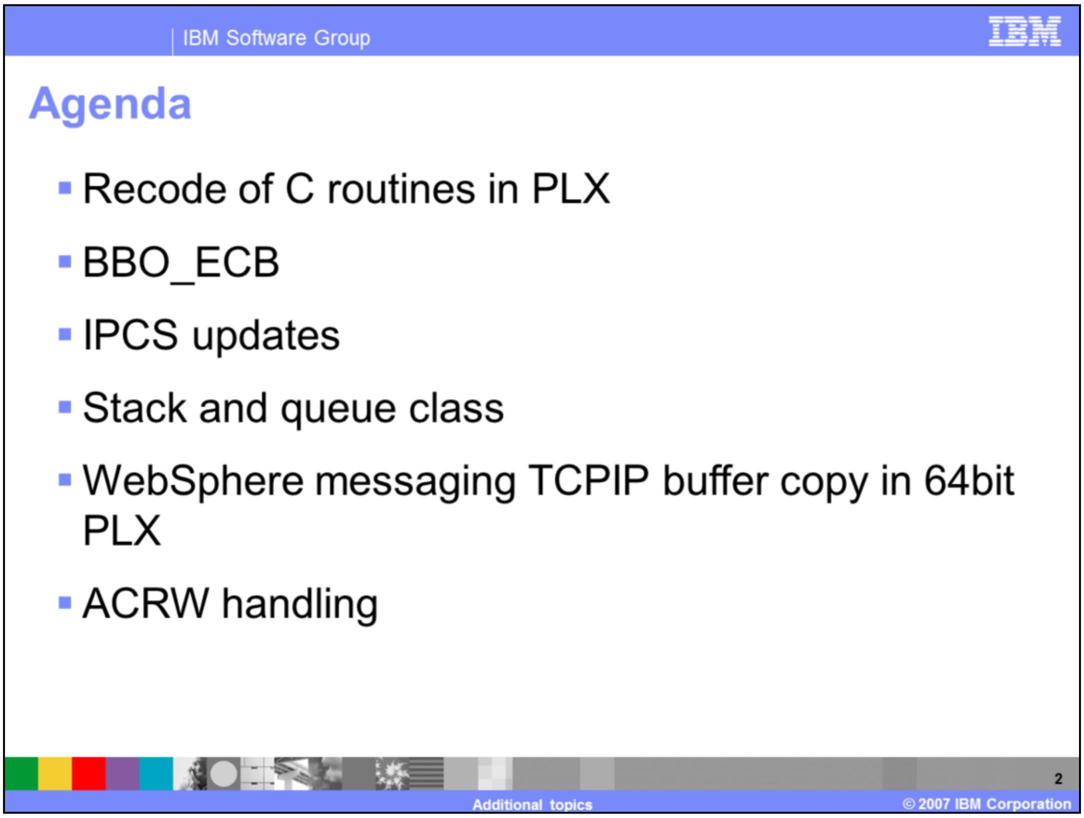
This presentation will address any additional 64-bit topics that have not already been covered for a WebSphere Base Application Server version 6.1 on z/OS.



This presentation will discuss the recoding of C routines in PLX and WebSphere Messaging TCPIP buffer copy in PLX. BBO_ECB, IPCS updates and ACRW handling will also be discussed.

C routines rewritten in PLX

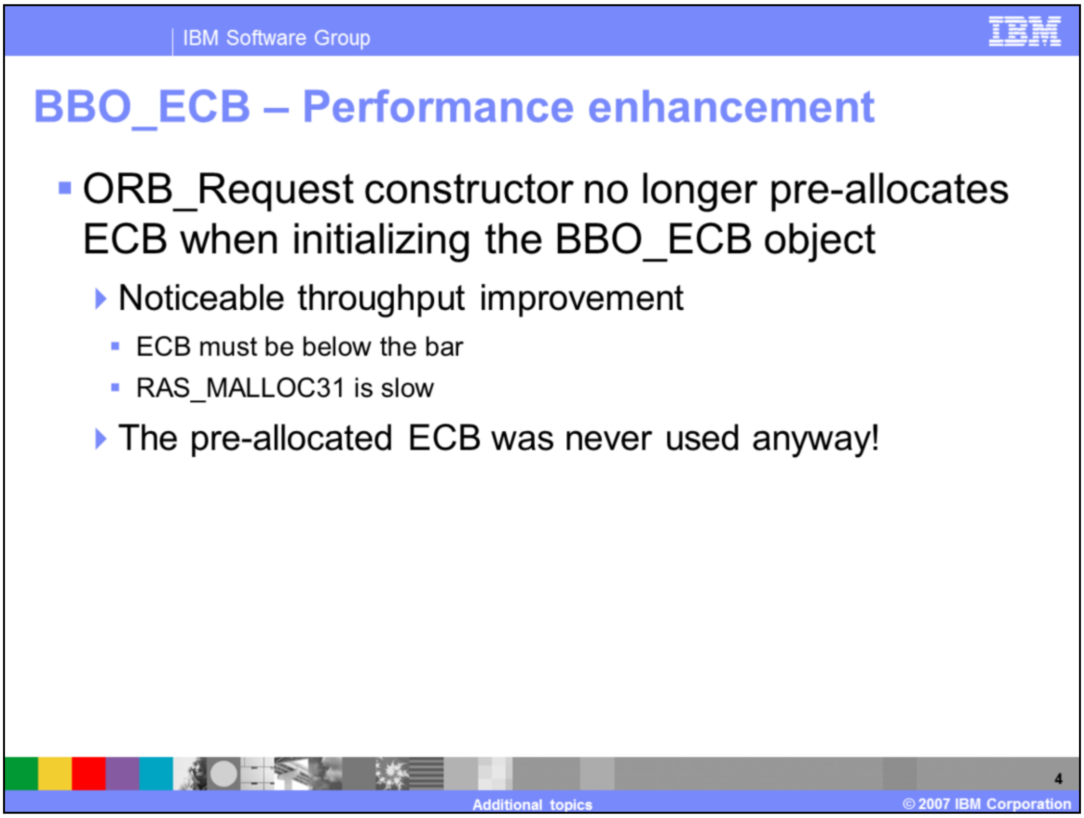
SMF utility routines moved from bbooutlc.c:

- GETSTOR, FREESTOR
  - Redesigned to allocate storage from a CPOOL
    - CPOOL allocated in bboossmf.plx
    - CPOOL ID stored in bacb_smf_cpool
- CVC (now in bboosmut.mac)
- LONGAVG (now in bboosmut.mac)
- StrToBin (now in bbooase3.plx)

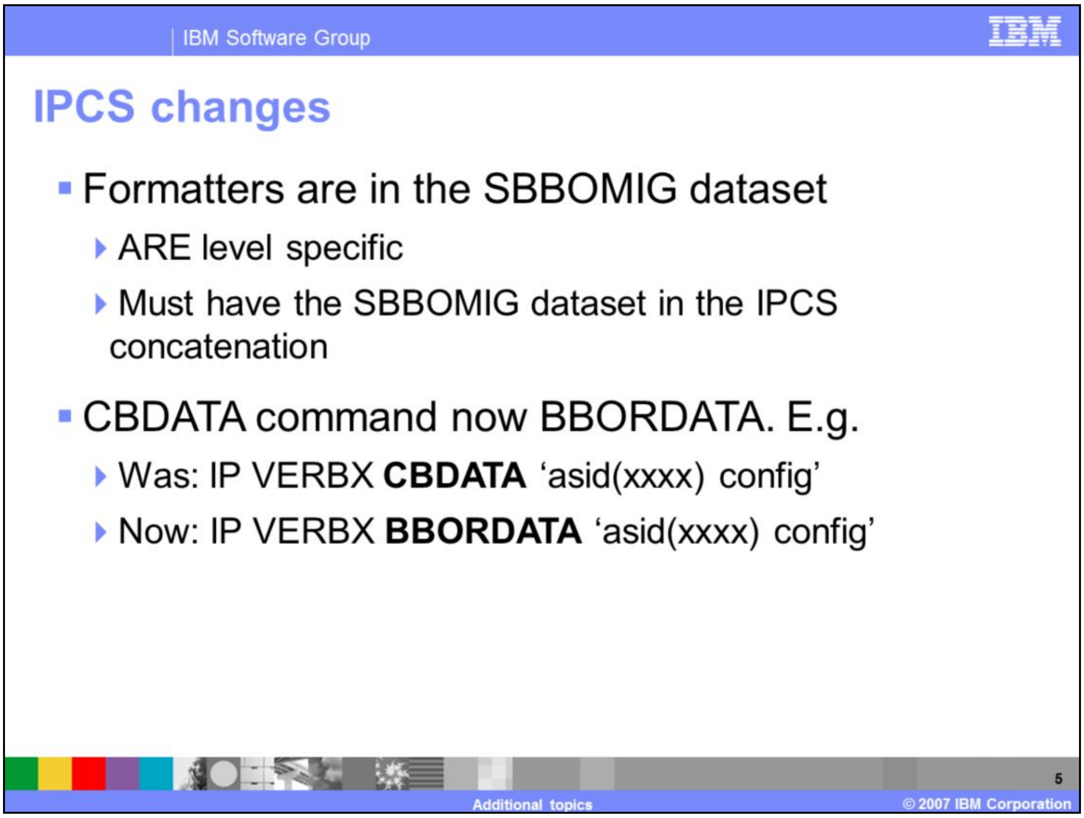
Several C utility routines for SMF have been replaced with PLX code.

The GETSTOR and FREESTOR functionality have been replaced by PLX code that allocates storage using a CPOOL. The CPOOL is created by bboossmf.plx during SMF initialization and the CPOOL ID is stored in bacb_smf_cpool.

The CVC and LONGAVG routines are now in bboosmut.mac, also StrToBin now resides in bbooase3.plx.

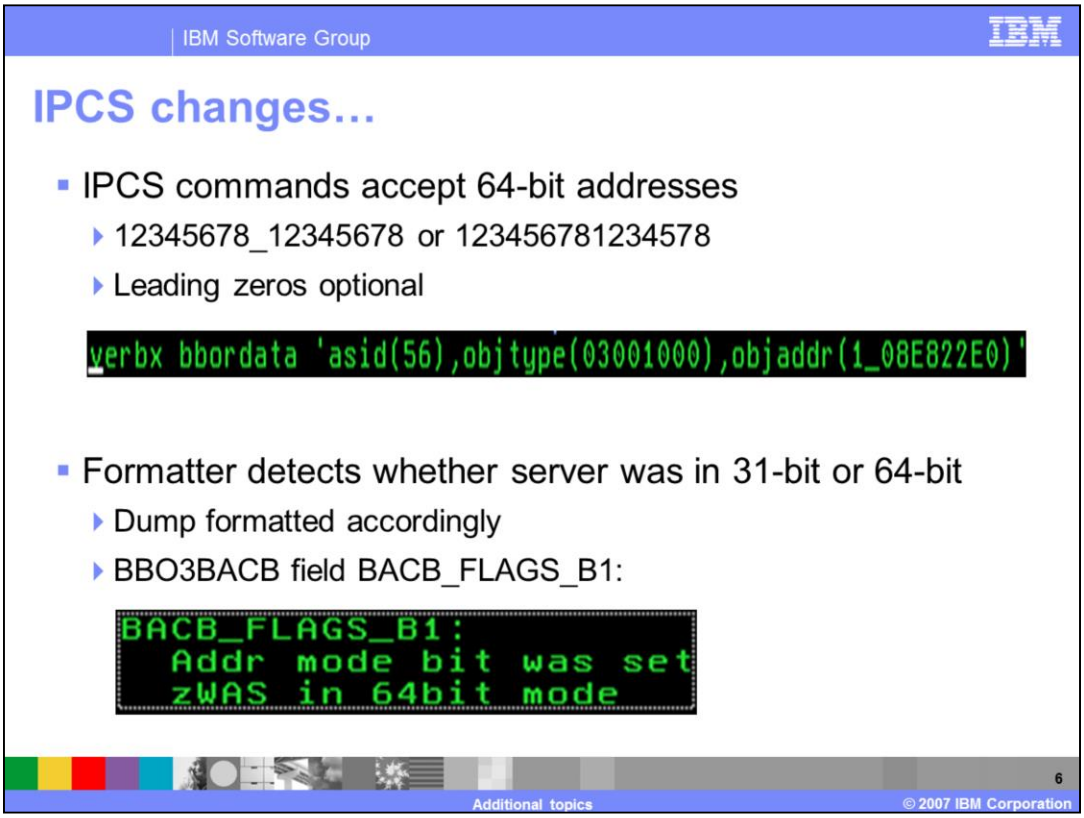


The ORB_Request constructor in bbooorbr.plx was changed to create the BBO_ECB object without a pre-allocated ECB. Since ECBs must be below the bar, the pre-allocation required RAS_MALLOC31, which is extremely slow. The pre-allocated ECB has not been used in the past.



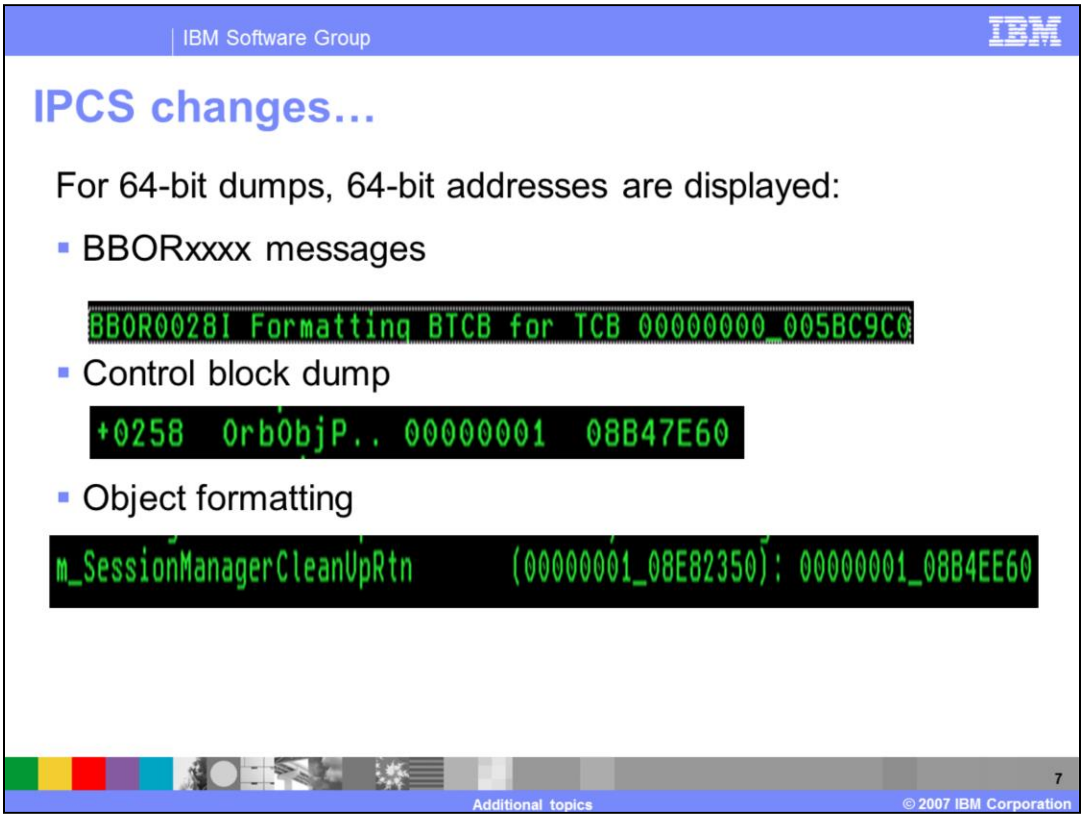
IPCS formatters for z/Websphere related dumps have been updated to handle 64bit data. Formatter utility code resides in SBBOMIG and tends to be specific to the version of the code that produced the dump.

The CBDATA verb to format z/Websphere data is now BBORDATA.



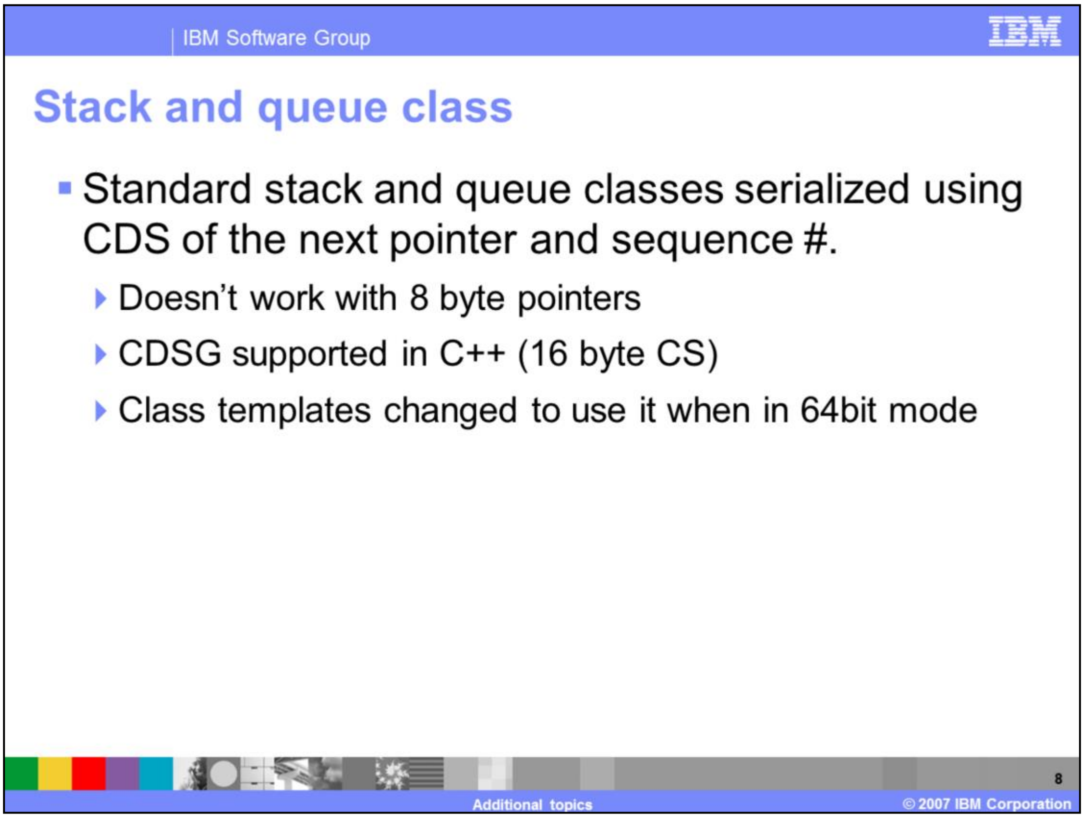
IPCS commands now accept 64-bit addresses, which can be entered with or without an underscore separator before the last 8 digits. Leading zeros are optional. Here is an example of a command to format the SessionManager object which is above the bar.

The formatter determines whether the dump was 64-bit or 31-bit mode by looking at a new flag in the BACB. For 64-bit mode dumps, you will see “zWAS in 64bit mode” when formatting the BACB. The absence of this message means the dump is 31-bit.

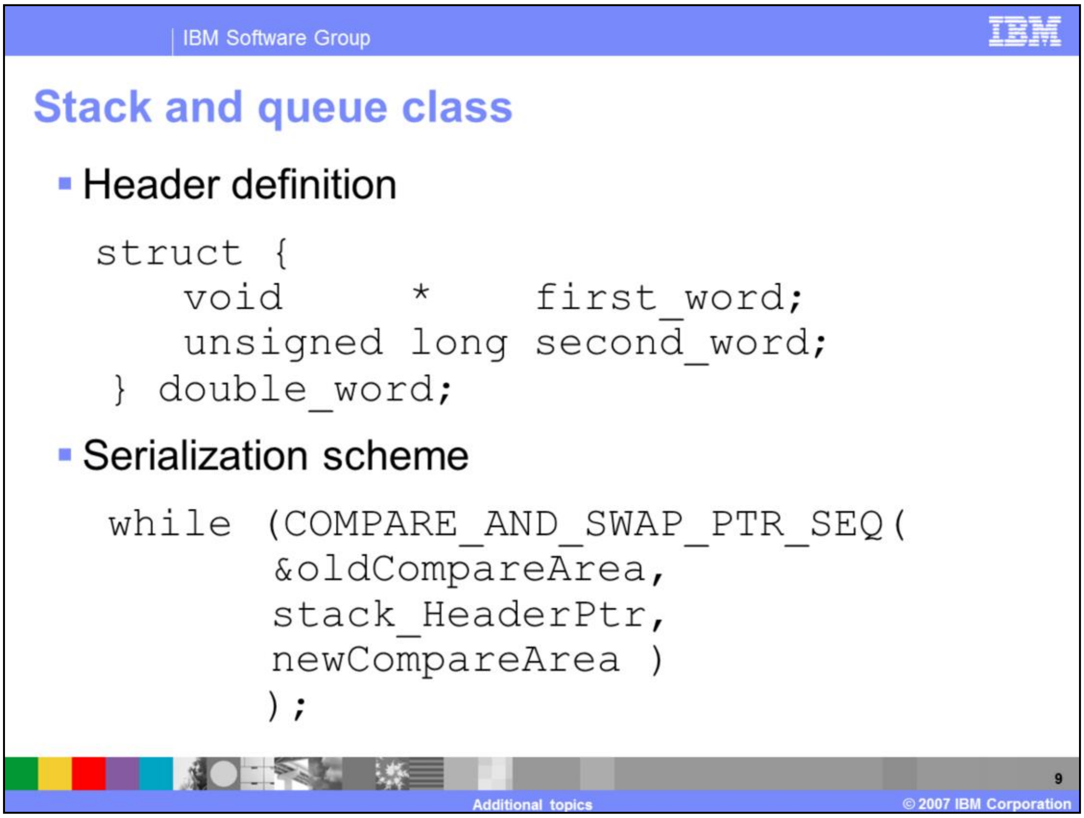


Addresses in BBORxxxx messages will always be formatted as 64-bit, even if they are only 31-bit, for example, TCB.

In a control block dump, a 64-bit address will appear as two words without the _ connector. In a formatted object, the address in the dump of an attribute is shown with the underscore connector and, if the value of the attribute is a 64-bit pointer, it will as well.



Several stack and queue classes depend on compare and swapping for the serialization when adding or deleting member elements. The next element address and the sequence number are what get tested and swapped using a CDS. This had to be changed for 64bit because the 8 byte pointer maxed out the CDS. Since CDSG is supported natively in the C/C++ compiler, classes that use a pointer plus sequence number for serialization were updated.



The first update is in the stack or queue element header. The design was to allow for an 8 byte compare and swap in 31-bit mode and 16 bytes in 64-bit mode. By making a long of the sequence number, or whatever the low order component is, the header is the right size in both build modes.

Other changes included updating the means for doing the compare and swap. It expands into a CDS or CDSG depending on the compile mode.

Stack and queue class

- The rub –
  - CDSG D(S) operand **MUST** be aligned on a quadword boundary
  - The template class is affected, for example:
  -
```
 // Queue Elements used to chain Local Connection Objects
static Stack_CDS<CF_Local_ConnectionQueElem> *
                       CF_Local_QueueElementPoolPtr
#ifdef _LP64
    __attribute__ ((aligned(16)));
#else
    ;
#endif
```
  - Affected parts
    - bboucds.h
    - bboustdt.h
    - bbouqnls.h
    - bboulong.h – CS types and macro definitions

Something that was discovered later was that the CDSG instructions require that their storage operand be on a quad word boundary. That was fixed by adding the C aligned attribute on the element declarations. Since it is unnecessary in 31-bit, the LP64 macro logic, as shown here, is used.

WebSphere messaging TCPIP buffer copy in 64bit PLX

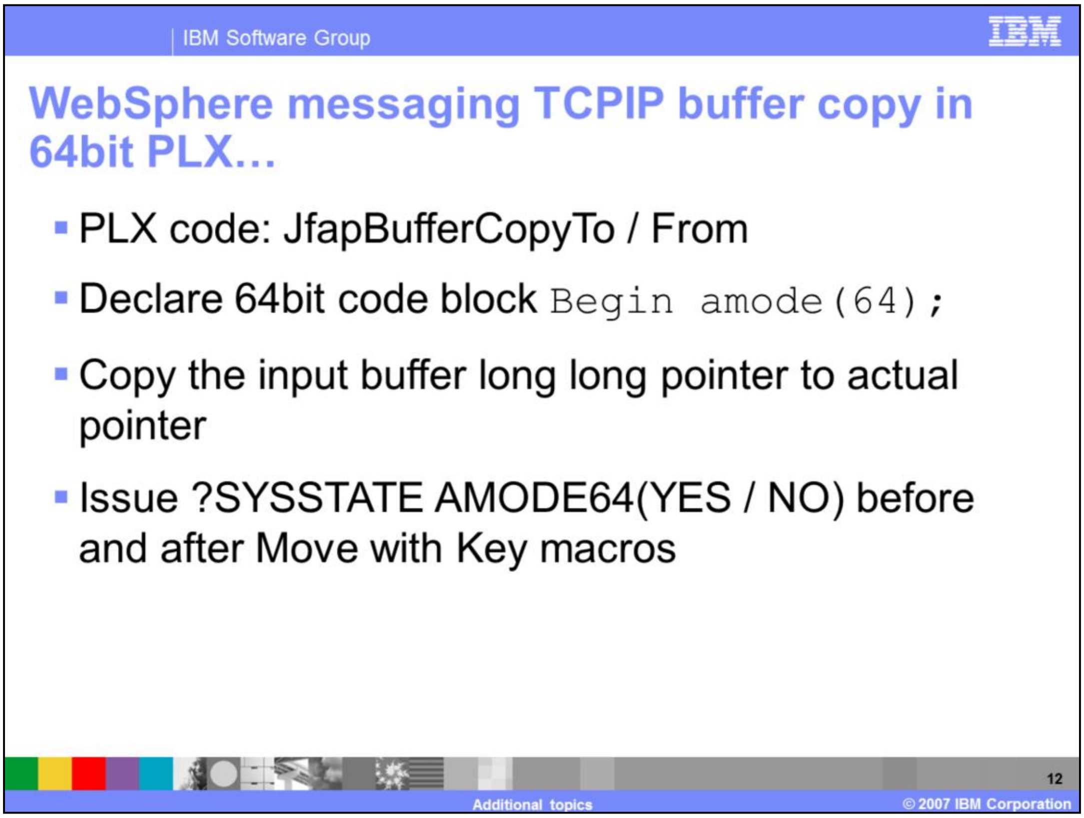
- Background
  - Buffers start out in JAVA code and get passed through C to PLX using glue code.
  - PLX copies data buffers to/from a dataspace.
  - Performance fix to eliminate glue
- Updates
  - Send and receive glue routine do not copy data buffer
  - Pass buffer address as long long

An enhancement was made to the CRA WebSphere Messaging buffer handling. These buffers contain data that was copied into or out of a z/OS data space. C code now deals with the data in buffers and there is an interface to the PLX code that handles the copying with the data space.

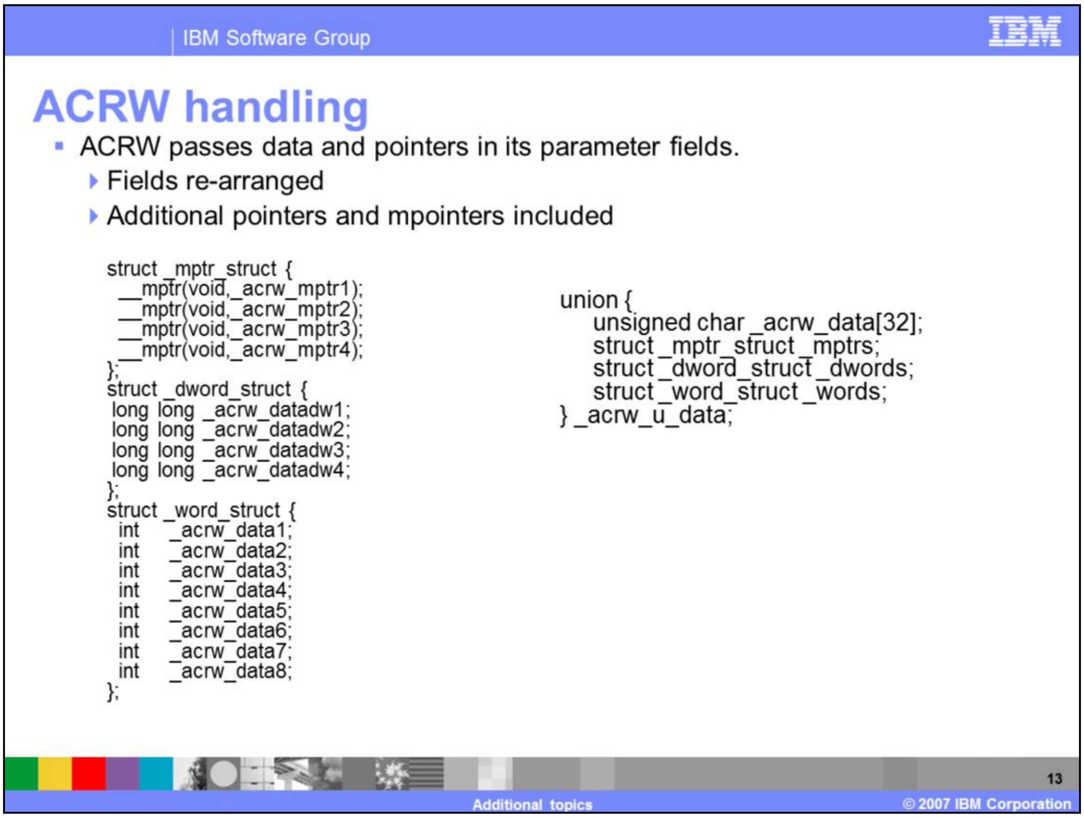
In 64-bit mode, the interface has to be through glue code. The primary function is to copy the buffers below the bar when the data is on the way in, and above the bar when the data is on the way back out. Since the buffers are big chunks of data, the glue copying takes time.

It turns out that the PLX code itself does not do much but copy the buffer. Given this, it was too much overhead to copy the data first in the glue code and very soon after copy it a second time in the PLX code.

The buffer handling was changed to eliminate the extra copying by making the PLX code smart enough to do it in 64-bit mode. The glue code was changed to not do the buffer copy. The glue code is still necessary, as the plist and parameters are needed to be below the bar because the PLX module itself is still in 31-bit mode. That also meant that the buffer address could not be a 64 pointer. The PLX compiler does not accept declaring ptr(64) in 31-bit mode so a fixed(64) or in C long is used.



In the receive and send buffer PLX modules, a 64-bit aware block of code is used by the begin Amode(64) construct. Within that code the compiler goes into 64-bit mode and allows specifying 64-bit pointers. The SYSSTATE macro enables the generated macro code to be 64-bit aware. For example, register operands are treated as 64-bit registers.



The last topic to cover is the ACRW. The ACRW which is a general purpose passer of data and pointer information, was affected by 64-bit support. The fields in the ACRW were rearranged and now can accommodate four 8 byte pointers or mptrs and 8 one word data areas. The areas are overlaid and so care needs to be taken when passing and extracting information in the ACRW.

Trademarks, copyrights, and disclaimers