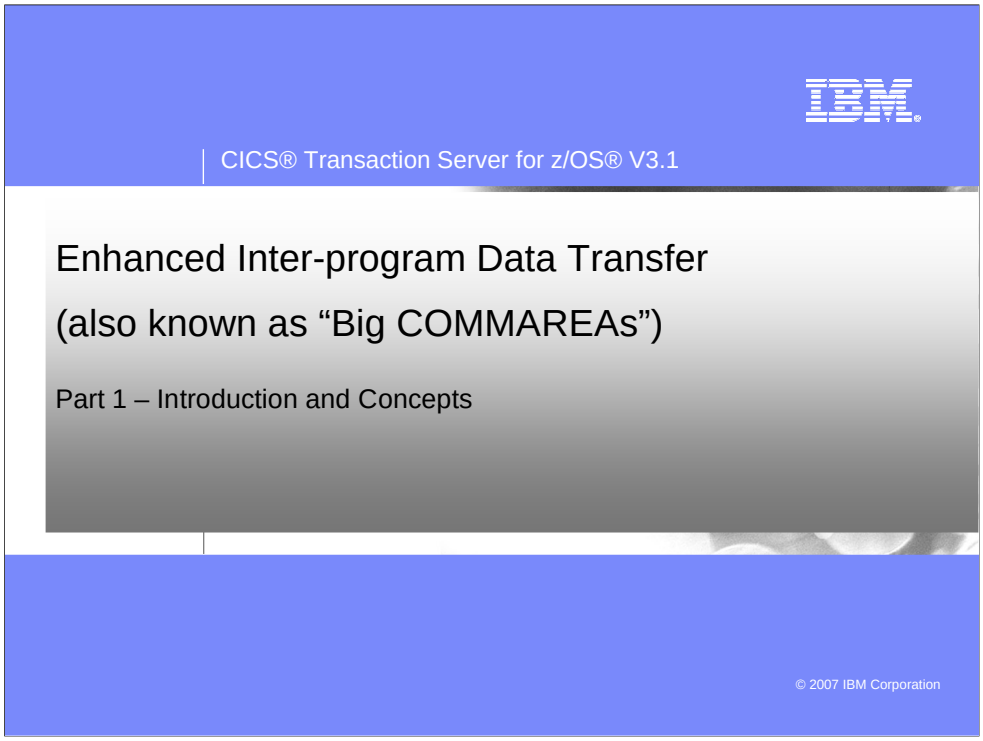
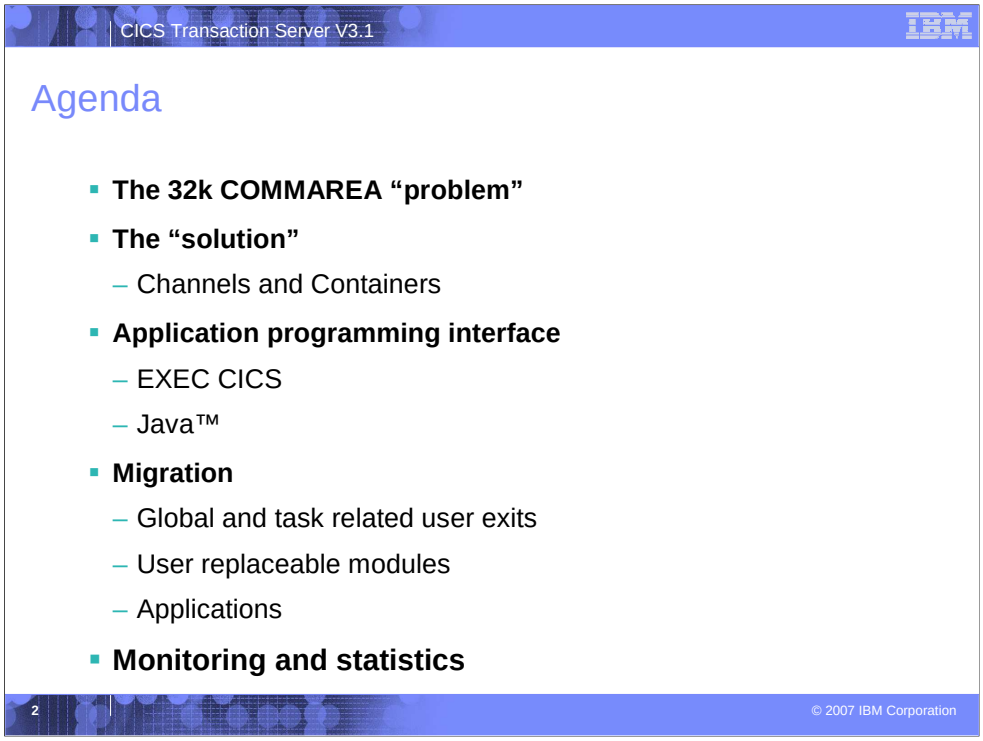


This presentation describes the capabilities of the Enhanced Inter-program Data Transfer function introduced in CICS Transaction Server 3.1. This function will allow programs and transactions to exchange more than 32K of data when using a LINK, XCTL, START or RETURN TRANSID command.

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



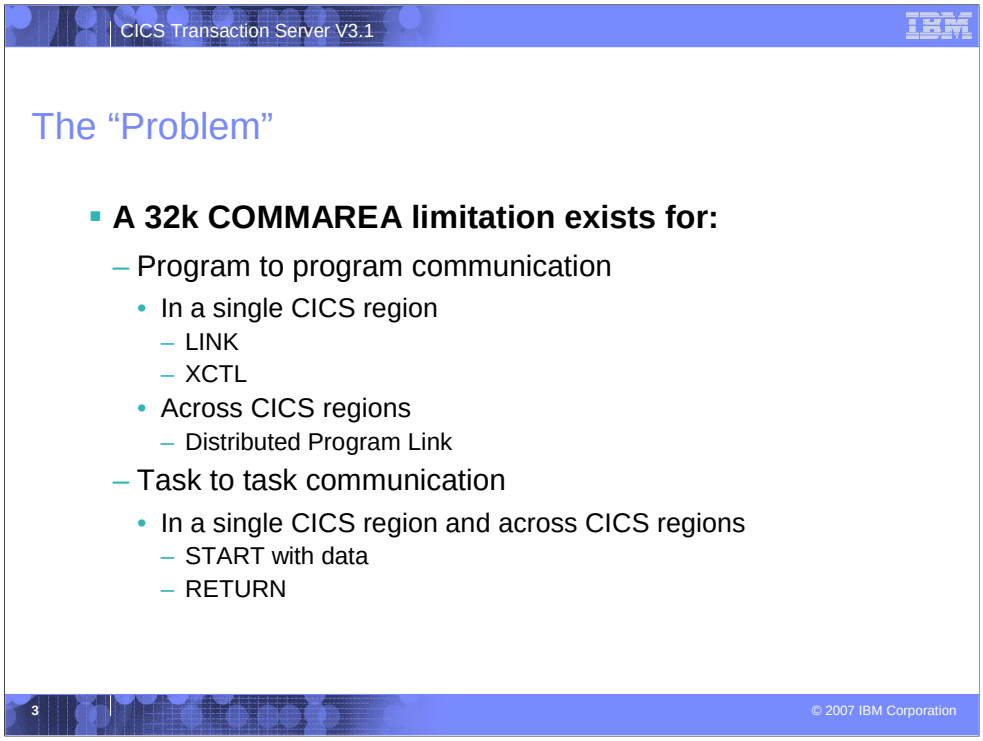
This presentation will discuss the problems that are encountered by programs encountering the 32K COMMAREA limitation, techniques that have been used to circumvent the 32K limitation and then will discuss the CICS solution to the problem.

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.

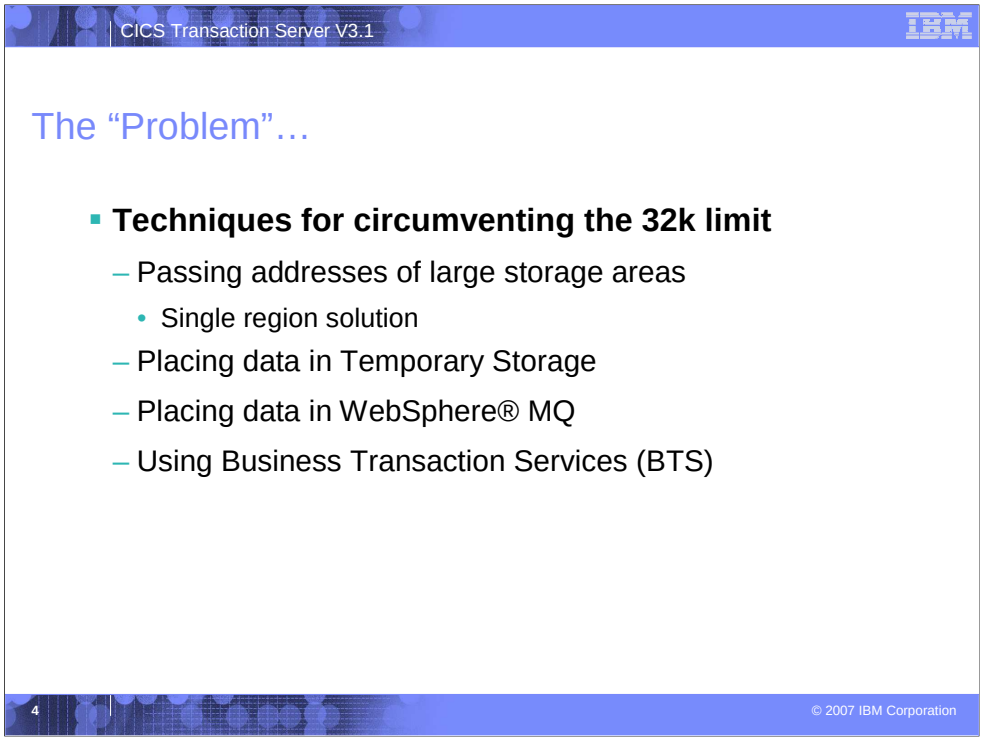


The 32K COMMAREA limitation has existed since command level program was introduced in 1975. The COMMAREA size restriction is applicable to both LINK and XCTL commands in a single region as while as applying to COMMAREAs used by programs participating in a Distributed Program Link (DPL) between two CICS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



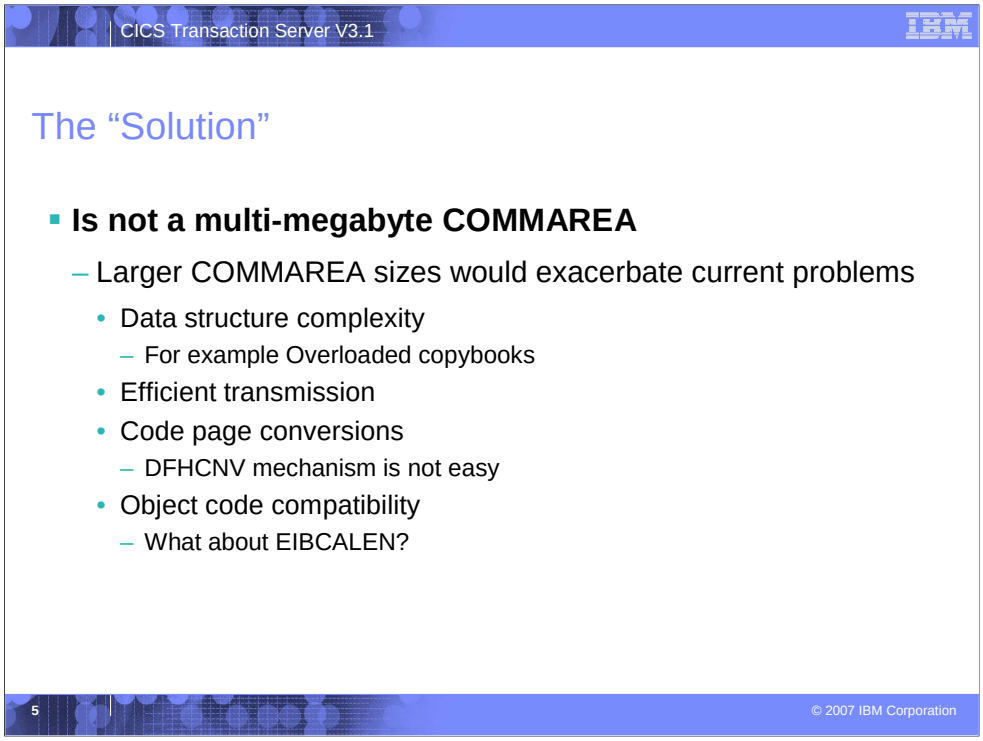
CICS programmers have developed a number of techniques for circumventing the 32K COMMAREA restriction both within a single CICS region and between multiple CICS regions.

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.



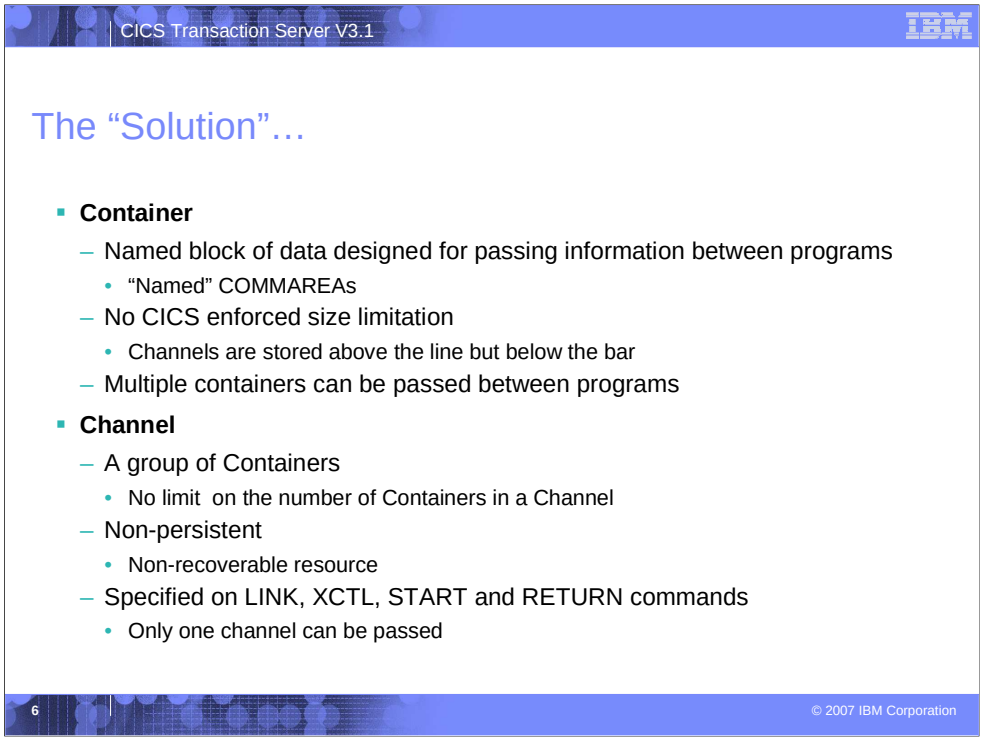
At first glance, it would seem that a reasonable solution to this problem would be to make the COMMAREA length greater than 32K and all problems would be resolved. While such an implementation would ease the 32K restriction it would not resolve all the “problems” that exist in exchanging data today and 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. The channel construct offers an easy way to get character data returned in the code page your application program requires.

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



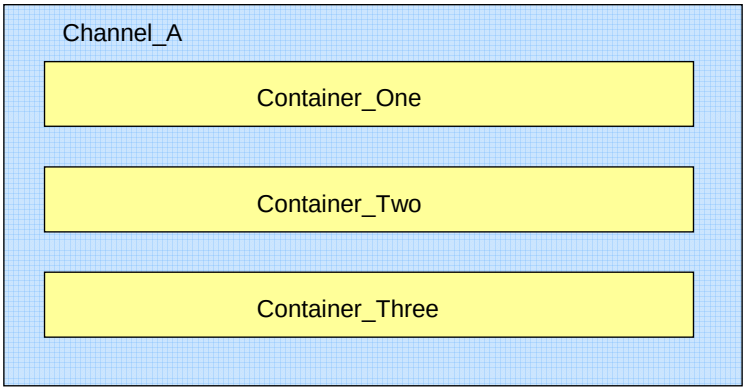
The solution to the 32K COMMAREA problem is to implement new constructs which solve the previously mentioned problems. CICS Transaction Server 3.1 has done this by implementing Channels and Containers. Channels and COMMAREAs are mutually exclusive. That is, you may use one technique or the other for 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, in terms of “best practices”, reasons will be provided for limiting the use of a single container and using multiple containers.

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.

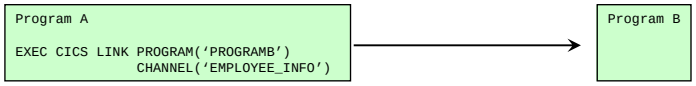
The “Solution”…



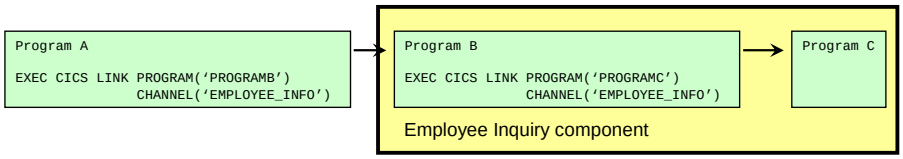
Here is a visual representation of a channel consisting of three containers. Both the channel and container are accessed by use of their name. In this example, the channel is named “Channel_A” and the containers are named “Container_One”, “Container_Two” and “Container_Three”.

Typical Scenarios for using Channels

- **One program / One channel**



- **One channel / Multiple programs**
  - The Channel is the interface to a Component



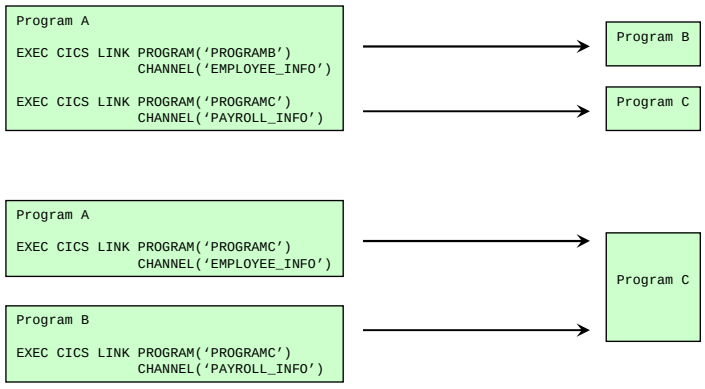
Here are several examples on how you might employ a channel in your application program flow.

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 of different containers, for their exchange of data. In this example there is only one copy of the data maintained.

Typical Scenarios for using Channels…

- **One program / Multiple channels**



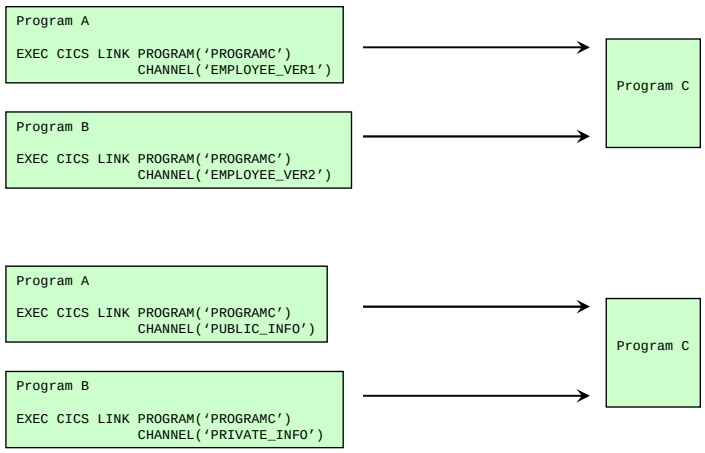
Here are further examples on how you might employ a channel in your application program flow.

Program A links to two programs, B and C. One option is to use a separate channel to invoke the two different programs. The same container names could be used in two channels.

In the second case, program C is a server program that can process requests from a number of different clients. Depending on the request type a different channel is used to pass the collection of containers from the client to program C. Program C can use the EXEC CICS ASSIGN command to determine which channel is present when it is called.

Typical Scenarios for using Channels…

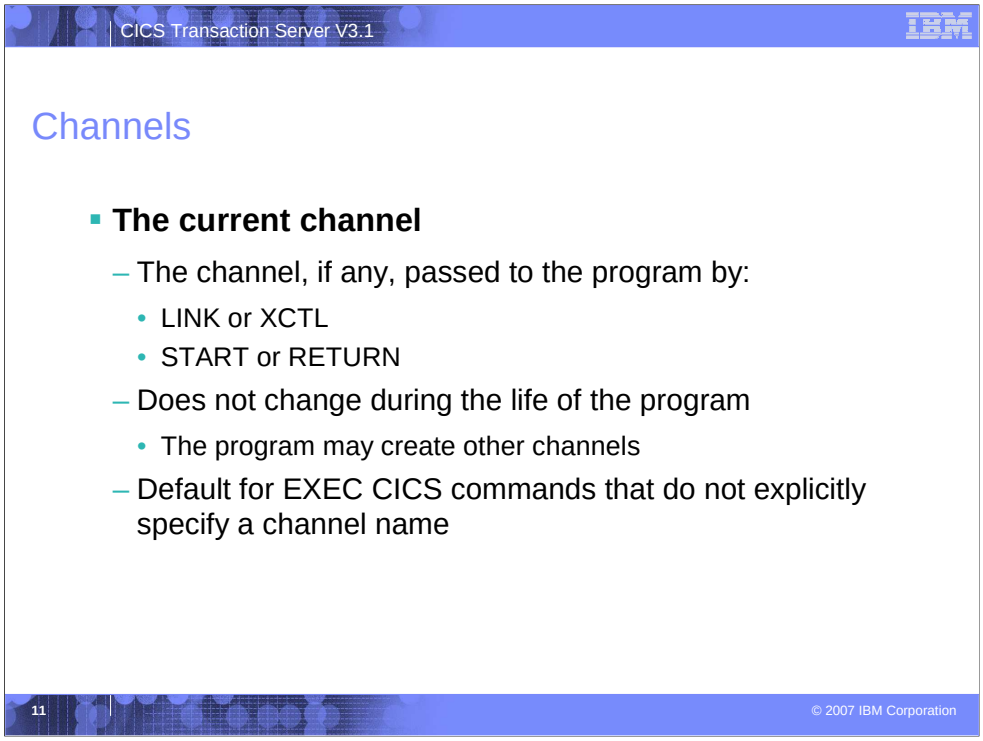
- **One program / Multiple channels**



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 processes. As time permits, the calling programs can be enhanced to use the new version on the container structure.

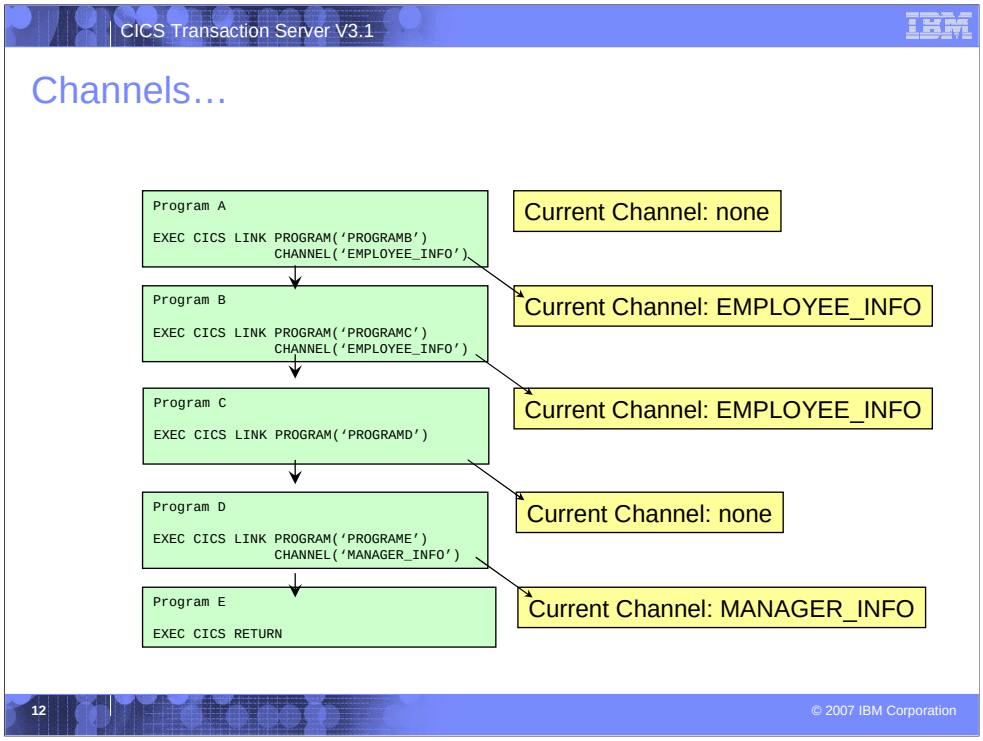
In the second case, program C normally will handle public requests from calling programs. A private data structure is used for special administrative programs. A different channel or container could be used to implement this “private protocol”.



A program's current channel is the channel, is 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 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.



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.

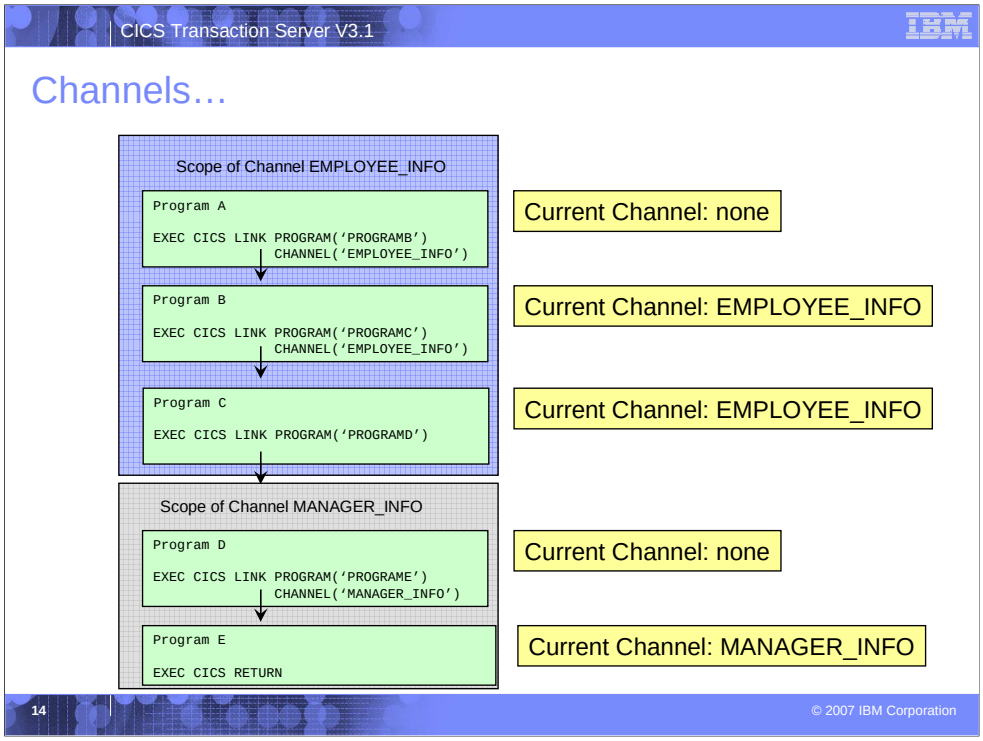
Channels…

- **Channel scope**
  - Application programs from which the channel can be accessed
    - Applications can only access the current channel
      - CICS may create read-only containers
    - Programs can create new channels
      - For use in program transfer to a different program
  - When no program in the link stack can access a channel it is deleted
    - Occurs at EXEC CICS RETURN or XCTL

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.

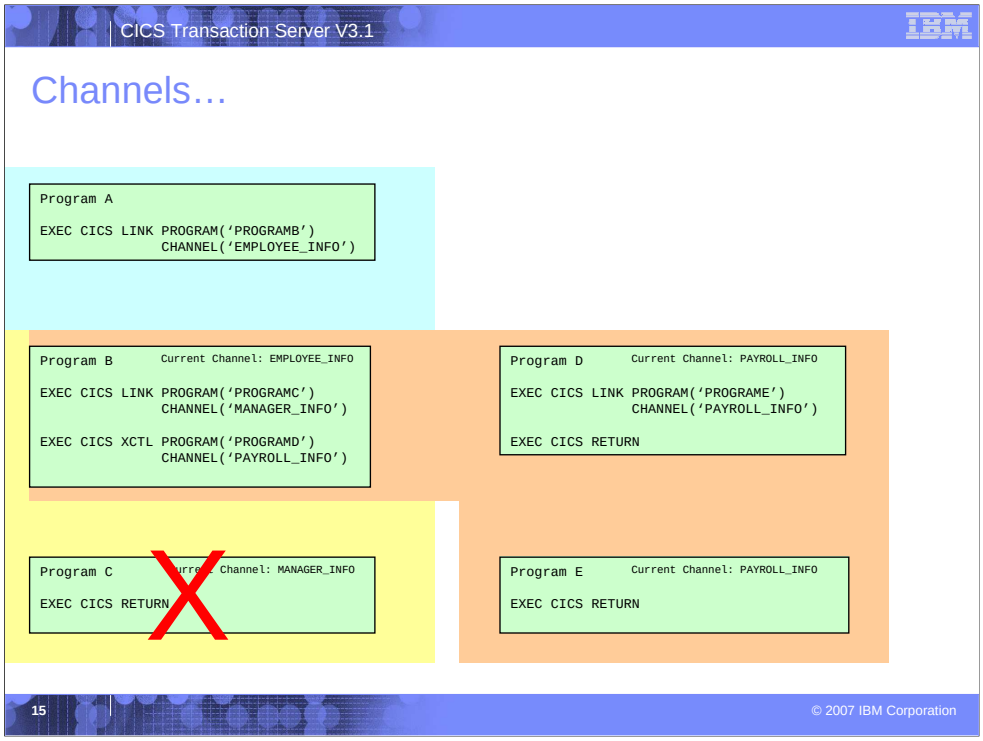
The other important thing to note about channel scope is that it controls when the channel will be deleted by CICS. When a channel goes out of scope, that is, no application program has the ability to access this channel, it is deleted. CICS will check to see if it can delete a channel at the time an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



This is an example of channel scope. This is the same example provided for determining 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 the channel (assuming it is not used in a pseudo conversational sequence).

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



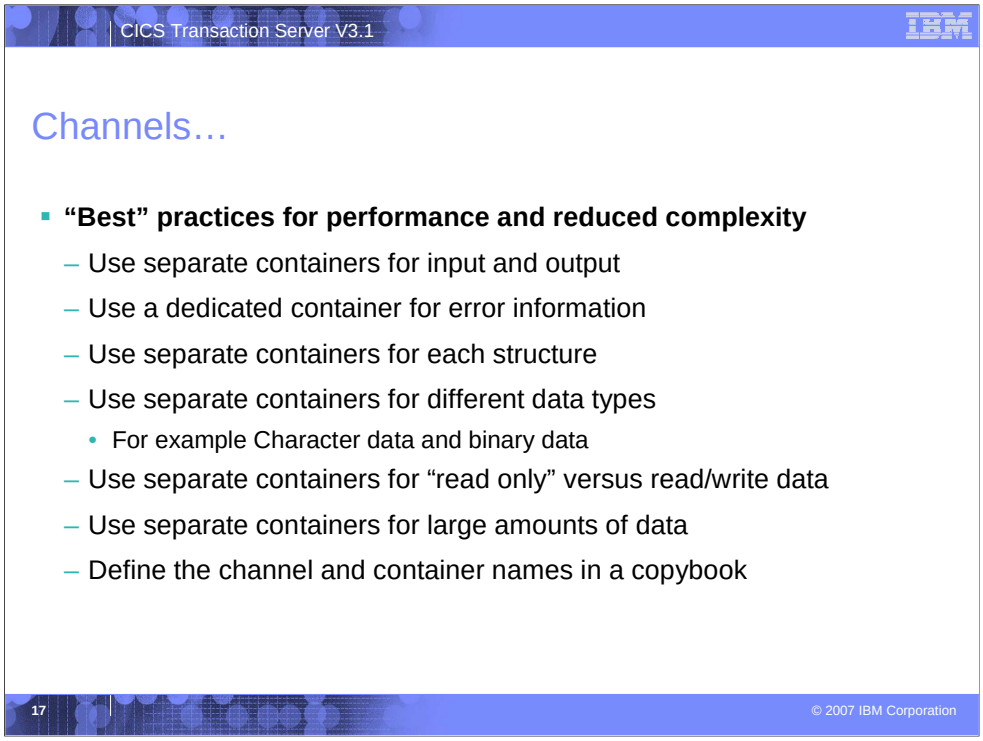
This is an example of channel scope. In this sequence of program LINKs and XCTLs the boxes will show the channel scope and where the channels will be deleted.

Channels…

- **Lifetime**
  - Creation
    - A channel is created by naming it on a command
      - EXEC CICS PUT CONTAINER CHANNEL
      - EXEC CICS MOVE CONTAINER TOCHANNEL
      - EXEC CICS LINK PROGRAM CHANNEL
      - EXEC CICS XCTL PROGRAM CHANNEL
      - EXEC CICS RETURN TRANSID CHANNEL
      - EXEC CICS START TRANSID CHANNEL
  - Deletion
    - A channel is deleted when it goes out of scope to any program in the linkage stack

A channel is created by naming it on an EXEC CICS command. The program will usually place data in the channel by using a PUT CONTAINER command. The PUT CONTAINER command will not only create the channel but create a named container associated with the channel.

A channel will be deleted when it goes out of scope to the programs in the linkage stack. When no application program is able to access the channel it will be destroyed.



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.

Use separate containers for input and output. This will allow you to simplify your copybook structure and make your programs easier to understand. When using channels with DPL, small amounts of data will be transferred between the CICS regions. 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.

Use a separate, dedicated container for 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 a separate container for each structure in the copybook. Consider defining

Channels…

- **EXEC CICS ASSIGN CHANNEL**
  - CHANNEL (data-area)
    - returns the 16 character name of the program's current channel, if one exists; otherwise returns blanks.
  - Used to discover the channel passed to a program

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.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

CICS WebSphere

Java, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.