Session 4091

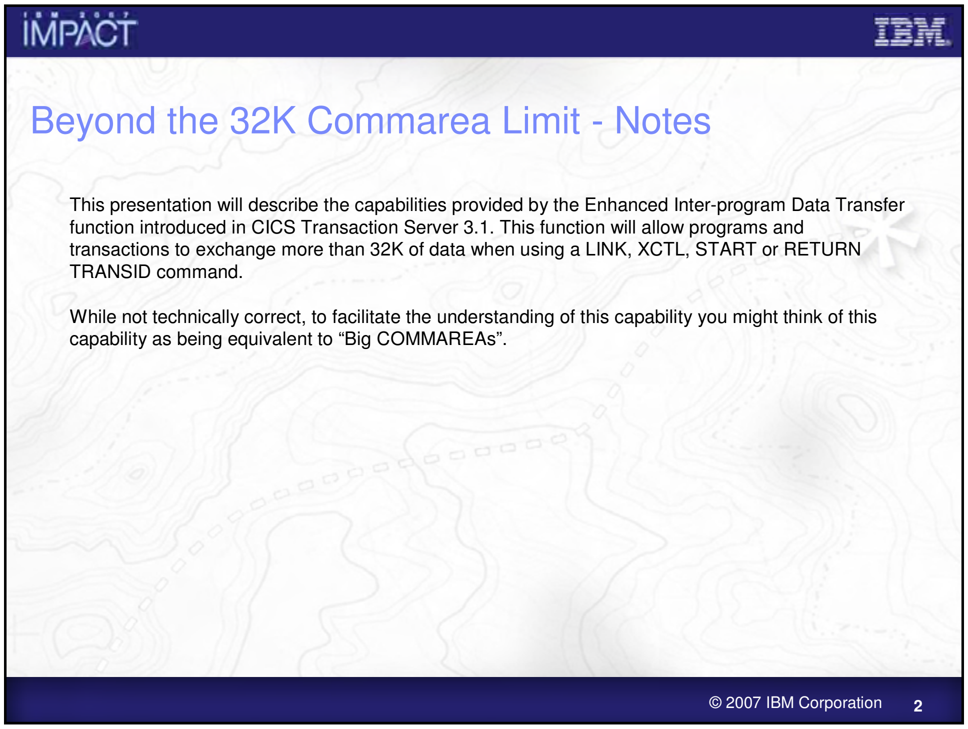
Beyond the 32K Commarea Limit

John Tilling

CICS Technical Planning and Strategy

Tilling@uk.ibm.com

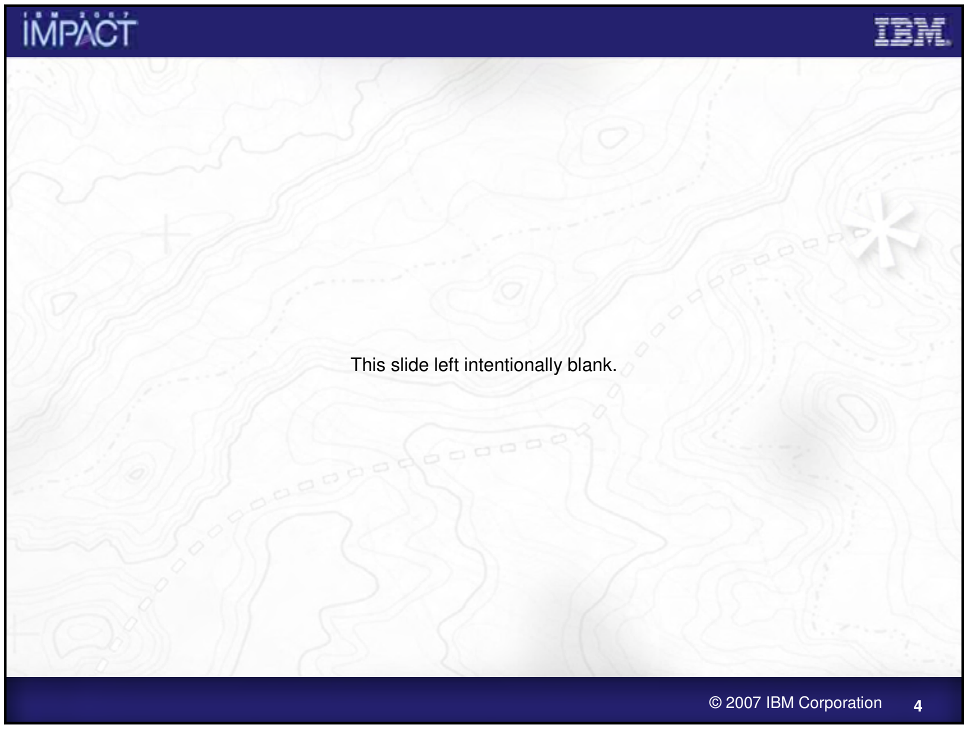
impact•venture*



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”.



This slide left intentionally blank.

Session Agenda

- The 32k COMMAREA
  - Why do we need to change?
- The "solution"
  - Containers and Channels
- Scenarios
  - Components
  - Migration
  - Best Practices
- API
- JCICS
- BTS
- System Programming
  - GLUEs, TRUEs, URMs
  - Monitoring and Statistics
- CICS TS 3.2 enhancements

Session agenda - 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.

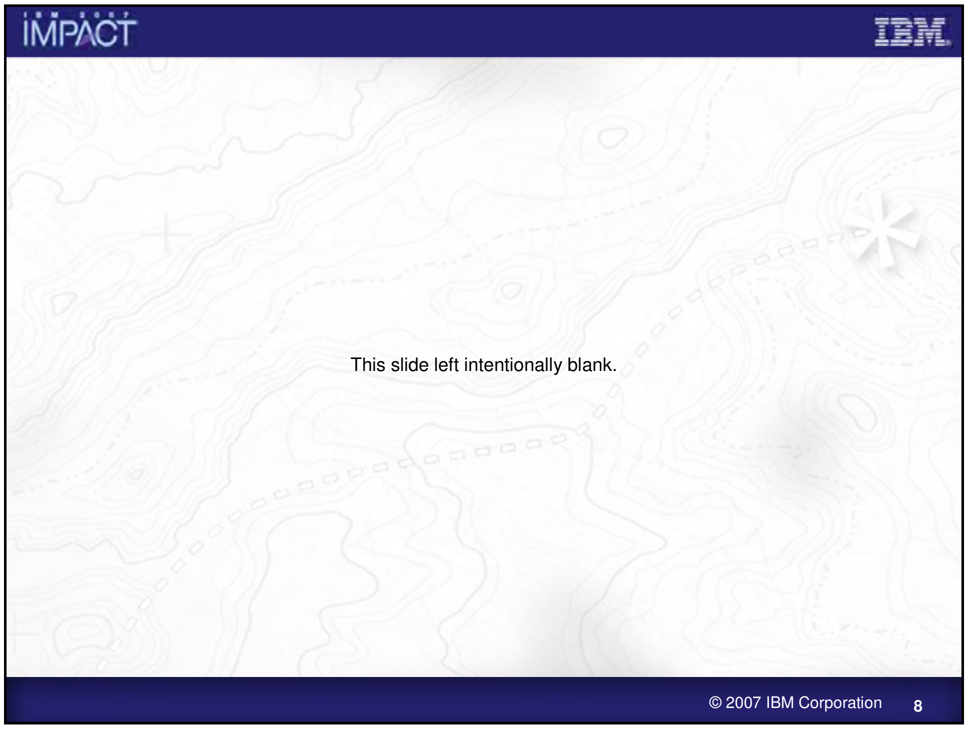
The CICS solution uses Channels and Containers to eliminate the problem. Channels are sets of Containers. Containers are named blocks of data that hold information to be passed between programs and transaction.

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

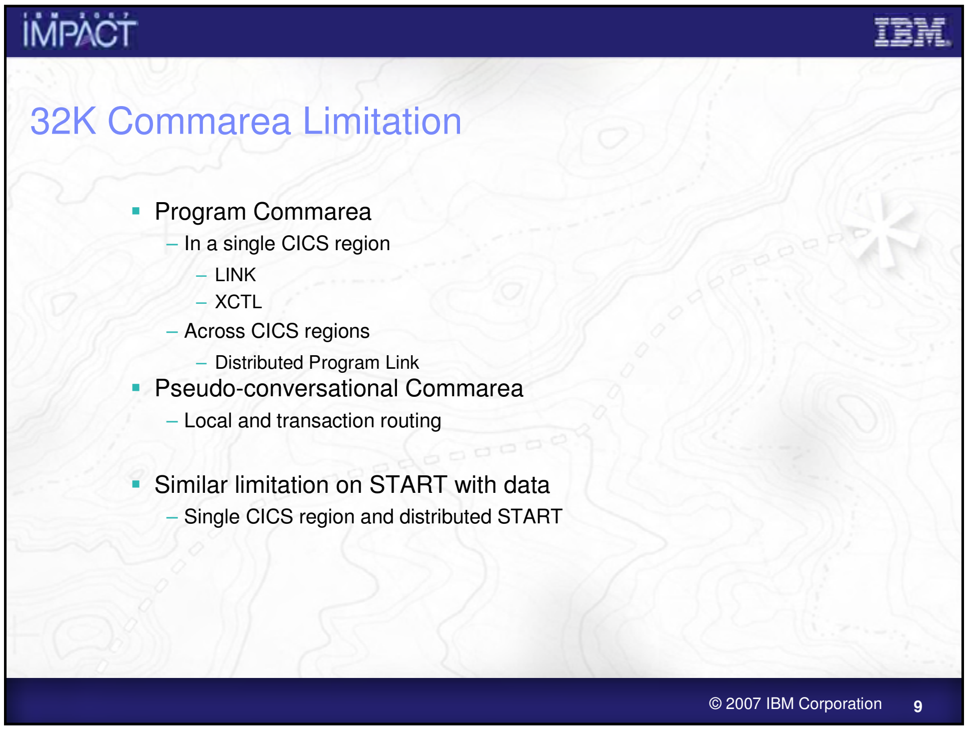
The effects of Channels and Containers on Global User Exits, Task Related User Exits and User Replaceable Modules will be described.

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

Why do we need to change?

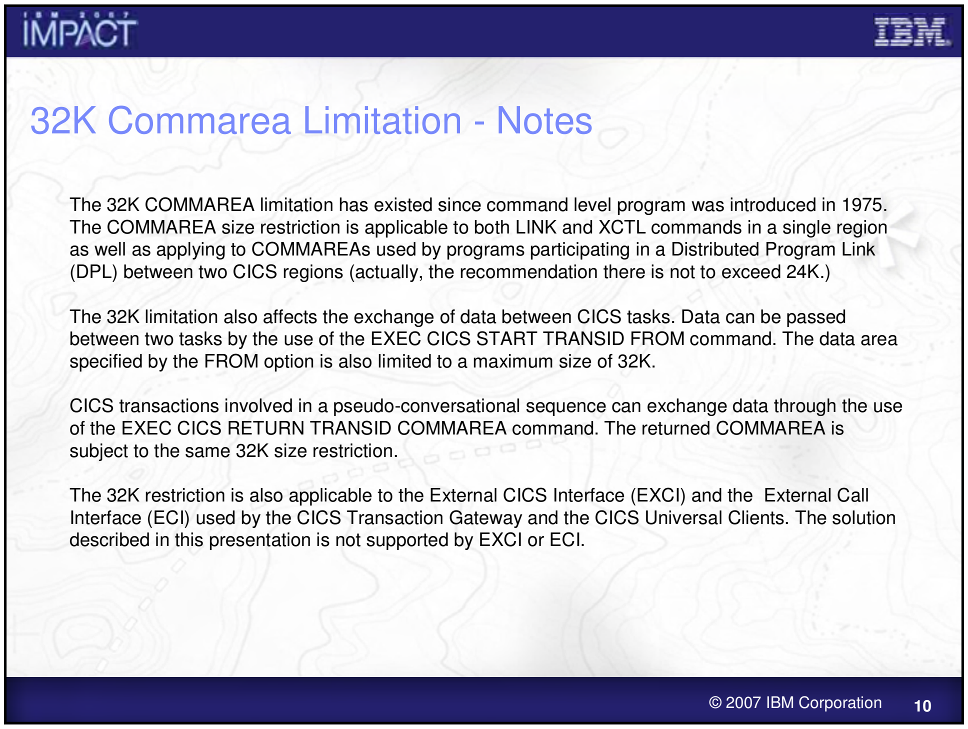


This slide left intentionally blank.



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



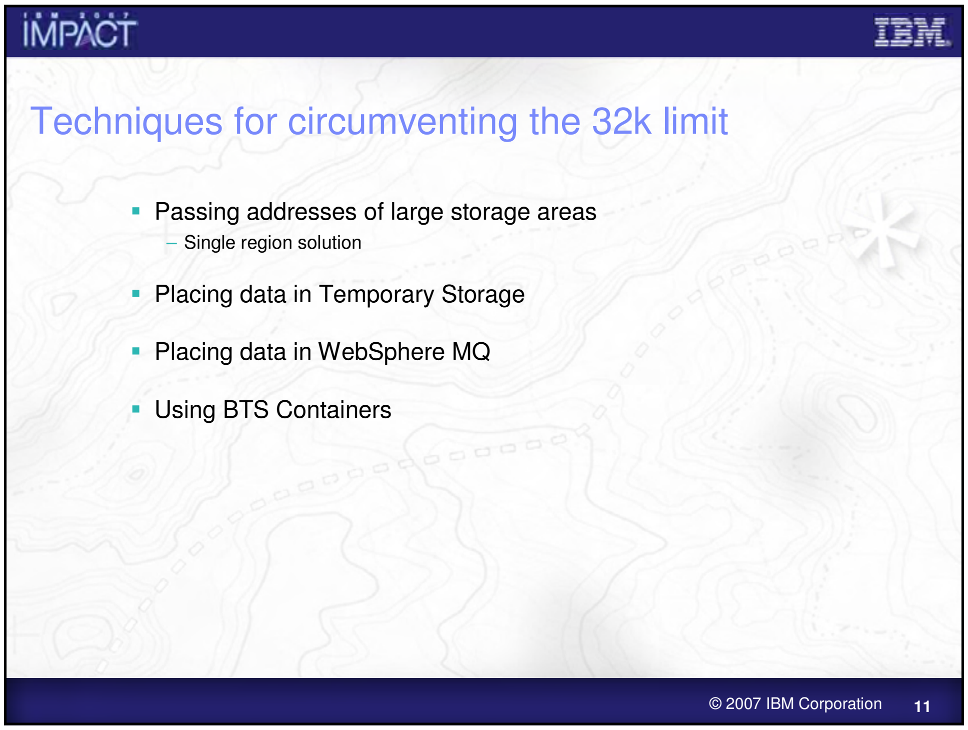
32K Commarea Limitation - Notes

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 (actually, the recommendation there is not to exceed 24K.)

The 32K limitation also affects the exchange of data between 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 also limited to a maximum size of 32K.

CICS transactions 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 the same 32K size restriction.

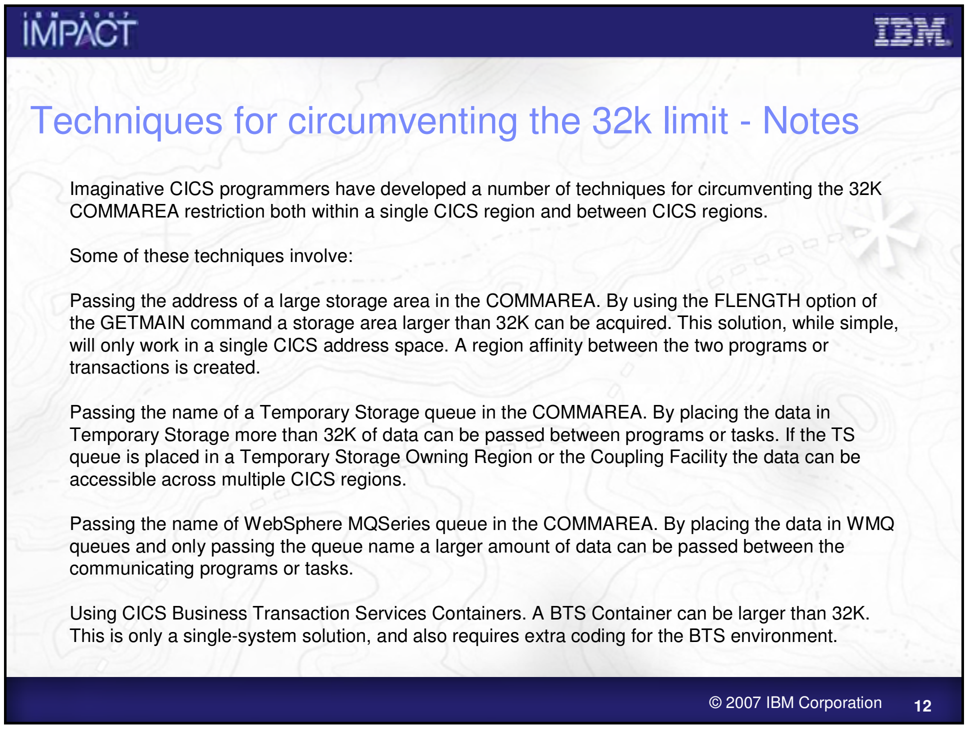
The 32K restriction is also applicable to the External CICS Interface (EXCI) and the External Call Interface (ECI) used by the CICS Transaction Gateway and the CICS Universal Clients. The solution described in this presentation is not supported by EXCI or ECI.



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 Containers

© 2007 IBM Corporation 11



Techniques for circumventing the 32k limit - Notes

Imaginative CICS programmers have developed a number of techniques for circumventing the 32K COMMAREA restriction both within a single CICS region and between 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 is 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 TS queue is placed in a Temporary Storage Owning Region or the Coupling Facility the data can be accessible across multiple CICS regions.

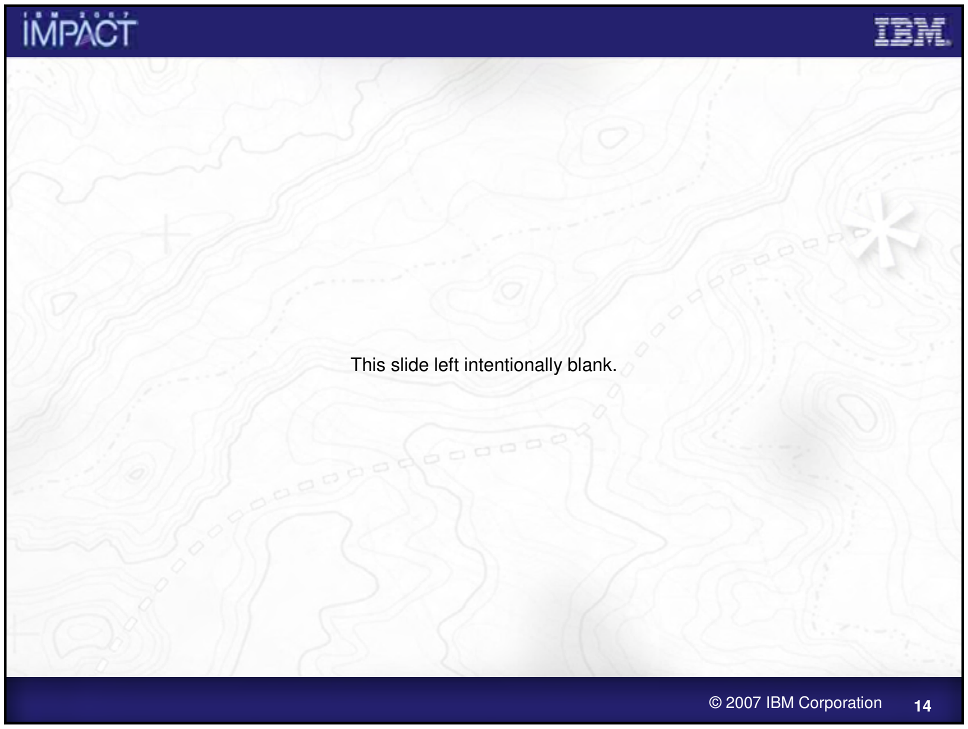
Passing the name of WebSphere MQSeries queue in the COMMAREA. By placing the data in WMQ queues and only passing the queue name a larger amount of data can be passed between the communicating programs or tasks.

Using CICS Business Transaction Services Containers. A BTS Container can be larger than 32K. This is only a single-system solution, and also requires extra coding for the BTS environment.

© 2007 IBM Corporation 12

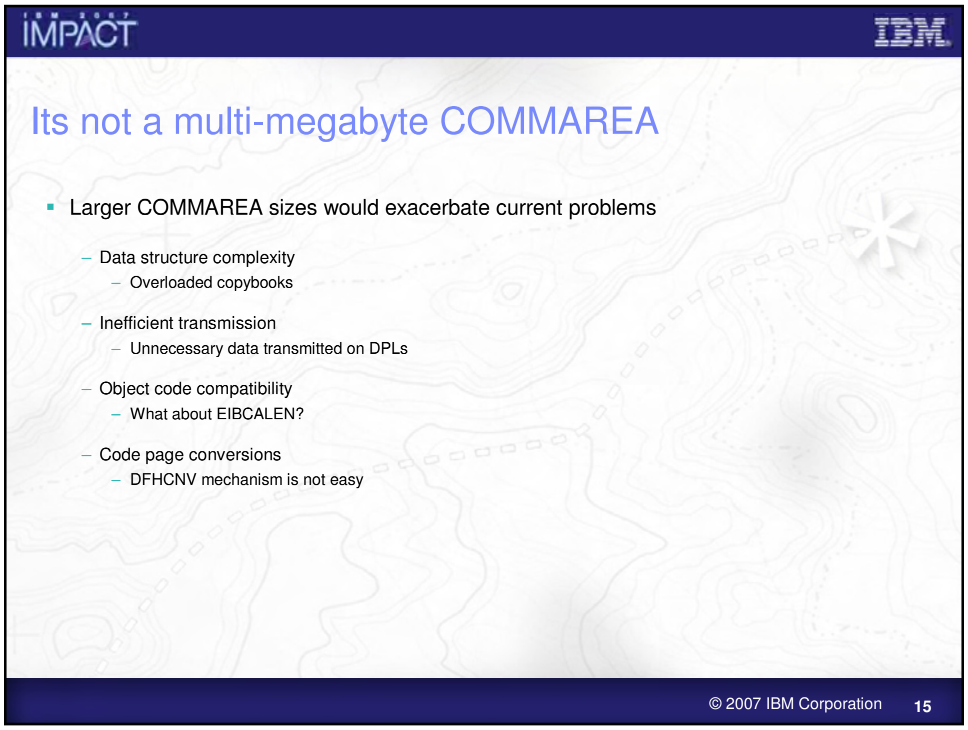
IMPACT IBM

The Solution



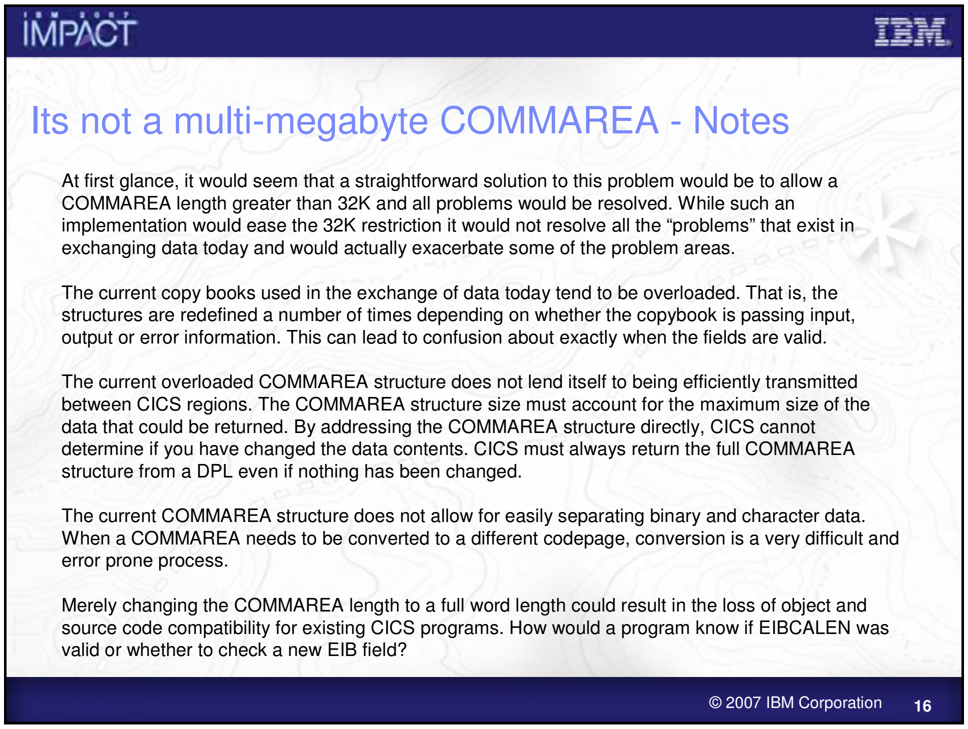
IMPACT IBM

This slide left intentionally blank.



Its 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



Its not a multi-megabyte COMMAREA - Notes

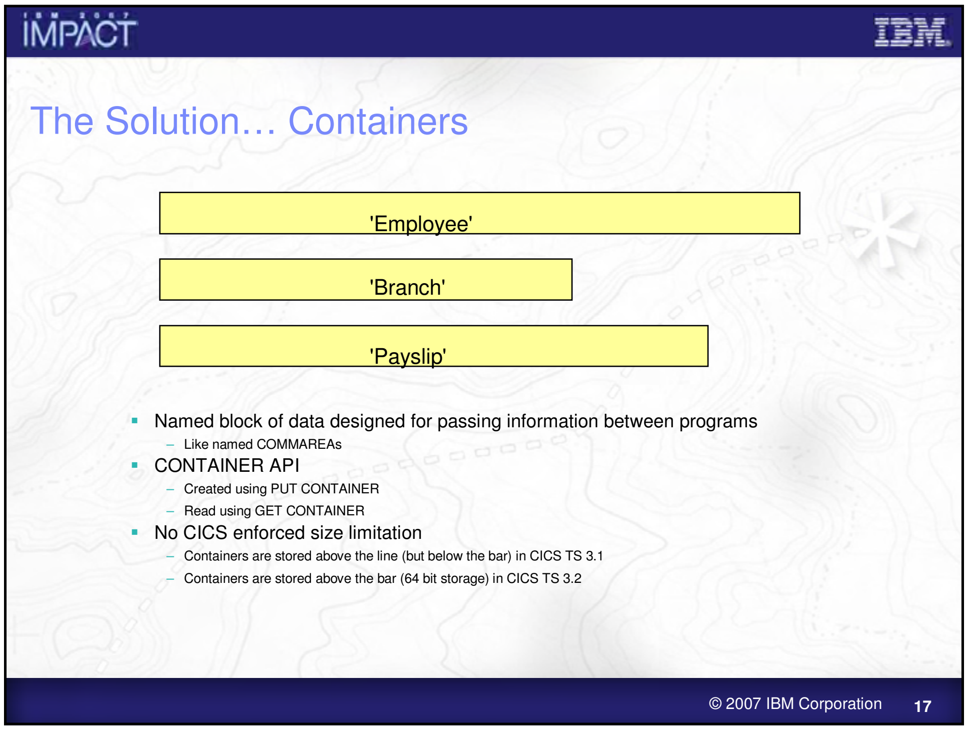
At first glance, it would seem that a straightforward solution to this problem would be to allow a COMMAREA length greater than 32K and all problems would be resolved. While such an implementation would ease the 32K restriction it would not resolve all the “problems” that exist in exchanging data today and would actually 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 can lead to confusion about exactly when the fields are valid.

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 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?



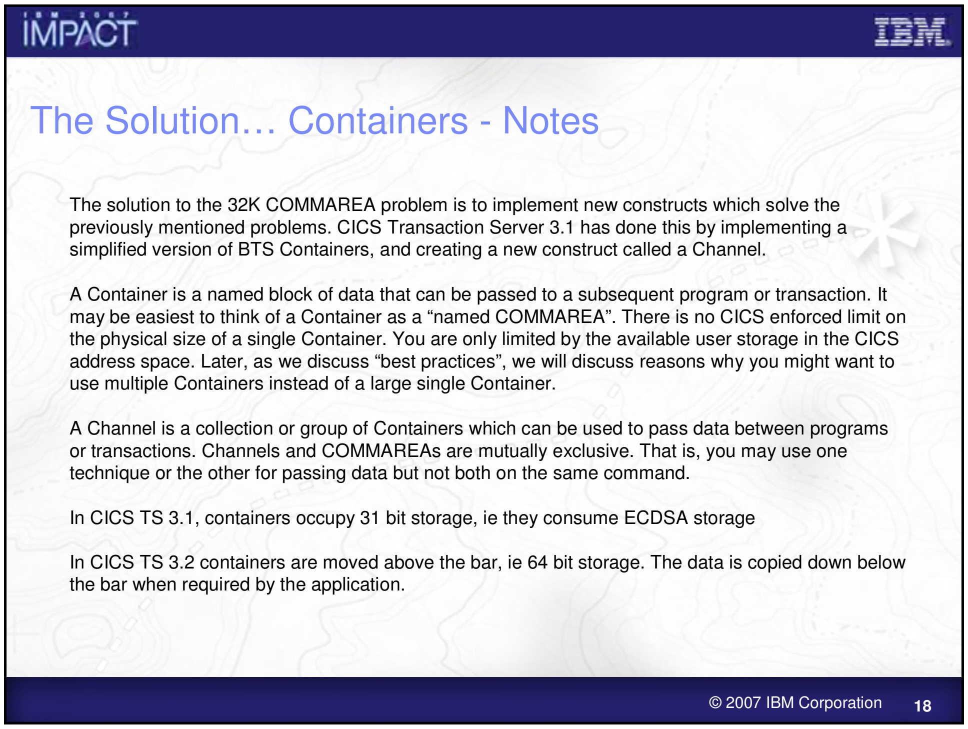
The Solution… Containers

'Employee'

'Branch'

'Payslip'

- Named block of data designed for passing information between programs
  - Like named COMMAREAs
- CONTAINER API
  - Created using PUT CONTAINER
  - Read using GET CONTAINER
- No CICS enforced size limitation
  - Containers are stored above the line (but below the bar) in CICS TS 3.1
  - Containers are stored above the bar (64 bit storage) in CICS TS 3.2



The Solution… Containers - 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 a simplified version of BTS Containers, and creating a new construct called a Channel.

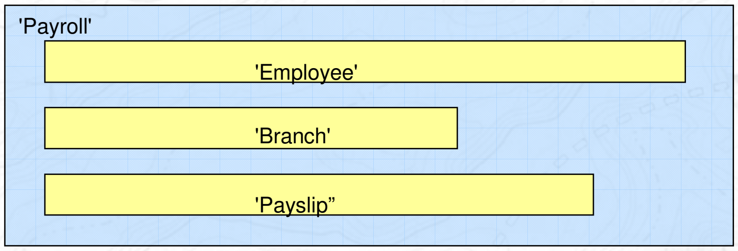
A Container is a named block of data that can be passed to a subsequent program or transaction. It may be easiest to think of a Container as a “named COMMAREA”. There is no CICS enforced limit on the physical size of a single Container. You are only limited by the available user storage in the CICS address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

A Channel is a collection or group of Containers which can be used to pass data between programs or transactions. Channels and COMMAREAs are mutually exclusive. That is, you may use one technique or the other for passing data but not both on the same command.

In CICS TS 3.1, containers occupy 31 bit storage, ie they consume ECDSA storage

In CICS TS 3.2 containers are moved above the bar, ie 64 bit storage. The data is copied down below the bar when required by the application.

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

The Solution… Channels - 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 release the storage they occupy.

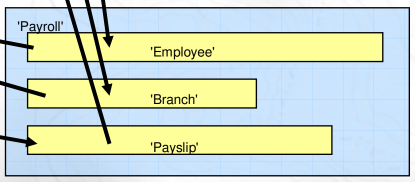
A Simple Example

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)
```



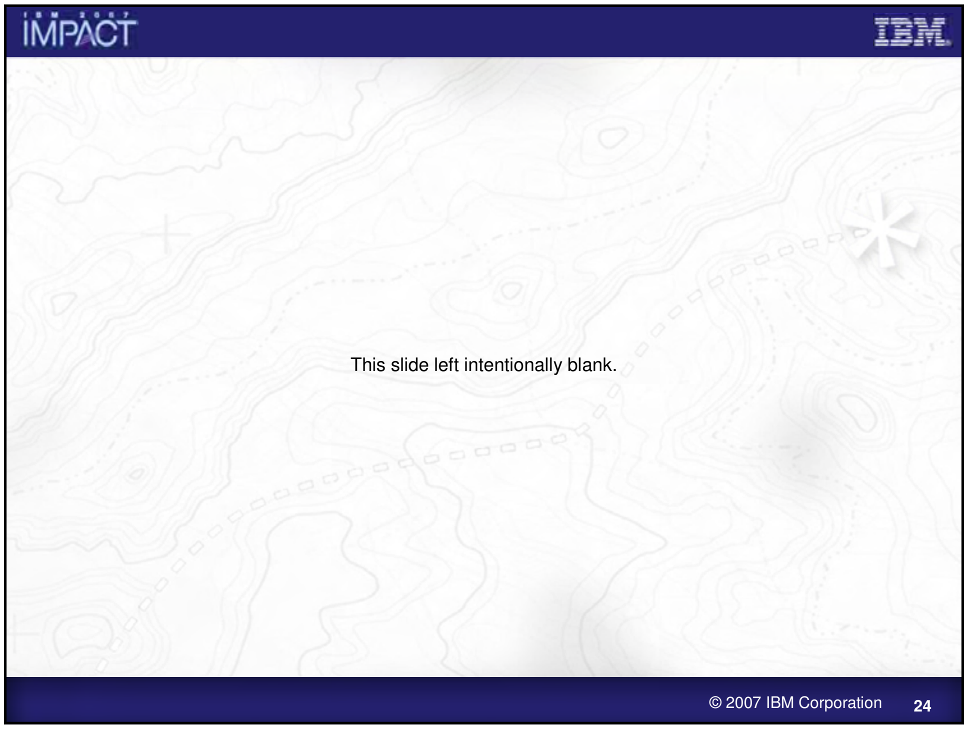
A Simple Example - Notes

Here an example of 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 LINK to another program. The name of the Channel is specified on the EXEC CICS LINK command and the target application program knows the name of the Channel it expects to be passed.

Scenarios

© 2007 IBM Corporation 23

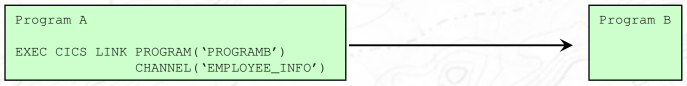


This slide left intentionally blank.

© 2007 IBM Corporation 24

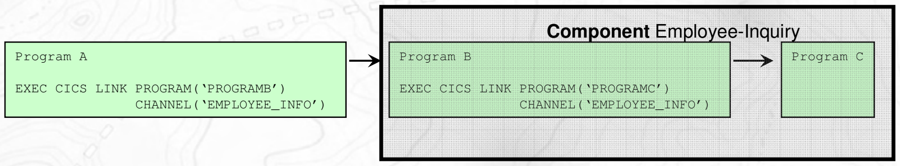
Basic Scenarios for using Channels

- One Channel / One Program



```
Program A

EXEC CICS LINK PROGRAM('PROGRAMB')
               CHANNEL('EMPLOYEE_INFO')
```

- One Channel / Multiple Programs
  - The Channel is the interface to a Component



```
Program A

EXEC CICS LINK PROGRAM('PROGRAMB')
               CHANNEL('EMPLOYEE_INFO')
```

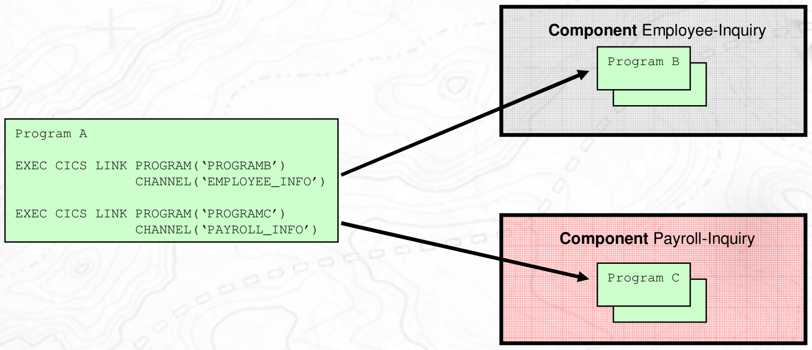
```
Program B

EXEC CICS LINK PROGRAM('PROGRAMC')
               CHANNEL('EMPLOYEE_INFO')
```

Basic Scenarios for using Channels - Notes

Another technique, shown here, is for the first program to create the Channel and Containers. This program then LINKs to another program passing the Channel along. The second program then LINKs to a third program, passing the same Channel. The same channel can be passsed along with the same Containers, or a different set of Containers, for the exchange of data.

Scenario - Multiple Components

- One Program / Multiple Channels



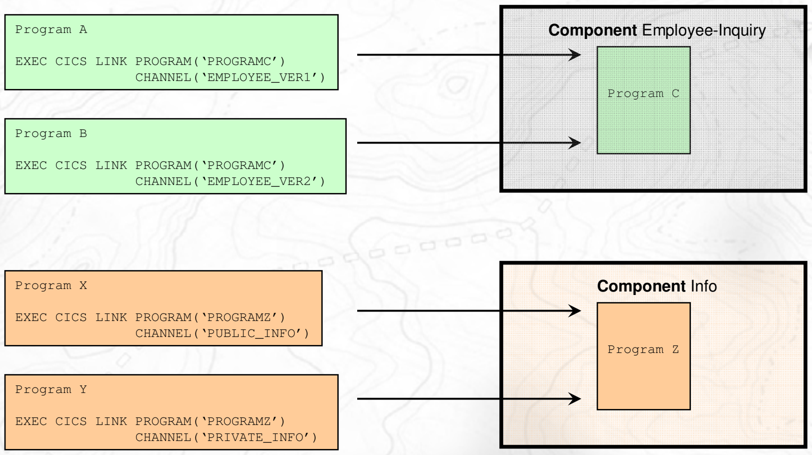
```
Program A

EXEC CICS LINK PROGRAM(‘PROGRAMB’)
               CHANNEL(‘EMPLOYEE_INFO’)

EXEC CICS LINK PROGRAM(‘PROGRAMC’)
               CHANNEL(‘PAYROLL_INFO’)
```

Scenario - Multiple Components - Notes

In this example, program A links to two other programs, B and C. Each program has a different Channel describing the different interfaces. This allows for much better separation of data than the old method of overloading the Commarea.

Scenario - Loose Binding

- Multiple Programs / Multiple Channels



```
Program A

EXEC CICS LINK PROGRAM('PROGRAMC')
               CHANNEL('EMPLOYEE_VER1')
```

```
Program B

EXEC CICS LINK PROGRAM('PROGRAMC')
               CHANNEL('EMPLOYEE_VER2')
```

```
Program X

EXEC CICS LINK PROGRAM('PROGRAMZ')
               CHANNEL('PUBLIC_INFO')
```

```
Program Y

EXEC CICS LINK PROGRAM('PROGRAMZ')
               CHANNEL('PRIVATE_INFO')
```

Scenario - Loose Binding - Notes

These two examples show what may be termed as loose binding, or loose coupling.

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 the 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 of 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

```
Program A

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

sing Channels

```
Program B

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

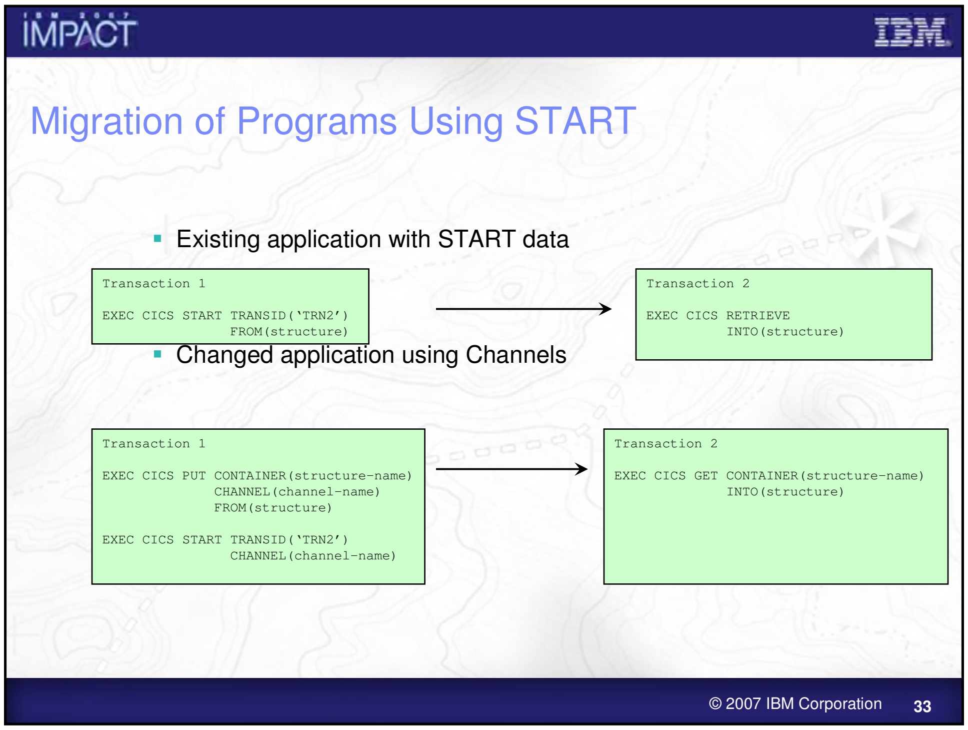
```
Program A

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)
SET(structure-ptr)

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

Migration of Programs Using LINK - 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. Note that, if Program B changes the Container data, it must PUT the Container back before returning, or the changes will not be visible to the caller.

There is no attempt in this example to describe how the copybook structure can be simplified. We will discuss the question of "Best Practices" later to see how the COMMAREA copybooks could be evaluated.



Migration of Programs Using START

- Existing application with START data

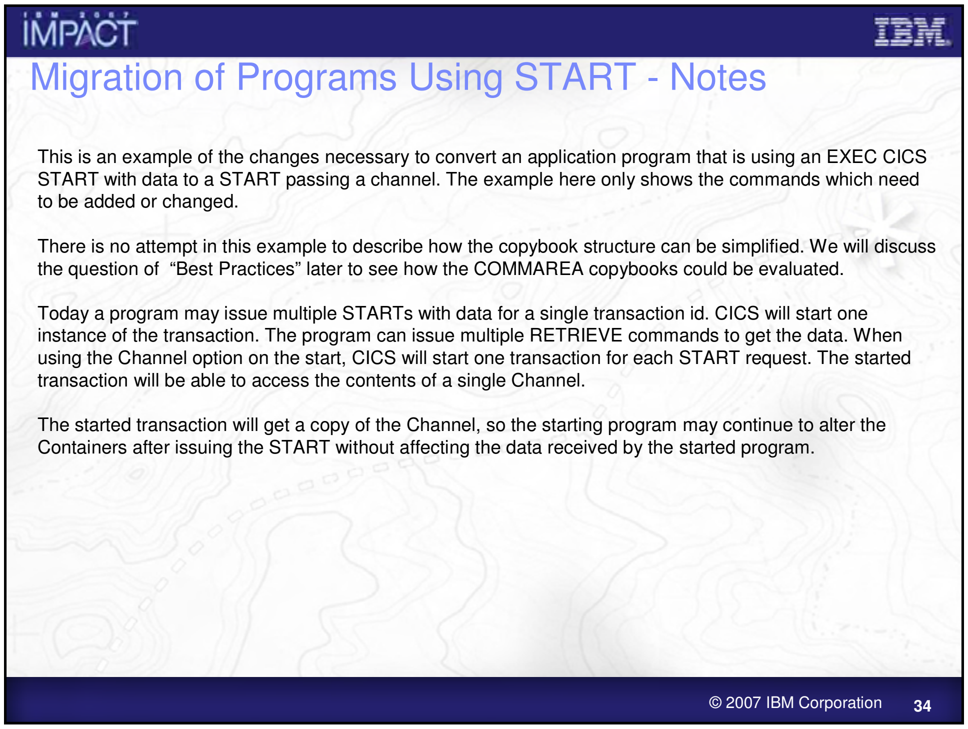
```
Transaction 1

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)
```



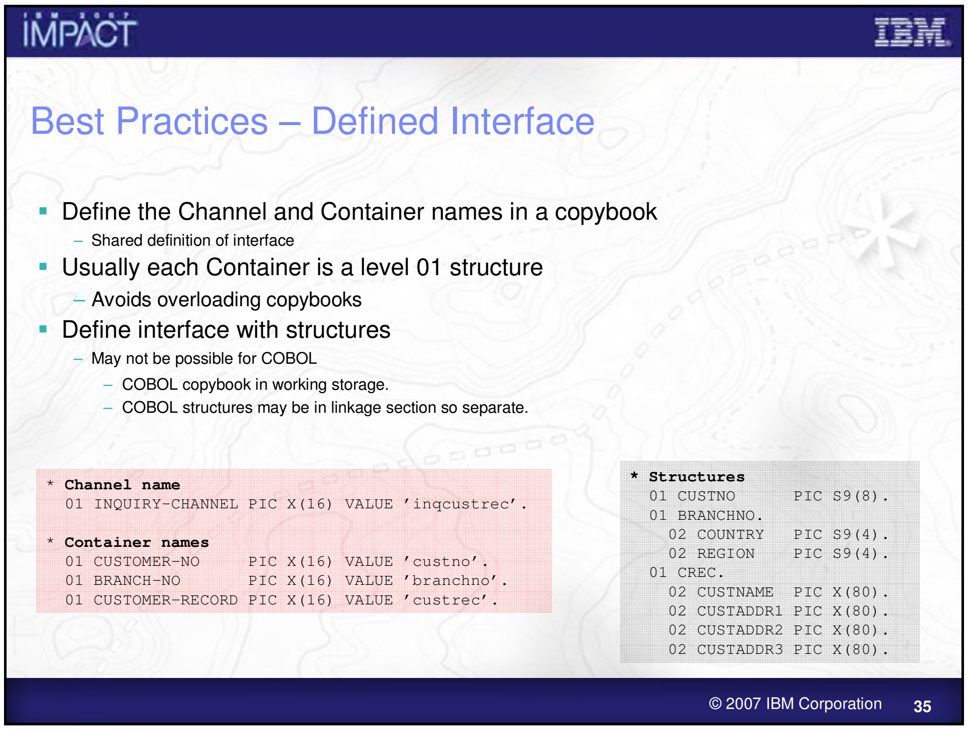
Migration of Programs Using START - Notes

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. We will discuss the question of "Best Practices" later to see how the COMMAREA copybooks could 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 RETRIEVE commands 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, so the starting program may continue to alter the Containers after issuing the START without affecting the data received by the started program.



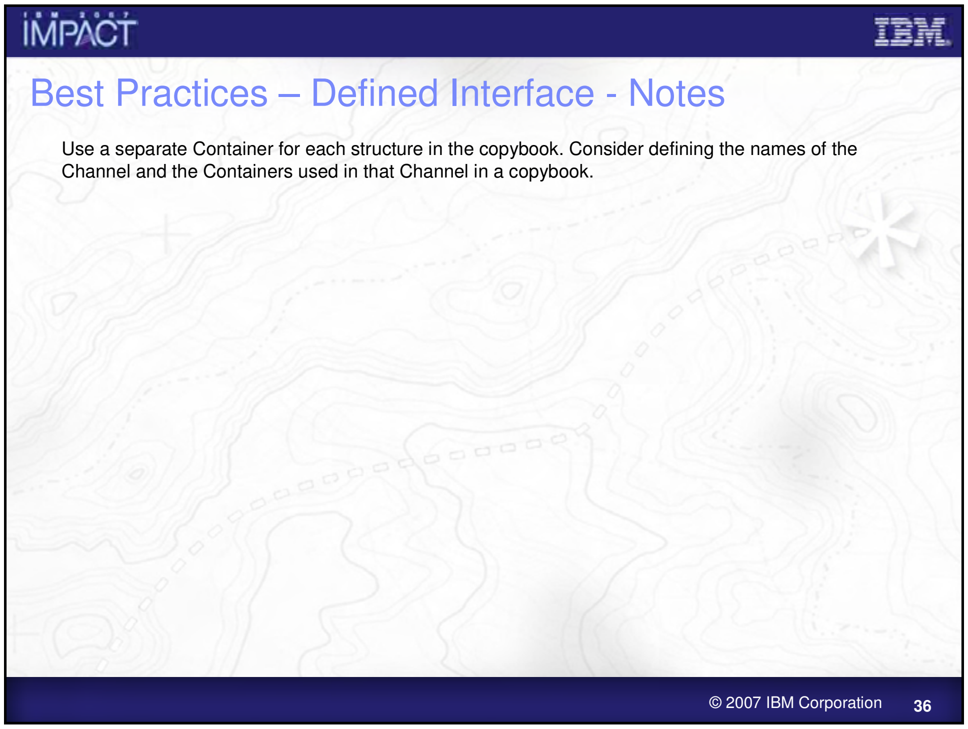
Best Practices – Defined Interface

- 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).
```

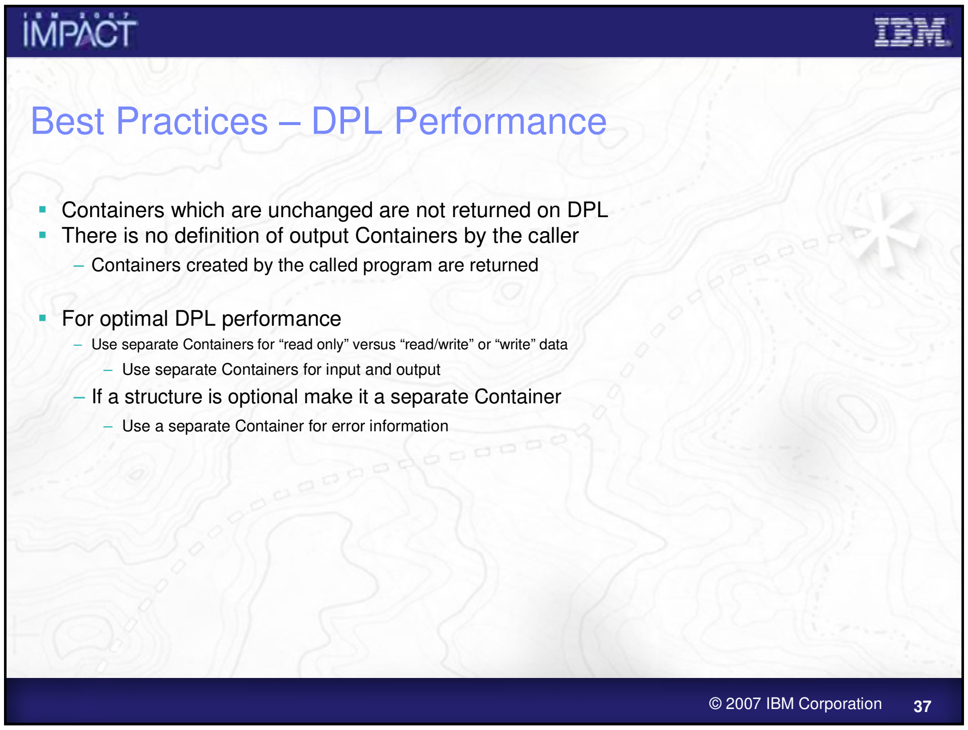
© 2007 IBM Corporation 35



Best Practices – Defined Interface - 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.

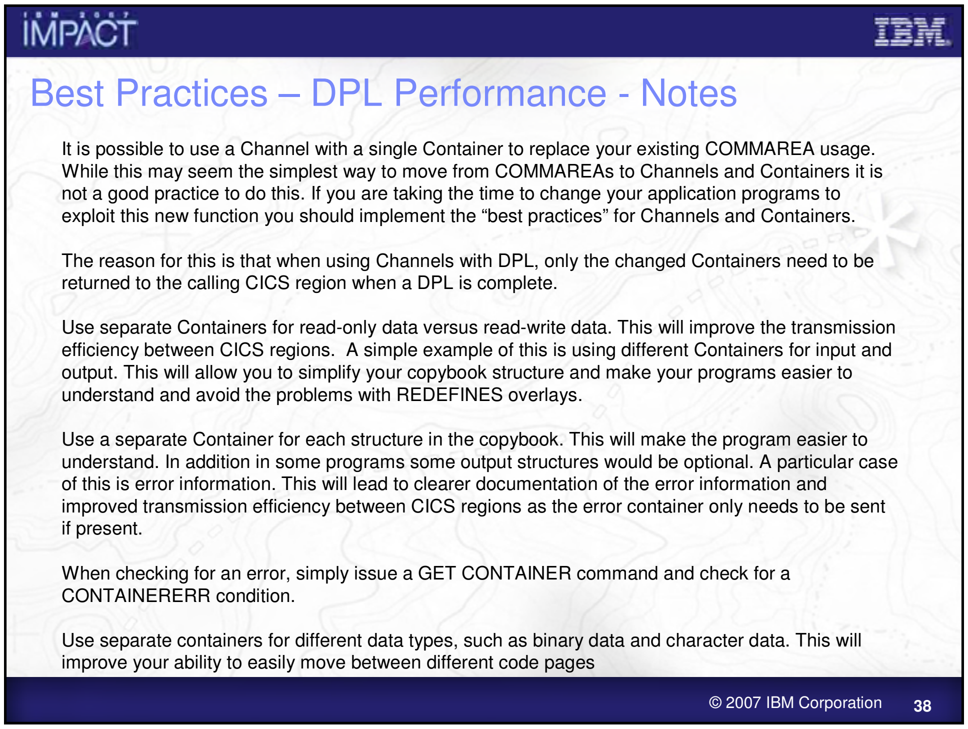
© 2007 IBM Corporation 36



Best Practices – DPL Performance

- Containers which are unchanged are not returned 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 a separate Container for error information

© 2007 IBM Corporation 37



Best Practices – DPL Performance - 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. If 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 improve the transmission efficiency between CICS regions. A simple example of this is using different 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 REDEFINES 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, simply issue a GET CONTAINER command and check for a CONTAINERERR condition.

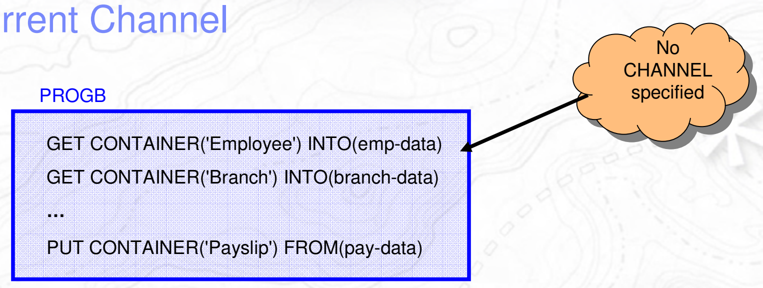
Use separate containers for different data types, such as binary data and character data. This will improve your ability to easily move between different code pages

© 2007 IBM Corporation 38

Terminology

This slide left intentionally blank.

The Current Channel



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

- 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

The Current Channel - Notes

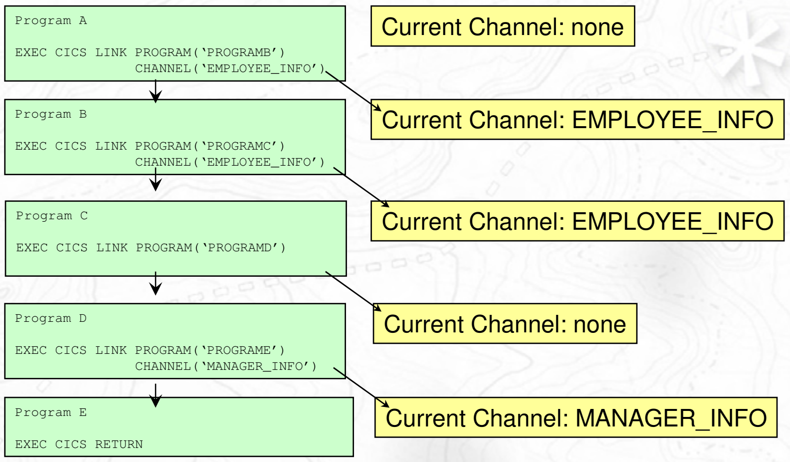
You may have spotted in the "Migration Example" that Program B didn't specify the channel name on the GET and PUT CONTAINER commands. It could have done so, but did not need to because it was dealing with its Current Channel

A program's Current Channel is the Channel, if any, that is passed to the program. The Current Channel is set when a program is started by a LINK, XCTL, START or pseudo-conversational RETURN command specifying a Channel.

While a called program can create new Channels for passing information to other called programs its Current Channel never changes.

If a Channel is not explicitly specified on a Container command, the Current Channel is used as the default value for the CHANNEL (channel-name) parameter.

Current Channel



```
Program A

EXEC CICS LINK PROGRAM('PROGRAMB')
               CHANNEL('EMPLOYEE_INFO')
```

```
Program B

EXEC CICS LINK PROGRAM('PROGRAMC')
               CHANNEL('EMPLOYEE_INFO')
```

```
Program C

EXEC CICS LINK PROGRAM('PROGRAMD')
```

```
Program D

EXEC CICS LINK PROGRAM('PROGRAME')
               CHANNEL('MANAGER_INFO')
```

```
Program E

EXEC CICS RETURN
```

Current Channel: none
Current Channel: EMPLOYEE_INFO
Current Channel: EMPLOYEE_INFO
Current Channel: none
Current Channel: MANAGER_INFO

Current Channel - 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 Container commands that it issues.

Program A then invokes program B with a LINK command with a Channel specified. Program B has a Current Channel of EMPLOYEE_INFO. Program B then invokes program C passing along the same EMPLOYEE_INFO Channel. Program C will also 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.

The Scope of a Channel

- The programs which can access a Channel
- A program can access
  - Its Current Channel
  - Any other 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

The Scope of a Channel - Notes

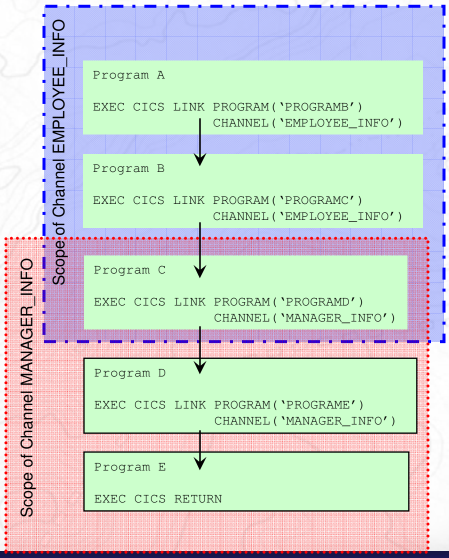
The scope of a Channel defines which programs have access to the Channel. Remember that the Channel and its associated Containers are only available to some of the programs in an executing transaction. The scope describes where the Channel and its Container data can be accessed.

An application program can only access its Current Channel and any new Channels that it creates. CICS itself can use Channels and Containers internally, 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 the 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.

It is not possible for a program in one task to access a Channel belonging to another task, even when both tasks own a Channel with the same name.

Channel Scope



```
Program A

EXEC CICS LINK PROGRAM('PROGRAMB')
               CHANNEL('EMPLOYEE_INFO')
```

Current Channel: none
Created Channel: EMPLOYEE_INFO

```
Program B

EXEC CICS LINK PROGRAM('PROGRAMC')
               CHANNEL('EMPLOYEE_INFO')
```

Current Channel: EMPLOYEE_INFO

```
Program C

EXEC CICS LINK PROGRAM('PROGRAMD')
               CHANNEL('MANAGER_INFO')
```

Current Channel: EMPLOYEE_INFO
Created Channel: MANAGER_INFO

```
Program D

EXEC CICS LINK PROGRAM('PROGRAME')
               CHANNEL('MANAGER_INFO')
```

Current Channel: MANAGER_INFO

```
Program E

EXEC CICS RETURN
```

Current Channel: MANAGER_INFO

Channel Scope - Notes

This is an example of how Channel scope operates. 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 an EXEC CICS return, CICS will then delete the Channel (assuming it is not passed to another transaction on the RETURN).

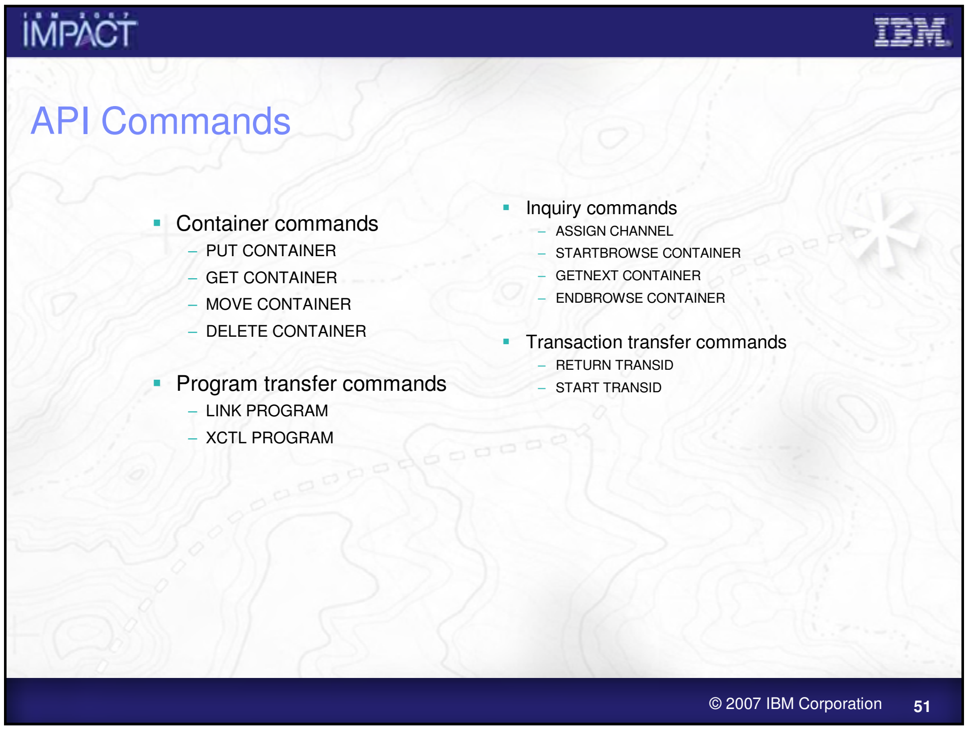
The Channel, MANAGER_INFO, is created by program C and is used to exchange information with programs D and E. The scope of the Channel is programs C, D and E. When program C issues a RETURN command the MANAGER_INFO Channel will go out of scope and be destroyed.

Note that both EMPLOYEE_INFO and MANAGER_INFO Channels 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).

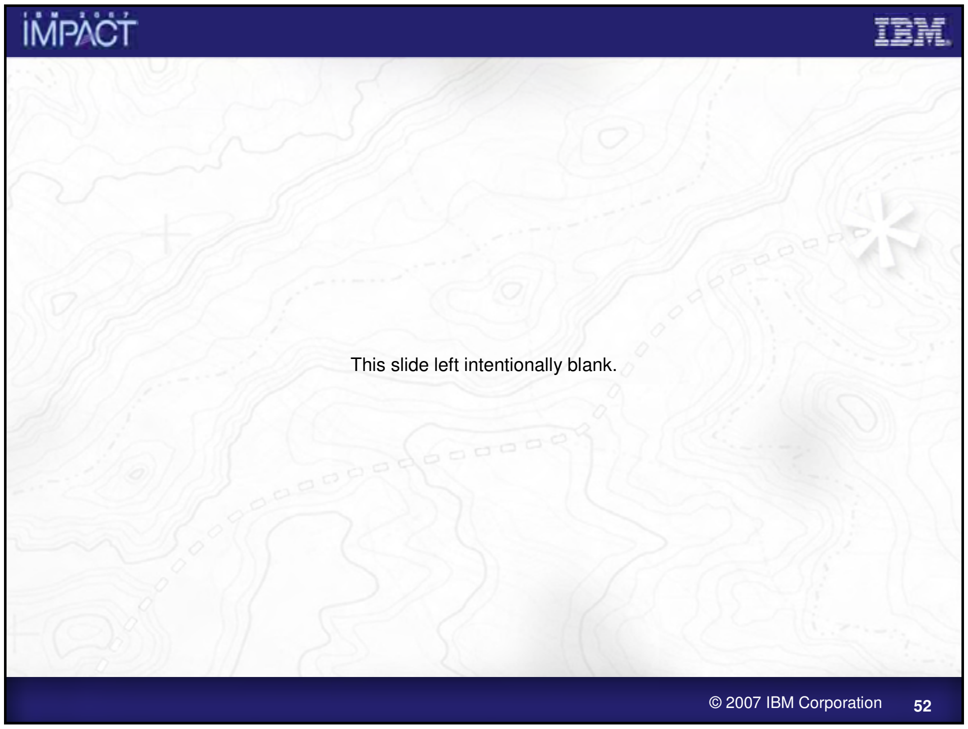
API Commands

This slide left intentionally blank.

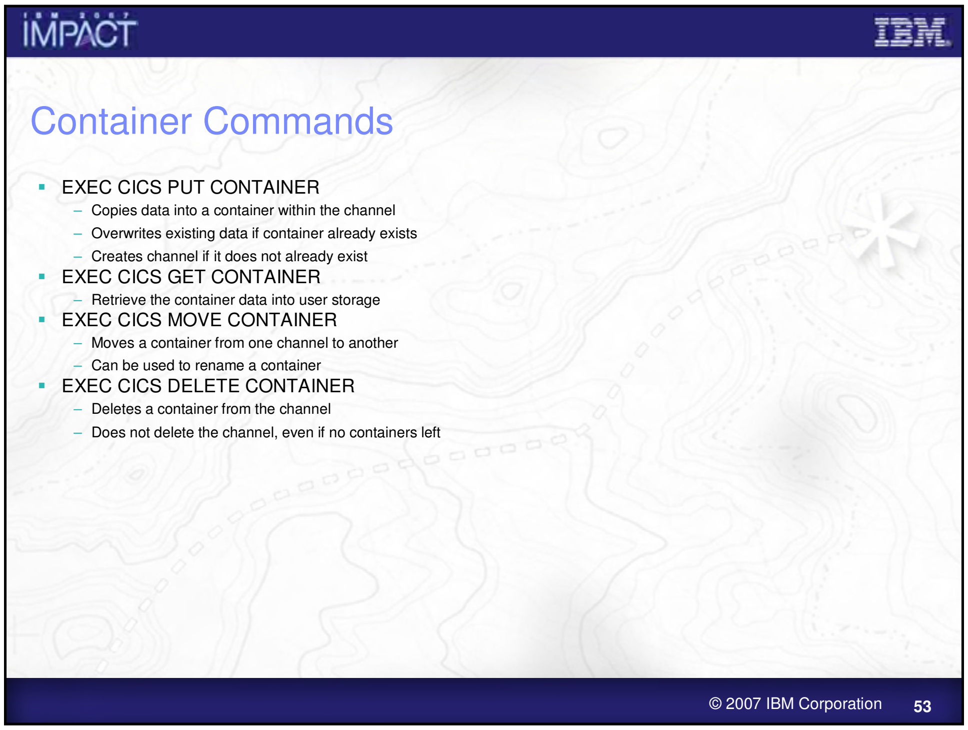


API Commands

- 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 commands
  - RETURN TRANSID
  - START TRANSID

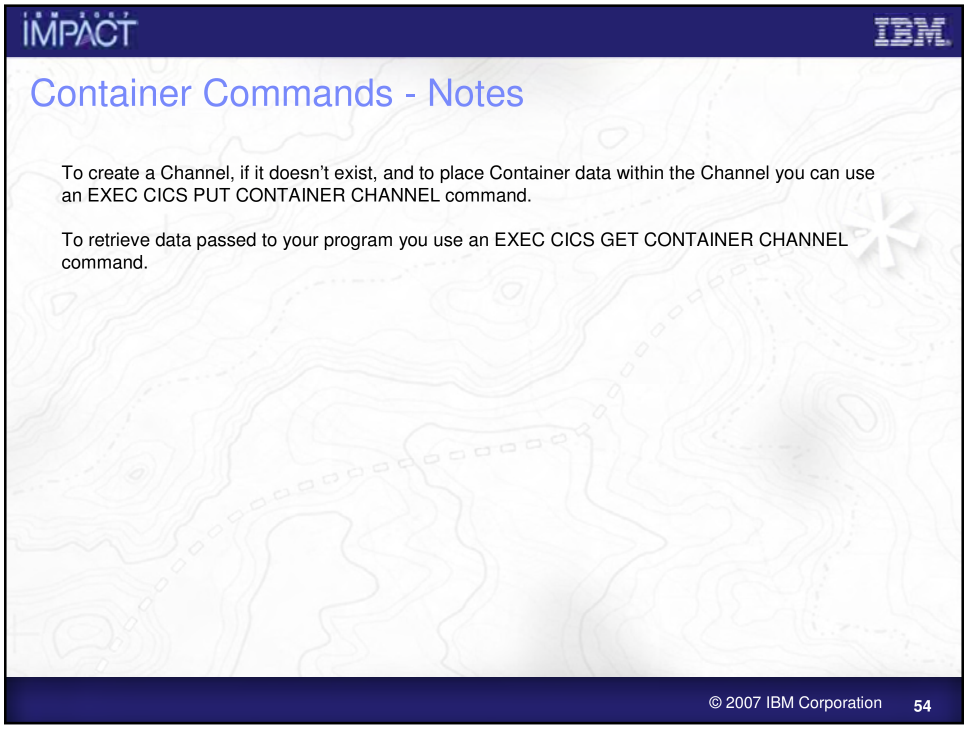


This slide left intentionally blank.



Container Commands

- EXEC CICS PUT CONTAINER
  - Copies data into a container within the channel
  - Overwrites existing data if container already exists
  - Creates channel if it does not already exist
- EXEC CICS GET CONTAINER
  - Retrieve the container data into user storage
- 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
  - Does not delete the channel, even if no containers left



Container Commands - Notes

To create a Channel, if it doesn’t exist, and to place Container data within the Channel you can use an EXEC CICS PUT CONTAINER CHANNEL command.

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

EXEC CICS PUT CONTAINER

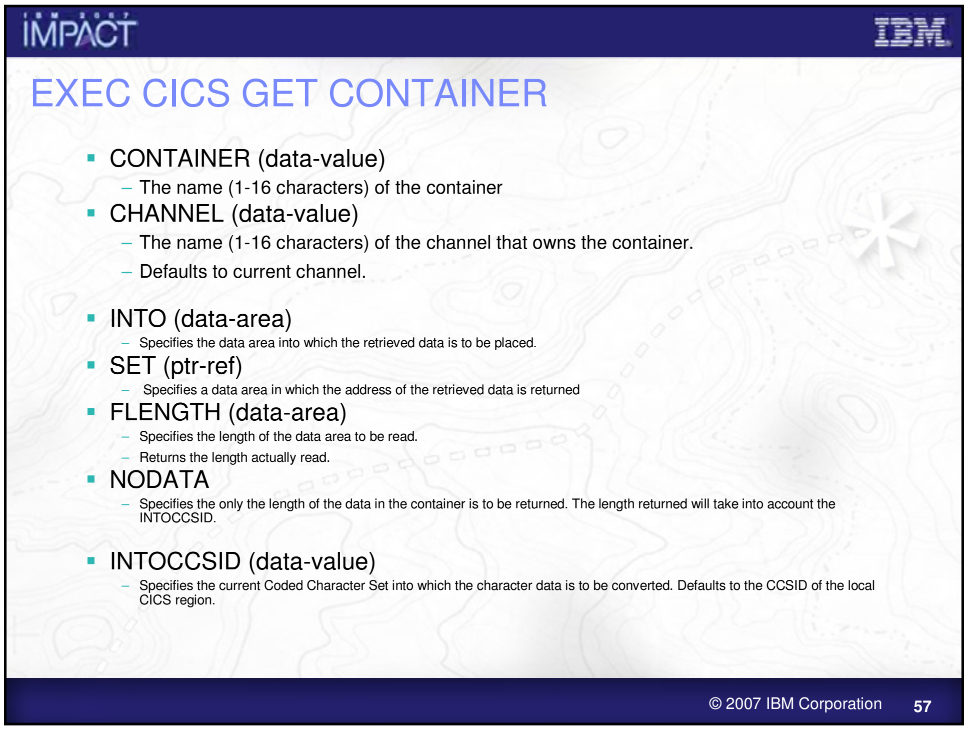
- CONTAINER (data-value)
  - The name (1-16 characters) of the container
- CHANNEL (data-value)
  - The name (1-16 characters) of the channel that owns the container.
  - Defaults to current channel.

- 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.

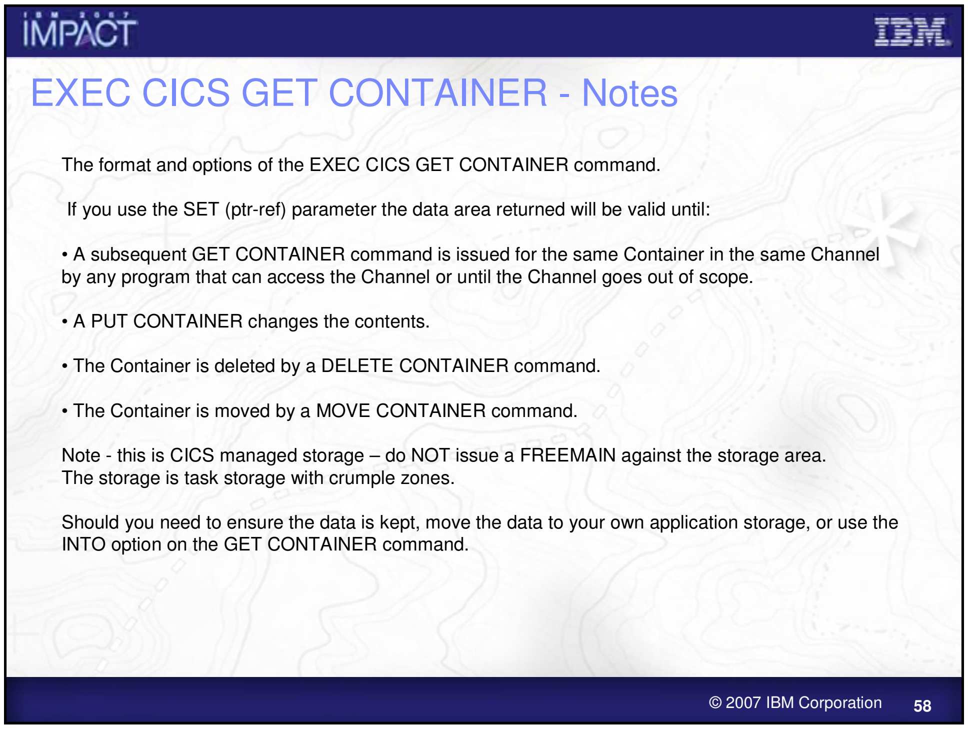
EXEC CICS PUT CONTAINER - Notes

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



EXEC CICS GET CONTAINER

- 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.
- SET (ptr-ref)
  - Specifies a data area in which the address of the retrieved data is returned
- 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.
- 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.



EXEC CICS GET CONTAINER - 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.

Note - 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, or use the INTO option on the GET CONTAINER command.

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)
```

Scenario – Simple Data Conversion - Notes

In this 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.

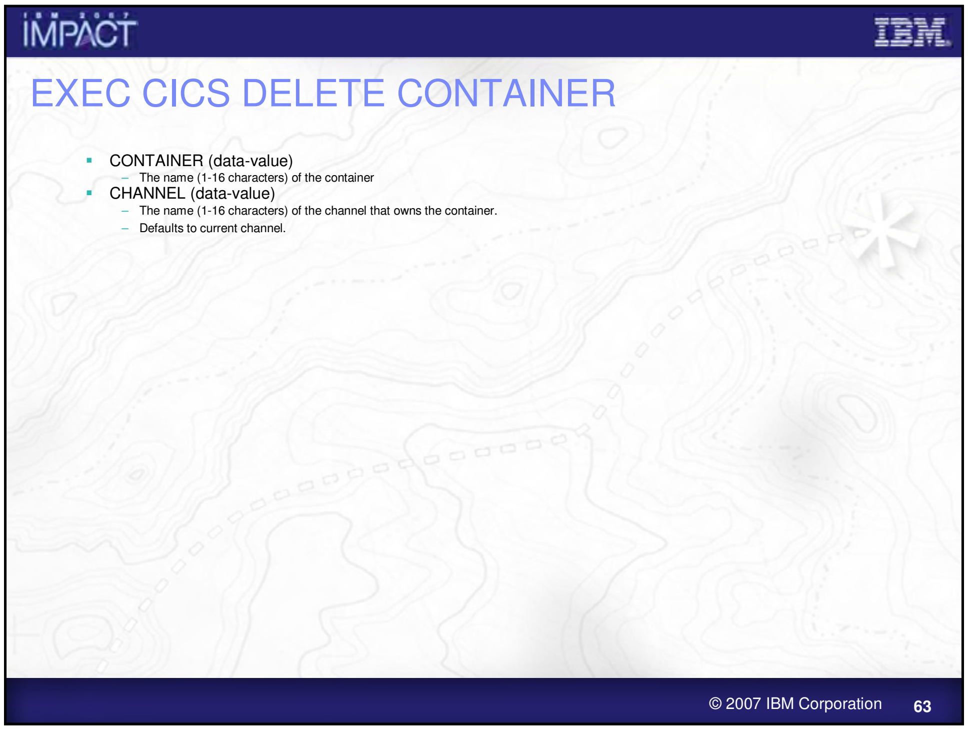
EXEC CICS MOVE CONTAINER

- 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

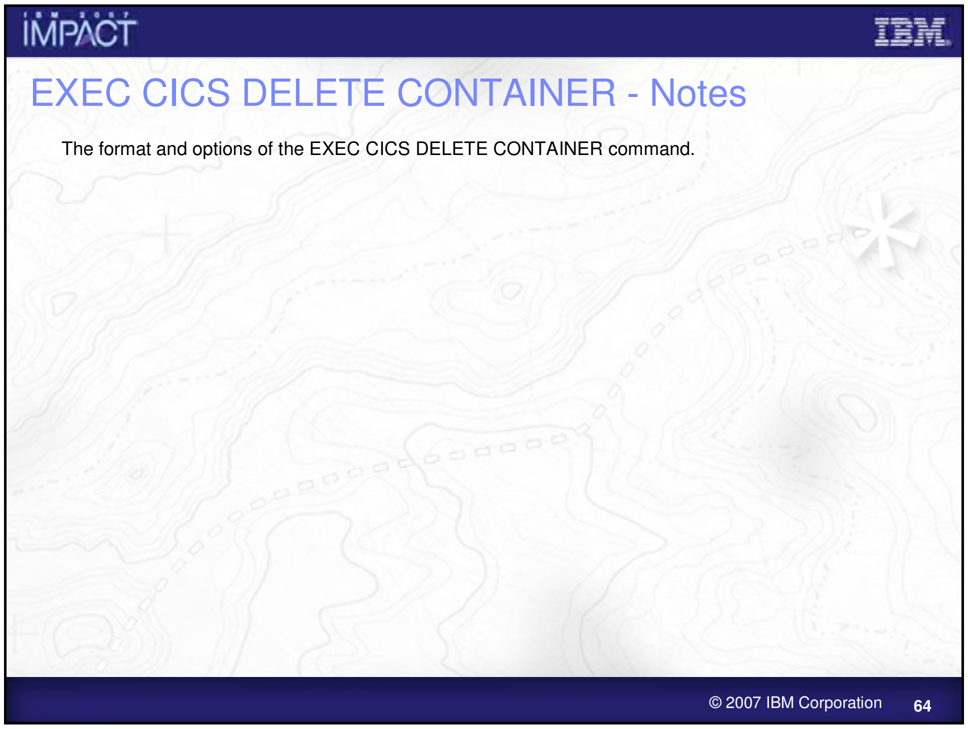
EXEC CICS MOVE CONTAINER - Notes

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



EXEC CICS DELETE CONTAINER

- 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.



EXEC CICS DELETE CONTAINER - Notes

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

Program Transfer Commands

- LINK PROGRAM [CHANNEL|COMMAREA]
  - Links to another 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

Program Transfer Commands - Notes

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

You may pass a Channel or a 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

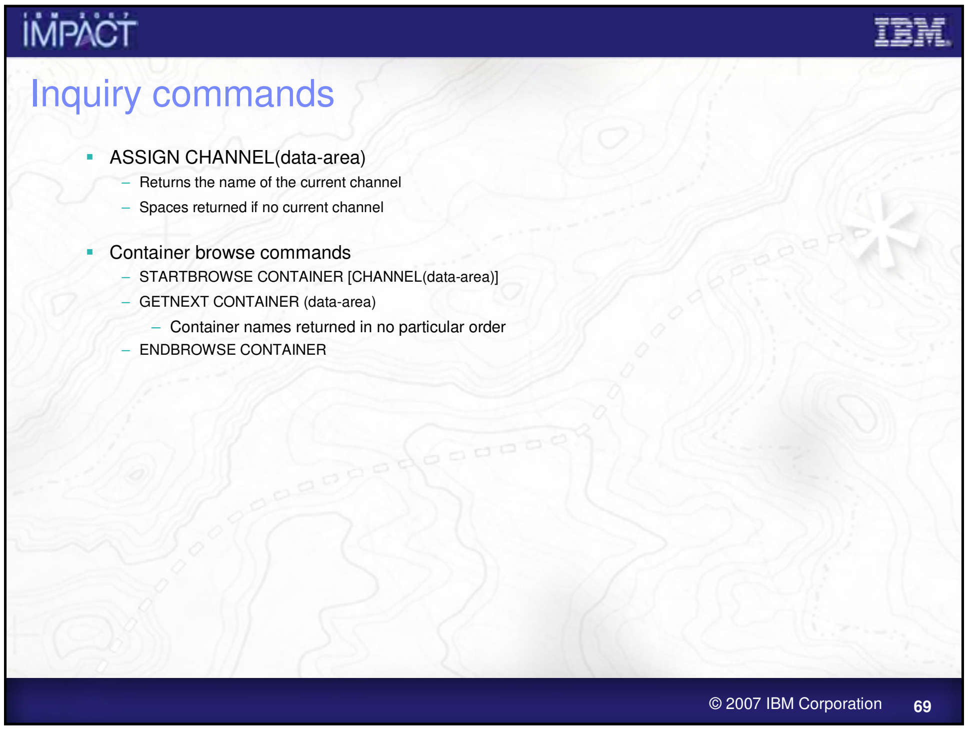
Transaction Transfer Commands - Notes

To begin or continue a pseudo-conversational transaction you will use an EXEC CICS RETURN TRANSID CHANNEL command. As with using a commarea, this command is only valid at the highest logical level, that is, a program that is returning control to CICS. You may pass a Channel or COMMAREA to the next transaction, but not both.

To start a new transaction and pass a Channel to the new task you use an EXEC CICS START TRANSID CHANNEL command. In the case of the START command the Channel and its Containers are copied from the original and passed to the started transaction. At this point there are two separate copies of the Channel and Container 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 STAT data, but not both.

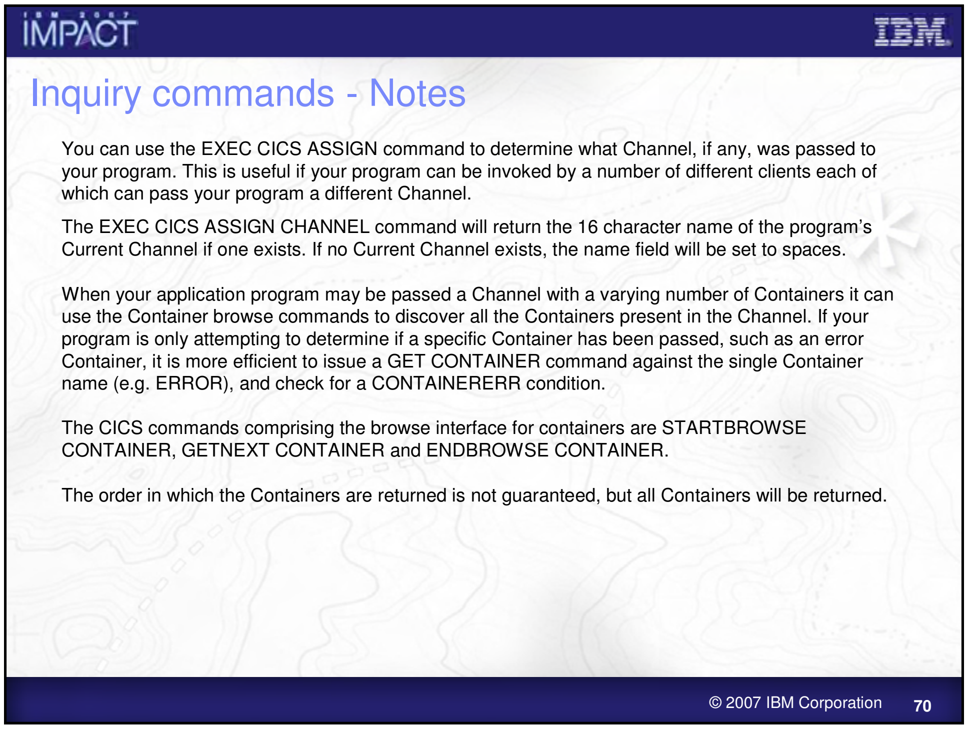
Transactions started with the CHANNEL option will have a STARTCODE of 'S'.

Timer options are not supported by the START command when a Channel is specified.



Inquiry commands

- ASSIGN CHANNEL(data-area)
  - Returns the name of the current channel
  - Spaces returned if no current channel
- Container browse commands
  - STARTBROWSE CONTAINER [CHANNEL(data-area)]
  - GETNEXT CONTAINER (data-area)
    - Container names returned in no particular order
  - ENDBROWSE CONTAINER



Inquiry commands - Notes

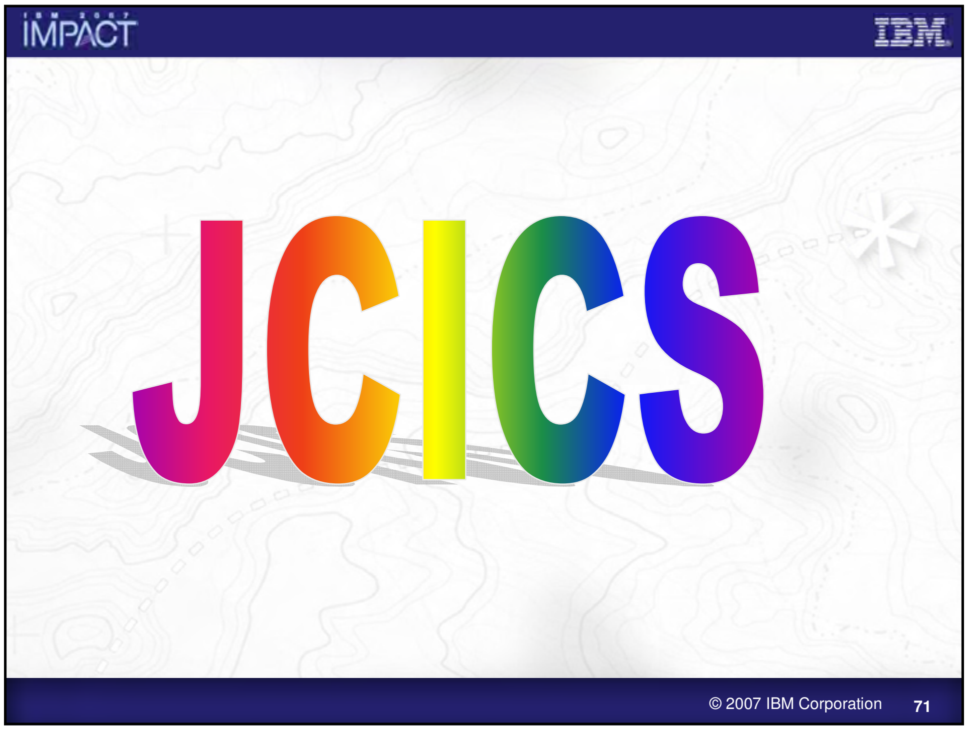
You can use the EXEC CICS ASSIGN command to determine what Channel, if any, was passed to your program. This is useful if your program can be invoked by a number of different clients each of which can pass your program a different Channel.

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, the name field will be set to spaces.

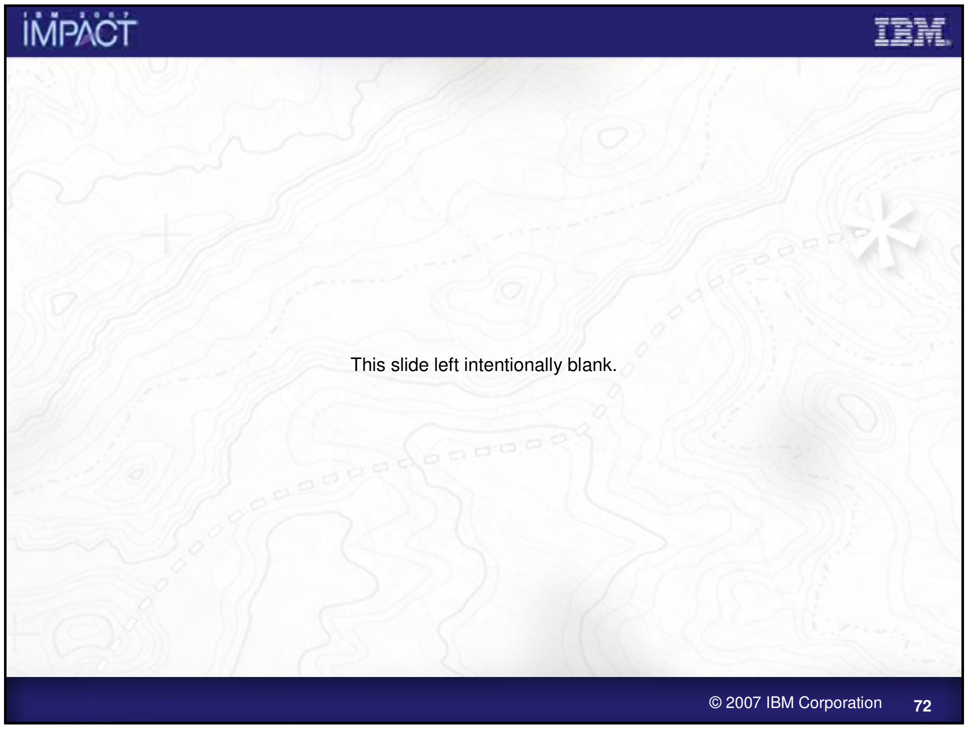
When your application program may be passed a Channel with a varying number of Containers it can use the Container browse commands to discover all the Containers present in the Channel. If your program is only attempting to determine if a specific Container has been passed, such as an error Container, it is more efficient to issue a GET CONTAINER command against the single Container name (e.g. ERROR), and check for a CONTAINERERR condition.

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

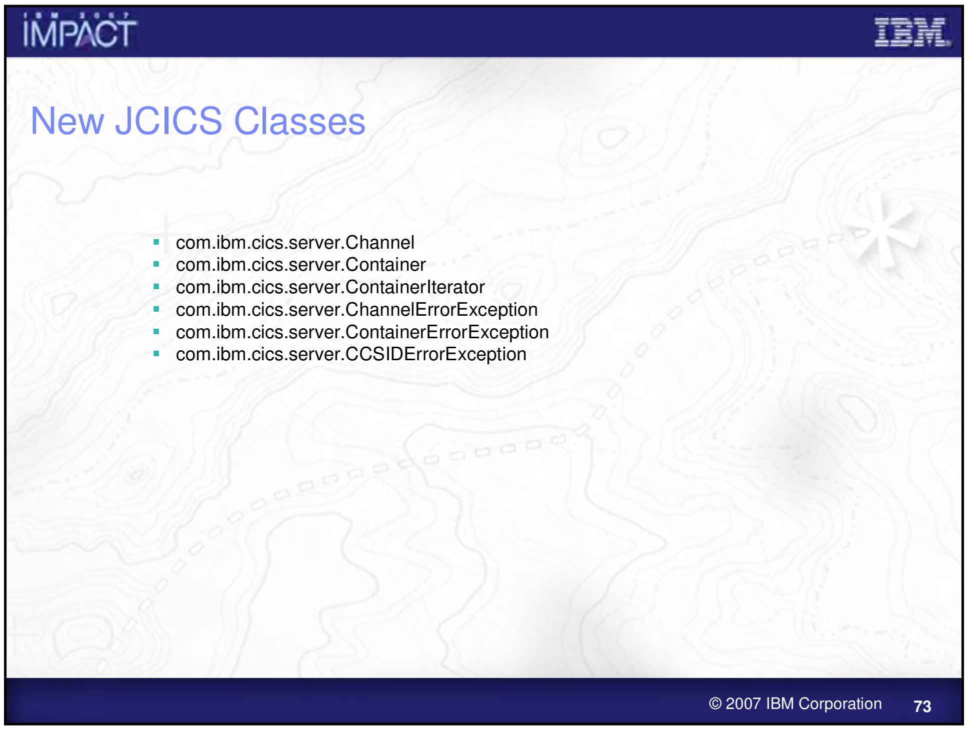
The order in which the Containers are returned is not guaranteed, but all Containers will be returned.



JCICS

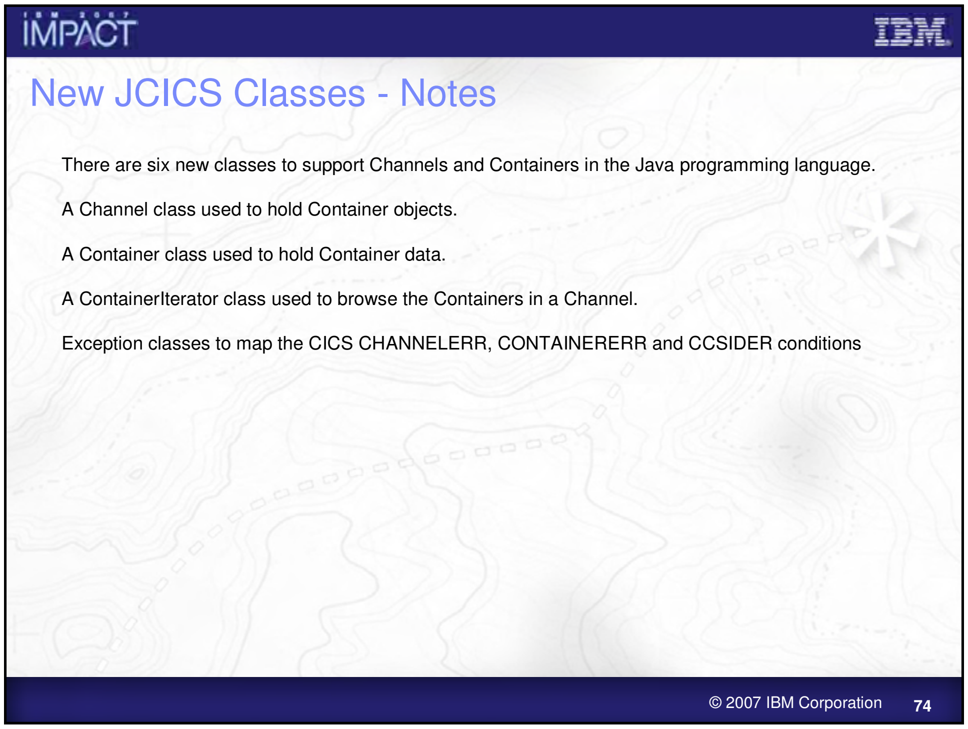


This slide left intentionally blank.



New JCICS Classes

- com.ibm.cics.server.Channel
- com.ibm.cics.server.Container
- com.ibm.cics.server.ContainerIterator
- com.ibm.cics.server.ChannelErrorException
- com.ibm.cics.server.ContainerErrorException
- com.ibm.cics.server.CCSIDErrorException



New JCICS Classes - Notes

There are six new classes to support Channels and Containers in the Java programming language.

A Channel class used to hold Container objects.

A Container class used to hold Container data.

A ContainerIterator class used to browse the Containers in a Channel.

Exception classes to map the CICS CHANNELERR, CONTAINERERR and CCSIDER conditions

IMPACT IBM

Creating a new Channel

- Use the createChannel() method of the Task class

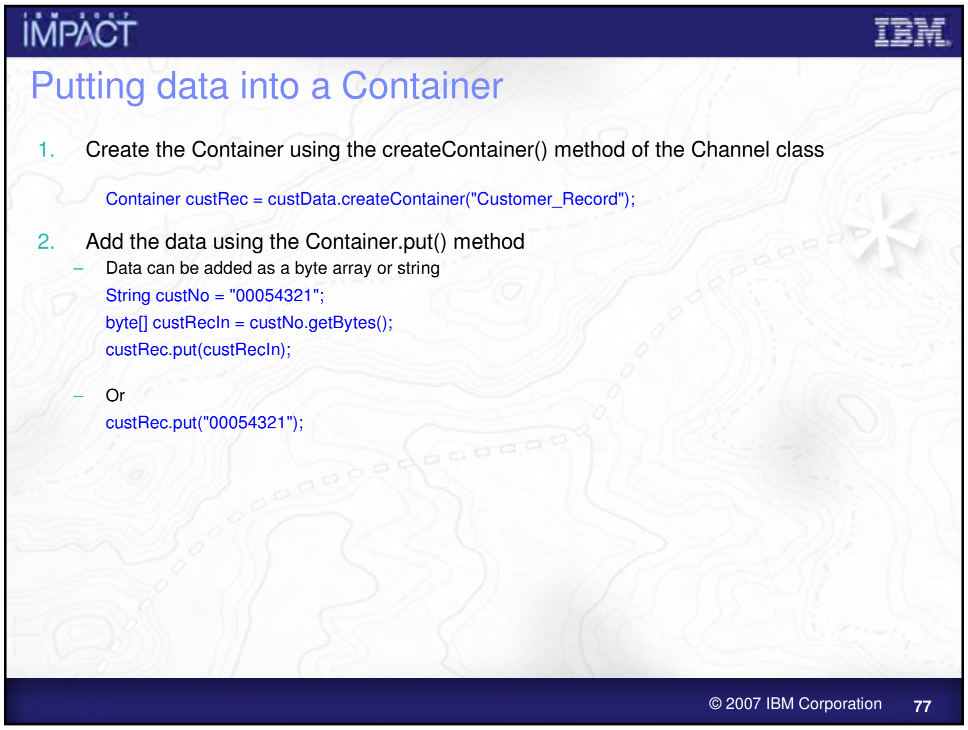
```
Task t = Task.getTask();
Channel custData = t.createChannel("Customer_Data");
```

IMPACT IBM

Creating a new Channel - Notes

Unlike the CICS API, in JCICS it is necessary to create a Channel object before creating any Containers to go into the Channel.

A Channel object is created by means of the createChannel() method of the Task class.



Putting data into a Container

1. Create the Container using the createContainer() method of the Channel class

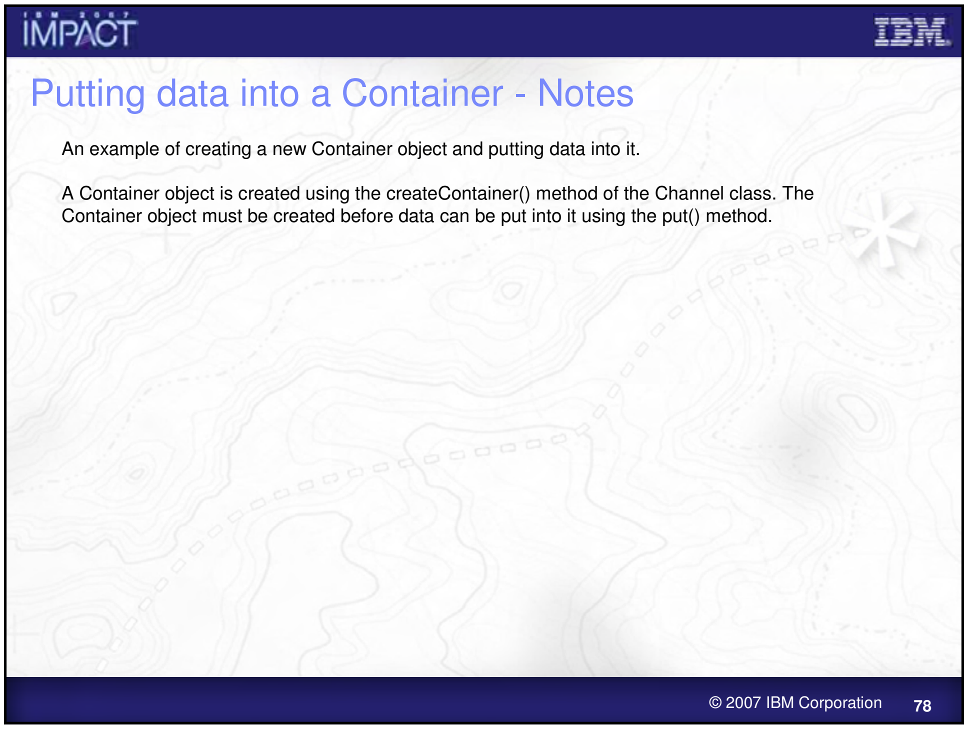
```
Container custRec = custData.createContainer("Customer_Record");
```

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");
```



Putting data into a Container - Notes

An example of creating a new Container object and putting data into it.

A Container object is created using the createContainer() method of the Channel class. The Container object must be created before data can be put into it using the put() method.

Passing a Channel to another program or task

- Passing a Channel to another program
  - Use the link() and xctl() methods of the Program class, passing the Channel object as the parameter

```
programX.link(custData);

programY.xctl(custData);
```

- Passing a Channel to another task
  - Use the issue() method of the StartRequest class, passing the Channel object as the parameter
    `StartRequest.issue(custData);`
  - Use the setNextChannel() method of the TerminalPrincipalFacility class
    `terminalPF.setNextChannel(custData);`

Passing a Channel to another program or task - Notes

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

The link() and xctl() methods of the Program class have been overloaded so that a Channel can be passed as the parameter instead of a CommAreaHolder.

Similarly with the issue() method of the StartRequest class.

For pseudo-conversational return, the setNextChannel() method of the TerminalPrincipalFacility class can be used.

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 object into a byte array:

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

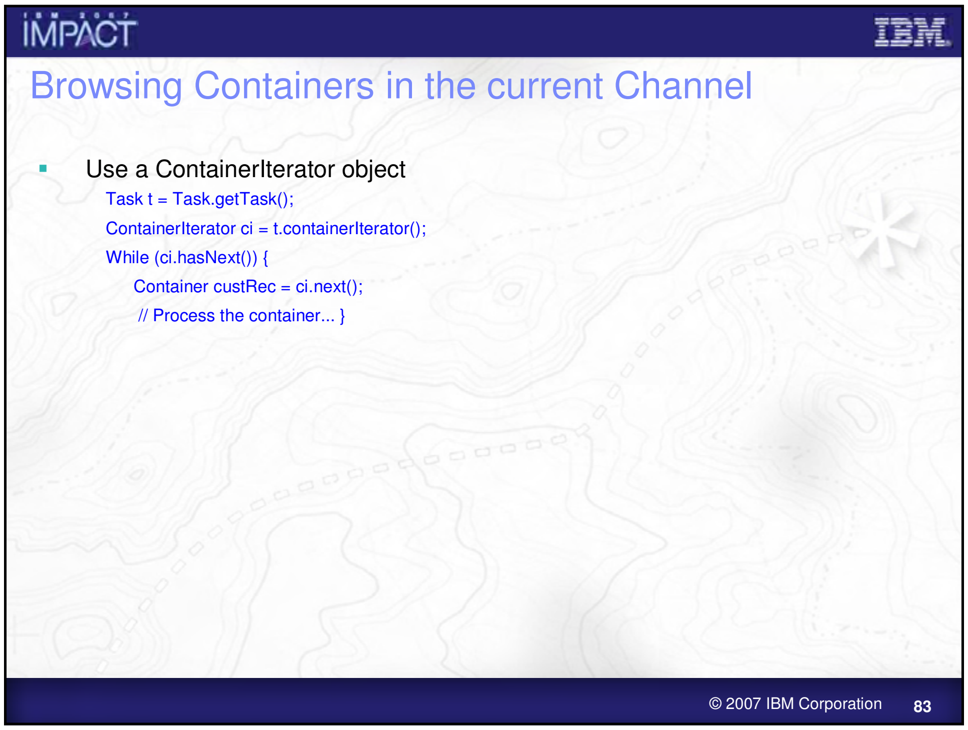
Getting data from the current Channel - Notes

An example of getting the current Channel and retrieving data from a Container within the Channel.

The getCurrentChannel() method of the Task class returns a Channel object representing the current Channel if it exists, otherwise it will return null – which should normally be tested for.

To get a Container object from a Channel, use the Channel's getContainer() method, specifying the Container name.

Finally, to get the data from the Container, use the Container class's get() method to read the data into a byte array.

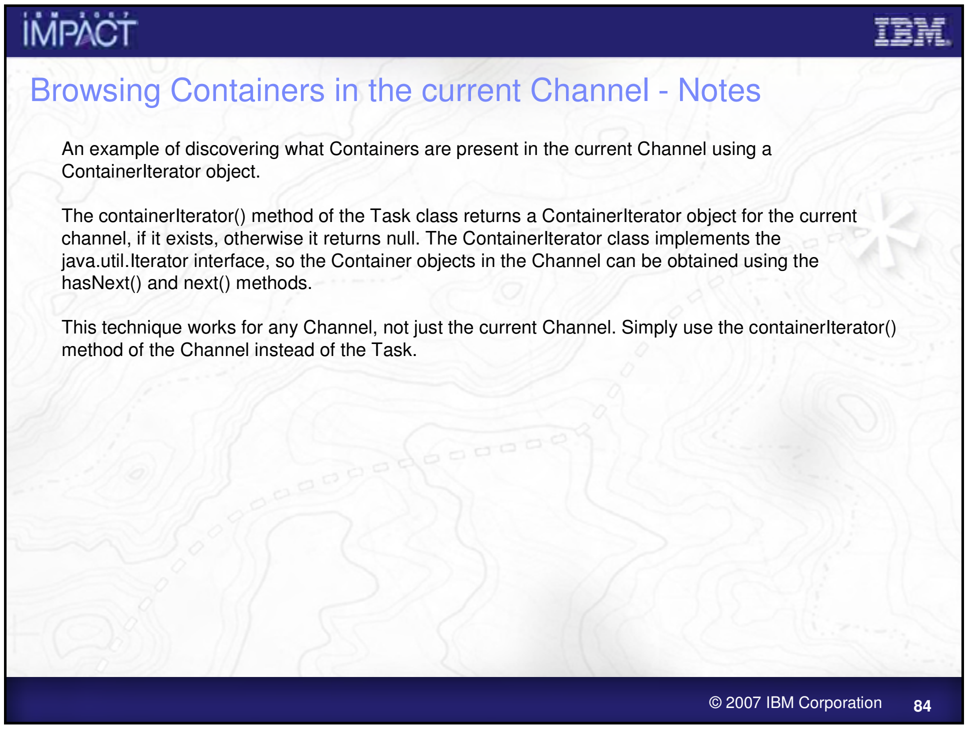


IMPACT IBM

Browsing Containers in the current Channel

- Use a ContainerIterator object

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



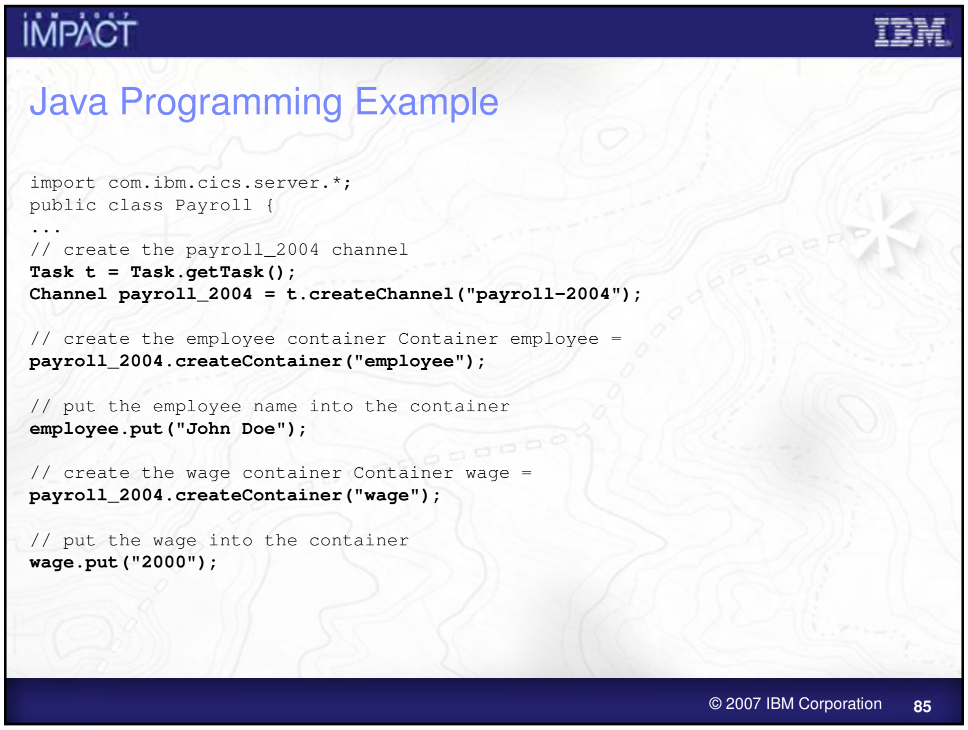
IMPACT IBM

Browsing Containers in the current Channel - Notes

An example of discovering what Containers are present in the current Channel using a ContainerIterator object.

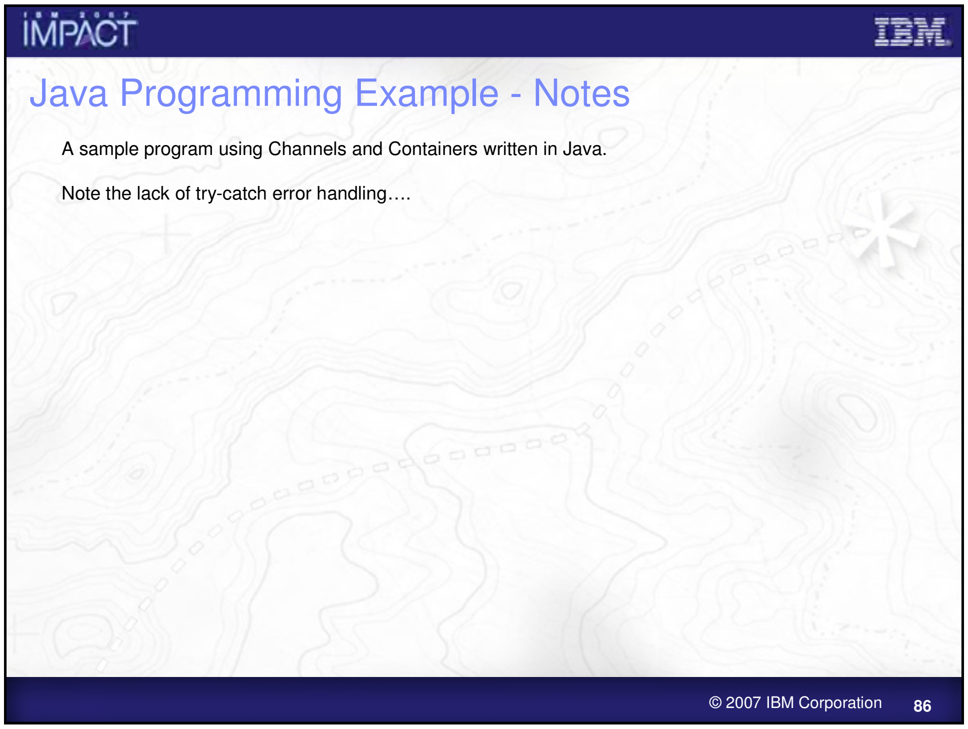
The containerIterator() method of the Task class returns a ContainerIterator object for the current channel, if it exists, otherwise it returns null. The ContainerIterator class implements the java.util.Iterator interface, so the Container objects in the Channel can be obtained using the hasNext() and next() methods.

This technique works for any Channel, not just the current Channel. Simply use the containerIterator() method of the Channel instead of the Task.



Java Programming Example

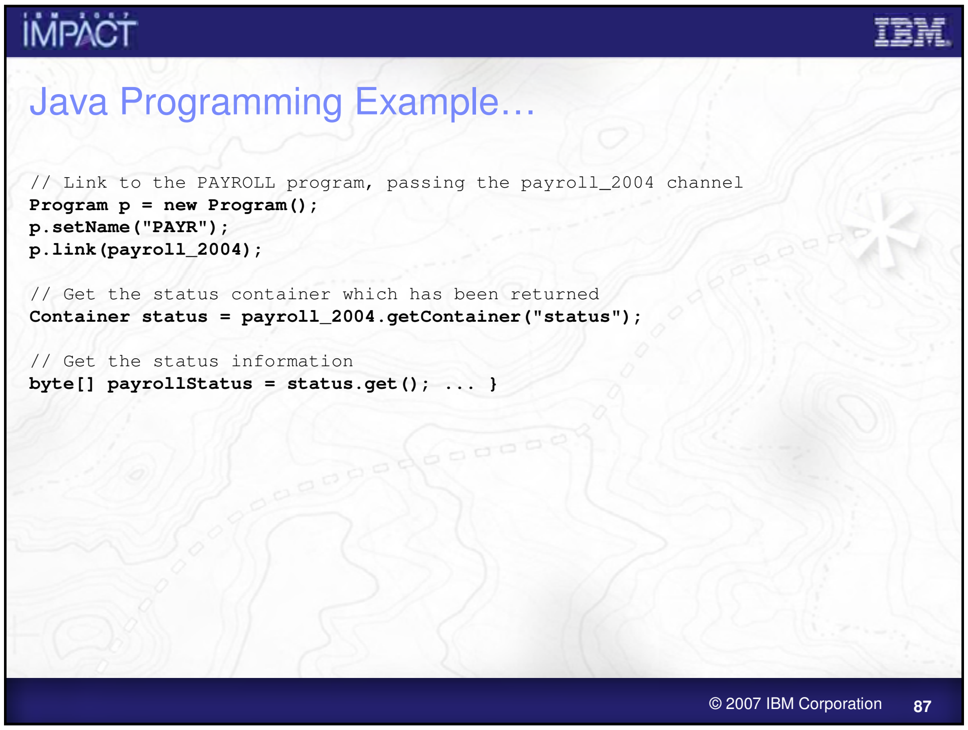
```
import com.ibm.cics.server.*;
public class Payroll {
...
// create the payroll_2004 channel
Task t = Task.getTask();
Channel payroll_2004 = t.createChannel("payroll-2004");

// 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");
```



Java Programming Example - Notes

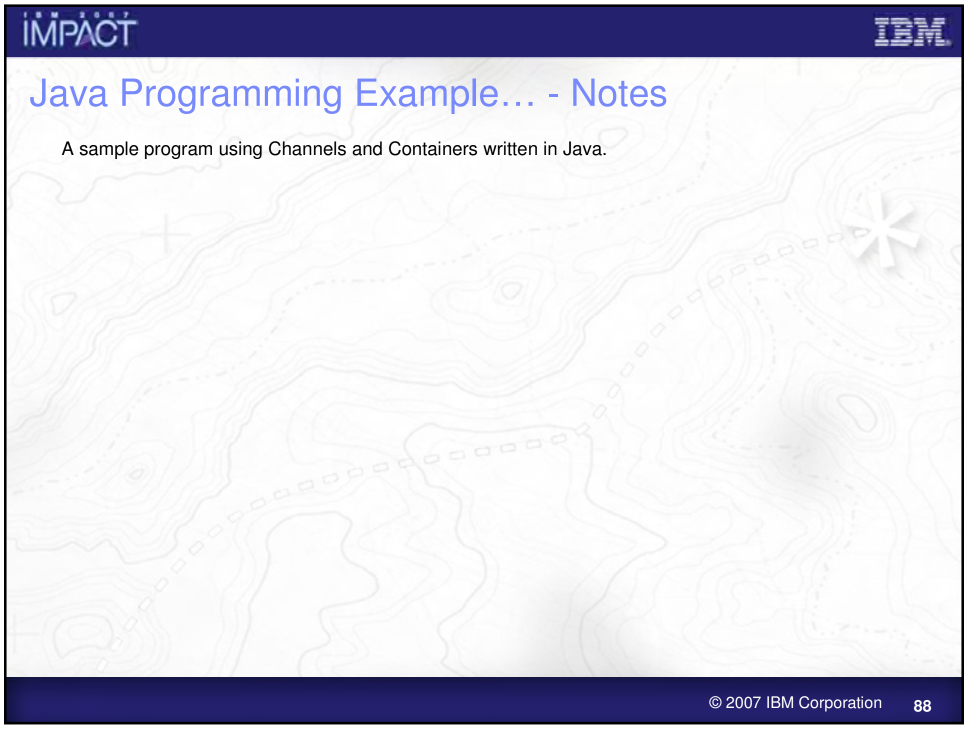
A sample program using Channels and Containers written in Java.

Note the lack of try-catch error handling….



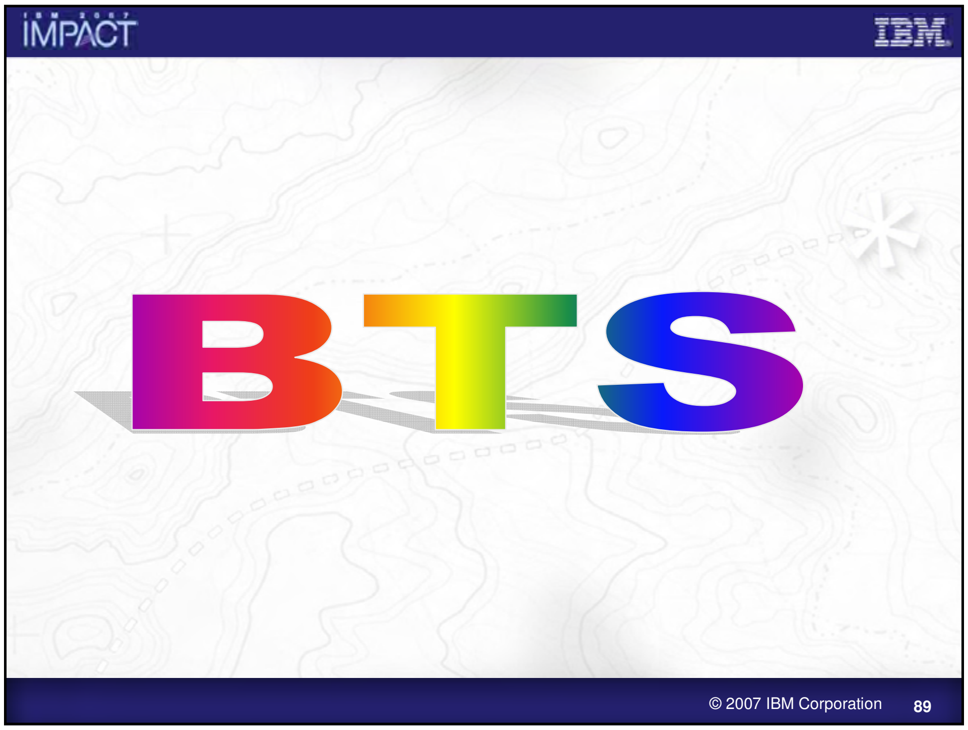
Java Programming Example…

```
// 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(); ... }
```

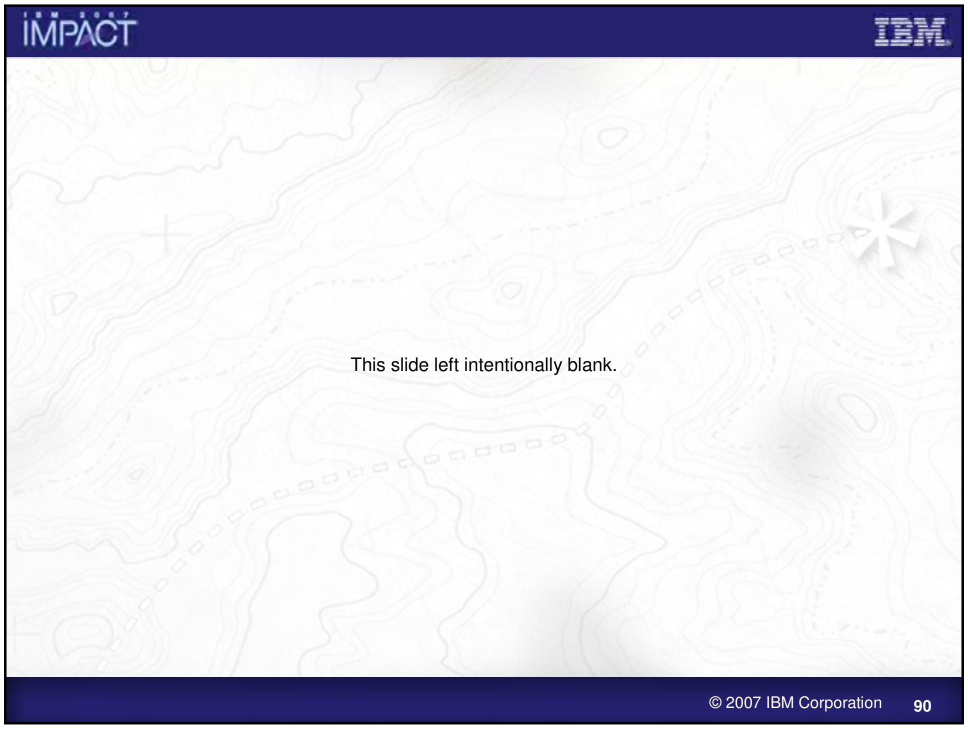


Java Programming Example… - Notes

A sample program using Channels and Containers written in Java.



BTS



This slide left intentionally blank.

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

Containers and Business Transaction Services - Notes

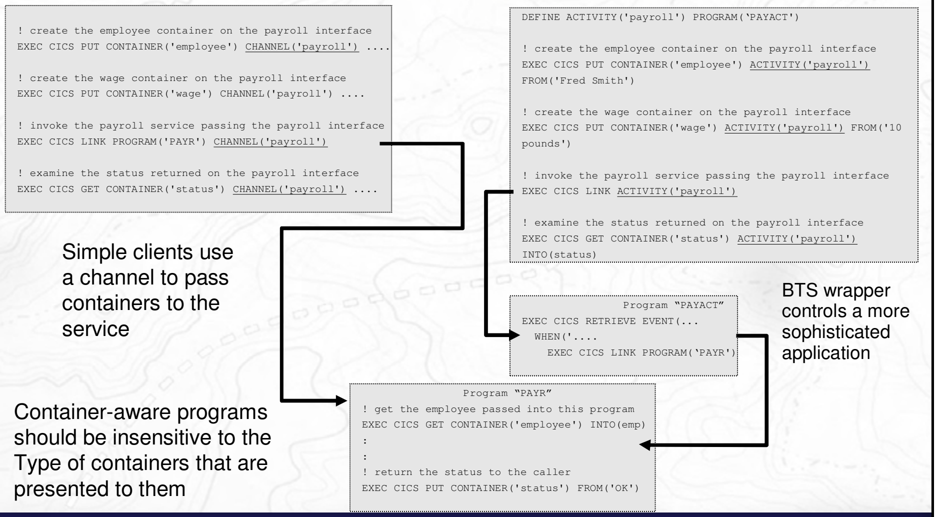
For those of you that have used CICS Business Transaction Services (BTS), available since CICS TS 1.3, you will be familiar with Containers. BTS implemented Containers as a way of passing information between Activities and Processes. There is no limit to the size of a Container in BTS. In fact, there 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). However, BTS Containers are hardened to VSAM, but Channel Containers are non-recoverable.

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 the 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)
```

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

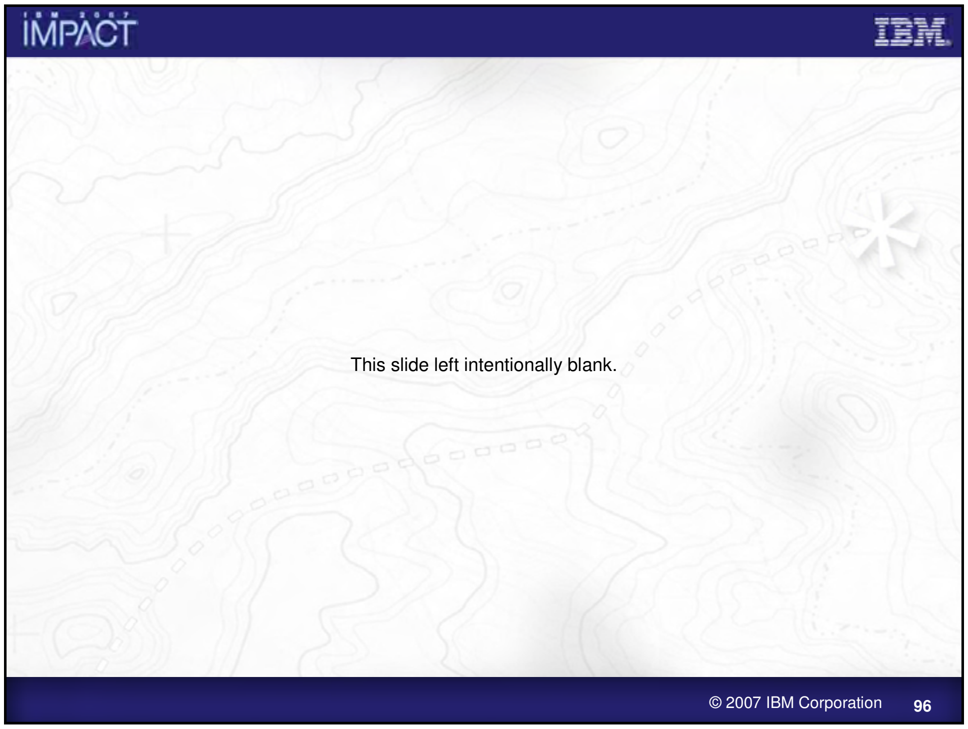
BTS wrapper controls a more sophisticated application

```
               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')
```

Containers and Business Transaction Services… - Notes

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

System Programming



This slide left intentionally blank.

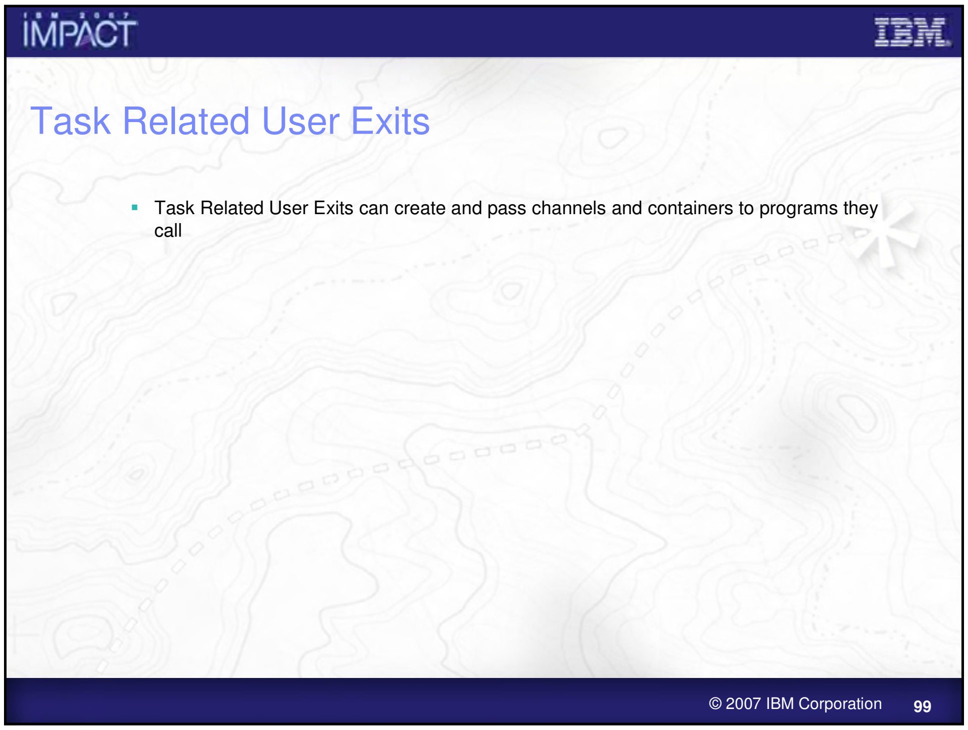
Global User Exits

- 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
    - XICEREQ, XICEREQC
    - XPCREQ, XPCEREQC
  - Channel name added to the PCUE parameter list
    - XPCFTCH
  - Exits cannot access contents of channels

Global User Exits - Notes

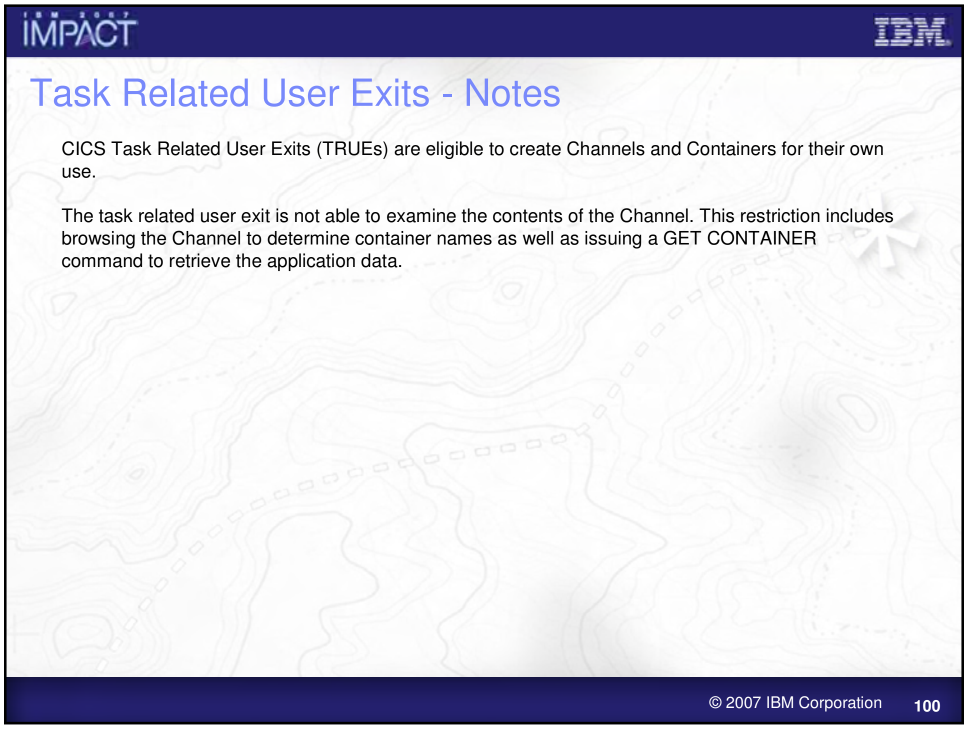
CICS Global User Exits (GLUEs) are eligible to create Channels and Containers for their own use.

The parameter list passed to a number of GLUEs has changed slightly with the addition of existence bits to signify the presence of an application’s Channel name. 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 commands to retrieve the application data.



Task Related User Exits

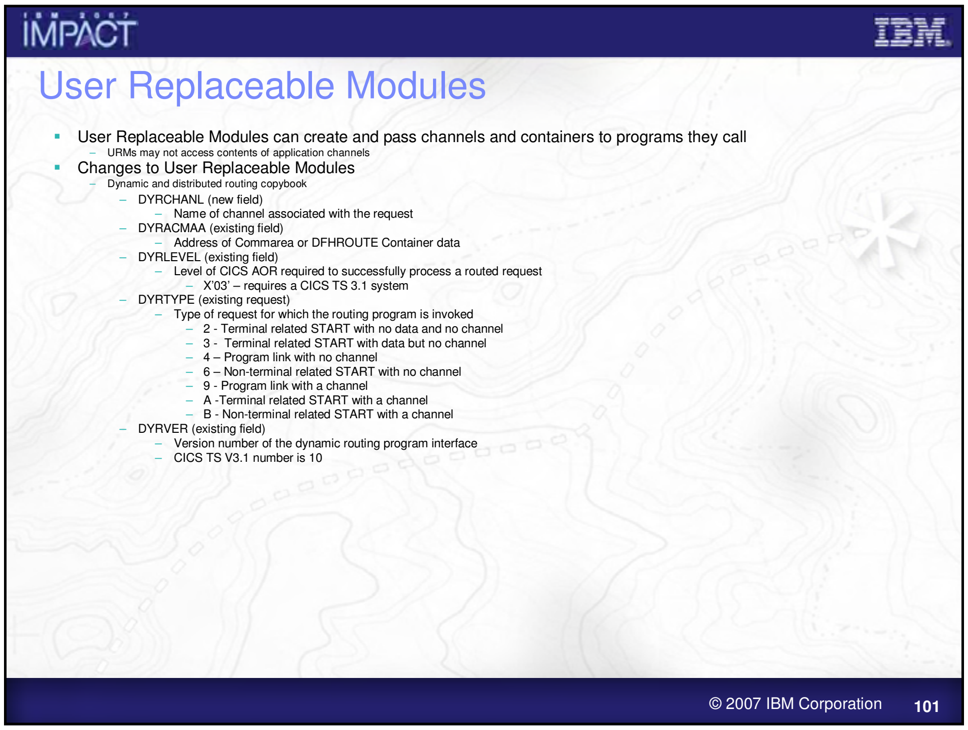
- Task Related User Exits can create and pass channels and containers to programs they call



Task Related User Exits - Notes

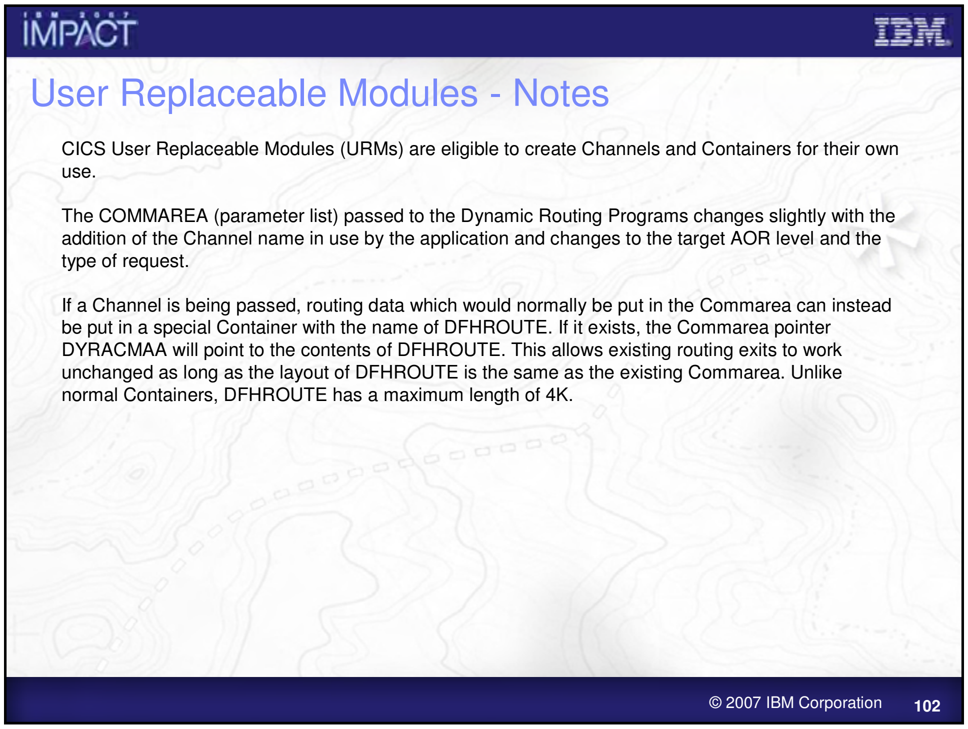
CICS Task Related User Exits (TRUEs) are eligible to create Channels and Containers for their own use.

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
    - DYRACMAA (existing field)
      - Address of Commarea or DFHROUTE Container data
    - 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



User Replaceable Modules - 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 use by the application and changes to the target AOR level and the type of request.

If a Channel is being passed, routing data which would normally be put in the Commarea can instead be put in a special Container with the name of DFHROUTE. If it exists, the Commarea pointer DYRACMAA will point to the contents of DFHROUTE. This allows existing routing exits to work unchanged as long as the layout of DFHROUTE is the same as the existing Commarea. Unlike normal Containers, DFHROUTE has a maximum length of 4K.

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
  - PGPUTCT
    - 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
  - PGPUTCDL
    - Total length, in bytes, of all the PUT CONTAINER data supplied

Monitoring - 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
    - Number of DPL requests issued with the chanel option for this task
  - PCRTNCCT
    - Number of RETURN requests issued with the channel option for this task
  - PCRTNCDL
    - Total length, in bytes, of the container data RETURNed

Monitoring… - 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, 'PCDLCSDL', 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, 'PCDLCRDL', 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, 'PCLNKCCT', 4 BYTES)
Number of program LINK requests issued with the CHANNEL option by the user task.

307 (TYPE-A, 'PCXCLCCT', 4 BYTES)
Number of program XCTL requests issued with the CHANNEL option by the user task.

308 (TYPE-A, 'PCDPLCCT', 4 BYTES)
Number of program distributed program link (DPL) requests issued with the CHANNEL option by the user task
.

309 (TYPE-A, 'PCRTNCCT', 4 BYTES)
Number of pseudoconversational RETURN requests issued with the CHANNEL option by the user task.

310 (TYPE-A, 'PCRTNCDL', 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
    - Total length of the data in the containers of all the remotely executed START CHANNEL requests.

Monitoring… - 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.

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.

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

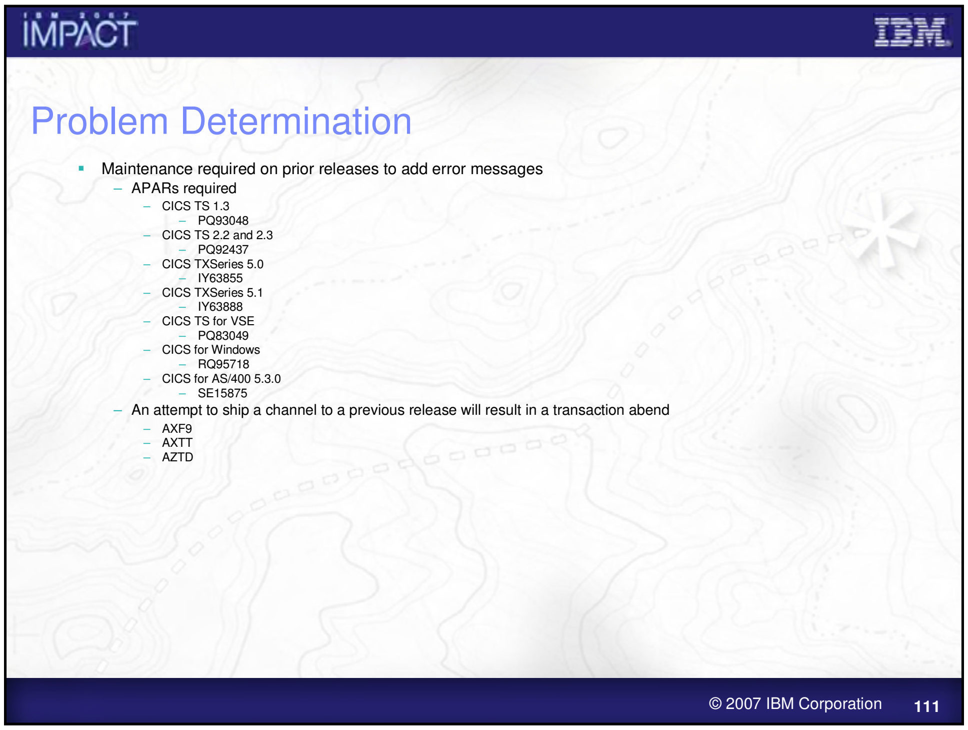
Statistics - Notes

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.

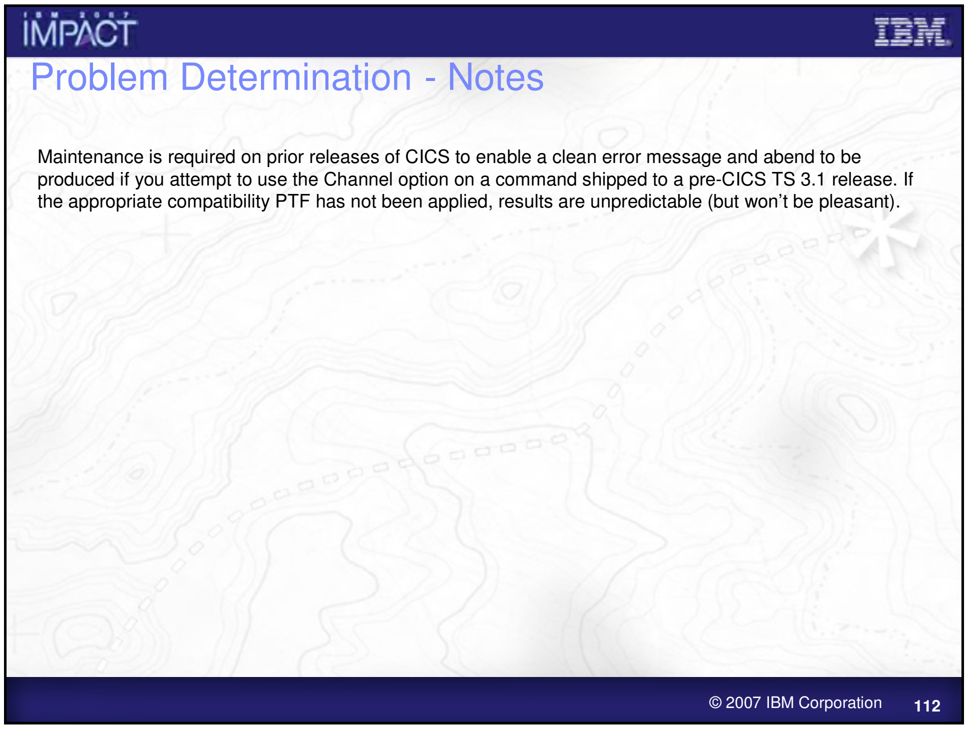
A14EST_CHANNEL_SENT: is the number of bytes sent on function-shipped Channel requests. This is the sum of the data in all the Containers.

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.



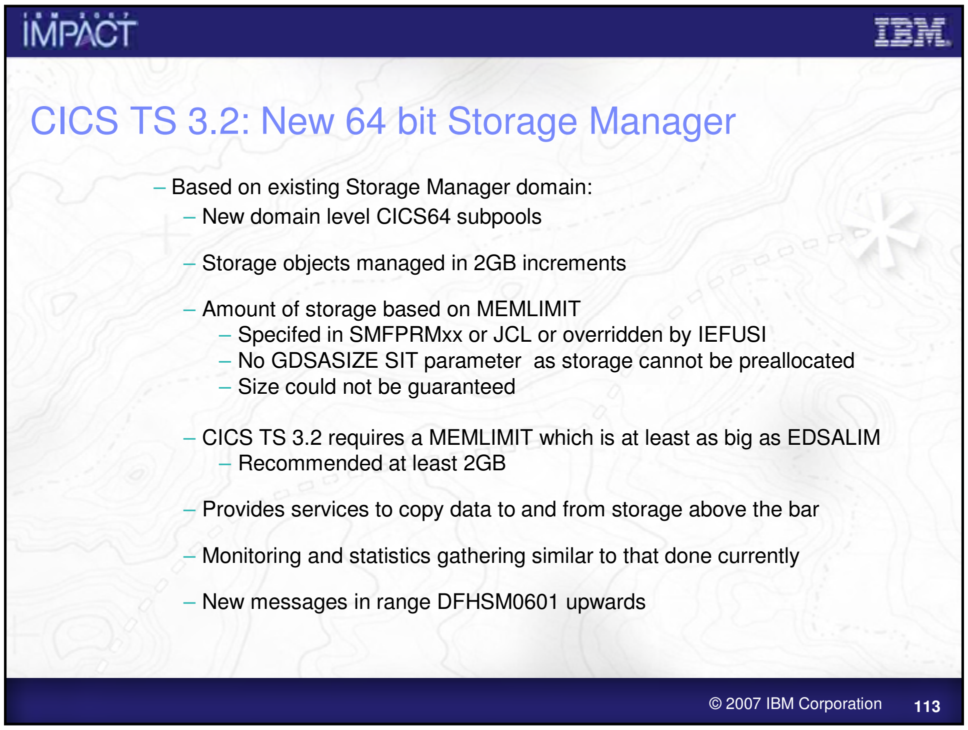
Problem Determination

- 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 5.0
      - IY63855
    - CICS TXSeries 5.1
      - IY63888
    - CICS TS for VSE
      - PQ83049
    - CICS for Windows
      - RQ95718
    - CICS for AS/400 5.3.0
      - SE15875
  - An attempt to ship a channel to a previous release will result in a transaction abend
    - AXF9
    - AXTT
    - AZTD



Problem Determination - Notes

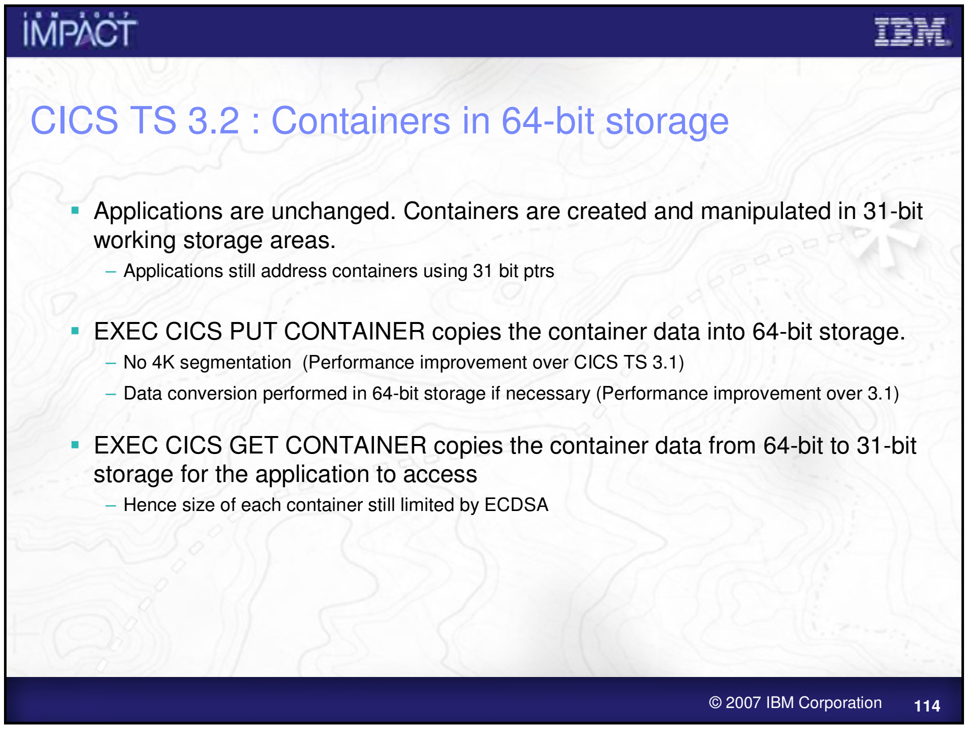
Maintenance is required on prior releases of CICS to enable a clean error message and abend to be produced if you attempt to use the Channel option on a command shipped to a pre-CICS TS 3.1 release. If the appropriate compatibility PTF has not been applied, results are unpredictable (but won't be pleasant).



CICS TS 3.2: New 64 bit Storage Manager

- Based on existing Storage Manager domain:
  - New domain level CICS64 subpools
  - Storage objects managed in 2GB increments
  - Amount of storage based on MEMLIMIT
    - Specifed in SMFPRMxx or JCL or overridden by IEFUSI
    - No GDSASIZE SIT parameter as storage cannot be preallocated
    - Size could not be guaranteed
  - CICS TS 3.2 requires a MEMLIMIT which is at least as big as EDSALIM
    - Recommended at least 2GB
  - Provides services to copy data to and from storage above the bar
  - Monitoring and statistics gathering similar to that done currently
  - New messages in range DFHSM0601 upwards

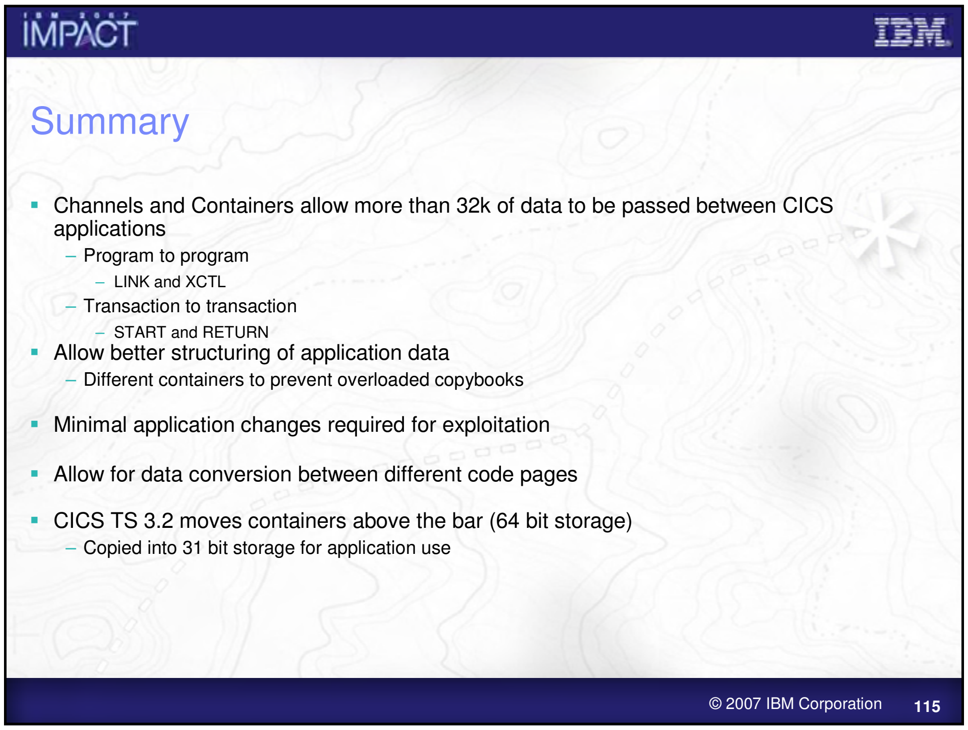
© 2007 IBM Corporation 113



CICS TS 3.2 : Containers in 64-bit storage

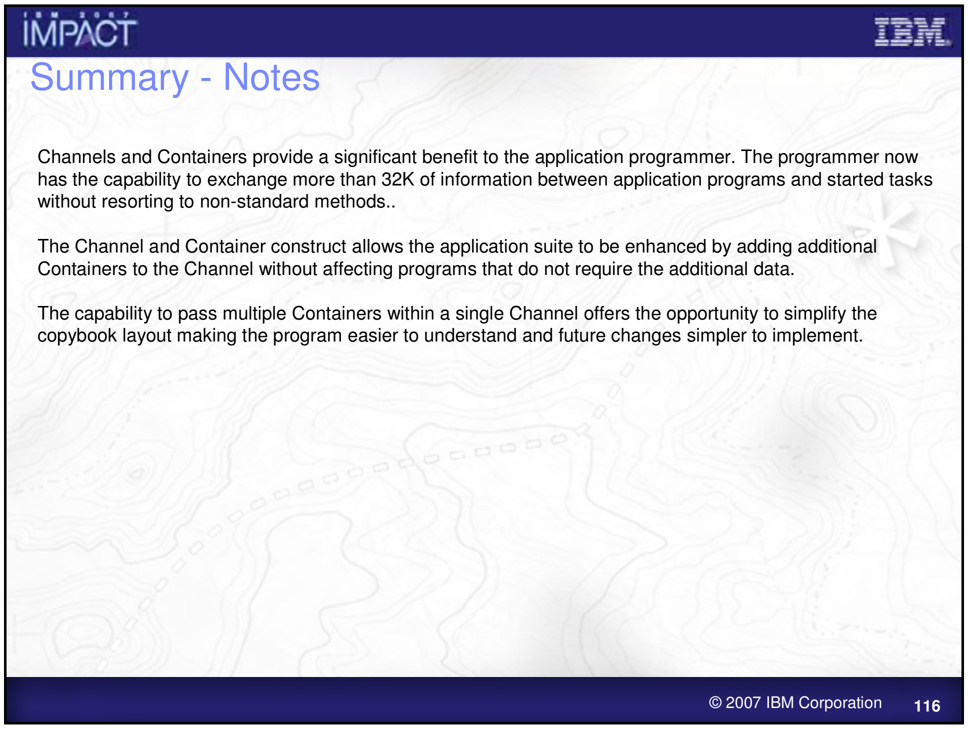
- Applications are unchanged. Containers are created and manipulated in 31-bit working storage areas.
  - Applications still address containers using 31 bit ptrs
- EXEC CICS PUT CONTAINER copies the container data into 64-bit storage.
  - No 4K segmentation (Performance improvement over CICS TS 3.1)
  - Data conversion performed in 64-bit storage if necessary (Performance improvement over 3.1)
- EXEC CICS GET CONTAINER copies the container data from 64-bit to 31-bit storage for the application to access
  - Hence size of each container still limited by ECDSA

© 2007 IBM Corporation 114



Summary

- Channels and Containers allow more than 32k of data to be passed between CICS applications
  - Program to program
    - LINK and XCTL
  - Transaction to transaction
    - START and RETURN
- Allow better structuring of application data
  - Different containers to prevent overloaded copybooks
- Minimal application changes required for exploitation
- Allow for data conversion between different code pages
- CICS TS 3.2 moves containers above the bar (64 bit storage)
  - Copied into 31 bit storage for application use



Summary - Notes

Channels and Containers provide a significant benefit to the application programmer. The programmer now has the capability to exchange more than 32K of information between application programs and started tasks without resorting to non-standard methods..

The Channel and Container construct allows the application suite to be enhanced by adding additional Containers to the Channel without affecting 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.

Questions
and
Answers