



IBM Software Group

IBM WebSphere Technical Conference

Featuring WebSphere, WebSphere Portal, WebSphere BI, WebSphere MQ and CICS – Nov 29th – December 3rd 2004 - Munich

CI21TS Beyond the 32K Commarea Limit
Colin Penfold - colin_penfold@uk.ibm.com

WebSphere software



@business on demand software

Beyond the 32K Commarea Limit - Notes

This presentation will describe the capabilities provided by the Enhanced Inter-program Data Transfer function introduced in CICS Transaction Server 3.1. This function will allow programs and transactions to exchange more than 32K of data when using a LINK, XCTL, START or RETURN TRANSID command.

While not technically correct, to facilitate the understanding of this capability you might think of this capability as being equivalent to “Big COMMAREAs”.

Acknowledgements

- The following are trademarks of International Business Machines Corporation in the United States, other countries, or both: IBM, CICS, CICS/ESA, CICS TS, CICS Transaction Server, DB2, MQSeries, OS/390, S/390, WebSphere, z/OS, zSeries, Parallel Sysplex.
- Java, and all Java-based trademarks and logos, are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Other company, product, and service names and logos may be trademarks or service marks of others.

Enhanced Inter-program Data Transfer - Notes

These foils are based on a presentation written by Steve Zemblowski

Session Agenda

- **The 32k COMMAREA “problem”**
- **The “solution”**
 - Containers and Channels
- **Scenarios**
 - Components
 - Migration
 - Best Practices
- **API**
- **JCICS**
- **BTS**
- **System Programming**
 - GLUEs, TRUEs, URM
 - Monitoring and Statistics

Enhanced Inter-program Data Transfer - Notes

This presentation will discuss the problems that are encountered by programs encountering the 32K COMMAREA limitation, techniques that have been used to circumvent the 32K limitation and then will discuss the CICS solution to the problem.

The CICS solution uses Channels and Containers to eliminate the problem. Channels are sets of Containers. Containers are name blocks of data that hold information to be passed between programs and transaction.

The CICS Application Programming Interface changes for both EXEC CICS commands and JCICS classes will be examined.

The effects of Channels and Containers on Global User Exits, Task Related User Exits and User Replaceable Modules will be described.

An example of how to migrate existing applications from their use of COMMAREAs to Channels and Containers will be presented.

The Problem

32K Commarea Limitation

- **Program Commarea**
 - In a single CICS region
 - LINK
 - XCTL
 - Across CICS regions
 - Distributed Program Link
- **Pseudo-conversational Commarea**
 - Local and transaction routing
- **Similar limitation on START with data**
 - Single CICS region and distributed START

Enhanced Inter-program Data Transfer - Notes

The 32K COMMAREA limitation has existed since command level program was introduced in 1975. The COMMAREA size restriction is applicable to both LINK and XCTL commands in a single region as well as applying to COMMAREAs used by programs participating in a Distributed Program Link (DPL) between two CICS regions.

The 32K limitation also effects the exchange of data between multiple CICS tasks. Data can be passed between two tasks by the use of the EXEC CICS START TRANSID FROM command. The data area specified by the FROM option is limited to a maximum size of 32K.

CICS transaction involved in a pseudo-conversational sequence can exchange data through the use of the EXEC CICS RETURN TRANSID COMMAREA command. The returned COMMAREA is subject to 32K size restriction.

The 32K COMMAREA restriction is also applicable to the External CICS Interface (EXCI) and the External Call Interface (ECI) used by the CICS Transaction Gateway and the CICS Universal Clients. The solution described in this presentation does not apply to this set of restrictions.

Techniques for circumventing the 32k limit

- **Passing addresses of large storage areas**
 - Single region solution
- **Placing data in Temporary Storage**
- **Placing data in WebSphere MQ**
- **Using BTS**

Enhanced Inter-program Data Transfer - Notes

CICS programmers have developed a number of techniques for circumventing the 32K COMMAREA restriction both within a single CICS region and between multiple CICS regions.

Some of these techniques involve:

Passing the address of a large storage area in the COMMAREA. By using the FLENGTH option of the GETMAIN command a storage area larger than 32K can be acquired. This solution, while simple, will only work in a single CICS address space. A region affinity between the two programs or transactions will be created.

Passing the name of a temporary storage queue in the COMMAREA. By placing the data in temporary storage more than 32K of data can be passed between programs or tasks. If the temporary storage queue is placed in a Temporary Storage Owning Region or the Coupling Facility the data can be accessible across multiple CICS regions.

Pass the name of WebSphere MQSeries queue in the COMMAREA. By placing the data in WMQ queues and only passing the queue name a larger amount of data can be passed between the communicating programs or tasks.

The Solution

Is not a multi-megabyte COMMAREA

- **Larger COMMAREA sizes would exacerbate current problems**
 - Data structure complexity
 - Overloaded copybooks
 - Inefficient transmission
 - Unnecessary data transmitted on DPLs
 - Object code compatibility
 - What about EIBCALEN?
 - Code page conversions
 - DFHCNV mechanism is not easy

Enhanced Inter-program Data Transfer - Notes

At first glance, it would seem that a reasonable solution to this problem would be to make the COMMAREA length greater than 32K and all problems would be resolved. While such an implementation would ease the 32K restriction it would not resolve all the “problems” that exist in exchanging data today and would exacerbate some of the problem areas.

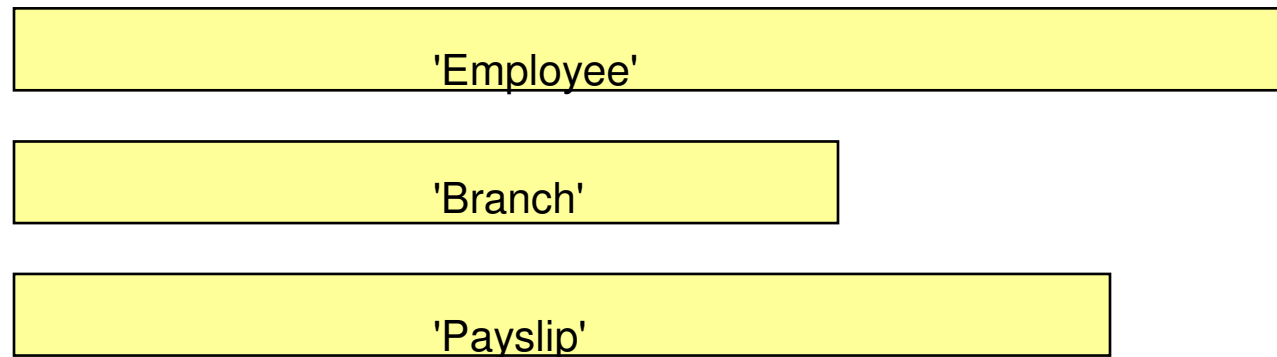
The current copy books used in the exchange of data today tend to be overloaded. That is, the structures are redefined a number of times depending on whether the copybook is passing input, output or error information. This leads to confusion on exactly when the fields are valid.

The current overload 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 could 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 allow for easily separating binary and character data. When a COMMAREA needs to be converted to a different codepage, conversion is a very difficult and error prone process.

Merely changing the COMMAREA length to a full word length could result in the loss of object and source code compatibility for existing CICS programs. How would a program know if EIBCALEN was valid or whether to check a new EIB field?

The “Solution”... Containers



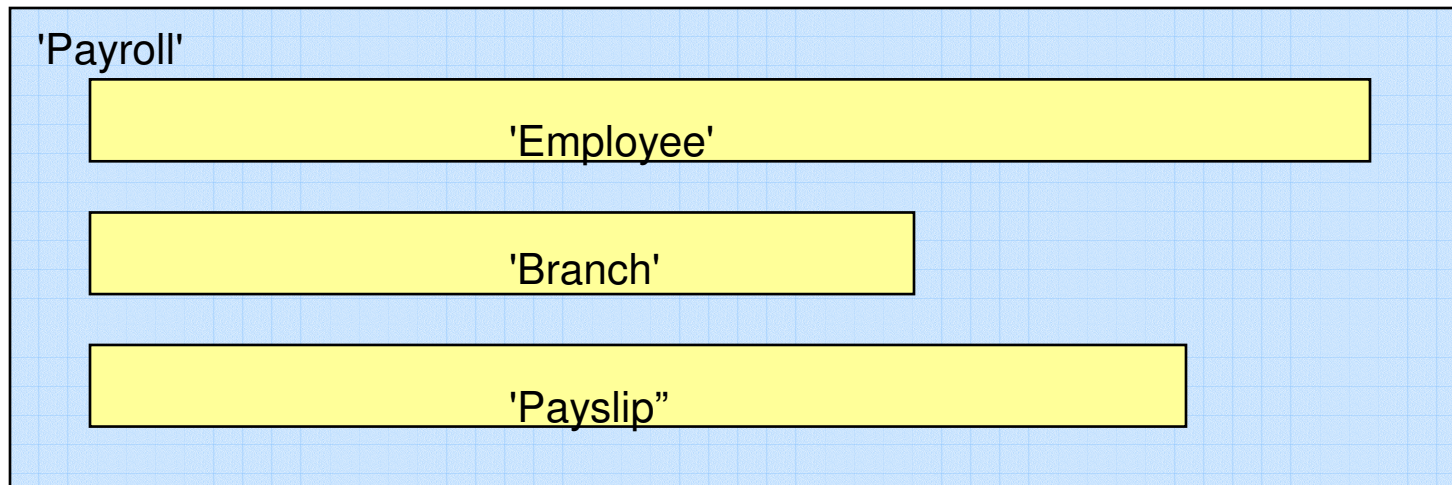
- **Named block of data designed for passing information between programs**
 - “Named” COMMAREAs
- **CONTAINER API**
 - Created using PUT CONTAINER
 - Read using GET CONTAINER
- **No CICS enforced size limitation**
 - Channels are stored above the line (but below the bar)

Enhanced Inter-program Data Transfer - Notes

The solution to the 32K COMMAREA problem is to implement new constructs which solve the previously mentioned problems. CICS Transaction Server 3.1 has done this by implementing Channels and Containers. Channels and COMMAREAs are mutually exclusive. That is, you may use one technique or the other for passing data but not both.

A Container is a named block of data that can be passed to a subsequent program or transaction. It may be easiest to think of a container as a “named COMMAREA”. There is no CICS enforced limit on the physical size of a single container. You are only limited by the available user storage in the CICS address space. Later, as we discuss “best practices”, there will be reasons why you will want to limit the use of a single container and use multiple containers.

The “Solution”... Channels



- **A group of Containers**
 - No limit on the number of Containers in a Channel
- **A Channel is a sort of program interface**
 - Passed on LINK, XCTL, pseudoconversational RETURN, and START commands
- **Non-persistent**
 - Non-recoverable resource similar to commareas

Enhanced Inter-program Data Transfer - Notes

A Channel is a set of containers that can be passed to another program or transaction. A channel is analogous to a parameter list.

Channels and containers are only visible to the programs that create them and to the programs they are passed to. When these programs terminate, CICS will automatically destroy the containers and storage they occupy.

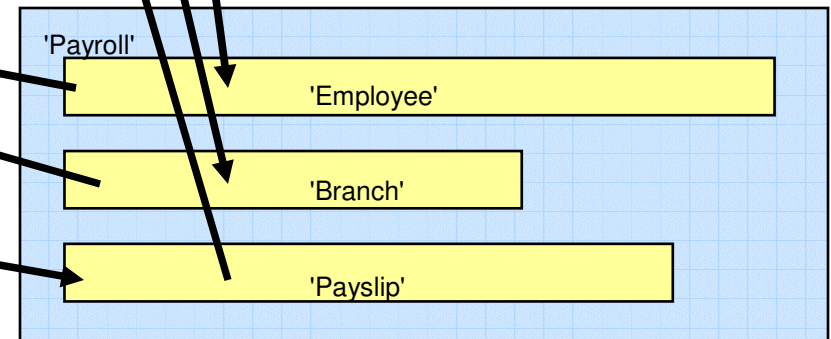
A Simple Example

PROGA

```
PUT CONTAINER('Employee') CHANNEL('Payroll') FROM(emp-data)
PUT CONTAINER('Branch') CHANNEL('Payroll') FROM(branch-data)
LINK PROGRAM('PROGB') CHANNEL('Payroll')
GET CONTAINER('Payslip') CHANNEL('Payroll') INTO(pay-data)
```

PROGB

```
GET CONTAINER('Employee') INTO(emp-data)
GET CONTAINER('Branch') INTO(branch-data)
...
PUT CONTAINER('Payslip') FROM(pay-data)
```



Enhanced Inter-program Data Transfer - Notes

Here are several examples on how you might employ a channel in your application program flow.

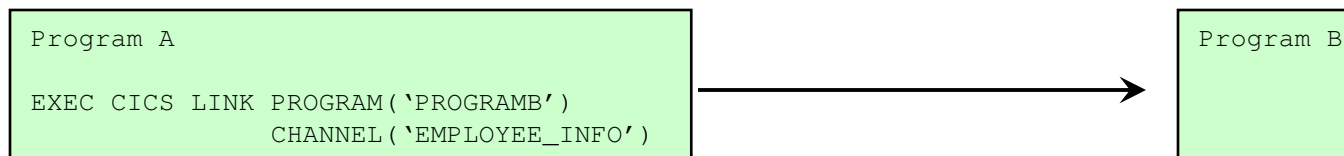
The simplest technique is to use one channel and its collection of containers to invoke one program. The channel name will be specified on the EXEC CICS LINK command and the target application program will always know exactly what channel it will be passed.

Another technique would be for the first program to create a channel and some containers. This program could then LINK to another program passing the channel along. Subsequent programs could use the same channel, with the same or different containers, for their exchange of data. In this example there is only one copy of the data maintained.

Scenarios

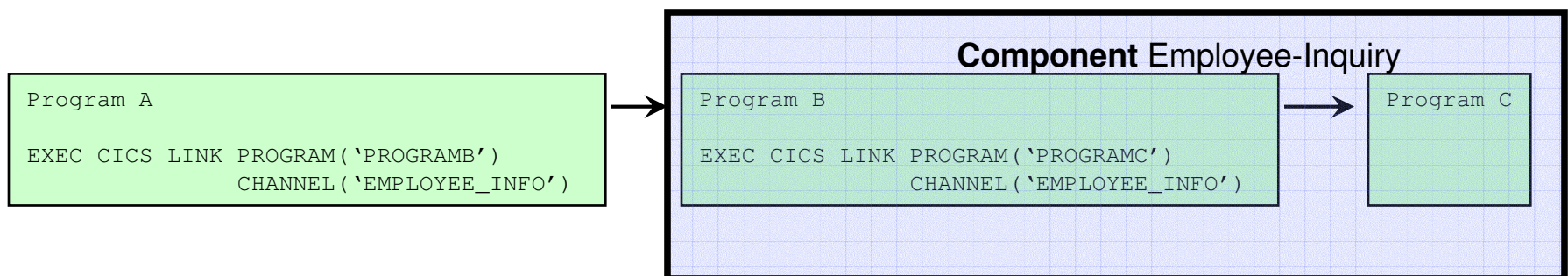
Basic Scenarios for using Channels

■ One channel / One program



■ One channel / Multiple programs

- The Channel is the interface to a Component



Enhanced Inter-program Data Transfer - Notes

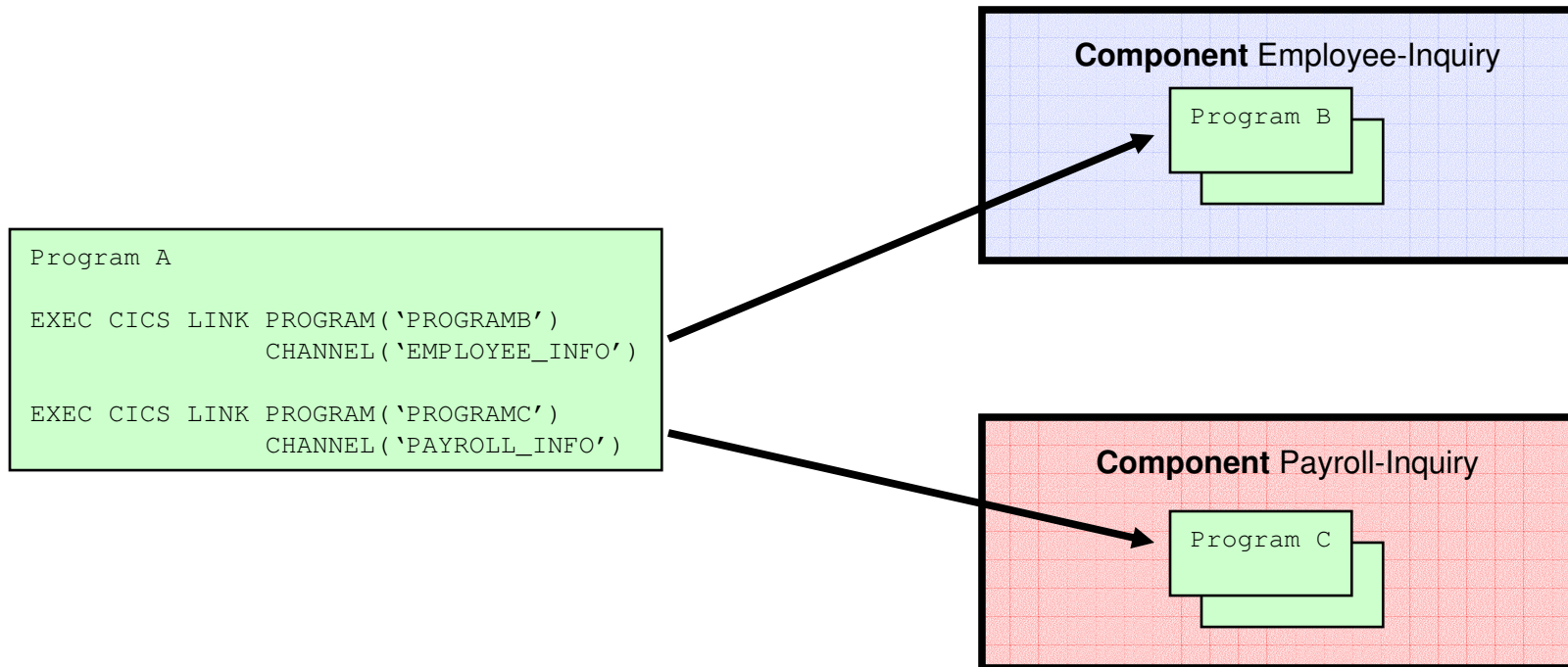
Here are several examples on how you might employ a channel in your application program flow.

The simplest technique is to use one channel and its collection of containers to invoke one program. The channel name will be specified on the EXEC CICS LINK command and the target application program will always know exactly what channel it will be passed.

Another technique would be for the first program to create a channel and some containers. This program could then LINK to another program passing the channel along. Subsequent programs could use the same channel, with the same or different containers, for their exchange of data. In this example there is only one copy of the data maintained.

Scenario - Multiple Components

- **One program / Multiple channels**



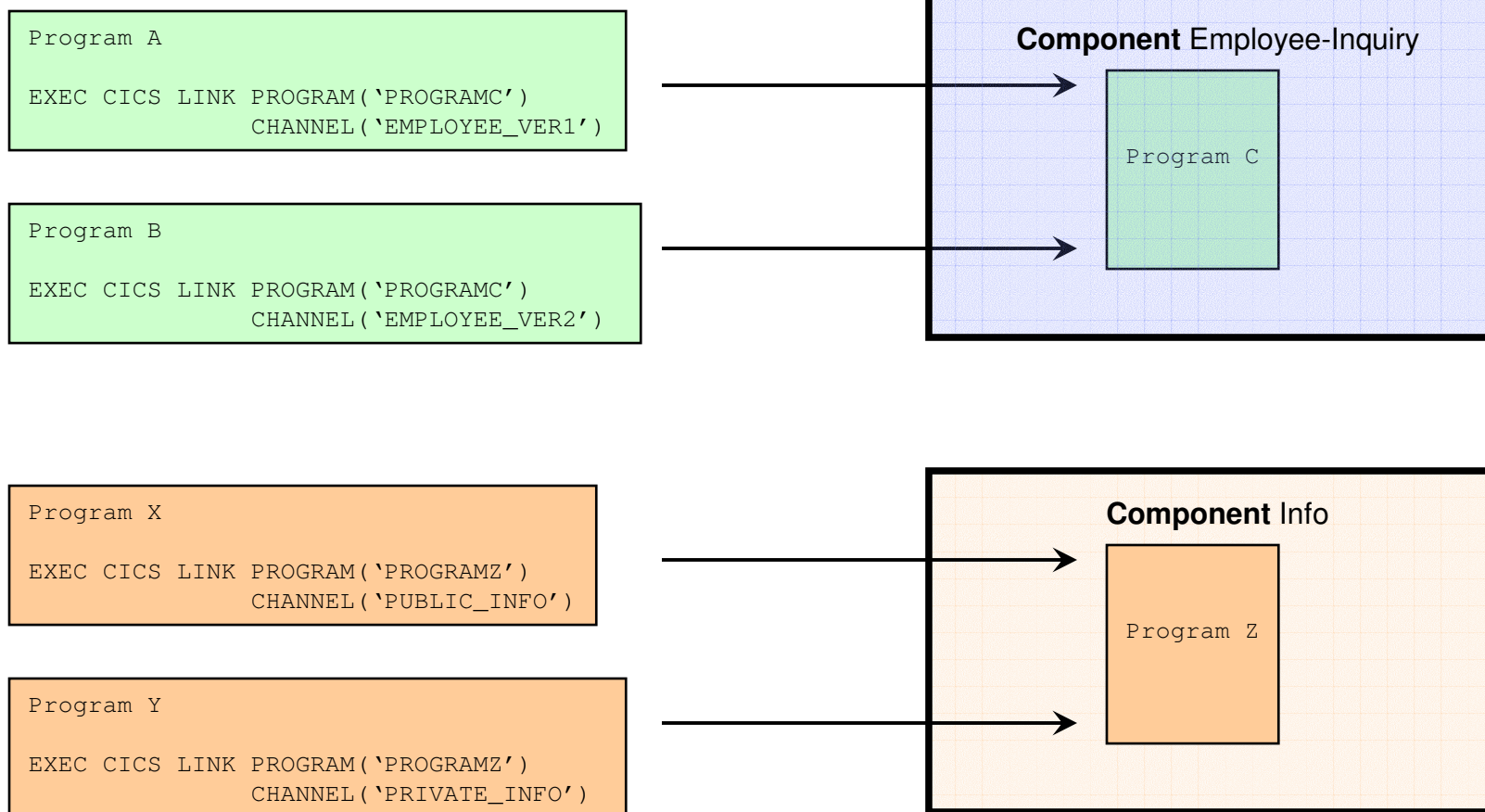
Enhanced Inter-program Data Transfer - Notes

Here are further examples on how you might employ a channel in your application program flow.

Program A links to two programs, B and C. Each program has a different channel describing the different interfaces.

Scenario - Loose Binding

- **Multiple programs / Multiple channels**



Enhanced Inter-program Data Transfer - Notes

Here are further examples on how you might employ a channel in your application program flow.

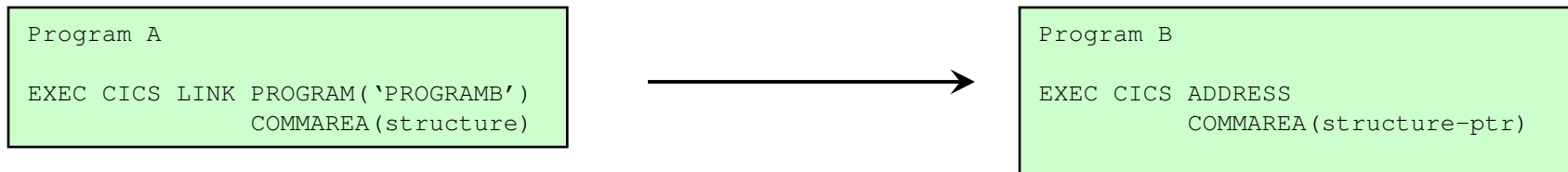
Program C is a server program that can process requests from a number of different clients. In the first example, there is a new version of data structure in a container. Program C could be enhanced to see which version of the container structure needs to be processed. As time permits, the calling programs can be enhanced to use the new version on the container structure.

In the second case, program Z normally will handle public requests from calling programs. A private data structure is used for special administrative programs. A different channel or container could be used to implement this “private protocol”.

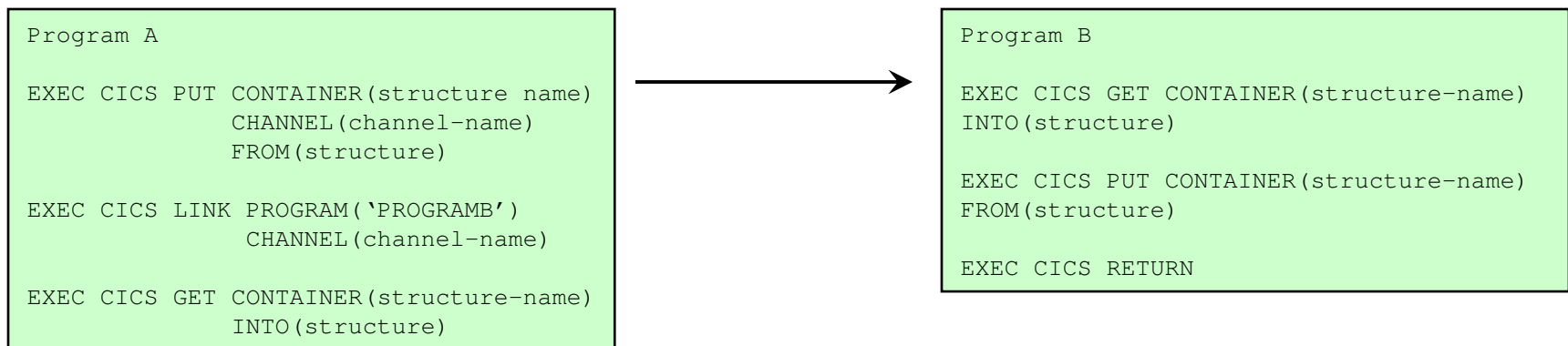
Note that CICS does not define any security mechanism to enforce who can use a channel name.

Migration of Programs Using LINK

- **Existing application with COMMAREA**



- **Changed application using Channels**

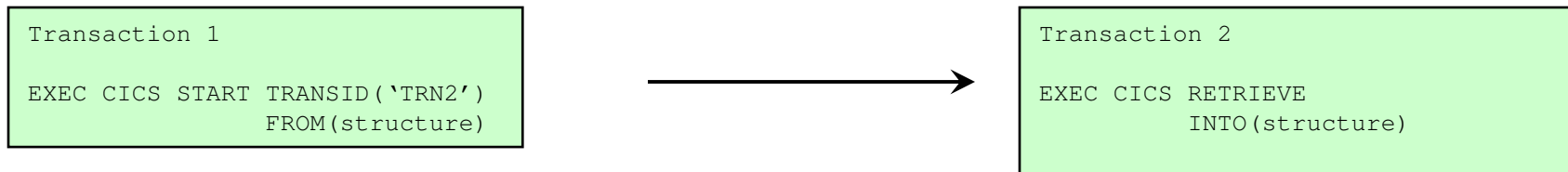


Enhanced Inter-program Data Transfer - Notes

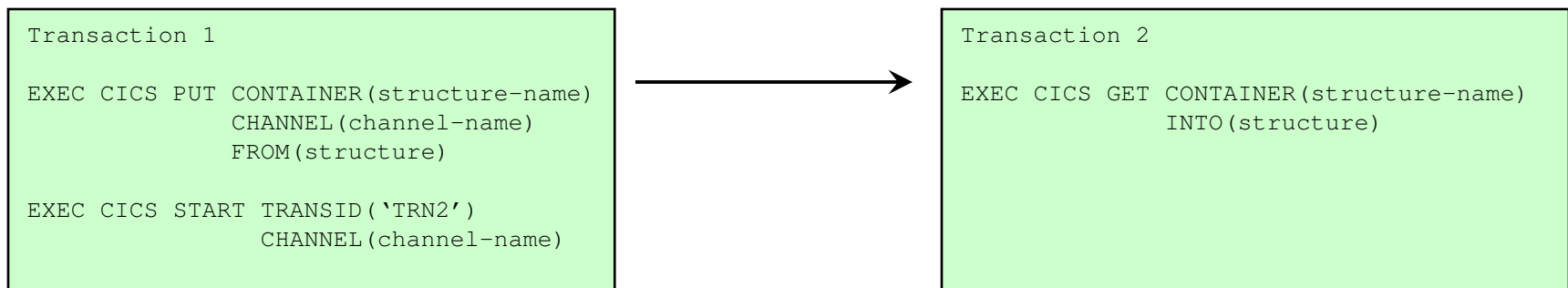
This is an example of the changes necessary to convert an application program that is using a COMMAREA to one using a channel and container. The example here only shows the commands which need to be added or changed. There is no attempt in this example to describe how the copybook structure can be simplified. Please refer to the “Best Practices” page for discussion on how the COMMAREA copybook should be evaluated.

Migration of Programs Using START

- Existing application with START data



- Changed application using Channels



Enhanced Inter-program Data Transfer - Notes

This is an example of the changes necessary to convert an application program that is using an EXEC CICS START with data to a START passing a channel. The example here only shows the commands which need to be added or changed. There is no attempt in this example to describe how the copybook structure can be simplified. Please refer to the "Best Practices" page for discussion on how the data area copybook should be evaluated.

Today a program may issue multiple STARTs with data for a single transaction id. CICS will start one instance of the transaction. The program can issue multiple RETRIEVES to get the data. When using the channel option on the start, CICS will start one transaction for each start request. The started transaction will be able to access the contents of a single channel. The started transaction will get a copy of the channel.

Best Practices – Defined Interface

- **Define the channel and container names in a copybook**
 - Shared definition of interface
- **Usually each container is a level 01 structure**
 - Avoids overloading copybooks
- **Define interface with structures**
 - May not be possible for COBOL
 - COBOL copybook in working storage.
 - COBOL structures may be in linkage section so separate.

```
* Channel name
01 INQUIRY-CHANNEL PIC X(16) VALUE 'inqcustrec'.

* Container names
01 CUSTOMER-NO      PIC X(16) VALUE 'custno'.
01 BRANCH-NO       PIC X(16) VALUE 'branchno'.
01 CUSTOMER-RECORD PIC X(16) VALUE 'custrec'.
```

```
* Structures
01 CUSTNO          PIC S9(8).
01 BRANCHNO.
  02 COUNTRY      PIC S9(4).
  02 REGION       PIC S9(4).
01 CREC.
  02 CUSTNAME     PIC X(80).
  02 CUSTADDR1   PIC X(80).
  02 CUSTADDR2   PIC X(80).
  02 CUSTADDR3   PIC X(80).
```

Enhanced Inter-program Data Transfer - Notes

Use a separate container for each structure in the copybook. Consider defining the names of the channel and the containers used in that channel in a copybook.

Best Practices – DPL Performance

- **Containers which are unchanged are not return on DPL**
- **There is no definition of output containers by the caller**
 - Containers created by the called program are returned

- **For optimal DPL performance**
 - Use separate containers for “read only” versus “read/write” or “write” data
 - Use separate containers for input and output
 - If a structure is optional make it a separate container
 - Use separate containers for error information

Enhanced Inter-program Data Transfer - Notes

It is possible to use a channel with a single container to replace your existing COMMAREA usage. While this may seem the simplest way to move from COMMAREAs to Channels and Containers it is not a good practice to do this. Since you are taking the time to change your application programs to exploit this new function you should implement the “best practices” for channels and containers.

The reason for this is that when using channels with DPL, only the changed containers need to be returned to the calling CICS region when a DPL is complete.

Use separate containers for read-only data versus read-write data. This will also improve the transmission efficiency between CICS regions. A simple example of this is containers for input and output. This will allow you to simplify your copybook structure and make your programs easier to understand and avoid the problems with REORDER overlays.

Use a separate container for each structure in the copybook. This will make the program easier to understand. In addition in some programs some output structures would be optional. A particular case of this is error information. This will lead to clearer documentation of the error information and improved transmission efficiency between CICS regions as the error container only needs to be sent if present.

When checking for an error container it is more efficient to issue a GET CONTAINER command and received a NOTFOUND condition than it is to initiate a browse of the containers in the channel.

Use separate containers for different data types, such as, binary data and character data. This will improve your ability to easily move between different code pages

Terminology

The Current Channel

PROGB

```
GET CONTAINER('Employee') INTO(emp-data)
GET CONTAINER('Branch') INTO(branch-data)
...
PUT CONTAINER('Payslip') FROM(pay-data)
```



No
CHANNEL
specified

- **The channel, if any, passed to the program by:**
 - LINK, XCTL, START or pseudo-conversation RETURN
- **Does not change during the life of the program**
 - The program may create other channels
- **Default for EXEC CICS commands that do not explicitly specify a channel name**

Enhanced Inter-program Data Transfer - Notes

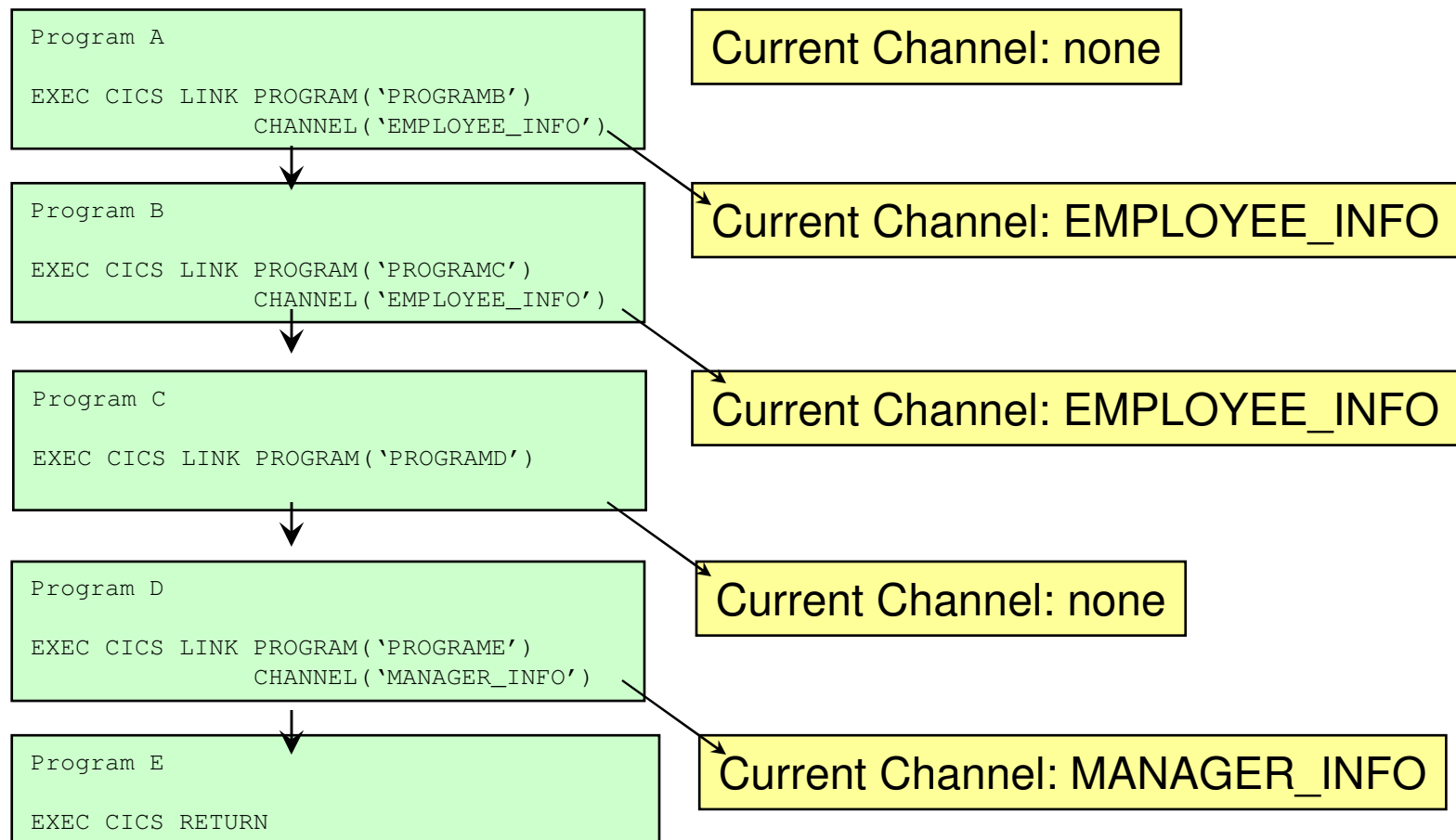
You may have spotted in the “Simple Example” that PROGB didn’t specify the channel name. It could have done so, but did not need to.

A program’s current channel is the channel, if any, that is passed to the program. The current channel is set by the calling program or transaction by issuing a transfer of control command with the channel parameter. The transfer of control commands that can utilize a channel are LINK, XCTL, START and pseudo-conversational RETURN.

While a called program can create new channels for passing information to other called programs its current channel does not change.

If a channel is not explicitly specified, the current channel is used as a default value for the CHANNEL (channel-name) parameter on EXEC CICS commands.

Current Channel



Enhanced Inter-program Data Transfer - Notes

This is an example of the program flow inside of an executing transaction. The programs link to each other passing information through the use of a channel. You will see that the initial program in the transaction, program A, does not have a current channel. This is because the transaction was not invoked by use of a RETURN TRANSID CHANNEL or by a START TRANSID CHANNEL command. Program A must explicitly specify the channel name in all channel commands that it invokes.

Program A then invokes program B with an LINK command with a channel specified. Program B has a current channel of EMPLOYEE_INFO. Program B then invokes program C passing along the EMPLOYEE_INFO channel. Program C will have a current channel of EMPLOYEE_INFO.

Program C then proceeds to LINK to program D but does not specify a channel (perhaps it continues to use a COMMAREA). Thus, program D does not have a current channel.

Finally, program D invokes program E with a LINK command and specifies a channel. Program E will have a current channel of MANAGER_INFO.

Channels Scope

- **The programs which can access a channel**

- **A program can access**
 - It's current channel
 - Any channels it creates

- **When no program in the link stack can access a channel it is deleted**
 - Can occur on RETURN or XCTL

- **Channels cannot be accessed by other tasks**

Enhanced Inter-program Data Transfer - Notes

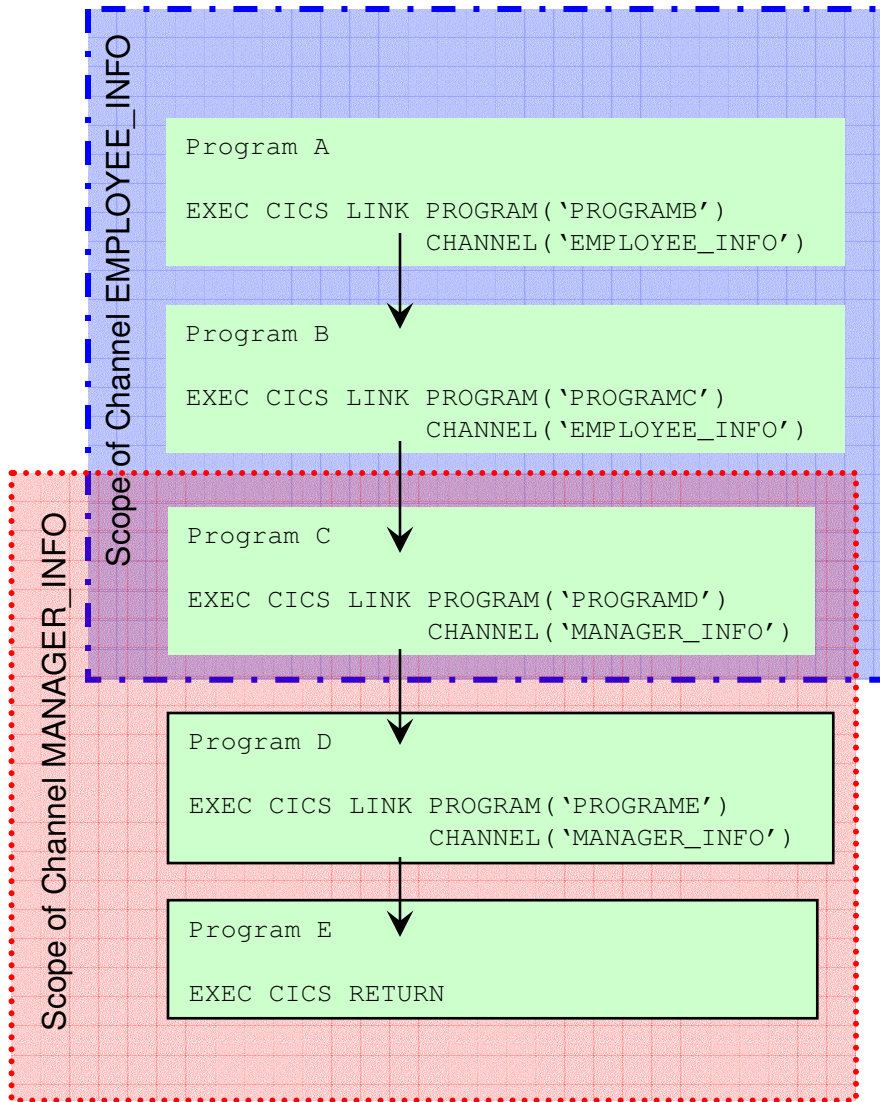
The scope of a channel is which applications can access this channel. Remember that the channel and its associated containers are only available to some of the programs in an executing transaction. The scope describes where the channel and its container data can be accessed.

An application program can only access its current channel and any new channels that it creates. CICS itself will use channels and containers and has the capability to create read-only containers within a channel. You will find read-only containers used in the new CICS Web Services function.

The other important thing to note about channel scope is that it controls when the channel will be deleted by CICS. When a channel goes out of scope, that is, no application program has the ability to access this channel, it is deleted. CICS will check to see if it can delete a channel at the time an EXEC CICS RETURN or XCTL command is issued.

Channels are usually created when the first PUT is done to them. They can also be created on a MOVE CONTAINER, and on the various commands to pass a channel (LINK, XCTL, START and pseudoconversational RETURN).

Channel Scope



Current Channel: none
Created Channel: EMPLOYEE_INFO

Current Channel: EMPLOYEE_INFO

Current Channel: EMPLOYEE_INFO
Created Channel: MANAGER_INFO

Current Channel: MANAGER_INFO

Current Channel: MANAGER_INFO

Enhanced Inter-program Data Transfer - Notes

This is an example of channel scope. This is the same example we looked at to determine the current channel of a program. The channel scope is indicated by the two overlay boxes.

The channel, `EMPLOYEE_INFO`, is created by program A and passed to subsequent programs B and C. This channel is not passed on the LINK to program D so the scope of the `EMPLOYEE_INFO` channel is programs A, B and C. When program A issues and EXEC CICS return, CICS will then delete this channel (assuming it is not passed to another transaction on the RETURN).

The channel, `MANAGER_INFO`, is created by program C and used to exchange information with programs D and E. The scope of the channel is programs C, D and E. When program C issues a RETURN command the `MANAGER_INFO` channel will go out of scope and be destroyed.

Note that both `EMPLOYEE_INFO` and `MANAGER_INFO` are in scope in program C.

Also note that programs D and E cannot access `EMPLOYEE_INFO`, but it is still in scope in programs in the link stack (i.e. A, B and C).

API Commands

API Commands

- **Container commands**
 - PUT CONTAINER
 - GET CONTAINER
 - MOVE CONTAINER
 - DELETE CONTAINER

- **Program transfer commands**
 - LINK PROGRAM
 - XCTL PROGRAM

- **Inquiry commands**
 - ASSIGN CHANNEL
 - STARTBROWSE CONTAINER
 - GETNEXT CONTAINER
 - ENDBROWSE CONTAINER

- **Transaction transfer**
 - RETURN TRANSID
 - START TRANSID CHANNEL

Container Commands

- **EXEC CICS PUT CONTAINER**
 - Copies data into a container within the channel
 - Overwrite existing data if container exists
 - Creates channel if that doesn't exist
- **EXEC CICS GET CONTAINER**
 - Retrieve the container data
- **EXEC CICS MOVE CONTAINER**
 - Moves a container from one channel to another
 - Can be used to rename a container
- **EXEC CICS DELETE CONTAINER**
 - Deletes a container from the channel
 - Doesn't delete the channel

Enhanced Inter-program Data Transfer - Notes

To create a channel, if it doesn't exist, and to place container data within the channel you will use an EXEC CICS PUT CONTAINER CHANNEL command.

To retrieve data passed to your program you will use an EXEC CICS GET CONTAINER CHANNEL command.

EXEC CICS PUT

- **CONTAINER (data-value)**
 - The name (1-16 characters) of the container
- **CHANNEL (data-value)**
 - The name (1-16 characters) of the channel that owns the container.
 - Defaults to current channel.
- **FROM (data-area)**
 - Specifies the data area from where the data to be saved is read.
- **FLENGTH (data-value)**
 - Specifies the length of the data area to be saved.
 - Can be 0 to very large.
 - This parameter is added by the translator if not specified (except C).
- **FROMCCSID (data-value)**
 - Specifies the current Coded Character Set of the character data to be put into the container. Defaults to the CCSID of the local CICS region.
- **DATATYPE (CVDA)**
 - BIT
 - The data in the container cannot be converted.
 - CHAR
 - Character data which can be converted.

Enhanced Inter-program Data Transfer - Notes

The format and options of the EXEC CICS PUT CONTAINER command.

EXEC CICS GET

- **CONTAINER (data-value)**
 - The name (1-16 characters) of the container
- **CHANNEL (data-value)**
 - The name (1-16 characters) of the channel that owns the container.
 - Defaults to current channel.
- **INTO (data-area)**
 - Specifies the data area into which the retrieved data is to be placed.
- **FLENGTH (data-area)**
 - Specifies the length of the data area to be read.
 - Returns the length actually read.
- **NODATA**
 - Specifies the only the length of the data in the container is to be returned. The length returned will take into account the INTOCCSID.
- **SET (ptr-ref)**
 - Specifies a data area in which the address of the retrieved data is returned
- **INTOCCSID (data-value)**
 - Specifies the current Coded Character Set into which the character data is to be converted. Defaults to the CCSID of the local CICS region.

Enhanced Inter-program Data Transfer - Notes

The format and options of the EXEC CICS GET CONTAINER command.

If you use the SET (ptr-ref) parameter the data area returned will be valid until:

- A subsequent GET CONTAINER command is issued for the same container in the same channel by any program that can access the channel or until the channel goes out of scope.
- A PUT CONTAINER changes the contents.
- The container is deleted by a DELETE CONTAINER command.
- The container is moved by a MOVE CONTAINER command.
- The channel goes out of scope.

This is CICS managed storage – do NOT issue a FREEMAIN against the storage area.
The storage is task storage with crumple zones.

Should you need to ensure the data is kept, move the data to your own application storage.

Scenario – Simple Data Conversion

- **PUT and GET can be used for data conversion**
- **Uses CICS or z/OS conversion tables**
- **Simple example of converting data to UTF-8**

```
EXEC CICS PUT CONTAINER('temp') CHANNEL('dummy')  
        FROM(ebcdic-data)  
        CHAR
```

```
EXEC CICS GET CONTAINER('temp') CHANNEL('dummy')  
        SET(utf8-ptr) FLENGTH(utf8-len)  
        INTOCCSID(1208)
```

Enhanced Inter-program Data Transfer - Notes

In the example the ebcdic data will be converted to utf-8 by the z/OS services.
In most cases SET should be used as the length of the resultant data may change.

Only character string data is supported.

EXEC CICS MOVE

- **CONTAINER (data-value)**
 - The name (1-16 characters) of the container
 - **CHANNEL (data-value)**
 - The name (1-16 characters) of the channel that owns the container.
 - Defaults to current channel.

 - **TOCHANNEL (data-value)**
 - Specifies the name of the channel that will own the target container
 - **AS (data-value)**
 - Specifies the name of the target container
-

Enhanced Inter-program Data Transfer - Notes

The format and options of the EXEC CICS MOVE CONTAINER command.

EXEC CICS DELETE

- **CONTAINER (data-value)**
 - The name (1-16 characters) of the container
- **CHANNEL (data-value)**
 - The name (1-16 characters) of the channel that owns the container.
 - Defaults to current channel.

Enhanced Inter-program Data Transfer - Notes

The format and options of the EXEC CICS DELETE CONTAINER command.

Program Transfer Commands

■ **LINK PROGRAM [CHANNEL|COMMAREA]**

- Links to the program, on a local or remote system, passing the channel and container data
- Creates the channel if it doesn't already exist

■ **XCTL PROGRAM [CHANNEL|COMMAREA]**

- Transfers control to the program on a local system passing the channel and container data
- Creates the channel if it doesn't already exist

Enhanced Inter-program Data Transfer - Notes

To link or transfer control to another program passing a channel and its associated containers you will use an EXEC CICS LINK PROGRAM CHANNEL or EXEC CICS XCTL PROGRAM CHANNEL command.

You may pass a channel or COMMAREA to a program but not both.

Transaction Transfer Commands

- **RETURN TRANSID [CHANNEL|COMMAREA]**
 - Returns control to CICS, passing the channel and container data to the next transaction id
 - Creates the channel if it doesn't already exist

- **START TRANSID [CHANNEL|FROM]**
 - Starts a task, on a local or remote system
 - Copies the named channel and container data and passing it to the started task
 - Creates the channel if it doesn't already exist

Enhanced Inter-program Data Transfer - Notes

To begin or continue a pseudo-conversational task you will use an EXEC CICS RETURN TRANSID CHANNEL command. Similar to commarea, this command is only valid at the highest logical level, that is, a program that is returning control to CICS. Remember that you may pass a channel or COMMAREA to a program but not both.

To start a new transaction and pass channel data to the new task you will use an EXEC CICS START TRANSID CHANNEL command. In the case of the START command the channel data is copied from the original channel and passed to the started transaction. At this point there are two separate copies of the channel data. If your starting program continues to make changes to the original container data in the channel it will not be reflected in the copy given to the started task. START can either pass a channel or start data.

Transactions started with the CHANNEL option will have a startcode of 'S'.
Timer options are not supported by the CHANNEL flavour of START.

Inquiry commands

- **ASSIGN CHANNEL(data-area)**
 - Blanks returned if no current channel

- **Channel browse commands**
 - STARTBROWSE CONTAINER [CHANNEL(data-area)]
 - GETNEXT CONTAINER (data-area)
 - Container names returned in no particular order
 - ENDBROWSE CONTAINER

Enhanced Inter-program Data Transfer - Notes

You may use the EXEC CICS ASSIGN command to determine what channel, if any, was passed to your program. This is useful if your program can be invoked by a number of different clients all of whom can pass your program a different channel.

The EXEC CICS ASSIGN CHANNEL command will return the 16 character name of the program's current channel if one exists. If no current channel exists blanks will be returned.

When your application program may be passed a channel with a varying number of containers it may use the container browse commands to determine all the containers present in the channel. If your program is only attempting to determine if a specific container has been passed, such as an error container, it is more efficient to issue a GET CONTAINER command against the single container name (e.g. ERROR).

The CICS commands comprising the browse interface for containers are STARTBROWSE CONTAINER, GETNEXT CONTAINER and ENDBROWSE CONTAINER.

The order in which the containers are returned is not guaranteed.



JCICS

New JCICS Classes

- **com.ibm.cics.server.Channel**
- **com.ibm.cics.server.Container**
- **com.ibm.cics.server.ContainerIterator**

Enhanced Inter-program Data Transfer - Notes

There are three new classes to support channels and containers in the Java programming language.

A Channel class used to create new containers in a channel.

A Container class used to place data in a container.

A ContainerIterator class used to browse the current channel.

Creating a new channel

- **Use the createChannel method of the task class**

```
Task t = Task.getTask();
```

```
Channel custData = t.createChannel("Customer_Data");
```

Enhanced Inter-program Data Transfer - Notes

This page intentionally left blank

Putting data into a container

1. **Create the container using the createContainer method of the Channel class**

```
Container custRec = custData.createContainer("Customer_Record");
```

2. **Add the data using the Container.put() method**

- Data can be added as a byte array or string

```
String custNo = "00054321";  
byte[] custRecIn = custNo.getBytes();  
custRec.put(custRecIn);
```

- Or

```
custRec.put("00054321");
```

Enhanced Inter-program Data Transfer - Notes

An example of creating a new container and placing data in the newly created container.

Passing a channel to another program or task

- **Passing a channel to another program**

- Use the link() and xctl() methods of the Program class

```
programX.link(custData);
```

```
programY.xctl(custData);
```

- **Passing a channel to another task**

- Use the issue method of the StartRequest class

```
StartRequest.issue(custData);
```

- Use the setNextChannel() method of the TerminalPrincipalFacility class:

```
terminalPF.setNextChannel(custData);
```

Enhanced Inter-program Data Transfer - Notes

An example of passing a channel to another program or another task.

Getting data from the current channel

1. Get the current channel object

- Use the `getCurrentChannel()` method of the `Task` class

```
Task t = Task.getTask();
```

```
Channel custData = t.getCurrentChannel();
```

```
Container custRec = custData.getContainer("Customer_Record");
```

2. Get data from a container

- Use the `Container.get()` method to read the data in a container into a byte array:

```
byte[] custInfo = custRec.get();
```

Enhanced Inter-program Data Transfer - Notes

An example of getting the current channel and retrieving data from a container within the channel.

Browsing the current channel

- **Use the ContainerIterator object**

```
Task t = Task.getTask();  
ContainerIterator ci = t.containerIterator();  
While (ci.hasNext()) {  
    Container custRec = ci.next();  
    // Process the container... }  
}
```

Enhanced Inter-program Data Transfer - Notes

An example of determining what containers are present in a channel using the ContainerIterator class.

Java Programming Example

```
import com.ibm.cics.server.*;
public class Payroll {
    ...
    // create the payroll_2004 channel
    Task t = Task.getTask();
    Channel payroll_2004 = t.createChannel("payroll-2004");

    // create the employee container Container employee =
    payroll_2004.createContainer("employee");

    // put the employee name into the container
    employee.put("John Doe");

    // create the wage container Container wage =
    payroll_2004.createContainer("wage");

    // put the wage into the container
    wage.put("2000");
}
```

Enhanced Inter-program Data Transfer - Notes

A sample program using channels and containers written in Java.

Java Programming Example...

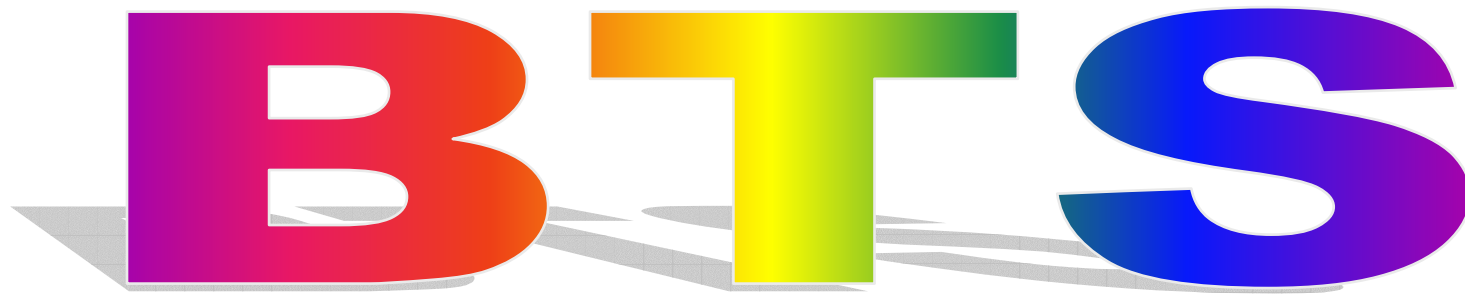
```
// Link to the PAYROLL program, passing the payroll_2004 channel
Program p = new Program();
p.setName("PAYR");
p.link(payroll_2004);

// Get the status container which has been returned
Container status = payroll_2004.getContainer("status");

// Get the status information
byte[] payrollStatus = status.get(); ... }
```

Enhanced Inter-program Data Transfer - Notes

A sample program using channels and containers written in Java.



The image features three large, bold, sans-serif letters: 'B', 'T', and 'S'. Each letter is filled with a horizontal rainbow gradient, transitioning from purple on the left to red, orange, yellow, green, and blue on the right. The letters are set against a white background and cast soft, grey shadows to the right, giving them a three-dimensional appearance.

Containers and Business Transaction Services

- **Containers have been used in BTS applications since CICS TS 1.3**
- **The container commands used in a channel context are similar to those in a BTS context**
 - GET, PUT
 - MOVE, DELETE
- **Programs that issue container commands can be used in both a channel and BTS context**
 - Some programming restrictions
 - Avoid specific reference to BTS or Channel context on commands
 - Cannot move containers between BTS and channels

Enhanced Inter-program Data Transfer - Notes

For those of you that have used CICS Business Transaction Services (BTS), available since CICS TS 1.3, you will be familiar with containers. 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. In fact, there have been white papers written to describe how a programmer might use BTS containers as a “Big COMMAREA”.

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 (e.g. GET, PUT, MOVE, DELETE).

It is possible to have the same server program invoked both a channel and 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 executed CICS will first check to see if there is a current channel. If there is, then the context of the command will be Channel. If there is no current channel, CICS will check to see if this is part of a BTS activity. If this is part of a BTS activity, then the context will be BTS. If the program has no channel context and no BTS context than an INVREQ will be raised.

Containers and Business Transaction Services...

```
! create the employee container on the payroll interface
EXEC CICS PUT CONTAINER('employee') CHANNEL('payroll') ....

! create the wage container on the payroll interface
EXEC CICS PUT CONTAINER('wage') CHANNEL('payroll') ....

! invoke the payroll service passing the payroll interface
EXEC CICS LINK PROGRAM('PAYR') CHANNEL('payroll')

! examine the status returned on the payroll interface
EXEC CICS GET CONTAINER('status') CHANNEL('payroll') ....
```

Simple clients use
a channel to pass
containers to the
service

Container-aware programs
should be insensitive to the
Type of containers that are
presented to them

```
DEFINE ACTIVITY('payroll') PROGRAM('payact')

! create the employee container on the payroll interface
EXEC CICS PUT CONTAINER('employee') ACTIVITY('payroll')
FROM('Fred Smith')

! create the wage container on the payroll interface
EXEC CICS PUT CONTAINER('wage') ACTIVITY('payroll') FROM('10
pounds')

! invoke the payroll service passing the payroll interface
EXEC CICS LINK ACTIVITY('payroll')

! examine the status returned on the payroll interface
EXEC CICS GET CONTAINER('status') ACTIVITY('payroll')
INTO(status)
```

BTS wrapper
controls a more
sophisticated
application

```
Program "PAYACT"
EXEC CICS RETRIEVE EVENT(...
WHEN('....
EXEC CICS LINK PROGRAM('payt')
```

```
Program "PAYR"
! get the employee passed into this program
EXEC CICS GET CONTAINER('employee') INTO(emp)
:
:
! return the status to the caller
EXEC CICS PUT CONTAINER('status') FROM('OK')
```

Enhanced Inter-program Data Transfer - Notes

This is an example of how a program might be designed and written to use containers in both a Channel and BTS context.

System Programming

Global User Exits

- **Global User Exits can create and pass channels and containers to programs they call**
- **Changes to Global User Exits**
 - Existence bits with channel name passed to exits
 - XICERREQ, XICEREQC
 - XPCREQ, XPCEREQC
 - Channel name added to the PCUE parameter list
 - XPCFTCH
 - Exits cannot access contents of channels

Enhanced Inter-program Data Transfer - Notes

CICS Global User Exits (GLUEs) are eligible to create channels and containers for their own use.

The parameter list passed to a number of GLUEs changes slightly with the addition of existence bits to signify the presence of a application's channel name. At this time the exit is not able to examine the contents of the channel. This restriction includes browsing the channel to determine container names as well as issuing a GET CONTAINER command to retrieve the application data.

Task Related User Exits

- **Task Related User Exits can create and pass channels and containers to programs they call**

Enhanced Inter-program Data Transfer - Notes

CICS Task Related User Exits (TRUEs) are eligible to create channels and containers for their own use.

At this time the task related user exit is not able to examine the contents of the channel. This restriction includes browsing the channel to determine container names as well as issuing a GET CONTAINER command to retrieve the application data.

User Replaceable Modules

- **User Replaceable Modules can create and pass channels and containers to programs they call**
 - URMs may not access contents of application channels
- **Changes to User Replaceable Modules**
 - Dynamic and distributed routing copybook
 - DYRCHANL (new field)
 - Name of channel associated with the request
 - DYRLEVEL (existing field)
 - Level of CICS AOR required to successfully process a routed request
 - > X'03' – requires a CICS TS 3.1 system
 - DYRTYPE (existing request)
 - Type of request for which the routing program is invoked
 - > 2 - Terminal related START with no data and no channel
 - > 3 - Terminal related START with data but no channel
 - > 4 – Program link with no channel
 - > 6 – Non-terminal related START with no channel
 - > 9 - Program link with a channel
 - > A -Terminal related START with a channel
 - > B - Non-terminal related START with a channel
 - DYRVER (existing field)
 - Version number of the dynamic routing program interface
 - CICS TS V3.1 number is 10

Enhanced Inter-program Data Transfer - Notes

CICS User Replaceable Modules (URMs) are eligible to create channels and containers for their own use.

The COMMAREA (parameter list) passed to the Dynamic Routing Programs changes slightly with the addition of the channel name in user by the application and changes to the target AOR level and the type of request.

At this time the URM is not able to examine the contents of the channel. This restriction includes browsing the channel to determine container names as well as issuing a GET CONTAINER command to retrieve the application data.

Monitoring

- **New monitoring group DFHCHNL**
 - PGTOTCCT
 - Total number of CICS requests for channel containers for the task
 - PGBRWCCT
 - Number of browse requests for channel containers for the task
 - PGGETCT
 - Number if GET CONTAINER requests for the task
 - PGPUTCCT
 - Number of PUT CONTAINER requests for the task
 - PGMOVCT
 - Number of MOVE CONTAINER requests for the task
 - PGGETCDL
 - Total length, in bytes, of all the GET CONTAINER data returned
 - PGPUTCDDL
 - Total length, in bytes, of all the GET CONTAINER data returned

Enhanced Inter-program Data Transfer - Notes

CICS adds new task performance monitoring information for channel and container usage.

Group DFHCHNL contains the following performance data:

321 (TYPE-A, 'PGTOTCCT', 4 BYTES)

The number of CICS requests for channel containers issued by the user task.

322 (TYPE-A, 'PGBRWCCT', 4 BYTES)

The number of CICS browse requests for channel containers issued by the user task.

323 (TYPE-A, 'PGGETCCT', 4 BYTES)

The number of GET CONTAINER requests for channel containers issued by the user task.

324 (TYPE-A, 'PGPUTCCT', 4 BYTES)

The number of PUT CONTAINER requests for channel containers issued by the user task.

325 (TYPE-A, 'PGMOVCCT', 4 BYTES)

The number of MOVE CONTAINER requests for channel containers issued by the user task.

326 (TYPE-A, 'PGGETCDL', 4 BYTES)

The total length, in bytes, of the data in the containers of all the GET CONTAINER CHANNEL commands issued by the user task.

327 (TYPE-A, 'PGPUTCDL', 4 BYTES)

The total length, in bytes, of the data in the containers of all the PUT CONTAINER CHANNEL commands issued by the user task.

Monitoring...

- **Changed monitoring group DFHPROG**
 - PCDLCSDL
 - Total length, in bytes, of the container data for a DPL
 - PCDLCRDL
 - Total length, in bytes, of the container data returned from a DPL
 - PCLNKCCT
 - Number of LINK requests issued with the channel option for this task
 - PCXCLCCT
 - Number of XCTL requests issued with the channel option for this task
 - PCDPLCCT
 - Total length, in bytes, of the container data passed on XCTLs
 - PCRTNCCT
 - Number of RETURN requests issued with the channel option for this task
 - PCRTNCDL
 - Total length, in bytes, of the container data RETURNed

Enhanced Inter-program Data Transfer - Notes

CICS adds new task performance monitoring information for channel and container usage.

The following new fields are added to group DFHPROG:

286 (TYPE-A, 'PCDLCS DL', 4 BYTES)

The total length, in bytes, of the data in the containers of all the distributed program link (DPL) requests issued with the CHANNEL option by the user task.

287 (TYPE-A, 'PCDL CR DL', 4 BYTES)

The total length, in bytes, of the data in the containers of all DPL RETURN CHANNEL commands issued by the user task.

306 (TYPE-A, 'PCLNK CCT', 4 BYTES)

Number of program LINK requests issued with the CHANNEL option by the user task.

307 (TYPE-A, 'PCXCL CCT', 4 BYTES)

Number of program XCTL requests issued with the CHANNEL option by the user task.

308 (TYPE-A, 'PCDPL CCT', 4 BYTES)

Number of program distributed program link (DPL) requests issued with the CHANNEL option by the user task

309 (TYPE-A, 'PCRTN CCT', 4 BYTES)

Number of pseudoconversational RETURN requests issued with the CHANNEL option by the user task.

310 (TYPE-A, 'PCRTN CDL', 4 BYTES)

The total length, in bytes, of the data in the containers of all the pseudoconversational RETURN CHANNEL commands issued by the user task.

Monitoring...

- **Changed monitoring group DFHTASK**
 - ICSTACCT
 - Number of START requests issued with the channel option
 - ICSTACDL
 - Length of the data in the containers of all the locally-executed START CHANNEL requests
 - ICSTRCCT
 - Number of interval control START CHANNEL requests to be executed on remote systems
 - ICSTRCDL', 4 BYTES)
 - Total length of the data in the containers of all the remotely executed START CHANNEL requests.

Enhanced Inter-program Data Transfer - Notes

CICS adds new task performance monitoring information for channel and container usage.

The following new fields are added to group DFHTASK:

065 (TYPE-A, 'ICSTACCT', 4 BYTES)

Total number of local interval control START requests, with the CHANNEL option, issued by the user task.

345 (TYPE-A, 'ICSTACDL', 4 BYTES)

Total length, in bytes, of the data in the containers of all the locally-executed START CHANNEL requests issued by the user task. This total includes the length of any headers to the data.

346 (TYPE-A, 'ICSTRCCT', 4 BYTES)

Total number of interval control START CHANNEL requests, to be executed on remote systems, issued by the user task.

347 (TYPE-A, 'ICSTRCDL', 4 BYTES)

Total length, in bytes, of the data in the containers of all the remotely-executed START CHANNEL requests issued by the user task. This total includes the length of any headers to the data.

Statistics

- **New fields in ISC/IRC system entry**
 - Number of LINK requests with channels, for function shipping
 - Number of bytes sent for function shipped channel requests
 - Number of bytes received for function shipped channel requests
- **New fields in Connections and Modenames**
 - Number of program control requests with channels function shipped for this connection
 - Number of bytes sent on channel function shipped requests for this connection
 - Number of bytes received on channel function shipped requests for this connection

Enhanced Inter-program Data Transfer - Notes

There are additions to "ISC/IRC system entry: Resource statistics" and to the "Connections and Modenames Report", both of which are mapped by the DFHA14DS DSECT. The new fields relate to channel data flowing across the connection.

A14EST_CHANNEL: is the number of program control link requests, with channels, for function shipping. This is a subset of the number in A14ESTPC.

A14EST_CHANNEL_SENT: is the number of bytes sent on function-shipped channel requests. This is the sum of the data in all the containers, excluding the file headers.

A14EST_CHANNEL_RECEIVED: is the number of bytes received on function-shipped channel requests. This is the sum of the data in all the containers, excluding the file headers.

Problem Determination

- **Maintenance required on prior releases to add error messages**
 - APARs required
 - CICS TS 1.3
 - PQ93048
 - CICS TS 2.2 and 2.3
 - PQ92437
 - CICS TXSeries
 - CICS TS for VSE
 - PQ83049
 - CICS for Windows
 - CICS for AS/400
 - SE15875
 - An attempt to ship a container to a previous release will result in a transaction abend
 - AXF9
 - AXTT
 - AZTD

Enhanced Inter-program Data Transfer - Notes

Maintenance will be required on prior releases of CICS to enable a clean error message to be produced if you attempt to use the channel option on a command shipped to a pre CICS TS 3.1 release

Summary

- **Allows more than 32k of data to be passed between CICS applications**
 - Program to program
 - LINK and XCTL
 - Transaction to transaction
 - START and RETURN
- **Minimal application changes required for exploitation**
- **Allows better structuring of application data**
 - Different containers to prevent overloaded copybooks
- **Allows for data conversion between different code pages**

Enhanced Inter-program Data Transfer - Notes

Channels and containers provide a significant benefit to the application program. The programmer now has the capability to exchange more than 32K of information between application programs and started tasks.

The channel and container construct allows the application suite to be enhanced by adding additional containers to the channel but will not affect programs that do not require the additional data.

The capability to pass multiple containers within a single channel offers the opportunity to simplify the copybook layout making the program easier to understand and future changes simpler to implement.