAREA(WS-CHN-DATA)
           END-EXEC.
```

13. When the VSAM data handler returns, move the return code and response message contents to the structure that maps the data fields of the RC-MSG container. After that, issue an **`EXEC CICS PUT CONTAINER(RC-MSG)`** command to provide the return and response messages to the caller of the catalog manager.

    Before you return to the caller of the catalog manager, create an output container for the result of the corresponding request.

14. Evaluate the request ID again to set up the output container using the appropriate structure that contains the result for the particular request-ID.

    See Example 5-17, where our example issues the relevant **`EXEC CICS PUT CONTAINER(RESULT)`** command, which returns to the caller of the catalog manager.

Example 5-17 Set up result output container

```
MOVE CA-RETURN-CODE TO RC.
MOVE CA-RESPONSE-MESSAGE TO RESPONSE-MESSAGE.

   EXEC CICS PUT CONTAINER(RC-MSG)
                 CHANNEL(CMN-CHANNEL)
```

```
                  FROM(RETCODE-MSG)
                  FLENGTH(LENGTH OF RETCODE-MSG)
                  END-EXEC.
    EVALUATE CA-REQUEST-ID
          WHEN '01INQC'

    EXEC CICS PUT CONTAINER(RESULT)
                  CHANNEL(CMN-CHANNEL)
                  FROM(CA-INQUIRE-REQUEST)
                  FLENGTH(LENGTH OF CA-INQUIRE-REQUEST)
                  END-EXEC

          WHEN '01INQS'

    EXEC CICS PUT CONTAINER(RESULT)
                  CHANNEL(CMN-CHANNEL)
                  FROM(CA-INQUIRE-SINGLE)
                  FLENGTH(LENGTH OF CA-INQUIRE-SINGLE)
                  END-EXEC

          WHEN '01ORDR'

    EXEC CICS PUT CONTAINER(RESULT)
                  CHANNEL(CMN-CHANNEL)
                  FROM(CA-ORDER-REQUEST)
                  FLENGTH(LENGTH OF CA-ORDER-REQUEST)
                  END-EXEC

          WHEN OTHER
*         Request is not recognized or supported
             PERFORM REQUEST-NOT-RECOGNISED

       END-EVALUATE.
* Return to caller
```

5.1.5 Installing and setting up the base application

Before you can run the ported version of the catalog manager application, you must define two VSAM data sets, and install the relevant resource definitions for the different migration stages. Our example describes the installation steps for the following application scenarios:

- The original base application.
- The first approach to port to the new function using a single container to replace the COMMAREA.
- The ported application at the stage 1 and stage 2 level.

Perform the following steps:

1. Define the VSAM KSDS data sets that the catalog manager example uses. One data set contains configuration information for the application. The other contains the sales catalog.
2. Use the job control language (JCL) shown in Example 5-18 to create the VSAM KSDS data sets.

   You can use the data sets for all application migration scenarios.
3. After this, copy the file resource definitions from CICS supplied group DFH$EXBS. In our example, we specify the data set name on the DSNAME option, and use the default values for all other attributes.

Example 5-18 JCL to create the VSAM KSDS data sets

```
//CICSPAZS JOB (999,POK),'CICS ZS',CLASS=A,MSGCLASS=T,
// NOTIFY=&SYSUID,TIME=NOLIMIT,REGION=0M
/*JOBPARM L=999,SYSAFF=SC04
//DELETE    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 DELETE CICSDSW.PAA1.EXMPLAPP.EXMPCONF
 DELETE CICSDSW.PAA1.EXMPLAPP.EXMPCAT
 SET MAXCC=0
/*
//DEFINE    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 /*                              */
 /* DEFINE A CICS GLOBAL CATALOG */
 /*                              */
 DEFINE CLUSTER(NAME(CICSDSW.PAA1.EXMPLAPP.EXMPCAT) -
         INDEXED -
         CYL(1 1)-
```

```
               KEYS(4 0) -
               RECORDSIZE(80,80) -
               SHR(2 3)-
               VOLUME(TOTCIH) REUSE)) -
             DATA(NAME(CICSDSW.PAA1.EXMPLAPP.EXMPCAT.DATA)) -
             INDEX(NAME(CICSDSW.PAA1.EXMPLAPP.EXMPCAT.INDEX))
      DEFINE CLUSTER(NAME(CICSDSW.PAA1.EXMPLAPP.EXMPCONF) -
               INDEXED -
               CYL(1 1)-
               KEYS(9 0) -
               RECORDSIZE(350,350) -
               SHR(2 3)-
               VOLUME(TOTCIH) REUSE)) -
             DATA(NAME(CICSDSW.PAA1.EXMPLAPP.EXMPCONF.DATA)) -
             INDEX(NAME(CICSDSW.PAA1.EXMPLAPP.EXMPCONF.INDEX))
```

5.1.6 Defining the 3270 interface

Our example uses the following transaction definitions to run the different migration scenarios. You can also use the following definitions:

- EGUI

  The EGUI transaction is part of the original catalog manager application. It runs program DFH0XGUI.

- SGUI

  The SGUI transaction runs the application that uses a single container to replace the COMMAREA. The transaction runs program EFH1XGUI, which replaces DFH0XGUI.

- XGUI

  The XGUI transaction runs program EFH2XGUI, which is the ported application at the Stage 1 and Stage 2 level.

To install the original and ported applications, perform the following steps:

1. Install the original base application:
   a. Define the two VSAM data sets that Example 5-18 on page 167 shows.
   b. Copy the file definitions from DFH$EXBS and alter the DSNAME option.
   c. Issue a **`CEDA DEFINE TRANSACTION(EGUI) PROG(DFHOXGUI) G(EXAMPLE)`** command to define the transaction.
   d. Issue a **`CEDA I G(EXAMPLE)`** command.

2. Install the first approach application (using a single container to replace the COMMAREA):
   a. Define the two VSAM data sets that Example 5-18 on page 167 shows, if you have not defined them already.
   b. Copy the file definitions from DFH$EXBS and alter the DSNAME option.
   c. Compile and link programs EFH1XGUI, EFH1XSEP, and EFH1XCMN.
   d. Issue a **`CEDA DEFINE TRANSACTION(SGUI) PROG(EFH1XGUI) G(EXAMPLE)`** command to define the transaction.
   e. Issue a **`CEDA I G(EXAMPLE)`** command.
3. Install the ported application at the Stage 1 and Stage 2 level:
   a. Define the two VSAM data sets that Example 5-18 on page 167 shows, if you have not defined them already.
   b. Copy the file definitions from DFH$EXBS and alter the DSNAME option.
   c. Compile and link programs EFH2XGUI, EFH2XSEP, and EFH2XCMN
   d. Issue a **`CEDA DEFINE TRANSACTION(XGUI) PROG(EFH2XGUI) G(EXAMPLE)`** command to define the transaction.
   e. Issue a **`CEDA I G(EXAMPLE)`** command.

The applications are now ready to use. You can also use the CICS Information Center to get a detailed installation description for the CICS catalog manager example application. The name of the topic is *Installing and setting up the base application*.

5.1.7 Running the application

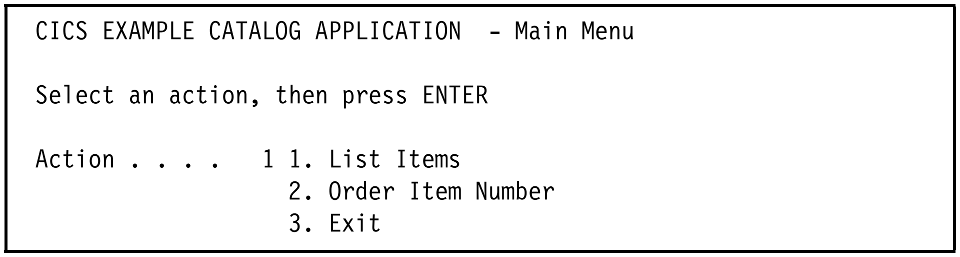
To make sure that the ported version of the application works as expected, check the flow of the programs.

After ensuring the basic function of the ported application, regression test your changes to determine the performance of the ported catalog manager application.

Before you start a regression test, this section shows you the sequence of the 3270 maps, and the operator input that you must simulate later on.

Figure 5-7 shows the first panel that opens when you enter the transaction EGUI. The transaction starts program DFH0XGUI, which sends the map.

Figure 5-7 shows you the main menu panel.



```
CICS EXAMPLE CATALOG APPLICATION  - Main Menu

Select an action, then press ENTER

Action . . . .   1 1. List Items
                   2. Order Item Number
                   3. Exit
```

Figure 5-7   Main menu panel

There are three options available:

- List items

  This function provides a list of the available catalog items. The inquire catalog function provides the list.

- Order item number

  This function enables you to order an item directly from the main menu panel. Option number 2 calls the inquire single function first.

- Exit

  This option ends the catalog manager example application.

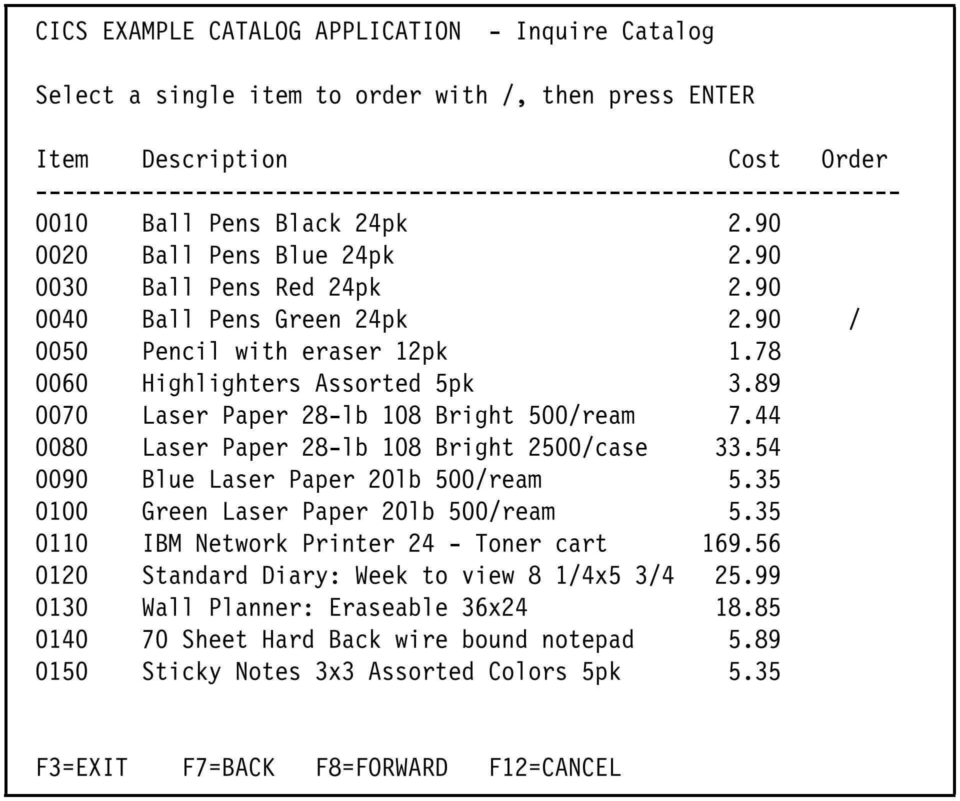
To walk through the options, follow these steps:

1. Select option 1. Figure 5-8 on page 171 shows the result of the inquire catalog function. It provides a list of the first 15 catalog items. The remaining catalog items are provided if you scroll forward using the F8 key.

   You can order an item from the inquire catalog panel from the main menu. If you type a forward slash (/) next to the catalog item cost column, it calls the inquire single function.

2. Figure 5-8 shows how our example selects the Ball Pens Green 24pk item. When you press Enter, it calls the inquire single function for item 0040.

   The inquire single function displays additional information about the item, and enables you to place the order for the item.

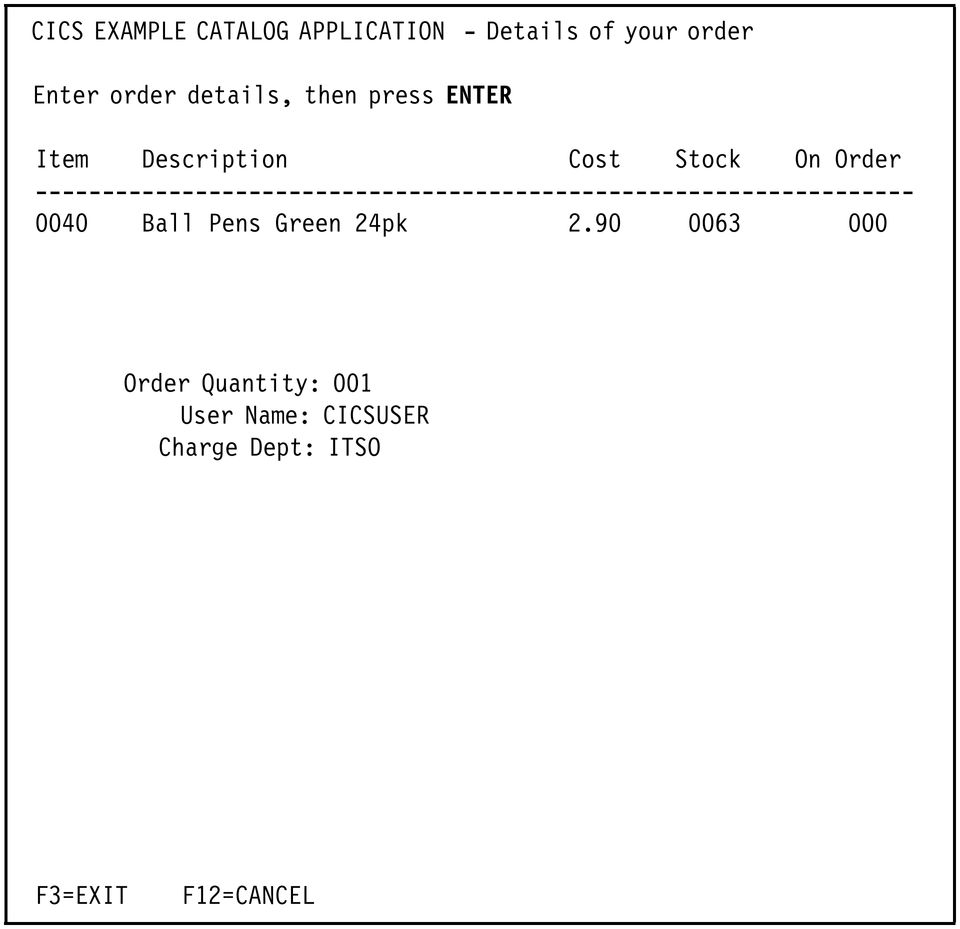


```
CICS EXAMPLE CATALOG APPLICATION  - Inquire Catalog

Select a single item to order with /, then press ENTER

Item    Description                                 Cost   Order
-----------------------------------------------------------------
0010    Ball Pens Black 24pk                        2.90
0020    Ball Pens Blue 24pk                         2.90
0030    Ball Pens Red 24pk                          2.90
0040    Ball Pens Green 24pk                        2.90     /
0050    Pencil with eraser 12pk                     1.78
0060    Highlighters Assorted 5pk                   3.89
0070    Laser Paper 28-lb 108 Bright 500/ream       7.44
0080    Laser Paper 28-lb 108 Bright 2500/case     33.54
0090    Blue Laser Paper 20lb 500/ream              5.35
0100    Green Laser Paper 20lb 500/ream             5.35
0110    IBM Network Printer 24 - Toner cart       169.56
0120    Standard Diary: Week to view 8 1/4x5 3/4   25.99
0130    Wall Planner: Eraseable 36x24              18.85
0140    70 Sheet Hard Back wire bound notepad       5.89
0150    Sticky Notes 3x3 Assorted Colors 5pk        5.35


F3=EXIT    F7=BACK   F8=FORWARD   F12=CANCEL
```

Figure 5-8 Inquire catalog map

3. Figure 5-9 shows the panel that opens when you select option 2 from the main menu, or when you type a forward slash (/) next to an item in the list of the catalog items.

   The detail of your order panel shows additional information about the item. It shows the number in stock, and the number of items that are currently on order.

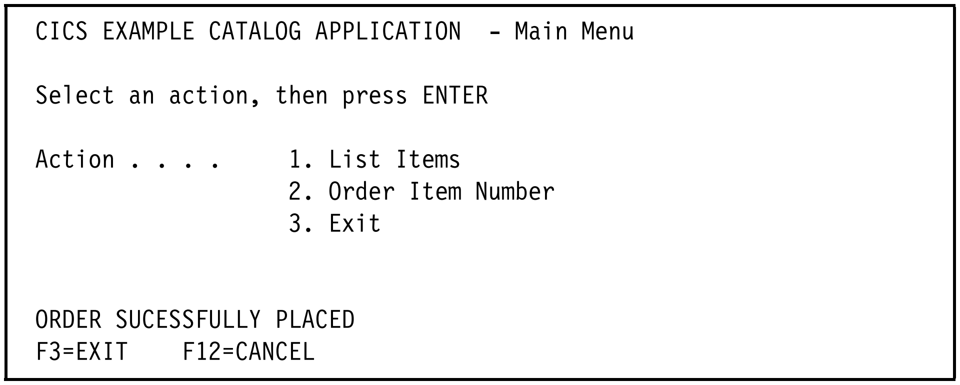


```
CICS EXAMPLE CATALOG APPLICATION - Details of your order

Enter order details, then press ENTER

Item    Description                     Cost    Stock    On Order
------------------------------------------------------------------
0040    Ball Pens Green 24pk            2.90     0063         000




      Order Quantity: 001
           User Name: CICSUSER
         Charge Dept: ITSO













F3=EXIT    F12=CANCEL
```

Figure 5-9 Order map

4. Specify the order details and press **enter**.

   When the order is successful, the main menu returns. The message `ORDER SUCCESSFULLY PLACED` indicates that you have placed the order, as shown in Figure 5-10.



```
CICS EXAMPLE CATALOG APPLICATION  - Main Menu

Select an action, then press ENTER

Action . . . .      1. List Items
                    2. Order Item Number
                    3. Exit


ORDER SUCESSFULLY PLACED
F3=EXIT    F12=CANCEL
```

Figure 5-10 Main menu map

5.1.8 Stage2: Catalog item images support

This section describes how to extend the catalog manager to enable catalog item image support.

When the Liberty catalog servlet links to the catalog manager, the corresponding catalog item image is provided by a JCICS program. This is done to extend the inquire single function of the catalog manager to provide the relevant images. Our example stores the item image files to an HFS directory. The name of the relevant image files corresponds to the item reference number, which the inquire single function of the catalog manager module uses.

We create an Open Services Gateway Initiative (OSGi) JCICS program that reads the image file from the HFS directory and puts its contents to an output binary container. The catalog manager issues an **`EXEC CICS LINK`** command to call the JCICS program. On return from the JCICS program, the item image is provided in the output container.

Figure 5-11 illustrates the application migration environment at the stage 2 level.

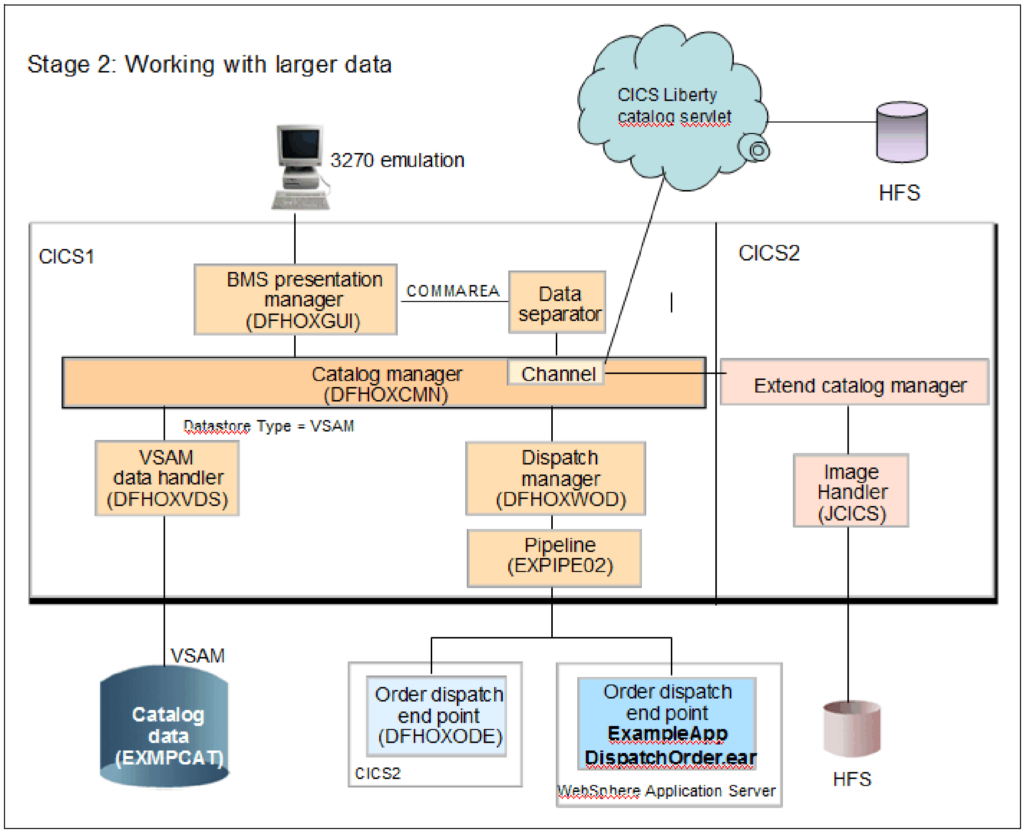

Stage 2: Working with larger data
3270 emulation
CICS Liberty catalog servlet
HFS
CICS1
BMS presentation manager (DFHOXGUI)
COMMAREA
Data separator
CICS2
Catalog manager (DFHOXCMN)
Channel
Extend catalog manager
Datastore Type = VSAM
VSAM data handler (DFHOXVDS)
Dispatch manager (DFHOXWOD)
Image Handler (JCICS)
Pipeline (EXPIPE02)
VSAM
Catalog data (EXMPCAT)
Order dispatch end point (DFHOXODE)
CICS2
Order dispatch end point ExampleApp DispatchOrder.ear
WebSphere Application Server
HFS

Figure 5-11 Application migration environment at the stage 2 level

Additional containers

Similar to our example, you can use four additional containers for the catalog item image support, which are as follows:

- Container *ref-no*

  The catalog manager creates it within the inquire single paragraph. Put the item reference number into the container and pass the channel to the JCICS program.

- Container *image-dir*

  The catalog manager creates it within the inquire single paragraph. Put the name of the HFS directory into the container and pass the channel to the JCICS program.

- Container *gif-data*

  The JCICS program creates this channel. It is the binary output channel that contains the image file.

- Container *error-msg*

  The JCICS program creates this container if there is a problem reading the image file from the HFS directory.

Extension to the catalog manager

Figure 5-12 on page 176 illustrates the new structure of the catalog manager module. You can call the module from either the data separation module, or from the CICS Liberty servlet web browser front end using an **`EXEC CICS LINK`** command passing a channel. The following steps describe the process:

1. On module entry, determine if there is a channel available. If there is no channel available, issue an **`EXEC CICS ABEND`** command. If the channel is available, issue an **`EXEC CICS GET CONTAINER`** command to figure out which request you are going to perform.
2. After that, issue an **`EXEC CICS READ`** command against the configuration file of the catalog manager example application. Do not change the original part of the code that reads the configuration file. However, if the read fails, put the return code and error messages into the RC-MSG container rather than using the COMMAREA to store the error information.
3. In the next step, evaluate the request ID that you got from the request container earlier on.
4. Run the **`EXEC CICS GET CONTAINER`** command that corresponds to the request ID. After that, issue an **`EXEC LINK PROGRAM`** command to call the VSAM data handler.
5. If the inquire single request ID runs, you additionally call the JCICS catalog server program to get the catalog item image from the HFS directory.

Figure 5-12 shows the new structure of the catalog manager module.

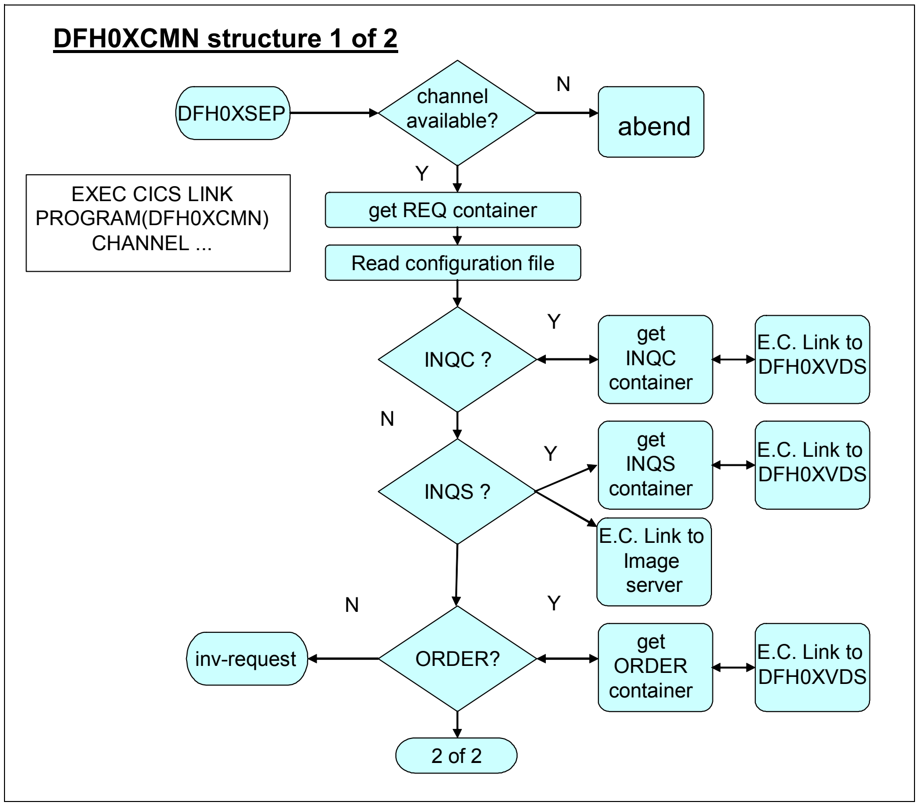


Figure 5-12 DFH0XCMN structure

Figure 5-13 on page 177 illustrates the structure of the catalog manager module before it returns to the caller. The steps are as follows:

1. Before you can return to the caller of the catalog manager, update the RC-MSG container with return code and error message information that might have occurred.
2. After that, evaluate the request ID and place the result to the output container that corresponds to the request ID.
3. You still have the catalog item image in the gif-data output container available. You do not have to take care of the item image within the catalog manager. The caller of the catalog manager module manages the container and passes the image to the web browser front end, which in this case is the CICS Liberty servlet.

Figure 5-13 shows the structure of the catalog manager module before it returns to the caller.

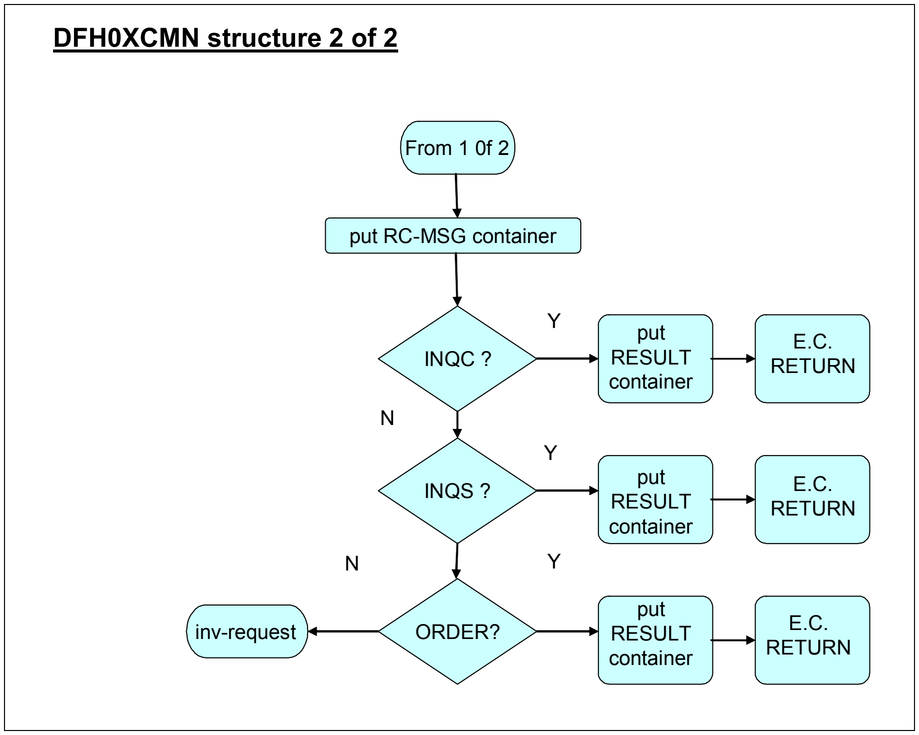


Figure 5-13 DFH0XCMN structure

Example 5-19 on page 178 shows how our example updates the catalog inquire single paragraph of the catalog manager program. It illustrates the following steps:

1. Place the image reference of the item that you want to get from the HFS directory to the ref-no container. Use the ITEM-REF-REQ data field of the inquire single structure to identify the item.
2. Place the name of the HFS directory into the image-dir container. Specify the IMG-DIR-CONTENTS data field on the `INTO` option of the **`put container`** command. Data field IMG-DIR-CONTENTS contains HFS directory `/u/klein4/images/`.
3. Issue an **`EXEC CICS LINK`** command passing the current channel, to call the catalog server JCICS program.

4. On return, the catalog server JCICS program places the image in the output gif-data container. To check that there are no error messages available in the error-msg container, run **EXEC CICS GET CONTAINER(GET-IMG-ERROR)**.

   You get a CONTAINERERROR condition if there are no error messages available. Therefore, use the RESP option to check if the response is normal. A response of DFHRESP(NORMAL) means that the container has been created and contains an error message, as shown in Example 5-19.

Example 5-19 Extended catalog-inquire-s paragraph

```
CATALOG-INQUIRE-S.
     MOVE 'EXCATMAN: CATALOG-INQUIRE' TO CA-RESPONSE-MESSAGE
     EXEC CICS GET CONTAINER(INQS)
                   CHANNEL(CMN-CHANNEL)
                   INTO(INQUIRE-SINGLE)
     END-EXEC

     MOVE ITEM-REF-REQ TO CA-ITEM-REF-REQ.

     EXEC CICS LINK   PROGRAM(WS-DATASTORE-PROG)
                     COMMAREA(WS-CHN-DATA)
     END-EXEC

*----------------------------------------------------------------*
   * before we return to the data separation routine we must     *
   * retrieve the corresponding image using the catalog server

*----------------------------------------------------------------*
     EXEC CICS PUT CONTAINER(IMAGE-REF)
                   CHANNEL(CMN-CHANNEL)
                   FROM(ITEM-REF-REQ)
     END-EXEC
   * PUT CONTAINER command to pass the HFS directory name

     EXEC CICS PUT CONTAINER(IMG-DIR)
                   CHANNEL(CMN-CHANNEL)
                   FROM(IMG-DIR-CONTENTS)
     END-EXEC
     EXEC CICS LINK   PROGRAM(CATALOG-SERVER)
                     CHANNEL(CMN-CHANNEL)
     END-EXEC
     EXEC CICS GET CONTAINER(GET-IMG-ERROR)
                   CHANNEL(CMN-CHANNEL)
                   INTO(RESPONSE-MESSAGE)
                   RESP(RESPONSE)
```

```
END-EXEC
IF RESPONSE = DFHRESP(NORMAL)
MOVE '54' TO RC
END-IF
EXIT.
```

5.2 Running the stage 2 code

Before running the CICS Liberty catalog servlet in the Stage 2 environment, you need to perform the steps that the following sections describe.

5.2.1 Installing and setting up the Stage 2 application

As Figure 5-11 on page 174 shows, the scenario now includes an image handler program and a CICS Liberty profile servlet. You need to configure both of these applications before running the scenario. You can find all Java source code and compiled classes in the additional material that we supply, described in Appendix B, “Additional material” on page 239.

Figure 5-14 on page 180 provides an overview of the applications and CICS resource definitions that we are going to implement to set up the Stage 2 application. We use two CICS regions to install the Stage 2 scenario.

On region IV3A69A3, we run the OSGi Java image handler program. The Java application requires a CICS JVMSERVER and a BUNDLE resource. In our environment, we use the following resources:

- JVMSERVER=JVMSERVX
- BUNDLE=IMGHNDL

We describe how to bundle the Java application using the IBM CICS Explorer® later on.

On CICS region IV3A69A4, we run the CICS Liberty profile servlet to get access to the catalog manager.

The servlet requires a Liberty JVMSERVER and a CICS BUNDLE definition.

1. JVMSERVER=DDWWLP
2. BUNDLE=CATLGAPP

We also describe how we use the IBM CICS Explorer to bundle the catalog server later on.

Requirements

We used CICS Transaction Server V5.2, which is required to run the Liberty profile servlet and, to deploy the servlet, we used IBM CICS Explorer V5.2.0.

The image handler program can be installed on any supported CICS version. With CICS Transaction Server Version 4.2 and later, it must be installed as an OSGi bundled application. We used a separate CICS region to run the image handler program, which can be any supported version.

Figure 5-14 shows an overview of the applications and CICS resource definitions.

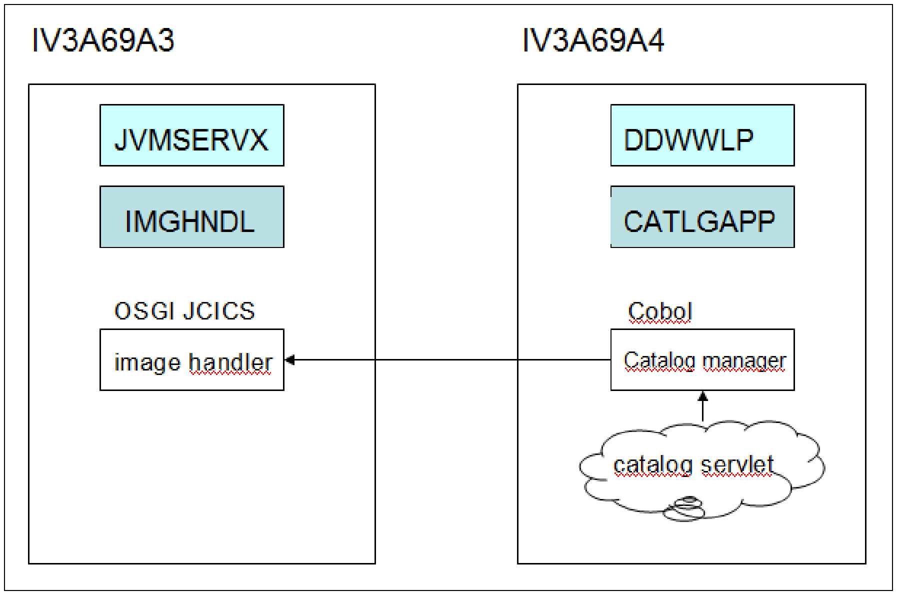


Figure 5-14 Required resources to run the stage 2 application

5.2.2 JCICS image handler program installation

The image handler program is written in Java, and uses the JCICS class library. It is linked to a channel that must have the following containers:

- ref-no

  The reference number of the item. This is used to find the corresponding `.gif` file for the item, for example, `0010.gif`.

- img-dir

  The HFS directory where item images are stored. The value for this is configured in the COBOL copybook DFH0XS02.

Setting up the image handler in CICS Transaction Server Version 4.1

Here are the required steps for setting up this program:

1. Add `saz099.catalog.jar` to the Java class path in your CICS Java virtual machine (JVM) Profile.
2. Define and install the following CICS resource:

   ```
   PROGRAM(CTLGSERV)
   GROUP(CATALOG)
   JVM(Yes)
   JVMClass(com.ibm.itso.saz099.catalog.server.ImageRetriever)
   ```

In CICS Transaction Server Version 4.2 and later, you need to install JCICS Java programs in OSGi Bundles. We used the IBM CICS Explorer to create an OSGi Bundle project, and then created a CICS Bundle that includes the OSGi Bundle project. This enables the image handler classes within the bundle to run in a CICS JVMServer.

In the following section, we describe how to set up the image handler Java application in CICS Transaction Server Version 5.2.

Setting up the image handler in CICS Transaction Server Version 5.2

See section 5.2.8, “Workspace setup for developing OSGi servlets and JSP” on page 203 for details about setting up your CICS Explorer workspace. After your CICS Explorer workspace is set up, you can start to create an OSGi project that contains the image handler JCICS program.

You must open the plug-in Development Perspective to perform the following steps:

1. From the Package Explorer view, right-click in an open area and select **New → plug-in Project**.

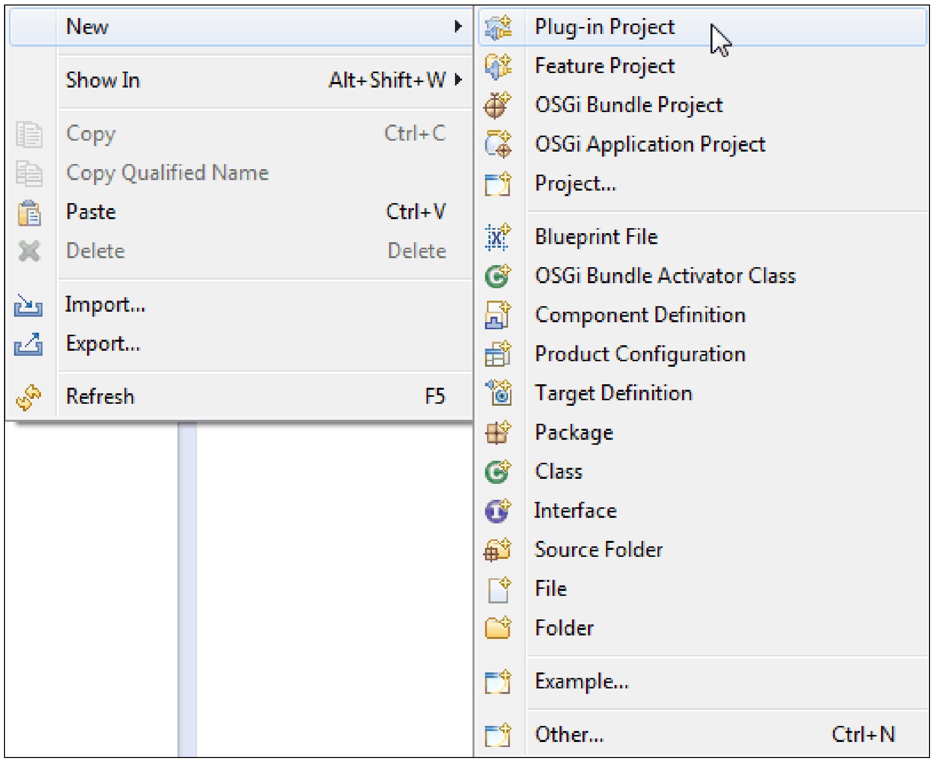
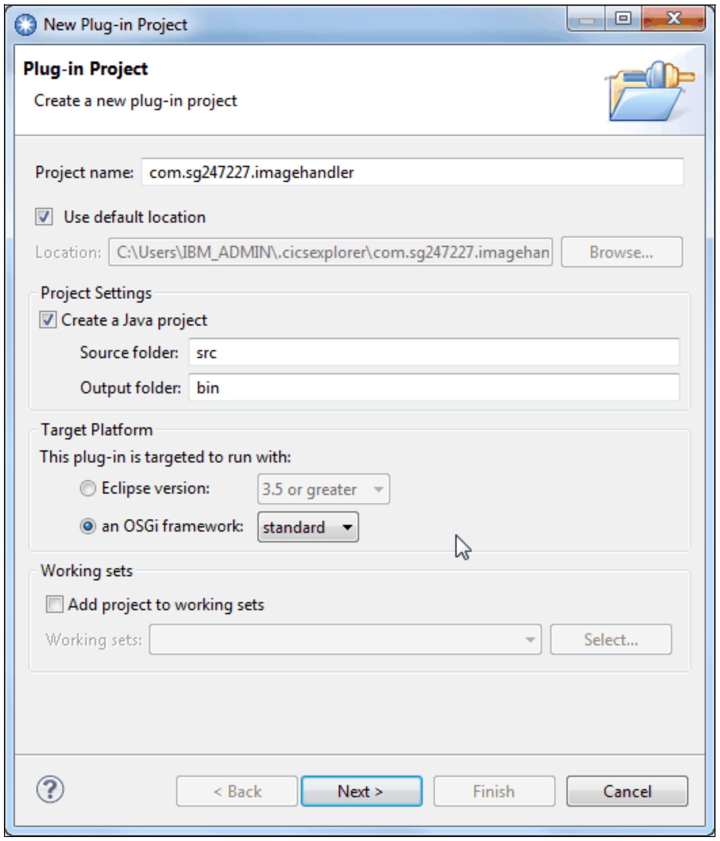


Figure 5-15 Create a new plug-in project

2. In Figure 5-15 on page 182, click **New** → **plug-in Project**. You see a window similar to Figure 5-16. We used `com.sg247227.imagehandler` as the name of the project. We chose to run with a standard OSGi framework.


New Plug-in Project

Plug-in Project

Create a new plug-in project

Project name: com.sg247227.imagehandler

[✓] Use default location

Location: C:\Users\IBM_ADMIN\.cicsexplorer\com.sg247227.imagehan

Project Settings

[✓] Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

( ) Eclipse version: 3.5 or greater

(●) an OSGi framework: standard

Working sets

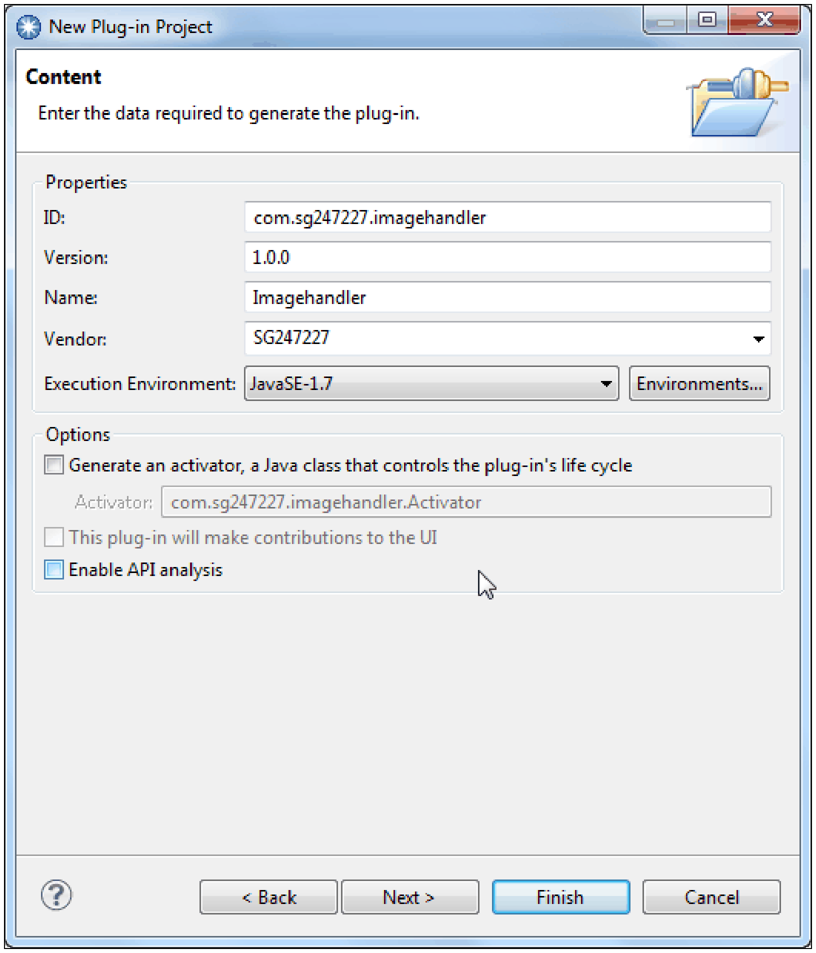
[ ] Add project to working sets

Working sets:

Figure 5-16 Create new plug-in Project

3. Click **Next** to open the New Plug-in Project Content window (Figure 5-17). It is important to remove the qualifier. The version we used is 1.0.0 without any qualifier. We also had to clear the **Generate an activator, a Java class that controls the plug-in's lifecycle** option.

   We chose an execution environment of **JavaSE1.7**, which is the default for CICS Transaction Server version 5.2. You can modify the execution environment if you run a different version of CICS Transaction Server.

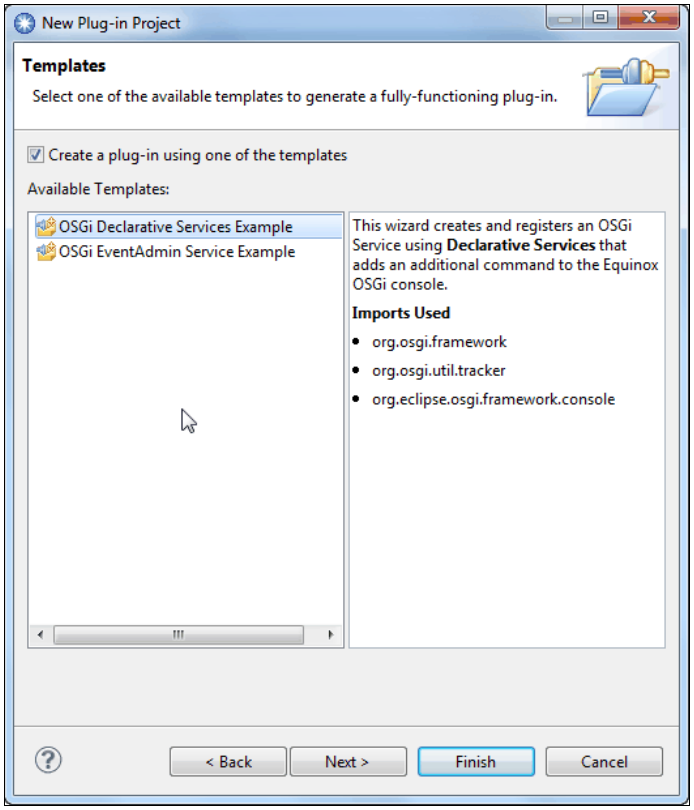


ID: com.sg247227.imagehandler
Version: 1.0.0
Name: Imagehandler
Vendor: SG247227
Execution Environment: JavaSE-1.7

☐ Generate an activator, a Java class that controls the plug-in's life cycle
Activator: com.sg247227.imagehandler.Activator
☐ This plug-in will make contributions to the UI
☐ Enable API analysis

Figure 5-17 New Plug-in Project content window

4. Click **Next** to get to the New plug-in Project Templates window (see Figure 5-18).



Templates

Select one of the available templates to generate a fully-functioning plug-in.

☑ Create a plug-in using one of the templates

Available Templates:

- OSGi Declarative Services Example
- OSGi EventAdmin Service Example

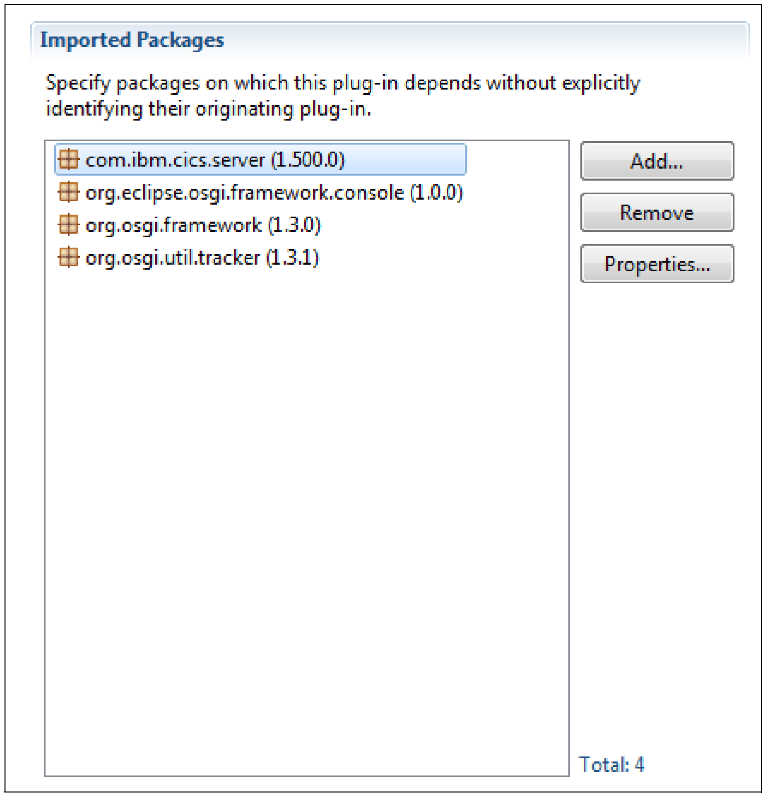
This wizard creates and registers an OSGi Service using **Declarative Services** that adds an additional command to the Equinox OSGi console.

Imports Used

- org.osgi.framework
- org.osgi.util.tracker
- org.eclipse.osgi.framework.console

Figure 5-18 New plug-in Project Templates

5. We selected OSGi Declarative Services and clicked **Next**. Note that the manifest editor opens automatically see Figure 5-19 on page 186.


Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

- com.ibm.cics.server (1.500.0)
- org.eclipse.osgi.framework.console (1.0.0)
- org.osgi.framework (1.3.0)
- org.osgi.util.tracker (1.3.1)

Figure 5-19 Imported packages

6. Our image handler application contains CICS API commands, so we needed to add a CICS dependency.

   a. In the manifest editor (Figure 5-20 on page 187), we clicked the **Dependencies** tab (along the bottom).

   b. Under Imported Packages, we clicked the **Add** button.

   c. Then, in the Package Selection dialog, we selected `com.ibm.cics.server (1.500.0)`, and clicked **OK**. Save and close the manifest editor.

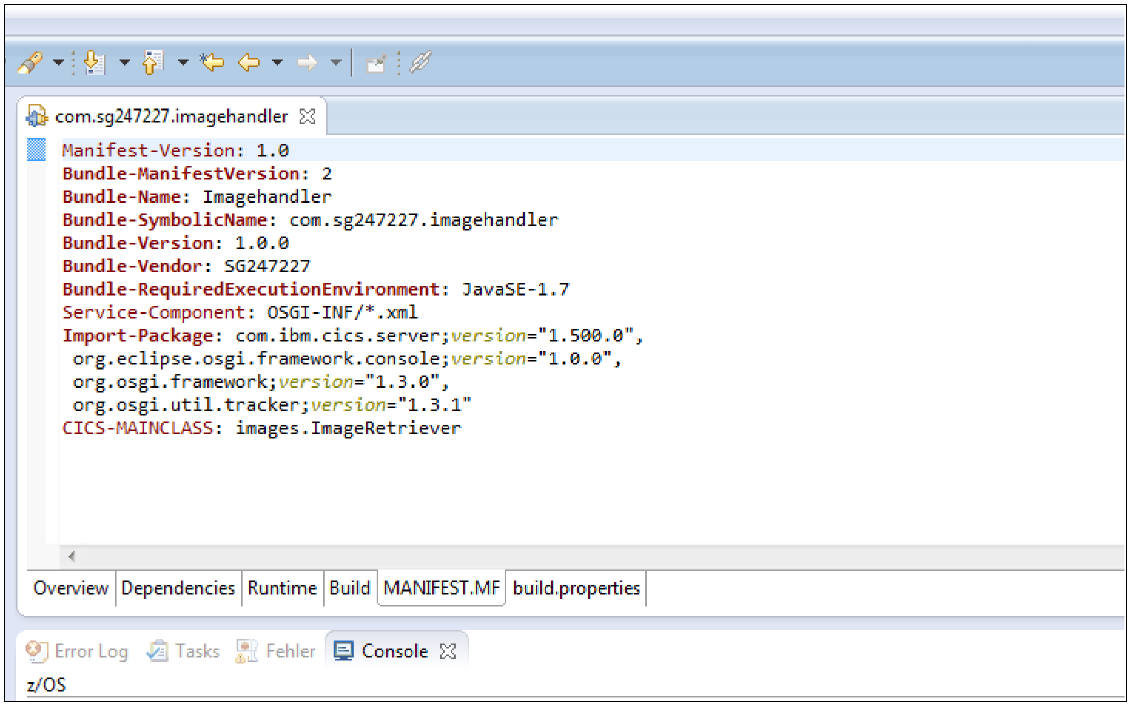
   Now we are ready to create a new package that contains the actual Java image handler program.

7. Right-click **com.sg247227.imagehandler** → **src**. Select **New** → **Package**.

8. On the New Java Package window, insert the package name (`images`).

9. Click **Finish** to create the package.
10. Next, we created the Java class for the image handler program. In the additional material, we supplied the `Imageretriever.java` file. We opened the file using Windows Notepad and press Ctrl+A and Ctrl+C to copy the contents to the clipboard.
11. After that, press Ctrl+V to paste the Java program to your `com.sg247227.imagehandler.src.images` package.
12. In the Package Explorer view, expand the `com.sg247227.imagehandler\WebContent\META-INF` directory and double-click the `MANIFEST.MF` file (this is the project's manifest file). We click the **MANIFEST.MF** tab (along the bottom).
13. We had to add the Java class name manually in the manifest. We inserted `CICS-MAINCLASS: images.ImageRetriever`.

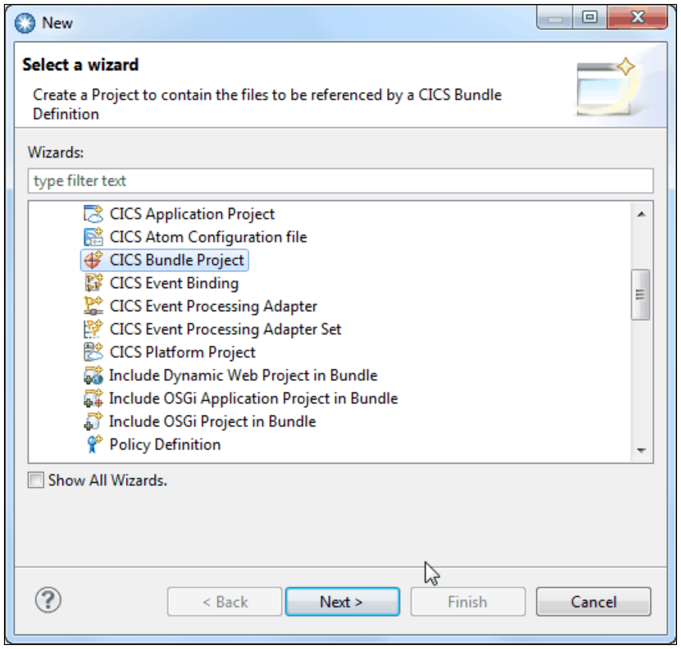
Note: The last statement of the manifest must include a line break. Therefore, we added a new empty line before we saved the manifest. See Figure 5-20, which shows our manifest.



```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Imagehandler
Bundle-SymbolicName: com.sg247227.imagehandler
Bundle-Version: 1.0.0
Bundle-Vendor: SG247227
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Service-Component: OSGI-INF/*.xml
Import-Package: com.ibm.cics.server;version="1.500.0",
 org.eclipse.osgi.framework.console;version="1.0.0",
 org.osgi.framework;version="1.3.0",
 org.osgi.util.tracker;version="1.3.1"
CICS-MAINCLASS: images.ImageRetriever
```

Figure 5-20 Manifest editor

14. From the Package Explorer, right-click and select **New** → **Other** → **CICS Resources** → **CICS Bundle Project** and click **Next**, as shown in Figure 5-21.


New

Select a wizard

Create a Project to contain the files to be referenced by a CICS Bundle Definition

Wizards:

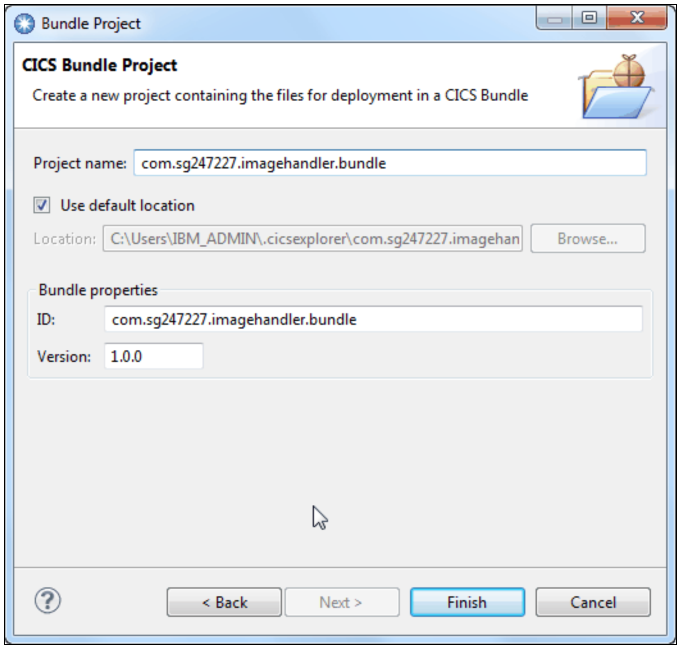
type filter text

- CICS Application Project
- CICS Atom Configuration file
- CICS Bundle Project
- CICS Event Binding
- CICS Event Processing Adapter
- CICS Event Processing Adapter Set
- CICS Platform Project
- Include Dynamic Web Project in Bundle
- Include OSGi Application Project in Bundle
- Include OSGi Project in Bundle
- Policy Definition

Show All Wizards.

Figure 5-21 Create CICS Bundle Project

15.We used the name `com.sg247227.imagehandler.bundle`, as shown in Figure 5-22. Click **Finish**.


Bundle Project

CICS Bundle Project

Create a new project containing the files for deployment in a CICS Bundle

Project name: com.sg247227.imagehandler.bundle

☑ Use default location

Location: C:\Users\IBM_ADMIN\.cicsexplorer\com.sg247227.imagehan

Bundle properties

ID: com.sg247227.imagehandler.bundle

Version: 1.0.0

Figure 5-22 Create CICS Bundle Project

The CICS Bundle has now been created. The idea was to include the OSGi `com.sg247227.imagehandler` project to the CICS Bundle that we just created.

16.From the Package Explorer, right-click the `com.sg247227.imagehandler.bundle` project.

17. Select **New** → **Other** → **CICS Resource** → **Include OSGi Project in Bundle**, as shown in Figure 5-23. Click **Next**.

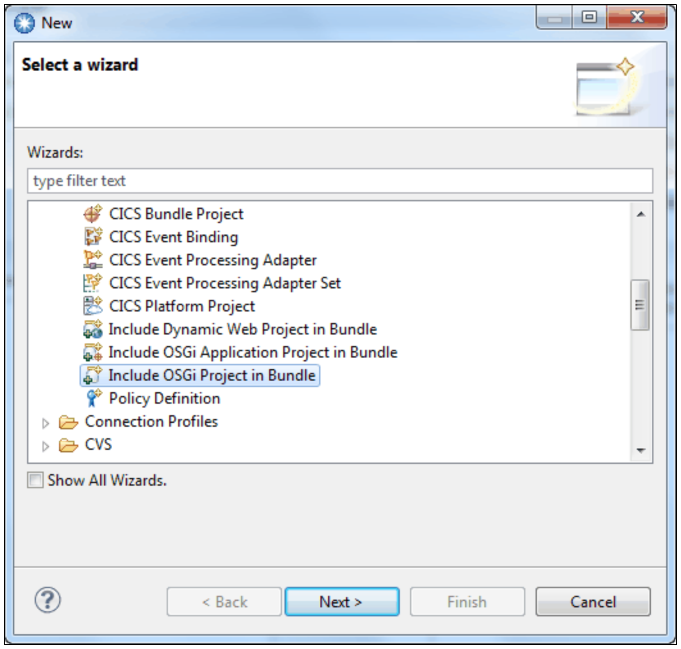
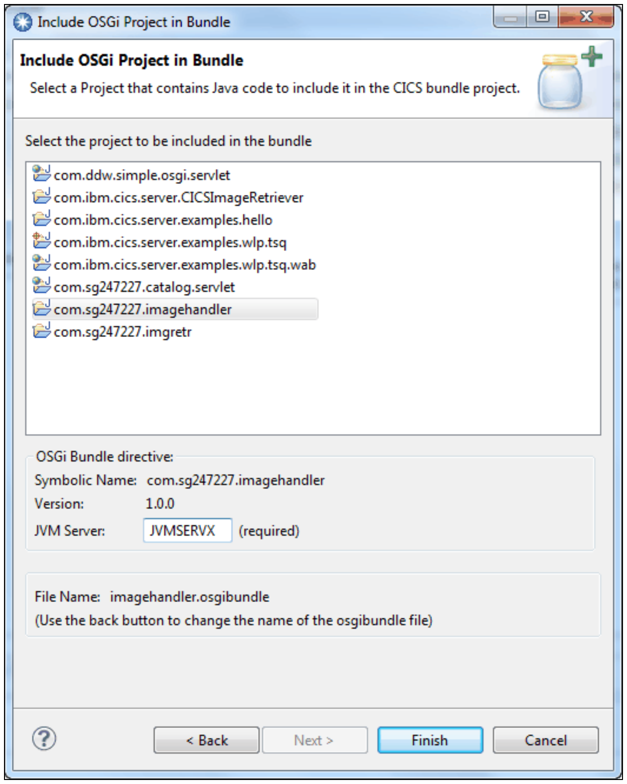


Figure 5-23 Include OSGi Project in Bundle

18. We selected the `com.sg247227.imagehandler` OSGi project. We also specified the name of a JVMSERVER resource definition. We wanted to install the image handler JCICS program on another CICS region that uses a different JVMSERVER. Therefore, we typed `JVMSERVX`, as shown in Figure 5-24.

19. Click **Finish** to include the OSGi project to the CICS Bundle.

    Now, we can transfer the CICS Bundle to the z/OS UNIX System Services file system.


Include OSGi Project in Bundle

Include OSGi Project in Bundle

Select a Project that contains Java code to include it in the CICS bundle project.

Select the project to be included in the bundle

- com.ddw.simple.osgi.servlet
- com.ibm.cics.server.CICSImageRetriever
- com.ibm.cics.server.examples.hello
- com.ibm.cics.server.examples.wlp.tsq
- com.ibm.cics.server.examples.wlp.tsq.wab
- com.sg247227.catalog.servlet
- com.sg247227.imagehandler
- com.sg247227.imgretr

OSGi Bundle directive:

Symbolic Name: com.sg247227.imagehandler

Version: 1.0.0

JVM Server: JVMSERVX (required)

File Name: imagehandler.osgibundle

(Use the back button to change the name of the osgibundle file)

Figure 5-24 Include OSGi project to the CICS bundle

5.2.3 Define a JVMSERVER definition in CICS TS

In this section, we create a CICS JVMSERVER definition in CICS Transaction Server. Depending on your preference, you can use the IBM CICS Explorer or the CICS-supplied CEDA transaction to create the definition.

The following steps can be used to create the JVMSERVER definition, using CEDA:

1. Copy the Liberty profile JVMProfile into your IBM z/OS UNIX System Services files. You can find the JVMProfile that we used for the catalog servlet in the additional material. The name of the file is `DFHOSGI.jvmprofile`. We used the CICS-supplied directory to store the JVMProfile, as shown in Example 5-20.

   Example 5-20 JVMProfile directory

   ```
   File  Directory  Special_file  Commands  Help
   ....
   ....
    EUID=0  /usr/lpp/cicsts/cicsts52/JVMProfiles/
      Type  Filename
    _ Dir   .
    _ Dir   ..
    _ File DFHOSGI.jvmprofile
   ```

2. We used the following parameters as described in Example 5-21 to create a JVMSERVER definition. We used CEDA to install the definition.

   Example 5-21 CEDA JVMSERVER definition

   ```
   OBJECT CHARACTERISTICS                                        CICS
   RELEASE = 0690
    CEDA  View JVmserver( JVMSERVX )
     JVmserver     : JVMSERVX
     Group        : LIBPROF
     DEScription   : JVM SERVER DEFINITION FOR OSGi BUNDLES
     Status       : Enabled         Enabled ! Disabled
      Jvmprofile     : DFHOSGI
   (Mixed Case)
     Lerunopts     : DFHAXRO
     Threadlimit    : 015             1-256
    DEFINITION SIGNATURE
     DEFinetime    : 10/07/14 10:48:24
     CHANGETime    : 10/07/14 16:30:46
     CHANGEUsrid    : CICSUSER
     CHANGEAGEnt    : CSDApi          CSDApi ! CSDBatch
      CHANGEAGRel    : 0690
   ```

5.2.4 Transfer the CICS Bundle to z/OS UNIX System Services

When deploying the image handler JCICS program to CICS, you define a CICS BUNDLE and place your image handler OSGi Project into the BUNDLE. Then, you export the CICS Bundle from your workstation to z/OS where it can be accessed by your CICS region.

To export the CICS Bundle, we created a directory in the z/OS UNIX file system. We used `<`*`home_directory`*`>/cicslab/bundles/`.

Export your CICS Bundle to UNIX System Services on z/OS

To export your CICS Bundle, complete the following steps:

1. In the Java Platform, Enterprise Edition (Java EE) perspective, in the Enterprise Explorer view, right-click your `com.sg247227.imagehandler.bundle` project and select **Export Bundle Project to z/OS UNIX File System**.
2. In the Export to z/OS UNIX File System dialog, click the **Export to a specific location in the file system** radio button, and click **Next**.
3. In the Export Bundle dialog, verify the following information:
   - In Bundle project, it should say `com.sg247227.imagehandler.bundle`.
   - In the parent directory, ensure that it says `<`*`home_directory`*`>/cicslab/bundles/`.
   - In the Bundle directory, ensure that it says `<`*`home_directory`*`>/cicslab/bundles/com.sg247227.imagehandler.bundle_1.0.0`, as shown in Example 5-22.

   Example 5-22 Exported Bundles

   ```
   EUID=0   /u/klein4/cicslab/bundles/
     Type  Filename
   _ Dir   .
   _ Dir   ..
   _ Dir   com.sg247227.catalog.cicsbundle_1.0.0
   _ Dir   com.sg247227.imagehandler.bundle_1.0.0
   ```

4. Still on the Export Bundle page, click **Finish**.

5. After that we used the information in Example 5-23 to create a BUNDLE definition in CICS Transaction Server by using the CEDA Transaction.This can also be done with the IBM CICS Explorer.

Example 5-23 CEDA BUNDLE definition

```
OBJECT CHARACTERISTICS                                          CICS
RELEASE = 0690
 CEDA  View Bundle( IMGHNDL  )
  Bundle        : IMGHNDL
  Group         : LIBPROF
  DEScription    :
  Status        : Enabled          Enabled ! Disabled
   BUndledir      :
/u/klein4/cicslab/bundles/com.sg247227.imagehandler.bundle
  (Mixed Case)   : _1.0.0
                :
                :
                :
  BAsescope      :
  (Mixed Case)   :
                :
                :
                :
 DEFINITION SIGNATURE
  DEFinetime    : 10/30/14 10:57:42
  CHANGETime    : 10/30/14 10:58:30

                                                          SYSID=69A3
APPLID=IV3A69A3
```

6. After that, we installed the BUNDLE definition. Both the JVMSERVER and BUNDLE definition should be ready to use and should show status enabled.

   You can also check the messages shown in Example 5-24 to make sure that the JVMSERVER and the BUNDLE definition are ready to use.

Example 5-24 JVMServer and Bundle successfully installed

```
DFHSJ0915 10/30/2014 11:06:52 IV3A69A3 CICSUSER JVMSERVER JVMSERVX
is now enabled and is ready for use.
DFHRL0132 I 10/30/2014 11:06:57 IV3A69A3 CEDA All defined resources
for BUNDLE IMGHNDL are now in the enabled state.
```

Preparing files in the additional material

The supplied additional material contains five images for use with the scenario. These are `0010.gif`, `0020.gif`, `0030.gif`, `0040.gif`, and `NoImage.gif`. You must place these in a directory on UNIX System Services, and the directory must have appropriate permissions to allow CICS to access them.

You must insert the explicit path of this directory into the IMG-DIR-CONTENTS field in the COBOL copybook DFH0XS02. This value is picked up when the catalog manager program EFH2XCMN is linked and compiled.

The path in the IMG-DIR-CONTENTS field must also be updated in two further files to provide for the image support. The servlet uses a Fileoutputstream to store the images from the img-dir container to an HFS directory. We do not use a server path to store the images.

We want to display the dynamic contents of the images from the HFS path that you specify, which can be the same path that you specify in the IMG-DIR-CONTENTS field. We use a URIMAP and TCPIPSERVICE definition in the IV3A69A4 region to supply the dynamic contents for the JSP file.

The following two files must be updated:

- The `LinkProgOSGI.java` file. Update the fileoutputstream statement. insert your path that should contain the `ximage.gif` dynamic image file:

  ```
  FileOutputStream outstr = new FileOutputStream ("/your
  path/ximage.gif");
  ```

- The `OrderItem2.JSP` file. Update the link to the directory that contains the dynamic image file. We insert the z/OS Internet Protocol (IP) address of the image handler CICS region and a port that is specified in a TCPIPSERVE definition. Note that the image that contains the dynamic contents does not need to be specified:

  ```
  <IMG src="http://9.155.11.209:4556/your path/" width="109"
  height="110"></DIV>
  ```

In the additional material, we provide a DFHCSDUP job that contains SYSIN statements to define the URIMAP and TCPIPSERVICE definitions. It also contains all of the required CEDA definitions for the project. Also see the readme file that we supply to get detailed instructions about what to modify.

5.2.5 The Liberty profile servlet access to the catalog manager

CICS supports a subset of the Liberty profile, including servlets and JSP. See the CICS Transaction Server version 5.2 Knowledge Center for a list of the CICS supported parts of the Liberty profile:

`http://www-01.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.java.doc/topics/liberty_features.html`

To provide a browser-based front end to the catalog manager, we used a CICS Liberty servlet. Within the servlet, we can issue a link to the catalog server using channel and containers directly.

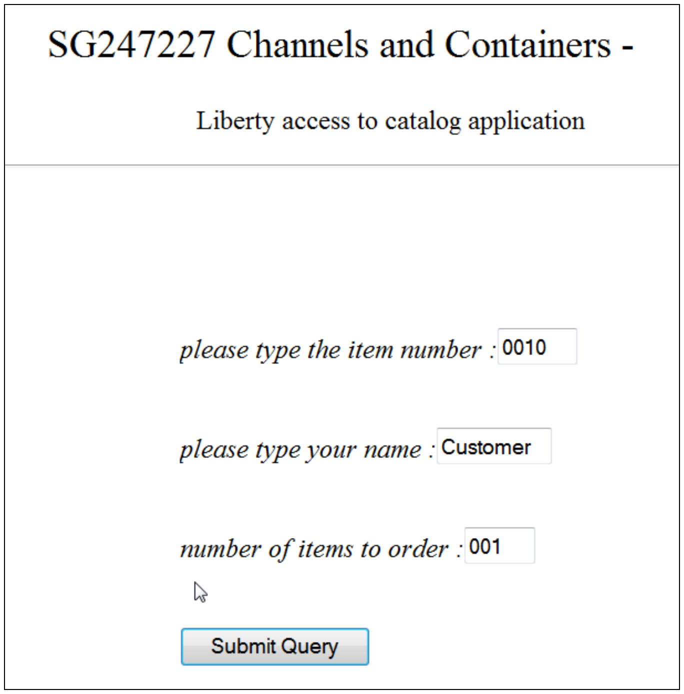
The servlet demonstrates how to develop a front end that can be used to place an order from the catalog. We used a Hypertext Markup Language (HTML) page to get the required data to place the order, as shown in Figure 5-25 on page 197:

- Item number
- A name
- Number of items

The catalog manager requires the following information in the channel and containers that get passed to the program:

- The container `request-type` contains the order request, which is 01ORDR.
- The container `order-item` contains the customer name, department, item, and number of items.

On return, we expect the result in the container `return-code-msg`.


SG247227 Channels and Containers -

Liberty access to catalog application

please type the item number : 0010

please type your name : Customer

number of items to order : 001

Figure 5-25 OrderItem HTML page

The JCICS snippets in Example 5-25 on page 198 show how we stored the required data to the containers, how we created the channel, and how we linked to the catalog manager. As you can see, we used two strings to store the data to the containers request-type and order-item. The strings `ca2Name` and `ca3Name` were created from the input fields of the `OrderItem.htm` file. To get the data to the container, we converted them to type `byte[]`.

Note that we put the data to the container using the following statements:

```
cont2input = ca2Name.getBytes();
reqtype.put(cont2input , myccsid);
```

We used two parameters on the `put` method. The `cont2input`, which is the data that is put to the container, and a string, which is the coded character set identifier (CCSID). We defined a string for the CCSID, such as `String myccsid = "1208";`.

When we issue the `put` method without `myccsid`, the catalog manager gets American Standard Code for Information Interchange (ASCII) data from the container. If we use the CCSID on the container `put` method, automatic code page conversion to CICS local CCSID takes place.

If you omit the CCSID on the container `put` method, it is not possible to convert the data on the `get` container command in the CICS COBOL program. We tried to use INTOCCSID and INTOCODEPAGE on the `get` container command in the catalog manager, which results in the following response:

```
CODEPAGEERR
RESP2 value 3
```

The data was created with a data type of `BIT`. Code page conversion is not possible. The data was returned without any code page conversion. Type of `BIT` is the default, so we suggest using the CCSID on the container `put` method, as shown in Example 5-25.

Example 5-25 JCICS snippets

```
// create the IMG-CHANNEL channel

           Channel imgchannel = t.createChannel("cmn-channel");

        // create containers

           Container reqtype = imgchannel.createContainer("request-type");

         ca2Name = "01ORDR";

              cont2input = ca2Name.getBytes();

              reqtype.put(cont2input , myccsid);


           Container orderitem = imgchannel.createContainer("order-item");
           byte[] cont3input = ca3Name.getBytes();

           orderitem.put(cont3input , myccsid);


        // Link to the catalog program, passing the channel
           Program p = new Program();
           p.setName("EFH2XCMN");
           p.link(imgchannel);

         Container returnmsgcont = imgchannel.getContainer("return-code-msg");
          byte[] catalogmsg = returnmsgcont.get(myccsid);

            String  resultmsg = new String(catalogmsg, "UTF-8");

        result = "Catalog Server returned: " +resultmsg + "for item number "
+ca1Name + " ";
```

On return from the catalog manager, we get the result back from container return-code-msg. The following statements are used to get the data from the container:

```
Container returnmsgcont = imgchannel.getContainer("return-code-msg");
byte[] catalogmsg = returnmsgcont.get(myccsid);
```

The data does not get converted from Extended Binary Coded Decimal Interchange Code (EBCDIC) to ASCII automatically. We use the INTOCCSID string on the container get method to convert the data with the following result:

```
CODEPAGEERR
RESP2 value 3
```

The data was created with a data type of `BIT`. Code page conversion is not possible. The data was returned without any code page conversion. The `BIT` type is also the default when we issue the **`put container`** command from the catalog manager. Therefore, we use the **`CHAR`** parameter on the **`put container`** command in the Cobol catalog manager, as shown in Figure 5-26.

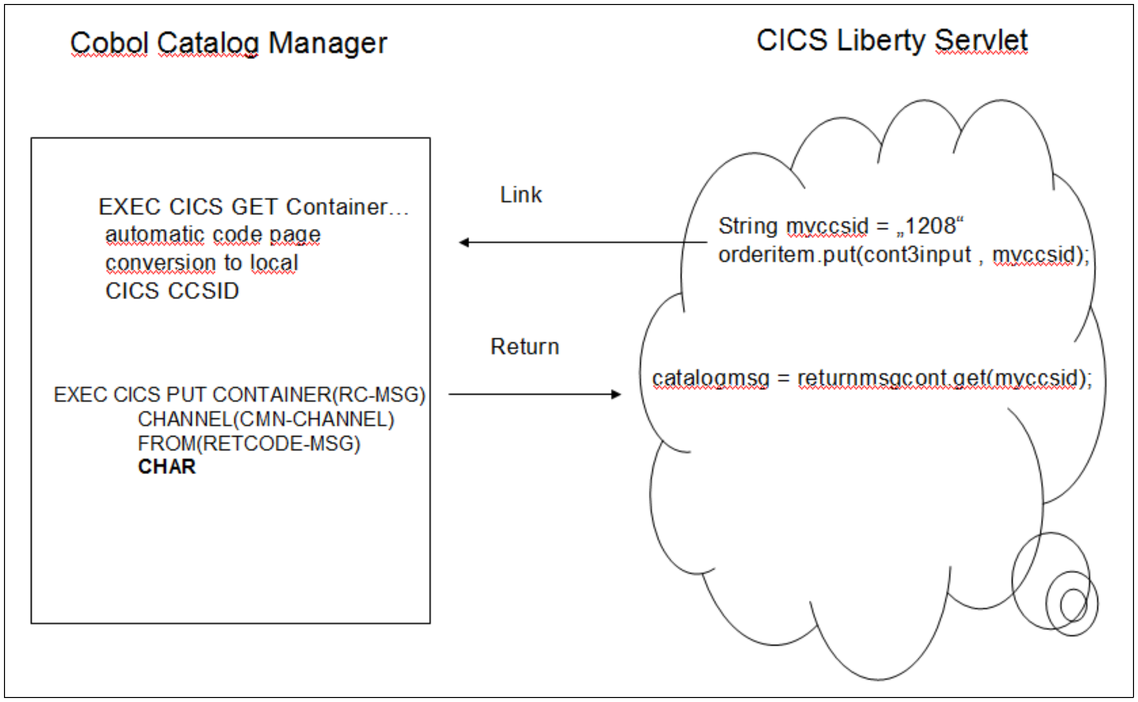

Cobol Catalog Manager
CICS Liberty Servlet
EXEC CICS GET Container… automatic code page conversion to local CICS CCSID
EXEC CICS PUT CONTAINER(RC-MSG) CHANNEL(CMN-CHANNEL) FROM(RETCODE-MSG) CHAR
Link
Return
String myccsid = „1208“
orderitem.put(cont3input , myccsid);
catalogmsg = returnmsgcont.get(myccsid);

Figure 5-26 Code page conversion using the catalog servlet

See 2.6, “Data conversion and code page conversion” on page 44 for additional information about code page conversion using channel and containers.

We put the JSP pages and Java packages in Appendix B, “Additional material” on page 239. You can look at them to see how we saved the data from the input fields, and how the logic works. To get the catalog servlet started, you can read through the next two sections, which describe how to run and deploy the servlet.

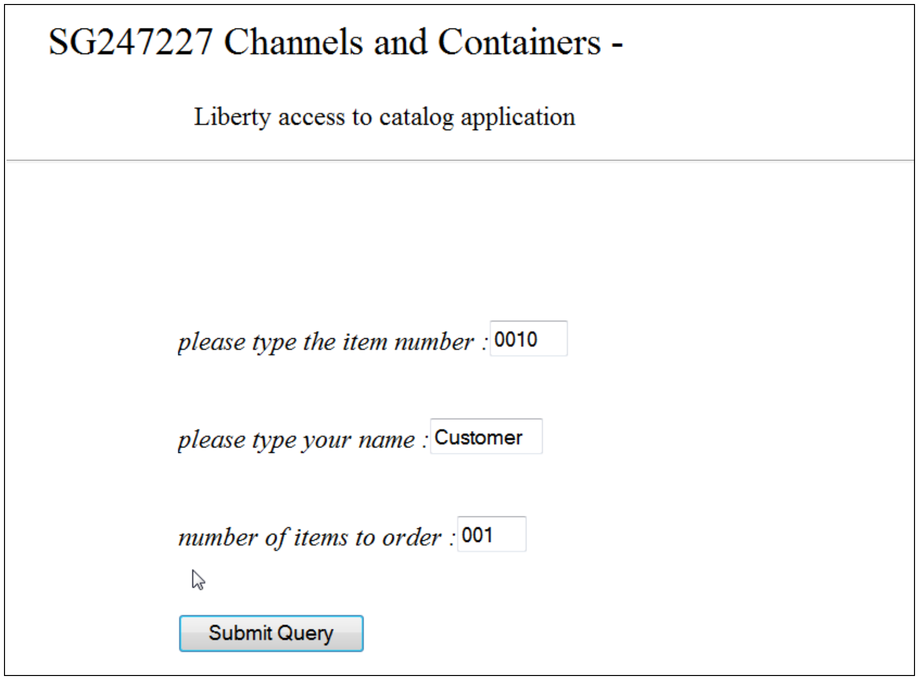
5.2.6 Running the catalog servlet

To run the catalog servlet, complete the following steps:

1. We use the following URL to start the catalog servlet application:

   ```
   http://9.155.11.209:4555/catalog/OrderItem.htm
   ```

2. The port we use is defined in the JVMProfile used for JVMServer DDWWLP. Note that there is no TCPIPSERVICE definition required to define the port to CICS. Next to the port, we specify the path, which is the Context root. We specified it on the web Module page of the New OSGi Bundle Project dialog.
3. After that, we specify the name of the first HTML page of the web application. We start the URL. The first page of the application starts, as shown in Figure 5-27.


SG247227 Channels and Containers -

Liberty access to catalog application

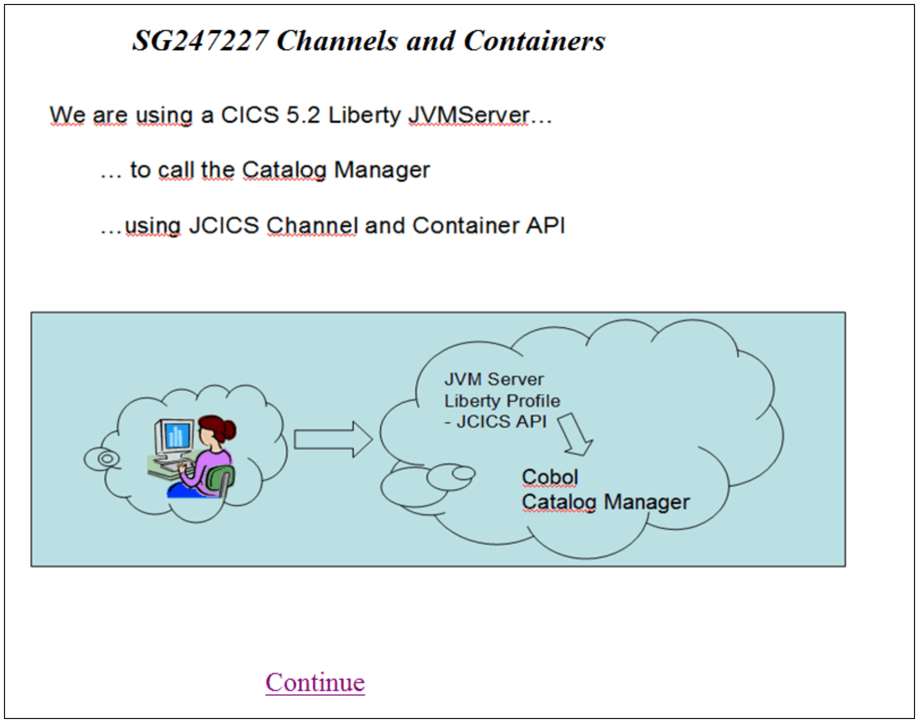
please type the item number : 0010

please type your name : Customer

number of items to order : 001

Figure 5-27   Entry page of the catalog servlet

4. The input fields are already set. You can enter the item number, name, and number of items if you want. We click **Submit Query** to get to the next page, which is shown in Figure 5-28.


SG247227 Channels and Containers

We are using a CICS 5.2 Liberty JVMServer…

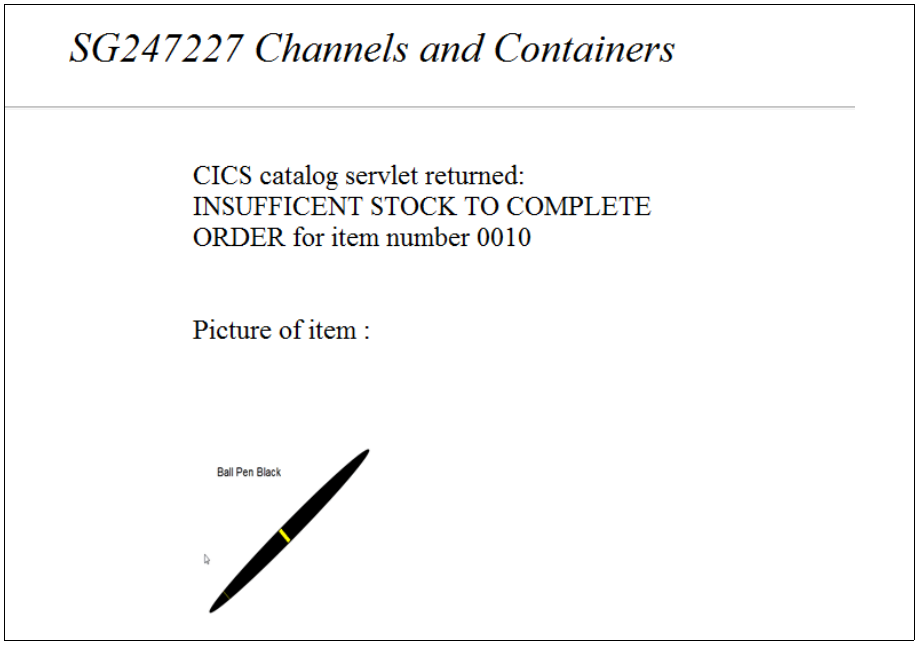
… to call the Catalog Manager

…using JCICS Channel and Container API

Continue

Figure 5-28 Info page of catalog servlet

5. The next page opens, which provides some information about the scenario. Next, click **Continue** and you see a window similar to Figure 5-29.



SG247227 Channels and Containers

CICS catalog servlet returned:
INSUFFICENT STOCK TO COMPLETE
ORDER for item number 0010

Picture of item :

Figure 5-29 Result page of catalog servlet

6. The resulting page displays and informs us that there is insufficient stock to complete the order. The image of the item is displayed as well.

   The image handler retrieved the image file from the z/OS UNIX file system directory and passed it back to the COBOL catalog manager in a container called gif-data. On return, however, the catalog manager returned the image to the servlet in the gif-data container.

   The servlet uses a fileoutputstream to store the image to a z/OS UNIX File directory that can be accessed by the JSP page we show in Figure 5-29.

5.2.7 Deploying the catalog servlet

In this section, we describe how to package the catalog servlet as an OSGi bundle in CICS Transaction Server V5.2. CICS supports a subset of the Liberty profile, including servlets and JSP. To run the servlet, you need to deploy the OSGi Bundle in CICS Transaction Server V5.2.

Requirements to perform the following steps:

We used the IBM CICS Explorer V5.2.0.0 to create and package the application as a CICS Bundle. To get instructions about how to set up the IBM CICS Explorer, you can use the following link:

http://www-01.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.java.doc/topics/installingthelibertyprofile.html?lang=en

Alternatively, you can use the following path to the CICS Transaction Server Version 5.2 Knowledge Center:

CICS Transaction Server 5.2.0 → Developing applications → Developing Java applications for CICS → Developing Java web applications to run in Liberty JVM server → Setting up the development environment

You also need to install the CICS Explorer software development kit (SDK) for servlets within the Explorer. This SDK contains the CICS Explorer parts needed to write Liberty applications. You also need to install additional tooling from IBM WebSphere. To get the support, you can install the following packages from the Eclipse Market Place:

- IBM CICS SDK for servlet and JSP support V5.2.0
- IBM WebSphere Application Server V8.5.5 Developer Tools for Eclipse Juno and Kepler

5.2.8 Workspace setup for developing OSGi servlets and JSP

You need to configure your workstation for developing OSGi-based servlets and JSP. We will change the OSGi support that is automatically added when you create a project.

This only has to be done one time per workspace. However, if you change the OSGi support to something else, or if you use a different workspace, you must change the OSGi support back to support CICS technology-based OSGi applications.

As described in the previous section, you need the IBM CICS Explorer 5.2.0.0 (includes the Java EE version of Eclipse), plus the WebSphere Application Server Development tools. See the CICS TS V5.2 Information Center for a comprehensive listing of how to set up your Eclipse environment to develop OSGi-based servlets in CICS.

You can use the following steps to set up the workspace for developing Liberty profile applications:

1. Start the Eclipse workbench if not already started.
2. In Eclipse, from the menu bar, select **Window** → **Preferences**. On the left, select **Plug-in Development** → **Target Platform**. On the right, select **Add**.
3. From the Target Definition dialog, select the radio button next to **Template**, then use the pull-down menu to select **CICS TS V5.2 with Liberty and PHP**, then click **Next**.
4. From the second page of the Target Content dialog, click the **Finish** button.
5. Back on the Target Platform page of the Preferences dialog, select the check box next to **CICS TS V5.2 with Liberty and PHP**, then click the **OK** button.

5.2.9 Create the catalog OSGi Project

In this section, we set up the OSGi project by creating the following components:

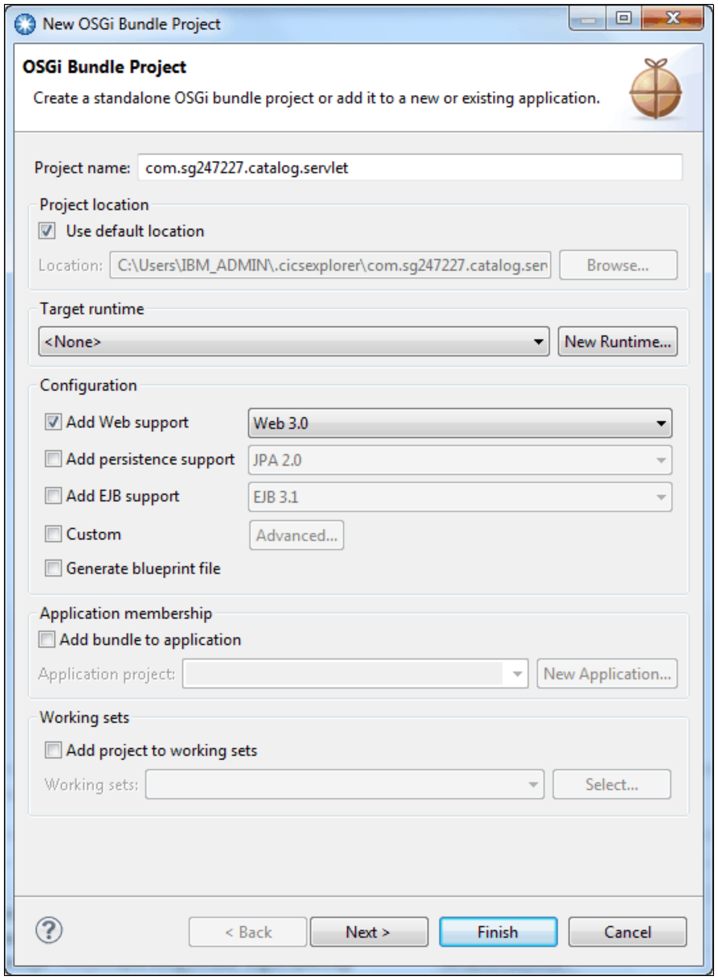
1. A plug-in Project that contains your servlet (this is the part that the user interacts with from their browser). The plug-in project contains JSP, cascading style sheets (CSS), graphics, and the JavaScript code.
2. You create an enterprise bundle archive (EBA) application bundle that points at the plug-in project containing your application.
3. A CICS BUNDLE that points to your EBA application bundle. This is the BUNDLE that you install into CICS.

The result is a CICS BUNDLE that contains an EBA, which contains a web application bundle (WAB). Your servlet is in the WAB.

Create a plug-in project

Perform the following steps:

1. Open or switch to the plug-in Development perspective.
2. From the Package Explorer view, right-click in an open area and select **New** → **OSGi Bundle Project**.
3. From the OSGi Bundle Project dialog, enter the following information, as shown in Figure 5-30 on page 205, then click the **Next** button.
   - Project name, which is `com.sg247227.catalog.servlet`.
   - **Use default location** must be selected.
   - **Add web support** must be selected.
   - We use Web3.0 support.
   - **Add bundle to application** must be cleared.
   - **Add project to working sets** must be cleared.


New OSGi Bundle Project

OSGi Bundle Project

Create a standalone OSGi bundle project or add it to a new or existing application.

Project name: com.sg247227.catalog.servlet

Project location

[✓] Use default location

Location: C:\Users\IBM_ADMIN\.cicsexplorer\com.sg247227.catalog.ser

Target runtime: <None>

Configuration

[✓] Add Web support: Web 3.0

[ ] Add persistence support: JPA 2.0

[ ] Add EJB support: EJB 3.1

[ ] Custom

[ ] Generate blueprint file

Application membership

[ ] Add bundle to application

Application project:

Working sets

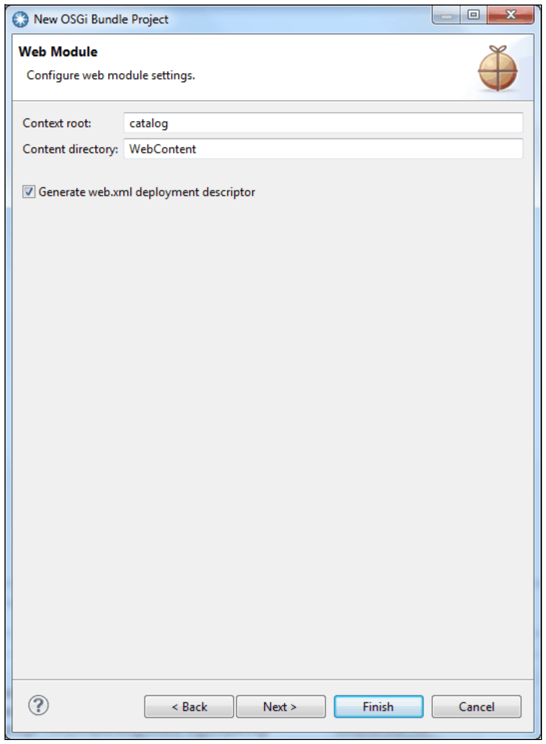
[ ] Add project to working sets

Working sets:

Figure 5-30 OSGi Bundle Project for catalog servlet

4. From the Java page of the New OSGi Bundle Project dialog, click **Next**.
5. From the Web Module page of the New OSGi Bundle Project dialog, change the Context root to `catalog`. Let the Content directory default to WebContent. Select the **Generate web.xml deployment descriptor**. Then click **Next**.
6. From the OSGi Bundle page of the New OSGi Bundle Project dialog, change the Version to 1.0.0. Also clear the box next to **Generate an activator, a Java class that controls the lifecycle of the bundle**.

7. Then click **Finish**. You see a window similar to Figure 5-31.



Web Module

Configure web module settings.

Context root: catalog

Content directory: WebContent

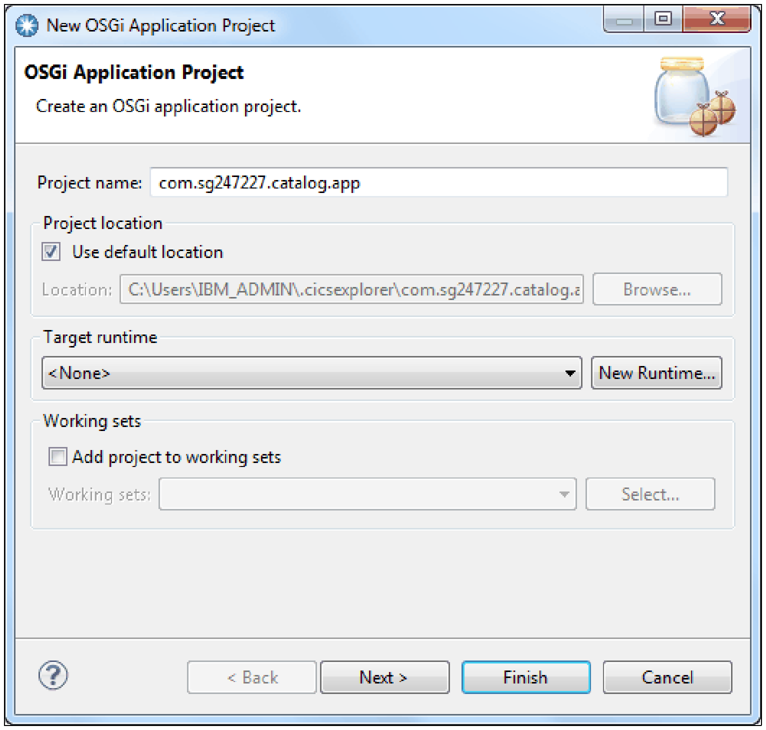
☑ Generate web.xml deployment descriptor

Figure 5-31 Context root of web Module

5.2.10 Create an EBA project to the OSGi Bundle Project

Perform the following steps to create an EBA project:

1. From the Package Explorer view, right-click in an open area and select **New** → **OSGi Application Project**.
2. From the OSGi Application Project dialog, enter the following information, as shown in Figure 5-32 on page 207, then click **Next**:
   – Project name, which is `com.sg247227.catalog.app`.
   – **Use default location** must be selected.
   – Clear **Add project to working set**.


New OSGi Application Project

OSGi Application Project

Create an OSGi application project.

Project name: com.sg247227.catalog.app

Project location

[✓] Use default location

Location: C:\Users\IBM_ADMIN\.cicsexplorer\com.sg247227.catalog.a

Target runtime

<None>

Working sets

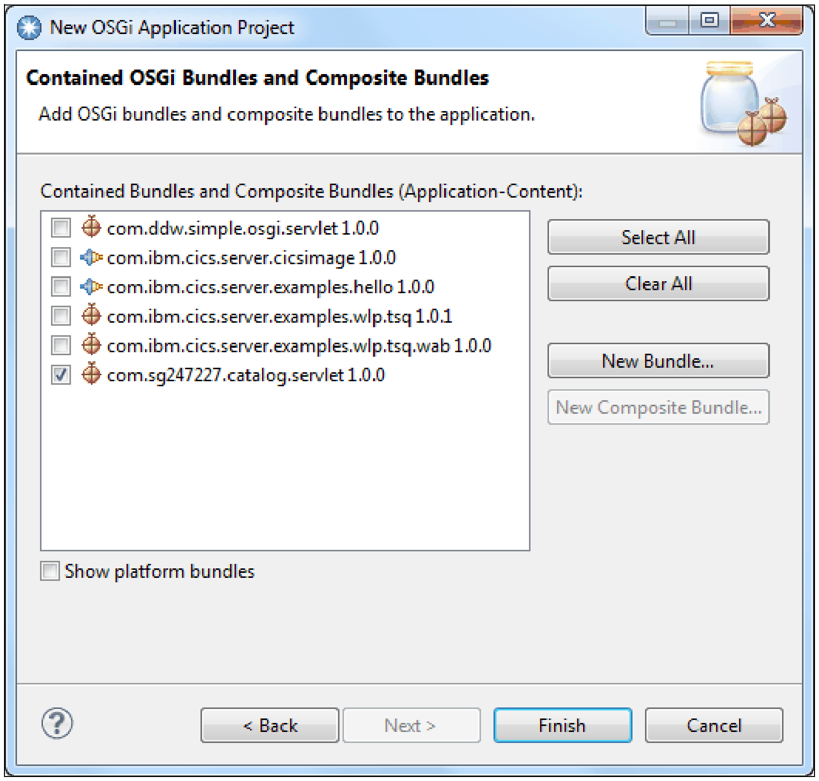
[ ] Add project to working sets

Working sets:

Figure 5-32 OSGi application project for the catalog servlet

3. From the Contained OSGi Bundles and Composite Bundles dialog, select the box next to **com.sg247227.catalog.servlet 1.0.0** (see Figure 5-33 on page 208), then click the **Finish** button. Note that you could put multiple OSGi Bundle Projects into your OSGi Application Project, but we only have one.
4. From the Package Explorer view, expand the `com.sg247227.catalog.app` project, and double-click the `APPLICATION.MF` manifest file.
5. On the Overview page of the manifest editor, toward the upper-left, change the Version to `1.0.0`.
6. Save and close the `com.sg247227.catalog.app` manifest editor.

Figure 5-33 shows the window for creating the project.


New OSGi Application Project

Contained OSGi Bundles and Composite Bundles

Add OSGi bundles and composite bundles to the application.

Contained Bundles and Composite Bundles (Application-Content):

- [ ] com.ddw.simple.osgi.servlet 1.0.0
- [ ] com.ibm.cics.server.cicsimage 1.0.0
- [ ] com.ibm.cics.server.examples.hello 1.0.0
- [ ] com.ibm.cics.server.examples.wlp.tsq 1.0.1
- [ ] com.ibm.cics.server.examples.wlp.tsq.wab 1.0.0
- [✓] com.sg247227.catalog.servlet 1.0.0

- [ ] Show platform bundles

Figure 5-33 New OSGi Application Project

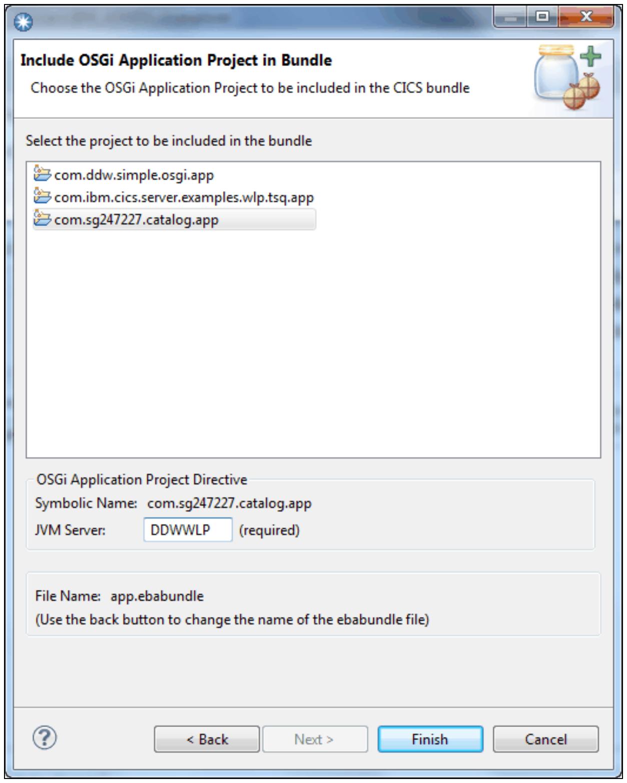
5.2.11 Create a CICS BUNDLE to the EBA project

Perform the following steps to create a CICS BUNDLE:

1. From the Package Explorer view, right-click in an open area and select **New** → **Other**, then from the Select a wizard dialog, select **CICS Resources** → **CICS Bundle Project**. Click **Next**.
2. From the CICS Bundle Project dialog, enter the following information, then click **Finish**:
   – Project name, which is `com.sg247227.catalog.cicsbundle`.
   – **Use default location** must be selected.
   – ID is com.sg247227.catalog.cicsbundle.
   – Version 1.0.0.
3. Note that the CICS bundle manifest editor opened.

4. From the CICS bundle manifest Overview view, in the Defined Resources section, click **New** and select **Include OSGi Application Project in Bundle**. Then, click **Next**.

   From the Include OSGi Project in Bundle dialog (Figure 5-34), highlight com.sg247227.catalog.app, and specify a **JVM Server** of `DDWWLP`.



Include OSGi Application Project in Bundle

Choose the OSGi Application Project to be included in the CICS bundle

Select the project to be included in the bundle

- com.ddw.simple.osgi.app
- com.ibm.cics.server.examples.wlp.tsq.app
- com.sg247227.catalog.app

OSGi Application Project Directive

Symbolic Name: com.sg247227.catalog.app

JVM Server: DDWWLP (required)

File Name: app.ebabundle

(Use the back button to change the name of the ebabundle file)

Figure 5-34 Include OSGi Application Project in Bundle

5. Still on the Include OSGi Application Project in Bundle dialog, note the verbiage in the parenthesis beneath the **JVM Server** (immediately under **File Name**). Click the **Back** button, because we are going to change the name to something other than `app.ebabundle`.
6. Still on the Include OSGi Application Project in Bundle dialog (you have changed to a different page of this wizard), change the **File name** to `com.sg247227.catalog.app.ebabundle`, then click **Finish**. You have just created the OSGi web bundle, your application bundle, and the CICS bundle for your servlet. Note that you could have put multiple EBA bundles in your CICS BUNDLE, however, we only have one.
7. Save and close the CICS bundle manifest editor.

5.2.12 Add the com.ibm.cics.server package to your OSGi project

In this section, we complete the parts of the OSGi project that you always complete for every servlet. In our servlet, we want to use JCICS channel and container commands. Therefore, the following steps can be used to see where to put imported packages, such as the `com.ibm.cics.server` package:

1. In the Package Explorer view, expand the **com.sg247227.catalog.servlet** → **WebContent** → **META-INF** directory and double-click the `MANIFEST.MF` file (this is the project's manifest file).
2. Our servlet contains CICS API commands, so we need to add a CICS dependency. In the manifest editor, click the **Dependencies** tab (along the bottom). Under Imported Packages, click **Add**. Then from the Package Selection dialog, we would select com.ibm.cics.server (1.500.0) and click **OK**.
3. Save and close the manifest editor.

5.2.13 Create the dynamic web project

In the provided Eclipse workbench, you create a Dynamic Web Project. This is the type of Eclipse project that is used to create a servlet and a JSP:

1. Right-click **com.sg247227.catalog.servlet** → **src** and select **New** → **Package**.
2. In the New Java Package dialog, under **Name**, type `com.sg247227.catalog.servlet`. Then click **Finish**.

From your desktop environment, copy the `LinkProgOSGI.java` program and paste it into your Eclipse `com.sg247227.catalog.servlet` project's `com.sg247227.catalog.servlet` package.

From your desktop environment, copy the following files and paste them into your Eclipse `com.sg247221.catalog.servlet` project's WebContent directory:

- `OrderItem.jsp` program
- `OrderItem1.jsp` program
- `OrderItem2.jsp` program
- `wsenv.gif`
- `ximage.gif`

5.2.14 Define a CICS JVMSERVER with Liberty profile

In this section, we create a CICS JVMSERVER definition in CICS Transaction Server. Depending on your preference, you can use the IBM CICS Explorer or the CEDA transaction to create the definition. The following steps can be used to create the JVMSERVER definition:

1. Copy the Liberty profile JVMProfile into your UNIX System Services files. You can find the JVMProfile that we used for the catalog servlet in the additional material. The name of the file is `DDWWLP.jvmprofile`. We used the CICS supplied directory to store the JVMProfile, as shown in Example 5-26.

   Example 5-26 JVMProfile directory

```
File  Directory  Special_file  Commands  Help
....
....
 EUID=0  /usr/lpp/cicsts/cicsts52/JVMProfiles/
   Type  Filename
 _ Dir   .
 _ Dir   ..
 _ File  DDWWLP.jvmprofile
```

2. We use the following parameters, described in Example 5-27, to create the JVMPROFILE definition.

   Example 5-27 CEDA JVMSERVER definition

```
OBJECT CHARACTERISTICS                                        CICS
RELEASE = 0690
 CEDA  View JVmserver( DDWWLP   )
  JVmserver     : DDWWLP
  Group         : LIBPROF
  DEScription   : JVM SERVER DEFINITION FOR LIBERTY PROFILE
  Status        : Enabled          Enabled ! Disabled
   Jvmprofile     : DDWWLP
(Mixed Case)
  Lerunopts     : DFHAXRO
  Threadlimit   : 015              1-256
 DEFINITION SIGNATURE
  DEFinetime    : 09/23/14 18:23:19
  CHANGETime    : 10/07/14 16:56:26
  CHANGEUsrid   : CICSUSER
  CHANGEAGEnt   : CSDApi           CSDApi ! CSDBatch
   CHANGEAGRel    : 0690
```

3. Before we install the JVMPROFILE, make sure that you have got a suitable port defined within your JVMProfile. We use the following port definition, as shown in Example 5-28. The port that we want to use must not be used somewhere else, for example in an existing Transmission Control Protocol/ Internet Protocol (TCP/IP) Service.

Example 5-28 Liberty JVMProfile

```
#WLP_SERVER_HOST=*
#
# WLP_SERVER_HTTP_PORT is used by CICS to configure the port used for
# HTTP requests to the web server. The default value is 9080. If unset
# in the JVM profile, CICS will set it to the
# WebSphere Application Server Liberty profile default value.
# In general you should not accept this default, instead set it to
# a free port number on your z/OS system. Note, WLP does not use CICS
# TCPIPServices, so you should take care not to use any port in use by
# a CICS TCPIPService (or lsewhere).
#
WLP_SERVER_HTTP_PORT=4555
-Dcom.ibm.cics.jvmserver.wlp.server.http.port=4555
#
```

4. When the port is set, we are ready to install the new JVMSERVER definition.

Note: Check that the following directives within the JVMProfile match your environment requirements:

- `JAVA_HOME=/usr/lpp/java/J7.0_64/`
- `WORK_DIR=/u/klein4/cicslab/logs/`
- `WLP_INSTALL_DIR=&USSHOME;/wlp`
- `WLP_SERVER_HTTP_PORT=4555`
- `-Dcom.ibm.cics.jvmserver.wlp.server.http.port=4555`

5.2.15 Export the CICS Bundle to z/OS

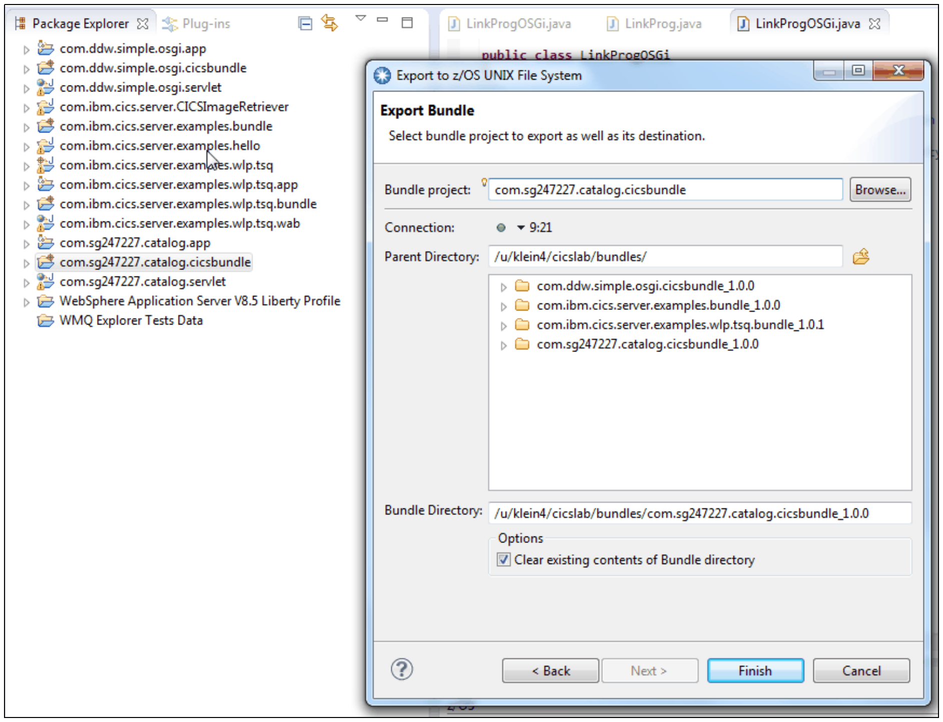
When deploying a servlet to CICS, you define a CICS BUNDLE and place your servlet into the BUNDLE. Then, you export the CICS Bundle from your workstation to z/OS, where it can be accessed by your CICS region.

To export the CICS Bundle, we create a directory in the z/OS UNIX file system. We use <*home_directory*>`/cicslab/bundles/`.

Export your CICS Bundle to UNIX System Services on z/OS

To export your CICS Bundle, complete the following steps:

1. In the Java EE perspective, the Enterprise Explorer view, right-click your `com.sg247227.catalog.cicsbundle` project and select **Export Bundle Project to z/OS UNIX file system**.
2. In the Export to z/OS UNIX file system pop-up dialog, select the **Export to a specific location in the file system** radio button, and then click **Next**.
3. Confirm the following information:
   - In the Export Bundle pop-up dialog (Figure 5-35), in **Bundle project**, it should say `com.sg247227.catalog.cicsbundle`.
   - Still on the Export Bundle page, in the Parent Directory, ensure that it says `<`*`home_directory`*`>/cicslab/bundles/`.
   - Still on the Export Bundle page, in Bundle Directory, ensure that it says `<`*`home_directory`*`>/cicslab/bundles/com.sg247227.catalog.cicsbundle_1.0.0`.
4. Still on the Export Bundle page, click **Finish**.



com.ddw.simple.osgi.app
com.ddw.simple.osgi.cicsbundle
com.ddw.simple.osgi.servlet
com.ibm.cics.server.CICSImageRetriever
com.ibm.cics.server.examples.bundle
com.ibm.cics.server.examples.hello
com.ibm.cics.server.examples.wlp.tsq
com.ibm.cics.server.examples.wlp.tsq.app
com.ibm.cics.server.examples.wlp.tsq.bundle
com.ibm.cics.server.examples.wlp.tsq.wab
com.sg247227.catalog.app
com.sg247227.catalog.cicsbundle
com.sg247227.catalog.servlet
WebSphere Application Server V8.5 Liberty Profile
WMQ Explorer Tests Data

Export to z/OS UNIX File System

Export Bundle

Select bundle project to export as well as its destination.

Bundle project: com.sg247227.catalog.cicsbundle Browse...

Connection: 9:21

Parent Directory: /u/klein4/cicslab/bundles/

com.ddw.simple.osgi.cicsbundle_1.0.0
com.ibm.cics.server.examples.bundle_1.0.0
com.ibm.cics.server.examples.wlp.tsq.bundle_1.0.1
com.sg247227.catalog.cicsbundle_1.0.0

Bundle Directory: /u/klein4/cicslab/bundles/com.sg247227.catalog.cicsbundle_1.0.0

Options

☑ Clear existing contents of Bundle directory

< Back Next > Finish Cancel

Figure 5-35 Export com.sg257227.catalog.cicsbundle Bundle

5.2.16 Define a CICS Bundle definition for the catalog servlet

In this section, we create a CICS BUNDLE definition in CICS TS for the catalog servlet. Depending on your preference, you can use the IBM CICS Explorer or the CEDA transaction to create the definition. The following steps can be used to create the BUNDLE definition that we use for the servlet:

1. First, we check if the `com.sg247227.catalog.cicsbundle_1.0.0` directory exists. See Example 5-29.

Example 5-29 Exported Bundle

```
EUID=0   /u/klein4/cicslab/bundles/
  Type  Filename
_ Dir   .
_ Dir   com.sg247227.catalog.cicsbundle_1.0.0
```

2. After that, we use the information in Example 5-30 to create a BUNDLE definition in CICS Transaction Server, by using either the IBM CICS Explorer or the CEDA Transaction.

Example 5-30 CEDA BUNDLE definition

```
OBJECT CHARACTERISTICS                                    CICS
RELEASE = 0690
 CEDA  View Bundle( CATLGAPP )
  Bundle        : CATLGAPP
  Group         : LIBPROF
  DEScription    :
  Status        : Enabled         Enabled ! Disabled
  BUndledir      :
/u/klein4/cicslab/bundles/com.sg247227.catalog.cicsbundle_
  (Mixed Case)   : 1.0.0
                :
                :
                :
  BAsescope      :
  (Mixed Case)   :
                :
                :
                :
 DEFINITION SIGNATURE
  DEFinetime    : 10/22/14 15:08:22
  CHANGETime    : 10/22/14 15:09:45

                                                       SYSID=69A4
APPLID=IV3A69A4
```

3. After that, we install the BUNDLE definition. Both the JVMSERVER and BUNDLE definition should be ready to use and should show a status of `enabled`.

   You can also check the messages shown in Example 5-31 to make sure that the JVMSERVER and the BUNDLE definition are ready to use.

   Example 5-31 JVMSERVER and BUNDLE successfully installed

   ```
   DFHSJ0915 10/24/2014 13:22:52 IV3A69A4 CICSUSER JVMSERVER DDWWLP is
   now enabled and is ready for use.
   DFHRL0132 I 10/24/2014 13:22:52 IV3A69A4 CEDA All defined resources
   for BUNDLE CATLGAPP are now in the enabled state.
   ```

6 Frequently asked questions

This chapter provides a list of the most frequently asked questions (FAQs). It provides a short answer for each of these questions, and a possible reference to certain chapters or sections in this book where you can access more detailed information. You can use this chapter to navigate through this channels and containers book, and start your investigation regarding this topic.

This chapter contains information about the following topics:

6.1 Administration questions

Table 6-1 contains FAQs dealing with the administration of channels and containers.

Table 6-1 Administration FAQs

No.	Question	Answer
1	How does IBM Customer Information Control System (CICS) resolve the restriction of 32 kilobytes (KB)?	CICS enhances inter-program data transfer using the constructs of channels and containers. A *container* is a named block of data that you can transfer to another program or transaction. You can also think of it as a *named communication area (COMMAREA)*. A *channel* is a set of containers that you can pass to another program or transaction, and can be thought of as a parameter list. See Chapter 2, "Application design and implementation" on page 27.
2	Can a CICS program continue to use a COMMAREA to pass data while implementing the channel and container constructs into its design?	Before CICS Transaction Server version 5.2, Channels and COMMAREA were mutually exclusive mechanisms for transferring data on **`EXEC CICS LINK`**, **`EXEC CICS XCTL`**, **`EXEC CICS START`**, and **`EXEC CICS RETURN`** commands. You could use one technique or the other during each operation, but not both at the same time. See Chapter 1, "Introduction to channels and containers" on page 1. However, with the introduction of the DFHTRANSACTION channel in CICS Transaction Server version 5.2, it is now possible to use both techniques at the same time. See 1.4.3, "The DFHTRANSACTION transaction channel" on page 15.
3	Can a CICS transaction, using channels to transfer data, be used in a pseudo-conversational mode?	Yes, you can use pseudo-conversational mode to preserve the container data between transactions. On the highest logical level return in the transaction flow (return to CICS), use the command format **`EXEC CICS RETURN TRANSID CHANNEL`**. To retrieve data passed to your program, use the command format **`EXEC CICS GET CONTAINER CHANNEL`**. See Chapter 3, "Programming" on page 65.
4	Is it possible to inquire against containers in a channel?	No, It is *not* possible to inquire using the system programming interface (SPI). This means that it is not possible to know the number of containers on a channel using **`EXEC CICS INQ CHANNEL(xxxxxxxx) CONTAINER COUNT`**.

No.	Question	Answer
5	Does the program free the storage areas that `EXEC CICS GET CONTAINER` acquires?	CICS manages the container storage. Do *not* issue a `FREEMAIN` function against the storage area. If necessary, you can use `EXEC CICS DELETE CONTAINER` to delete the container and free the storage. Otherwise, the program reclaims the container storage when the channel in which it was created goes out of scope. If you require to preserve some or all of the data, you can copy it into your own application storage. See 3.1, “EXEC CICS application programming interface” on page 66.
6	Are the containers recoverable?	No they are not. If you require recoverability for the container’s content, you must use business transaction services (BTS). See Chapter 3, “Programming” on page 65.
7	Can you access channels and containers using task-related user exit (TRUE) or global user exit (GLUE)?	TRUEs and GLUEs can create their own channels and, additionally, access transaction channels.
8	Are channels and containers suitable for dynamic transaction routing in an IBM CICSPlex System Manager environment?	Yes, but you must use a special container - DFHROUTE - if you require to access the data for dynamic routing decisions. See 2.9, “Best practices” on page 58.

6.2 Application programming questions

Table 6-2 contains FAQs regarding the application programming using channels and containers.

Table 6-2 Application programming FAQs

No.	Question	Answer
1	How can you know if a subroutine is started by programs using COMMAREA or channel, if both of the methods are in use?	The `EXEC CICS ASSIGN CHANNEL` command is provided for this scope. See 2.4, “STARTBROWSE application programming interface” on page 40. Additionally, you can check EIBCALEN.
2	Can you use channels and containers in Common Business Oriented Language (COBOL) dynamic calls?	Calling programs cannot pass channels and containers in the call. The called program can access the current channel, if any.

No.	Question	Answer
3	How do I know if I have a transaction channel (DFHTRANSACTION)?	Issue the following command: **`EXEC CICS GET CHANNEL('DFHTRANSACTION') CONTAINER('DUMMY') NODATA`** `'CHANNELERR'` is returned if the transaction channel does not exist.
4	Can I use transaction channels on all link types?	Transaction channels can only be used on multiregion operation (MRO) and Internet Protocol interconnectivity (IPIC) link types.
5	Is the transaction channel available after a remote DPL?	Yes it is, if the distributed program link (DPL) is made to a region at CICS Transaction Server version 5.2
6	What happens if I create a transaction channel and DPL to a pre-CICS Transaction Server version 5.2 region?	The transaction channel is not shipped.
7	When porting existing COMMAREAs, is it better to use channels with single or multiple containers?	It is suggested to use multiple containers. See 4.4.1, “Configuration” on page 103.
8	How can a program know what are the containers associated to the current channel?	Browse application programming interfaces (APIs) are provided for this scope. See 2.4, “STARTBROWSE application programming interface” on page 40 and 3.1.6, “Browsing the current channel” on page 73.
9	Why must you use data conversion in channel API rather than the CICS-provided one?	The data conversion model offered by channel API is simpler to manage compared to the CICS one. For more considerations on the matter, see 2.6.1, “Data conversion with channels” on page 44.
10	What are the factors that you must consider when designing a channel?	Consider the following factors when designing a channel: ► Use separate containers for input and output data. ► The server program, not the client, should create the output containers. ► Use separate containers for read-only and read/write data. ► Use separate containers for `CHAR` and `BIT` data. ► If a structure is optional, make it a separate container. ► Use dedicated containers for error information. More information about these factors is provided in 2.9.1, “Designing a channel” on page 58.

No.	Question	Answer
11	Is it there any type of API used to create a channel?	There is no explicit API to create a channel, but it is created when performing certain commands. More information about this topic is available in 3.1, “EXEC CICS application programming interface” on page 66.
12	How can a program pass one channel to another one?	**`EXEC CICS LINK`**, **`EXEC CICS XCTL`**, **`EXEC CICS START`**, and **`EXEC CICS RETURN`** are the commands that you can use to pass channels between programs. See 3.1.3, “Passing a channel to another program or task” on page 69.
13	Are CICS Java classes (JCICS) provided to manage channels and containers?	Yes they are. See 3.2, “CICS Java” on page 76.

6.3 Performance questions

Table 6-3 contains FAQs dealing with the performance of channels and containers.

Table 6-3 Performance FAQs

No.	Question	Answer
1	Does the terminal input/output area (TIOA) size interfere in any way with channels transmission over MRO session?	The TIOA used to process messages over MRO links has a maximum length of 32,767 bytes, as defined in the **`IOAREALEN`** parameter. In case of channels and containers usage, where data can be larger than 32 KB, CICS splits the data into more 32 KB TIOAs as required, and sends them to the remote region. In the remote region, CICS reassembles the 32 KB chunks of TIOA and presents them to the application as a channel. Note that using intersystem communication (ISC) links between CICS to pass container’s data can affect performance, depending on the amount of data passed. The larger the data, the more response time is affected by the process of sending them. See 4.6, “Problem determination” on page 109.

No.	Question	Answer
2	Any suggestion about containers and storage usage?	You must be careful when you use channels and containers to define the amount of extended dynamic storage area (DSA) size. Also, in this case, if large amounts of data are to be `PUT` in a container, it can affect the extended dynamic storage area allocation. The data area is subject to a `GETMAIN` function in user task storage above the 16 megabyte (MB) line and associated to the task. When you issue the `put container` command, it is also allocated in 64-bit storage, within the PGCSCB subpool of the above-the-bar CICS dynamic storage area (GCDSA) so that it can pass to the next program. The use of 64-bit storage influences the value that you choose for the IBM z/OS `MEMLIMIT` parameter that applies to the CICS region. You must also consider other CICS facilities that use 64-bit storage. See 4.6, “Problem determination” on page 109.
3	Is there any limitation to the container size used for data transfer in CICS	CICS has a 2 gigabyte (GB) limit on container size, which is, in effect, no limit. However, you are limited only by the available storage in the CICS address spaces, and by any architected structure limits in the application programming language. Container storage is acquired in the CICS address space above the 4 GB bar in GCDSA, subpool PGCSDB. CICS creates a copy of the container data within task storage above the 16 MB line for user access. Rather than using a single large container for transferring data between programs, effective programming practice suggests defining multiple containers within a channel, based on purpose, such as input, output, data type, data size, and so on. SeeChapter 4, “Systems management and configuration” on page 99.

6.4 Functions not supporting channels and containers

Table 6-4 shows the functions and languages that currently do *not* support the channels and containers API.

Table 6-4 Functions not supporting channels and containers

No.	Function	Comment
1	C++	Channels and containers support for C or C++ foundation classes is not in place.
2	External call interface (ECI)	CICS Transaction Gateway using ECI is not yet able to use these new enhancements.

6.5 Online information about channels and containers

The following list includes websites containing interesting information about channels and containers:

- The following link goes to the *Channels: Quick start* section of the Knowledge Center for CICS Transaction Server Version 5.2:

  http://www-01.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.applicationprogramming.doc/topics/dfhp3_ch_quickst.html?lang=en

- Find a white paper describing using channels and containers to enhance CICS interprogram data transfer on the following File Transfer Protocol (FTP) site:

  ftp://service.boulder.ibm.com/software/htp/cics/pdf/cics-g224753500.pdf

- Find a CICS Transaction Server product support page that you can use to search for channels and containers, and obtain useful tips, on the following website:

  http://www-306.ibm.com/software/htp/cics/tserver/support/

6.6 Hints and tips

The following list provides some hints and tips:

- To determine if a container was put as a `BIT` or `CHAR`, perform a **`GET CONTAINER NODATA CCSID(00037)`** and check the `RESP` code. If it comes back as `RESP=X`, `RESP2=Y`, the container is of type `BIT`.
- To access the current channel in JCICS other than using **`Task.getCurrentChannel();`** you can try the following call.

  If the name of the current channel is `CURRCHAN`, then the following JCICS call explicitly returns a reference to the current channel:

  ```
  Task.createChannel("CURRCHAN");
  ```

- To point at which the container is actually created in CICS while using JCICS to create a container: When the data is put into the container using JCICS, the container is created inside of CICS, as the following code snippet shows:

  ```
  Channel myChannel = Task.createChannel();
  Container myContainer = myChannel.createContainer();
  myContainer.put("Hello"); // It is at this point that the container
  is created in CICS
  ```

A

CICS channels and containers Liberty servlet example

This appendix is a description of how to start an existing IBM Customer Information Control System (CICS) assembly language application from a servlet running in a CICS Transaction Server version 5.2 Liberty environment.

This is achieved using Java classes, from the CICS Java (JCICS) class library, to link to the existing channel-aware business logic. We have developed a small sample to demonstrate how to use the JCICS link with channel application programming interface (API) within a servlet.

The CICS servlet issues a JCICS link request that passes a channel, rather than a communication area (COMMAREA), to the CICS back-end application.

Channels and containers JCICS servlet example

The following components are used, and are prerequisites to run the application:

- IBM CICS Explorer v5.2.0
- IBM WebSphere Application Server Liberty profile technology built in IBM CICS Transaction Server v5.2
- Web application and servlet implementation included in the additional material
- CICS JVMServer and Bundle definition

  The CICS region must be set up to run Liberty servlets.
- The CICS back-end program

  Create an assembly language or Common Business Oriented Language (COBOL) program LNKFRST1 that gets passed a channel from the servlet JCICS link request.

Figure A-1 shows the flow of a CICS Liberty servlet that links to a CICS COBOL program.

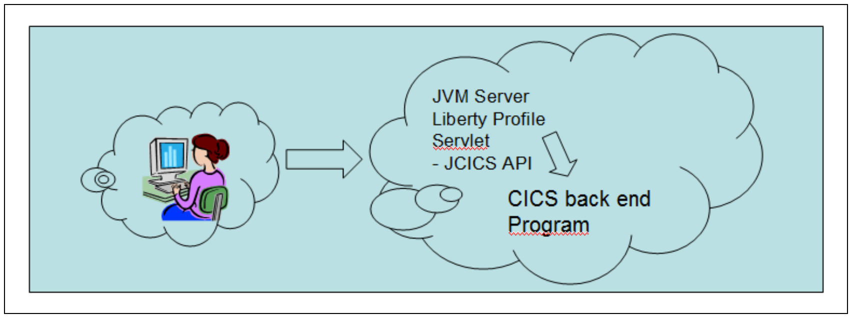


Figure A-1 Structure of the JCICS servlet sample

What the servlet channels and containers JCICS example does

The `LinkProg` servlet, which is deployed as a web application, requests an input phrase from an Hypertext Markup Language (HTML) form, which is passed to the servlet implementation. The phrase is copied to a container, which is passed to a channel-aware CICS back-end program. The back end appends a string in front of the phrase and returns to the web application.

The sample consists of the following components:

- An HTML form
- A Java servlet, which is supplied in the additional material. The servlet is running in a CICS Transaction Server v5.2 Liberty profile environment.
- A Java class that contains the JCICS channels and containers API. The class also uses a JCICS link request to call the back-end assembly language program.
- A back-end assembly language program

The sample works according to the following steps:

1. The user starts the application from a web browser. A form is displayed.
2. The form prompts the user to input a phrase. When the user clicks the **Submit** button, the servlet is started.

   The servlet performs the following functions:

   a. Set the response attributes

      - response.setContentType("text/xml");
      - response.setCharacterEncoding(ASCII);

   b. Obtain the **str** name from the request:

      - String **str** = request.getParameter("str");

   c. starts a method that contains the JCICS API, passing the input parameters entered by the user:

      - **LinkProgOSGi lnkprog** = new **LinkProgOSGi**();
      - result = lnkprog.linkProg(str);

   d. Construct the response:

      - // Construct the response
      - ServletOutputStream out = response.getOutputStream();
      - out.write("<result>".getBytes(ASCII));
      - out.write(result.getBytes(ASCII));
      - out.write("</result>".getBytes(ASCII));

3. The servlet runs method linkProg, which uses the JCICS classes to create a channels and container. It copies the phrase in string `str` into the container. After that, the servlet issues a JCICS link to the CICS back-end program, passing the channel.
4. The CICS back-end program issues a **`get container`** command. It appends the phrase to its program name and returns.

5. On return, the CICS servlet issues a **`get container`** JCICS API and returns the data in a `Byte[]` array. After that we convert the `Byte[]` array to a string and append it to the following result string:

   ```
   result = "Program linked using channel and container " +resultmsg ;
   ```

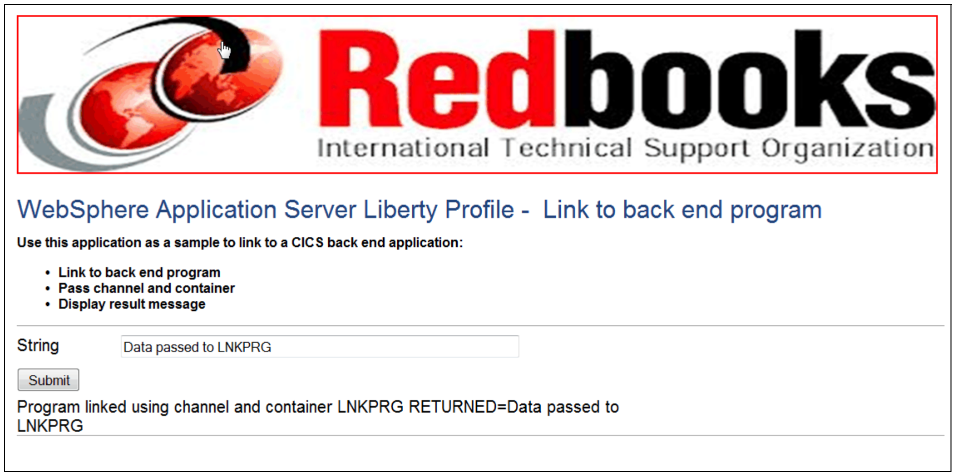
6. The servlet uses the HTML page to display the result on the user's browser.

HTML page and java script

As mentioned earlier, CICS Transaction Server V5.2 provides a web container that can run Java servlets and JavaServer Pages (JSP). We can use the features of the Java servlet and JSP specifications to write modern web applications for CICS. The web container runs in a Java virtual machine (JVM) server, and is built on the IBM WebSphere Application Server Liberty profile technology.

To demonstrate how to use the channels and containers API using a servlet we use an HTML page to get a data string from the users browser. We pass the data string to the servlet, which converts the data to a `Byte[]` array. After that, we create a channel and put the `Byte[]` array to a container using the JCICS API.

We pass the channel on a JCICS link to the back-end CICS program. See Figure A-2, which shows the HTML page. Within the HTML page, we use JavaScript to call the servlet implementation.



WebSphere Application Server Liberty Profile - Link to back end program

Use this application as a sample to link to a CICS back end application:

- Link to back end program
- Pass channel and container
- Display result message

String: Data passed to LNKPRG

Program linked using channel and container LNKPRG RETURNED=Data passed to LNKPRG

Figure A-2 CICS Liberty servlet using channels and containers API

Figure A-2 also shows the resulting message that is returned from the servlet. When you click **Submit**, the result is displayed below the **Submit** button.

Example A-1 shows the HTML that we use to call the CICS servlet implementation. See the following lines in the HTML to get an idea about how we pass the input string `str` when the servlet is called:

1. At this point, the JavaScript section starts. We use JavaScript within the HTML page to get the input data from the form and to call the servlet.
2. We create variable `str` and read the value from the input field.
3. In this section, we submit the request to the servlet.
4. This line shows how we define the input field, which is used to get the data string from the users browser.

Example: A-1 Define the input field used to get the data string

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>

<!-- <copyright --><!-- notice="cics-lm-source-asis" --><!-- pids="5655-Y04" --><!--
years="2009,2012" --><!-- crc="4151234751" > --><!-- Licensed Materials - Property of
IBM --><!-- 5655-Y04 --><!-- (C) Copyright IBM Corp. 2009, 2012 --><!-- This source
material is provided "AS-IS" under the terms and --><!-- conditions of the IBM Customer
Agreement and of the associated --><!-- Licensed Program Specifications documentation
for CICS Transaction --><!-- Server for z/OS. --><!-- --><!-- The terms and conditions
of this license permit users to modify this --><!-- source material and DO NOT provide
for any entitlement to defect --><!-- correction. --><!-- </copyright> -->
  <meta content="text/html; charset=ISO-8859-1" http-equiv="Content-Type">

  <style type="text/css">
<!--
body {
font-family: arial;
}
-->
.formClass {
width: 100px;
float: left;
position: relative;
}
.header {
background-image: url('images/banner.jpg');
background-repeat: no-repeat;
color: white;
}
<!--
A:visited {
color: blue;
text-decoration: underline;
}
-->
```

```
<!--
A:hover {
color: red;
text-decoration: underline;
}
-->
<!--
A:link {
color: blue;
text-decoration: underline;
}
-->
<!--
A:active {
color: green;
text-decoration: underline;
}
-->
  </style>
  <script type="text/javascript"> <<<<<<<<<<<<<<<< 1

/*
function to write a string to a container */
function postData() {
initPage(); // create the HTTP Request object
var xmlhttp = new XMLHttpRequest();
// read the values from the form
var str = document.myForm.str.value; <<<<<<<<<<<<<<<<<< 2

// build up the request string
var requestStr = "str=" + str;

// submit the request to the servlet
xmlhttp.open("POST", "LinkProg", true); <<<<<<<<<<<<<<<<<<<<<<<<<<<<< 3
xmlhttp.setRequestHeader("If-Modified-Since","0");
xmlhttp.setRequestHeader("Cache-Control","no-cache");
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlhttp.onreadystatechange=function()
{
if(xmlhttp.readyState==4 && xmlhttp.status == 200 ){
var xmlDoc = xmlhttp.responseText;
document.getElementById("messages").innerHTML = xmlDoc;
// refresh(str);
// myForm.elements["str"].value = '';
}
};
xmlhttp.send(requestStr);
}


function initPage()
{ document.getElementById("messages").innerHTML = '';
 }
  </script>
  <title>CICS WebSphere Liberty profile - Link Program Example</title>


</head>


<body>

<a href="https://www.ibm.com/developerworks/mydeveloperworks/blogs/cicsdev/"
style="text-decoration: none;"> <img src="images/banner.jpg" style="width: 70%; height:
150px;"></a>
<div style="color: rgb(34, 69, 125); font-style: normal; font-size: x-large;
padding-top: 20px;">WebSphere Application Server Liberty profile -  Link to back end
program</div>

<div style="padding-top: 10px; font-weight: bold; font-size: small;">
Use this application as a sample to link to a CICS back end application:
<ul>

  <li>Link to back end program</li>

  <li>Pass channel and container</li>

  <li>Display result message</li>

</ul>

</div>

<hr>
<div id="form" style="float: left; width: 50%; clear: right;">
<form name="myForm">
  <div class="formClass">String</div>

  <div> <input name="str" style="width: 60%;" type="text"> </div> <<<<<<<<<<<<< 4



  <div id="buttonsPanel" style="margin-top: 10px; clear: both;">
  <div> <input value="Submit" onclick="postData()" style="float: left;" type="button">
</div>

  <div> <br>

  </div>

  </div>

</form>

<div id="messages" style="clear: both; padding-top: 5px;"></div>

</div>
```

```
<hr style="clear: both;"><!-- This div is for the results from the file -->
<div id="fileA" style="clear: both;"><br>
<br>
<br>
<br>
<br>
</div>
</body>
</html>
```

Servlet implementation LinkProg

A servlet is a server-side web technology based on Java. A servlet extends some prewritten code in the `javax.servlet` classes. Extending these classes provides standard methods that a servlet can use to respond to web requests.

The servlet contains methods to get started when the Hypertext Transfer Protocol (HTTP) requests issued from our HTML form arrives. A **`GET`** request from the web browser is processed by the servlet's `doGet() method`, a **`PUT`** by the `doPut()` method, and a **`POST`** by the servlet's `doPost()` method.

We use the `doGet()` and `doPost()` methods, which are the two most common forms of input from a web browser, for **`GET`** and **`POST`** processing from a web page.

The code in Example A-2 demonstrates how we use the `doGet()` method to call method `linkProg` to link to the CICS back-end program passing a channel. We use the following Java statements in the `doGet()` method to obtain the data string from the HTML form. After that, we call method linkProg and pass the data string. In method `linkProg`, we create the channel and container using the JCICS API:

- // obtain the **str** name from the request
- String **str** = request.getParameter("str");
- **LinkProgOSGi lnkprog** = new **LinkProgOSGi**();
- result = lnkprog.linkProg(str);

Example A-2 shows the servlet implementation that we use.

Example: A-2 Servlet implementation class LinkProg

```
package com.sg247227.linkprog.servlet;

import java.io.IOException;


import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


/**
 * servlet implementation class LinkProg
 */
@WebServlet("/LinkProg")
public class LinkProg extends HttpServlet
{
    private static final long serialVersionUID = 1L;
    private static final String ASCII = "ISO-8859-1";

    /**
     * @see HttpServlet#HttpServlet()
     */






    public LinkProg()
    {
        super();

    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws 
ServletException, IOException
    {
        // Set the response attributes
        response.setContentType("text/xml");
        response.setCharacterEncoding(ASCII);




        // obtain the str name from the request
        String str = request.getParameter("str");


        // initialise a variable to hold the result of the action
        String result = "";

        // write the record
        LinkProgOSGi lnkprog = new LinkProgOSGi();
        result = lnkprog.linkProg(str);



        // Construct the XML response
        ServletOutputStream out = response.getOutputStream();
        out.write("<result>".getBytes(ASCII));
        out.write(result.getBytes(ASCII));
        out.write("</result>".getBytes(ASCII));
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
    {
        doGet(request, response);
    }
}
```

JCICS business class LinkProgOSGI

We use Java class `LinkProgOSGI` that contains method `linkProg` to create a channel and container that can be passed on the link to the back-end program. In Example A-3, we show how we use the `get` and `put` methods on the container that we create. We specify the coded character set identifier (CCSID) on the `get` and `put` methods to make sure that we have data conversion in place. The following lines show the key functions:

1. The line shows how we created the channel that we use to link to the back-end program.
2. At next we create the container that contains the data string from the users browser.
3. We convert string `str` to a `Byte[]` array.
4. The `Byte[]` array can now be put to the container. Note the CCSID.
5. In this section, we link to CICS program LNKPRG passing the channel.
6. On return, we get the data from the container into a `Byte[]` array.
7. We convert the `Byte[]` array to a string and append it to the result string.

Example: A-3 Use of the get and put methods on the container

```
package com.sg247227.linkprog.servlet;


import com.ibm.cics.server.*;

public class LinkProgOSGi

{
    public String linkProg(String ca1Name)
    {

        String result = "";
        String myccsid = "1208";



        try
        {

            Task t = Task.getTask();



              // create the channel
              Channel mychannel = t.createChannel("MYCHANNEL");<<<<<<<<<<<<<<<< 1

              // create the MIGRCNT container
              Container migrcnt = mychannel.createContainer("MAINCNT1");<<<<<<<<<<< 2


              // put the data string into the container

              byte[] continput = ca1Name.getBytes();<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< 3


              migrcnt.put(continput, myccsid);<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< 4



              // Link to the LNKPRG program passing a channel<<<<<<<<<<<<<<<<<<<<< 5
              Program p = new Program();
              p.setName("LNKPRG");
              p.link(mychannel);




          // Get the returned data

             byte[] retdata = migrcnt.get(myccsid);<<<<<<<<<<<<<<<<<<<<<<<<<<<<< 6

              String  resultmsg = new String(retdata, "UTF-8");<<<<<<<<<<<< 7





            result = "Program linked using channel <mychannel> and container <migrcnt> "
+resultmsg ;
        }
        catch (ChannelErrorException e)
        {
           result = "Channel error";
        }
        catch (ContainerErrorException e)
        {
           result = "An IO Error occurred";
        }
        catch (InvalidRequestException e)
        {
           result = "Invalid Request";
        }
        catch (CCSIDErrorException e)
        {
           result = "Invalid CCSID.";
        }



        catch (Exception e)
        {
           result = "Invalid CCSID.";
        }
        // return the result to the calling servlet
        return result;
    }
}
```

CICS back end program

Example A-4 shows the assembly language program that gets called by the CICS servlet. We issue an **`ASSIGN PROGRAM()`** command to get the program name to which you append the hello string.

Then we issue a **`GET CONTAINER(CA1STR)`** command to append the string to the program name. After this, we issue a **`put container`** command. We specified `CA1` on the `FROM` option, which copies the assembled string into the container. After this, return to the caller, which is the CICS servlet. See Example A-4.

Example: A-4 CICS back-end program

```
DFHEISTG DSECT
 RESP     DS    F
 RESP2    DS    F
 CA1      DS    0CL50
 PROGRAM  DS    CL8
 RETMSG   DS    CL10
 CA1STR   DS    CL30
 MIGRCHN  DS    CL16
 MIGRCNT  DS    CL16
 *CA      DSECT
 CASTRING DS    CL30
 LINKFRST CSECT
 BEGIN    DS    0H
 *        L 4,DFHEICAP
 *        USING CA,4
          MVC   MIGRCNT,=CL16'MAINCNT1        '
          MVC   RETMSG,=CL10' RETURNED='
EXEC CICS ASSIGN PROGRAM(PROGRAM) RESP(RESP) RESP2(RESP2)
          EXEC CICS GET CONTAINER(MIGRCNT) INTO(CA1STR)
          EXEC CICS PUT CONTAINER(MIGRCNT) FROM(CA1)
 RETURN   DS    0H
          EXEC CICS RETURN
          END
 //LKED.SYSIN  DD  *
     MODE RMODE(ANY),AMODE(31)
               NAME  LNKFRST1(R)
 /*
 //
```

Installation

To install the CICS servlet channels and containers example, complete the following steps:

1. Compile and link the CICS back-end program LNKFRST1.
2. Install the servlet using the procedure that we use in Chapter 5, “Sample application” on page 139 to install the catalog servlet. You can also use the installation procedure that we provide in the additional material.

   All of the required files needed for this example can be found in Appendix B, “Additional material” on page 239.

B

Additional material

This chapter refers to the additional material that you can download from the Internet. These additional materials provide you with further information about the various aspects involving Customer Information Control System (CICS) channels and containers, and also communication areas (COMMAREAs).

Locating the web material

The web material associated with this book is available in soft copy on the Internet from the IBM Redbooks publication web server. Browse to the following website:

ftp://www.redbooks.ibm.com/redbooks/SG247227

Alternatively, you can go to the IBM Redbooks publication website:

ibm.com/redbooks

Select the **additional materials** and open the directory that corresponds with the IBM Redbooks publication form number, SG247227.

Using the web material

The additional web material that accompanies this IBM Redbooks publication includes the following files:

File name	*Description*
SG24-7227Web_Content.zip	There are four compressed code samples. The `catalog.xls` file shows the fluctuations in user CPU time. The `cattrace.xls` file shows the outputs for XEIIN and XEIOUT global user exit points. The `CICSHelloWorld.ear` file and the `HelloWorldEJB.jar` are figures that show sample panels of CICS and JCICS Enterprise JavaBeans (EJB).

System requirements for downloading the web material

The following system configuration is advised:

Hard disk space:	1.2 megabytes (MB) minimum
Operating System:	Microsoft Windows
Processor:	500 megahertz (MHz) or higher
Memory:	256 MB preferably 512 MB

How to use the web material

Create a subdirectory or folder on your workstation and extract the contents of the web material zip file into this folder.

Sample server program used in some examples

In some examples in Chapter 2, "Application design and implementation" on page 27, we used a program that could receive a name in a channel or a communication area (COMMAREA). This subroutine changes the name into a phonetic string and returns the string to the caller through the channel or the COMMAREA. This subroutine is provided in Example B-1.

Example: B-1 Subroutine to return string

```
PROCESS LIST CICS('SP,COBOL3') APOST TRUNC(BIN)
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    PHONETIC.
      *****************************************************************
      * CHANGE A NAME (40Bytes) to a phonetic name (40Bytes)          *
      * COMMAREA AND CHANNEL INTERFACDE                               *
```

```
*****************************************************************
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*------- counter --------*
 01  I  PIC 9(2).
 01  K  PIC 9(2).
*------- TABLES ---------*
 01  TAB1-1X PIC X(36) VALUE
         'SCSZCZTZTSDSKSQUPFPHUEAEOEEIEYEUAUOU'.
 01  TAB1-1 REDEFINES TAB1-1X.
     03  TAB1-1-FIELD PIC X(2) OCCURS 18 TIMES.
 01  TAB1-3X PIC X(36) VALUE
         'C C C C C C X KVV V Y E E AYAYOYA$$ '.
 01  TAB1-3 REDEFINES TAB1-3X.
     03  TAB1-3-FIELD PIC X(2) OCCURS 18 TIMES.

 01  TAB2-1X PIC X(15) VALUE "ZKGQ   IJ FWPT$".
 01  TAB2-1 REDEFINES TAB2-1X.
     03  TAB2-1-FIELD PIC X OCCURS 15 TIMES.
 01  TAB2-3X PIC X(15) VALUE 'CCCCEYEYYSVVBDU'.
 01  TAB2-3 REDEFINES TAB2-3X.
     03  TAB2-3-FIELD PIC X OCCURS 15 TIMES.

 01  VALID-CHARACTERS PIC X(15) VALUE 'ABCDLMNORSUVSXY'.
 01  VALID-CHARACTERS-TAB REDEFINES VALID-CHARACTERS.
     03  ZEICHEN PIC X OCCURS 15 TIMES.

 01  P POINTER.
*------- flags -------------------*
 01  SWITCH PIC X(2).
     88  LETTER-OK VALUE IS 'OK'.

*------- CHANNEL STUFF ---------*
 01  CURRENTCHANNELNAME PIC X(16).
 01  MYBROWSETOKEN PIC 9(8) BINARY.
 01  INPUTCONTAINER  PIC X(16) VALUE IS 'NAMECONTAINER   '.
 01  OUTPUTCONTAINER PIC X(16) VALUE IS 'PHONETICNAMECONT'.
*------- RETURN AND REASON CODES -----------*
 01  RC  PIC 9(8) BINARY.
 01  RSN PIC 9(8) BINARY.

 LINKAGE SECTION.
 01  DFHCOMMAREA.
     03  LNAME PIC X(40).
     03  PNAME PIC X(40).
     03  REDEFINES PNAME.
         05  LETTER PIC X OCCURS 40 TIMES.
/
 PROCEDURE DIVISION USING DFHCOMMAREA.
* establish addressibility.
     EXEC CICS ASSIGN CHANNEL(CURRENTCHANNELNAME)
     END-EXEC.
     IF CURRENTCHANNELNAME IS EQUAL TO SPACES
        THEN  CONTINUE
*             continue with commarea
        ELSE
*             try to find out the name of the input container
              EXEC CICS GETMAIN SET(P)
                        FLENGTH(LENGTH OF DFHCOMMAREA)
                        INITIMG(X'40')
              END-EXEC
              SET ADDRESS OF DFHCOMMAREA TO P
              EXEC CICS GET CONTAINER(INPUTCONTAINER)
                        CHANNEL(CURRENTCHANNELNAME)
                        INTO(LNAME)
              END-EXEC
     END-IF.
* uppercase the input string
     MOVE FUNCTION UPPER-CASE (LNAME) TO PNAME.
* change double characters
     PERFORM WITH TEST BEFORE VARYING I FROM 1 BY 1
             UNTIL I > ( LENGTH OF TAB1-1X / 2 )
         INSPECT PNAME REPLACING ALL
             TAB1-1-FIELD (I) BY TAB1-3-FIELD (I)
     END-PERFORM.
* change single characters
     MOVE X'0A3A39' TO TAB2-1X(5:3)
     MOVE X'45'     TO TAB2-1X(10:1)
     PERFORM WITH TEST BEFORE VARYING I FROM 1 BY 1
             UNTIL I > LENGTH OF TAB2-1X
         INSPECT PNAME REPLACING ALL
             TAB2-1-FIELD (I) BY TAB2-3-FIELD (I)
     END-PERFORM.
* delete non valid characters
     PERFORM WITH TEST BEFORE VARYING I FROM 1 BY 1
             UNTIL I > 40
         MOVE SPACE TO SWITCH
         PERFORM WITH TEST BEFORE VARYING K FROM 1 BY 1
         UNTIL K > LENGTH OF VALID-CHARACTERS-TAB
             IF LETTER (I) = ZEICHEN (K)
```

```
                 THEN MOVE 'OK' TO SWITCH
                 ELSE CONTINUE
             END-IF
         END-PERFORM
         IF LETTER-OK
             THEN CONTINUE
             ELSE MOVE SPACE TO LETTER (I)
         END-IF
     END-PERFORM.
* delete double characters
     PERFORM COMPRESS-STRING.
     PERFORM WITH TEST BEFORE VARYING I FROM 1 BY 1
         UNTIL I = 40
         IF LETTER (I) = LETTER (I + 1)
             THEN MOVE SPACE TO LETTER (I)
             ELSE CONTINUE
         END-IF
     END-PERFORM.
     PERFORM COMPRESS-STRING.

* if a channel is  available, then move output into container
     IF CURRENTCHANNELNAME IS EQUAL TO SPACES
        THEN  CONTINUE
*             continue with return to cics
        ELSE
             EXEC CICS PUT CONTAINER(OUTPUTCONTAINER)
                 FROM(PNAME)
             END-EXEC
     END-IF.

* return to caller
     EXEC CICS RETURN
     END-EXEC.

     GOBACK.

*****************************************************************
* subroutine                                                    *
*****************************************************************
* delete all blanks
 COMPRESS-STRING.
     PERFORM 40 TIMES
         PERFORM WITH TEST BEFORE
         VARYING I FROM 1 BY 1 UNTIL I = 40
             IF LETTER (I) = ' '
```

```
                       THEN MOVE LETTER (I + 1) TO LETTER (I)
                            MOVE ' ' TO LETTER (I + 1)
                       ELSE CONTINUE
                   END-IF
               END-PERFORM
           END-PERFORM.
      *
       END PROGRAM PHONETIC.
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed description of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 246. Note that some of the documents referenced here might be available in soft copy only:

- *Application Development for IBM CICS Web Services*, SG24-7126
- *Java Application Development for CICS*, SG24-5275
- *CICS Transaction Server V3R1 Channels and Containers Revealed*, SG24-7227

Online resources

These websites and Uniform Resource Locators (URLs) are also relevant as further information sources:

- CICS family product support page

  http://www-306.ibm.com/software/htp/cics/support/
- CICS product main page, a summary of the function

  http://www-306.ibm.com/software/htp/cics/tserver/v31/apptrans/
- CICS Transaction Server product support page

  http://www-306.ibm.com/software/htp/cics/tserver/support/
- Description of channels and containers that can be downloaded from the following File Transfer Protocol (FTP) site

  ftp://service.boulder.ibm.com/software/htp/cics/pdf/cics-g224753500.pdf
- Information center for CICS Transaction Server V5.2 containing the learning path to *Introduction to Channels and Containers*

  http://www-01.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.home.doc/welcomePage/welcomePage.html?cp=SSGMGV

How to get IBM Redbooks

You can search for, view, or download IBM Redbooks, Redpapers, Hints and Tips, draft publications, and Additional materials, and order hardcopy IBM Redbooks or CD-ROMs, from the following website:

`ibm.com/redbooks`

Help from IBM

IBM Support and downloads

`ibm.com/support`

IBM Global Services

`ibm.com/services`

Index

IBM

Redbooks

Using IBM CICS Transaction Server Channels and Containers

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Using IBM CICS Transaction Server Channels and Containers

Convert a COMMAREA-based application to use channels and containers

Learn how to configure systems with a sample application

Simplify the process for code page conversion

This IBM Redbooks publication describes the new channels and containers support in IBM Customer Information Control System (CICS) Transaction Server V5.2. The book begins with an overview of the techniques used to pass data between applications running in CICS.

This book describes the constraints that these data techniques might be subject to, and how a channels and containers solution can provide solid advantages alongside these techniques. These capabilities enable CICS to fully comply with emerging technology requirements in terms of sizing and flexibility.

The book then goes on to describe application design, and looks at implementing channels and containers from an application programmer point of view. It provides examples to show how to evolve channels and containers from communication areas (COMMAREAs).

Next, the book explains the channels and containers application programming interface (API). It also describes how this API can be used in both traditional CICS applications and a Java CICS (JCICS) applications. The business transaction services (BTS) API is considered as a similar yet recoverable alternative to channels and containers. Some authorized program analysis reports (APARs) are introduced, which enable more flexible web services features by using channels and containers.

The book also presents information from a systems management point of view, describing the systems management and configuration tasks and techniques that you must consider when implementing a channels and containers solution.

The book chooses a sample application in the CICS catalog manager example, and describes how you can port an existing CICS application to use channels and containers rather than using COMMAREAs.

SG24-7227-01 ISBN 0738440507