64-Bit User Space in H06.24/J06.13

Tom Eastep – Program Architect
Billy Jasieniecki – Languages & Tools Architect
Scott Norton – NonStop Kernel Architect

April 19, 2012

© 2012 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice
Agenda

• Overview
• 64-bit Memory
• LP64 Compile Model
• Tools
• Migration Aids
• Debugging
• OSS / Guardian Interoperability
• Mixed Data Model Programming
Overview

• Goal: Give application programs access to much more virtual memory.

• 58 T-numbers participated, producing 61 NPVs/SPRs
  - Many dependencies
  - Most of these NPVs/SPRs are not installable on earlier RVUs (see softdocs for details)
Overview

• Provides true 64-Bit OSS processes
  – The heap used by malloc() is in 64-bit addressable memory
  – Default maximum heap size is 12GB
  – Limit can be raised (or lowered) either at link time or at process creation time

• Guardian processes remain 32-bit

• 32-bit Guardian, 32-bit OSS and 64-bit OSS processes all have access to large flat segments in 64-bit addressable memory
  – File system I/O is done directly to/from these segments through a set of new 64-bit APIs
    • Both OSS and Guardian I/O APIs are provided
  – 64-bit flat segments can be shared among the three process types
Overview - continued

• Both Guardian and OSS processes can have up to 510GB of process-private virtual address space vs. 2GB previously
Overview – memory models

- We expect 64-bit OSS processes to be used when:
  - An existing OSS application does not fit well within the 32-bit model
  - An existing application is being ported to OSS, especially if it already uses the 64-bit model
  - New applications that don’t require features not yet supported on 64-bit applications.
Overview – memory models

• We expect the vast majority of existing applications to remain 32-bit
  – Migration requires some effort
  – Data footprint for a 64-bit application is larger than the footprint of the same application compiled for 32-bit operation
Overview - restrictions

- Restrictions
  - 64-bit C++ programs must use the version 3 library
  - 64-bit threaded OSS programs must use the POSIX User Threads (PUT) library
  - Full access to only a subset of the Guardian API from 64-bit OSS programs
  - In a 64-bit OSS process, only the CRE (malloc()) heap and flat segments are currently outside of the 32-bit addressable range
  - Currently no access to SQL from 64-bit applications.
Overview - restrictions

• Restrictions – continued
  – While the architectural limit on process-private address space is 510GB, the practical limit is much less

Depends on:
• Swap space configuration
• Physical memory size
• Number and size of the programs contending for these resources
• Locality of reference
Overview – continued

• Documentation
  – Overview
    • *OSS Programmer’s Guide* – Chapter 11
    • *Guardian Programmer’s Guide* – Appendix A
Overview – continued

• Documentation – continued
  – Detailed
    • OSS System Calls Reference Manual
    • OSS Library Calls Reference Manual
    • OSS Porting Guide
    • Guardian Procedure Calls Reference Manual
    • Guardian Errors and Messages Manual
    • ENSCRIBE Programmer’s Guide
    • eld Manual
    • EMS Manual
    • TCP/IP Programming Reference Manual
    • SPI Programming Manual
    • TSMP Pathsend and Server Programming Manual
    • Native Inspect Manual
    • TNS/E Inspect Manual
    • TMF Application Programmer’s Guide
    • C/C++ Programmers Guide
    • pTAL Reference Manual
    • White Papers and Quick Reference Guides
Agenda

• Overview
• 64-bit Memory
• LP64 Compile Model
• Tools
• Migration Aids
• Debugging
• OSS / Guardian Interoperability
• Mixed Data Model Programming
Sign-extended Addresses

• A 64-bit address can be formed from a 32-bit address through *sign extension*.
  – If the high-order bit of the 32-bit address is zero, then the high-order 32 bits of the corresponding 64-bit address are zero.
  – If the high-order bit of the 32-bit address is one, then the high-order 32 bits of the corresponding 64-bit address are one.
  – We call these *sign-extended 32-bit addresses*.

• A sign-extended 32-bit address can be converted to a 32-bit address through simple truncation.
Memory Map (Not to Scale)

- Implemented in H06.20/J06.09
- The dark blue regions are referred to as 32-bit addressable
- In this presentation, the light-colored regions are referred to as 64-bit addressable, but 32-bit addressable areas are also 64-bit addressable.
Memory Map

- Process Private 32-bit Addressable Space
  - 0x00000000-0x7FFFFFFFF
  - The content of this space in a 64-bit OSS process is the same as that for a 32-bit process.
    - Globals
    - Program code
    - Private DLL code
    - User Stacks
    - PFS
    - DLL instance data
  - Because flat segments and heap are now in 64-bit addressable space, there is more space for these.
64-bit heap

• For a 64-bit OSS process:
  − By default, the CRE heap used by malloc() is 12GB, with its base at 0x100000000 (4GB) and its limit at 0x400000000 (16GB)
  − The maximum size can be changed:
    • At link time using `-set heap_max`
    • At run-time
      − PROCESS_SPAWN(), tdm_spawn() and the tdm_exec() family allow the maximum heap size to be set.
      − The OSS run utility accepts a new `-heapmax=N` where N is the maximum heap size in MB. The size can be specified in GB by using a ‘GB’ suffix (e.g., 50GB)
    • If the maximum heap size is >= 28GB, its base is placed at address 0x800000000 (32gb) to optimize the memory management table structure.
    • Upper limit is 480GB
64-bit flat segments

- 64-bit flat segments can be allocated using `SEGMENT_ALLOCATE64()`.
- Guardian programs and both 32-bit and 64-bit OSS programs can allocate (and share) 64-bit addressable flat segments.
- Allocation strategy is ‘first-fit with good alignment’.
  - If the heap is placed at 0x8000000000, the 28GB range from 0x1000000000-0x8000000000 is still available for flat-segment allocation.
Agenda

• Overview
• 64-bit Memory
• LP64 Compile Model
• Tools
• Migration Aids
• Debugging
• OSS / Guardian Interoperability
• Mixed Data Model Programming
Data Models

- Our current native C/C++ uses the *ILP32* data model.
  - Integers are 32 bits wide
  - Long integers are 32 bits wide.
  - Pointers are 32 bits wide

- The industry standard for 64-bit C/C++ is *LP64*
  - Integers are 32 bits wide.
  - Long integers are 64 bits wide.
  - Pointers are 64 bits wide

- We have adopted LP64 for 64-bit OSS processes.
Object Files

- Object files can be categorized by data model
  - 32-bit – an object file that uses the ILP32 data model. All existing native object files have this data model
  - 64-bit – an object file that uses the LP64 data model
  - Neutral – An object file with an unspecified data model

- 64-bit linkfiles are produced by specifying the `-Wlp64` option to the TNS/E C/C++ compiler

- The data model is displayed by ENOFT’s FILEHDR command
Loadfiles

• OSS program files can be 32-bit (ILP32) or 64-bit (LP64)
  − Running a 32-bit program creates a 32-bit process
  − Running a 64-bit program creates a 64-bit process
    • `exec()` and `tdm_exec()` families where the `path` parameter specifies a 64-bit program file.
    • `tdm_spawn()` where the `file` parameter specifies a 64-bit program file.
    • `fork()` of a running 64-bit process.
    • `PROCESS_SPAWN_()` where the `program-file` names a 64-bit program.

• Guardian program files can only be 32-bit

• DLLs can be 32-bit, 64-bit or Neutral
Loading DLLs

• A 64-bit DLL may only be loaded into a 64-bit process.
• A 32-bit DLL may only be loaded into a 32-bit process.
• A Neutral DLL may be loaded into either a 32-bit or a 64-bit processes.
  - Neutral DLLs may depend only on other neutral DLLs and must explicitly declare pointers in entry point functions as 32-bit or 64-bit pointers.
  - The Implicit DLLs are Neutral.
  - The Networking DLLs loaded by QIO are Neutral.
  - ZPGODLL (Performance Guided Optimization) is Neutral.
  - ZIMPIMP is Neutral
Public DLLs

• The public DLLs have traditionally had names of the form ZaaaDLL or ZbbbbDLL. These names are retained for 32-bit and Neutral DLLs.

• New 64-Bit DLLs have names of the form YaaaDLL or YbbbbDLL.

• Example:
  – ZCRTLDLL – 32-bit C runtime
  – YCRTLDLL – 64-bit C runtime
Agenda

• Overview
• 64-bit Memory
• LP64 Compile Model
• Tools
• Migration Aids
• Debugging
• OSS / Guardian Interoperability
• Mixed Data Model Programming
Tools - ENOFT

- ENOFT has been enhanced to display the data model

```
enoft> filehdr

******* ELF File Header

Ident:        ELF64-bit Big_Endian NSK
Type:         Linkfile
Target Machine:  IA64
...
Flags:        (0x830000)
         NO_SHARED_GLOBALIZED
Neutral DATA MODEL
...
enoft>
```
Tools - ELD

• You may not specify both 32-bit and 64-bit linkfiles in a single execution of ELD.

• Neutral linkfiles can be created from either 32-bit or 64-bit linkfiles.
  - A neutral linkfile can be produced using eld:
    • by using the \textit{–r} option and specifying \texttt{–set data_model neutral}
    • by specifying \texttt{–change data_model neutral}
  - Neutral linkfiles may be specified to the linker along with either 32-bit or 64-bit linkfiles

3/7/2012

NonStop Technical Update
Tools – ELD - Continued

• ELD output can be:
  – 32-bit if no 64-bit files have been input
  – 64-bit if no 32-bit files have been input
  – Neutral

• By default ELD assumes the data model of the first linkfile it is given.

• You can override using `-set data_model`
  – `-set data_model ilp32`
  – `-set data_model lp64`
  – `-set data_model neutral`
Tools – ELD - Continued

• When building a 64-bit DLL or Program:
  - ELD looks in the location of ZREG for YabcDLL when searching for a 64-bit public DLL (which normally is what is listed in the dependent loadfile’s liblist). If that name is not found and the environment is not Guardian, then ELD will also look for libabc\text{y}.so.
  - When in the Guardian environment, ELD will look in $\text{SYSTEM.YDLL}$ for a 64-bit DLL prior to looking in $\text{SYSTEM.ZDLL}$
  - When in the OSS environment, ELD looks in /lib64, /usr/lib64 and /usr/local/lib64 for 64-bit DLLs.
Tools – C/C++

- **-Wlp64** option (or LP64 in the Guardian environment) for specifying the LP64 data model.
  - Error if **-Wsystype=guardian**
  - Error if **-Wversion2**
  - Error if **-Wsql**

- C “Main”
  - cmain64.o on OSS and Windows
  - cmain64 on Guardian

- New Macros
  - **__ILP32** – defined with value 1 if 32-bit compilation.
  - **__LP64** – defined with value 1 if 64-bit compilation.
Tools – C/C++ – Explicitly Declared Pointers

- In both 32-bit and 64-bit mode, the default C/C++ data model can be overridden using modifiers:
  - `'_ptr64 *'` creates a 64-bit pointer or reference
  - `'_ptr32 *'` creates a 32-bit pointer or reference
- Examples:
  - char _ptr64 * longPtr;
  - char _ptr32 * shortPtr;

- New builtins (require `#include <builtin.h>`):
  - `(void _ptr32 *)_ptr64_to_ptr32(void _ptr64 *)`
    - Generates an overflow trap if the passed pointer is not a sign-extended 32-bit pointer
  - `int _is_32bit_addr(void _ptr64 *)`
    - Returns true if argument is a sign-extended 32-bit pointer
Tools – C/C++ -- Alignment

• In 64-bit compilation units, the following data types are disallowed with shared2 and shared8 aligned structures and classes:
  − long
  − unsigned long
  − * (without explicit _ptr32 or _ptr64)
Agenda

• Overview
• 64-bit Memory
• LP64 Compile Model
• Tools
• Migration Aids
• Debugging
• OSS / Guardian Interoperability
• Mixed Data Model Programming
The following messages are always enabled

- Any direct assignment from a 64-bit pointer to a 32-bit pointer will emit:
  
  error(611): a value of type "<long pointer type>" cannot be assigned to an entity of type "<short pointer type>"

- Any cast of a 64-bit pointer to a 32-bit pointer will emit:
  
  warning(2404): explicit cast of a 64-bit pointer to a 32-bit pointer: truncation possible

- Any conversion from a 64-bit pointer to a shorter integer type will emit:
  
  C compiles:
    warning(912): conversion from pointer to smaller integer
  
  C++ compiles:
    error(256): invalid type conversion
Tools – C/C++ – Migration Aids – continued

- **Wmigration_check=32to64** on ILP32 compilation
  
  - Any conversion (using either a cast or a direct assignment) from a long to an int will emit:
    
    warning(2412): 64 bit migration: type conversion may truncate value

  - Any conversion from a pointer to long to a pointer to int or from a pointer to int to a pointer to long type will emit:
    
    warning(2414): 64 bit migration: type conversion may cause target of pointers to have a different size

  - Any relational compare involving an unsigned int and a signed long will emit: warning(2411):
    
    64 bit migration: comparison of unsigned integer with signed long
Tools – C/C++ – Migration Aids – continued

- **-Wmigration_check=32to64 – continued**
  - Any call of a memory allocation function (malloc, etc.) without first declaring a prototype will emit:
    
    ```
    warning(2415): no prototype or definition in scope for call to memory allocation routine "<function name>"
    
    Note that this condition always generates an error in c99 or C++.
    ```
  - Any conversion from a pointer to an integer of the same size type will emit:
    
    ```
    warning(2161): conversion from pointer to same-sized integral type (potential portability problem)
    ```
Agenda

• Overview
• 64-bit Memory
• LP64 Compile Model
• Tools
• Migration Aids
• Debugging
• OSS / Guardian Interoperability
• Mixed Data Model Programming
Debugging

• Snapshot files
  – New snapshot file format (64-bit)
    • Based on the industry standard open ELF64 core file format
    • Uses industry-standard IA64 Linux/GCC data structures
  – The new format is used when the target process has one or more 64-bit user segments
  – Legacy (32-bit) format is used otherwise
  – Both have filecode 130
  – 64-bit snapshot can be Enscribe format-1 (max 2GB) or format-2 (max 1TB minus 4GB).
  – Visual Inspect will not read the new format
Debugging – continued

• Snapshot file compression
  – 64-bit snapshot files can be compressed when they are created using the `save` command:
    ```
    save filename [compression] ![]
    ```
  – `compression` may be `bzip2`, `gzip` or `none`
  – The `snapshot` command automatically detects compression and acts accordingly.
Debugging – continued

- Snapshot file compression:
  - There are significant time/space tradeoffs with compression:
    - none is the fastest but takes the most space
    - gzip is next fastest but takes up less space than none
    - bzip2 is the slowest but the compressed dump takes up the least space
  - In the absence of a save option, and in the saveabend case, the file size is used to determine the compression option.
  - Note that dumps can be compressed after the fact using the standard utilities from the ITUG Library
Debugging – continued

• Reading Snapshot files
  – The Snapshot Read/Write routines (T0745) continue to be available to read 32-bit snapshot files.
  – 64-bit files can be accessed directly using the ELF64 core file headers included with T0745.
Debugging - continued

• eInspect
  – Re-based to GDB 6.8 from gdb 5.0
  – Re-based to TCL 8.5 from 8.0
    • Supports 64-bit arithmetic
  – Can be used to debug 64-bit processes
  – Can be used to analyze both 32-bit and 64-bit snapshot files

• eGarth
  – will create 64-bit snapshot files from a CPU dump for processes having 64-bit segments (including Guardian and OSS 32-bit processes)
Debugging – continued

• Visual Inspect
  – Can be used to debug 32-bit processes with 64-bit segments, including the contents of the 64-bit segments.
  – Cannot be used to debug 64-bit processes.
  – Cannot be used to analyze the new 64-bit format snapshot files that contain one or more 64-bit segments.
   • Attempt to open a 64-bit snapshot is rejected with an error
Agenda

- Overview
- 64-bit Memory
- LP64 Compile Model
- Tools
- Migration Aids
- Debugging
- OSS / Guardian Interoperability
- Mixed Data Model Programming
OSS/Guardian Interoperability

- Guardian Calls in 64-bit programs
  - Many, but not all, Guardian APIs can be called from 64-bit OSS processes.
    - Listed in the OSS Programmer’s Guide
OSS/Guardian Interoperability

- For allocating and managing 64-bit addressable memory, a new set of APIs has been created:
  - Allocate and manage 64-bit flat segments
  - Dynamically allocate/deallocate memory through 64-bit pool routines
  - For Guardian Filesystem I/O, a new set of APIs has been created:
    - Directly return a file management error as opposed to a condition-code.
    - Accept 64-bit buffers
    - Accept 32-bit transfer sizes
    - Have 64-bit no-wait tags so that 64-bit addresses can be used as tags.
Interoperability - Continued

- 64-bit flat segments can be allocated and managed with:
  - ADDRESS_DELIMIT64_
  - REFPARAM.BoundsCheck64_
  - SEGMENT_ALLOCATE64_
  - SEGMENT_DEALLOCATE_
  - SEGMENT_GETINFO64_
  - SEGMENT_GETINFO_STRUCT_
  - SEGMENT_GET_MIN_ALIGN_
  - SEGMENT_GET_PREF_ALIGN_
  - SEGMENT_RESIZE64_
Interoperability - Continued

• New 64-bit pool routines can be used for dynamic allocation/deallocation of 64-bit memory:
  − Much higher performance than the pre-existing Guardian pool management routines
  − A pool need not be a contiguous block of virtual memory
    • May be grown by allocating an additional segment; and
    • ‘augmenting’ the pool to include that segment.
  − New APIs for pool management:
    − POOL64_AUGMENT_
    − POOL64_CHECK_
    − POOL64_CHECKSHRINK_
    − POOL64_DEFINE_
    − POOL64_DIMINISH_
    − POOL64_GET_
    − POOL64_GETINFO_
    − POOL64_PUT_
    − POOL64_RESIZE_
Interoperability - Continued

• New ENSCRIBE I/O APIs:
  - FILE_AWAITIO64_()
  - FILE_CONTROL64_()
  - FILE_CONTROLBUF64_()
  - FILE_LOCKFILE64_()
  - FILE_LOCKREC64_()
  - FILE_READ64_()
  - FILE_READLOCK64_()
  - FILE_READUPDATE64_()
  - FILE_READUPDATELOCK64_()
  - FILE_REPLY64_()
Interoperability – continued

- ENSCRIBE APIs – continued:
  - FILE_SETMODENOWAIT64()
  - FILE_UNLOCKFILE64()
  - FILE_UNLOCKREC64()
  - FILE_WRITE64()
  - FILE_WRITEREAD64()
  - FILE_WRITEUPDATE64()
  - FILE_WRITEUPDATEUNLOCK64()
  - FILENAME_FINDNEXT64()

- These existing APIs work with the above:
  - CANCELREQL()
  - FILE_COMPLETEL()
Agenda

• Overview
• 64-bit Memory
• LP64 Compile Model
• Tools
• Migration Aids
• Debugging
• OSS / Guardian Interoperability
• Mixed Data Model Programming
Mixed Data-model Programming

- Definition: Use of both 32-bit and 64-bit addresses and pointers in the same compilation unit.

- Only needed for:
  - Access to 64-bit segments from 32-bit programs
  - Using 32-bit system calls from 64-bit OSS programs.
Mixed Data-model Programming - 64-bit segments in 32-bit Programs

• 32-bit programs can allocate a 64-bit segment using SEGMENT_ALLOCATE64_() (kmem.h) and access the segment using 64-bit pointers

• 64-bit pool routines can be used for dynamic allocation/deallocation (kpool64.h)

• In string.h are prototypes for string manipulation functions that:
  – Accept 64-bit pointers and data lengths.
  – Return pointers that are also 64-bit wide.
Mixed Data-model Programming - 64-bit segments in 32-bit Programs continued

• Functions:
  - `void _ptr64* memchr64(const void _ptr64*, int, unsigned long long);`
  - `int memcmp64(const void _ptr64*, const void _ptr64*, unsigned long long);`
  - `void _ptr64* memcpy64(void _ptr64*, const void _ptr64*, unsigned long);`
  - `void _ptr64* memmove64(void _ptr64*, const void _ptr64*, unsigned long);
  - `void _ptr64* memset64(void _ptr64*, int, unsigned long long);`
  - `char _ptr64* strcat64(char _ptr64*, const char _ptr64*);`
  - `char _ptr64* strchr64(const char _ptr64*, int);
  - `int strcmp64(const char _ptr64*, const char _ptr64*);
  - `char _ptr64* strcpy64(char _ptr64*, const char _ptr64*);
  - `unsigned long long strlen64(const char _ptr64*);`
  - `char _ptr64* strncat64(char _ptr64*, const char _ptr64*, unsigned long);
  - `char _ptr64* strchr64(const char _ptr64*, int);
  - `int strncmp64(const char _ptr64*, const char _ptr64*, unsigned long long);
  - `char _ptr64* strncpy64(char _ptr64*, const char _ptr64*, unsigned long);
  - `char _ptr64* strstr64(const char _ptr64*, const char _ptr64*);`
Mixed data-model Programming - 64-bit segments in 32-bit Programs continued

- The 64-bit Enscribe I/O functions described earlier (FILE_READ64_(), etc) also are available from 32-bit OSS and Guardian programs
  - Allow I/O operations directly to/from 64-bit segments
  - Accept 64-bit buffers
  - Accept 32-bit transfer sizes
  - Have 64-bit no-wait tags so that 64-bit addresses can be used as tags.
Mixed data-model Programming - 64-bit segments in 32-bit Programs continued

• The following new Guardian Socket APIs also accept 64-bit buffer addresses:
  - send64_()
  - sendto64_()
  - recv64_()
  - recvfrom64_()
  - send_nw64_()
  - send_nw2_64_()
  - sendto_nw64_()
  - t_sendto_nw64_()
  - recv_nw64_()
  - recvfrom_nw64_()
  - t_recvfrom_nw64_()
Mixed Data-model Programming – 64-bit segments in 32-bit Programs continued

- The following OSS APIs provide 64-bit I/O buffer support from 32-bit programs:
  - `read64_()`
  - `recv64_()`
  - `recvfrom64_()`
  - `recvmsg64_()`
  - `send64_()`
  - `sendto64_()`
  - `sendmsg64_()`
  - `write64_()`
Mixed Data-model Programming - 64-bit applications using 32-bit buffers

• Given that not all of the Guardian API has been converted to accept 64-bit pointers, there can be a need for 32-bit pointers in a 64-bit program.

• The 32-bit heap (malloc32()) is used for this purpose.
  – While the process’s stack is currently 32-bit addressable, that might not be true in the future
  – We therefore discourage creating 32-bit pointers referring to local variables.
Mixed Data-model Programming - 64-bit applications using 32-bit buffers continued

Example – from cextdecs.h:

```c
// #pragma section definesave
// #if (_TANDEM_ARCH_ > 1)
_tal _extensible short DEFINESAVE (
  const char _ptr32 *, /* IN */
  short _ptr32 *,      /* OUT */
  short ,              /* IN */
  short _ptr32 *,      /* OUT */
  short               /* IN */
);
// #else
_tal _extensible short DEFINESAVE (
  const char _far *,  /* IN */
  short _far *,       /* OUT */
  short ,             /* IN */
  short _far *,       /* OUT */
  short               /* IN */
);
// #endif
```

3/7/2012
NonStop Technical Update 58
Mixed Data-model Programming – 64-bit applications using 32-bit buffers continued

```c
#include <cextdecs.h(DEFINESAVE)>
...
struct savedDef {
    char name[24];
    short length;
    short data[1024];
}

char _ptr32 * defineName;
struct savedDef _ptr32 * savedDefine;
short result;
savedDefine = (struct savedDef _ptr32 *)malloc32( sizeof savedDef );
if ( ! savedDefine ) {
    ...
}
memcpy( savedDefine->name, "=MYDEFINE" );
result = DEFINESAVE( savedDefine->name, savedDefine->data, 1024,
                    &savedDefine->length );
if ( result != 0 ) {
    ...
}
... free32( savedDefine );
```
Mixed Data-model Programming – epTAL: 64-bit addresses

• epTAL
  - The epTAL compiler does not support an equivalent of LP64.
  - It does include support for dealing with 64-bit addresses.
  - This support is enabled using the __EXT64 option.
    • Defines a TOGGLE of the same name.
    • If you wish to use the toggle in definition files that are also used in TAL, it’s a good idea to include \texttt{?DEFINE\_TOG \_\_EXT64} in any section that references \_\_EXT64
  - If you always want to enable the option for pTAL, use:
    \texttt{\_\_\_}\texttt{IF PTAL}
    \texttt{\_\_\_}\texttt{\_\_EXT64}
    \texttt{\_\_\_}\texttt{ENDIF PTAL}
    \texttt{\_\_\_}\texttt{DEFINE\_TOG \_\_EXT64}
Mixed Data-model Programming --
epTAL: 64-bit addresses continued

• New keywords when __EXT64 is set:
  – .EXT64 – 64-bit indirection
    • Example - INT .EXT64 FOO;
  – EXT64ADDR – A 64-bit address
  – PROC64PTR – A 64-bit pointer to a procedure
  – PROC64ADDR – The address of a 64-bit procedure
  – …
Mixed Data-model Programming -- epTAL: 64-bit addresses continued

• New built-ins when __EXT64:
  - $EXT64ADDR_TO_EXTADDR()
  - $EXT64ADDR_TO_EXTADDR_OV()
  - $EXTADDR_TO_EXT64ADDR()
  - $FIXED0_TO_EXT64ADDR()
  - $FIX() (converts EXT64ADDR expression to a FIXED(0))
  - $XADR64()
  - $PROCADDR() (also accepts a PROC64ADDR expression)
  - $PROC64ADDR() (converts a PROCADDR or PROC64ADDR expression to a PROC64ADDR)
  - $IS_32BIT_ADDR()
  - $UFIX()
Mixed Data-model Programming -- epTAL: 64-bit addresses continued

• To access the new 64-bit ENSCRIBE I/O APIs from epTAL:
  − Specify the __EXT64 compiler option
  − ?SETTOG _64BIT_CALLS
  − ?SOURCE EXTDECS,…sections…)

• The _64BIT_CALLS toggle is also used in many of the declaration files in the ZGUARD ISV to hide 64-bit extensions from callers that don’t want them.
Mixed Data-model Programming -- epTAL: 64-bit addresses continued

• To link an epTAL module into a 64-bit OSS program:
  − All reference parameters must be .EXT64
  − The data model of the generated 32-bit linkfile must be changed to neutral before it can be linked with the 64-bit OSS linkfiles and 64-bit DLLs.
Mixed Data-model Programming - Gotchas

• The following generates a warning 2404 when compiled in 32-bit mode:

```c
char * shortPtr;
char _ptr64 * longPtr;
...
shortPtr = (char *)longPtr;
```

• Recode as:

```c
shortPtr =
        (char *)__ptr64_to_ptr32( longPtr );
```
Mixed Data-model Programming - Gotchas

• This is common when converting a 32-bit function to accept 64-bit pointers. Worse, the actual argument is truncated. In a 32-bit compilation:

```c
void f(char _ptr64 *);
...
void * _ptr64 previouslyShortPtr; /* used to be 32-bit */
...
f((char *)previouslyShortPtr);
```

• The above generates a 2405 warning.
  - No consequences unless `previouslyShortPtr` is not a sign-extended 32-bit address.

• Recode as:

```c
f((char _ptr64 *)longPtr);
```
Mixed Data-model Programming -- Pointer Widening in NSK Compilers

- The native compilers have always sign-extended pointers and references when passing them to a function/procedure.

- This was done to permit functions to be converted to accept 64-bit pointers/references while remaining compatible with existing callers.

- 32-bit pointer arguments have always been sign-extended before being used since IPF is a 64-bit processor.

- Once a function has been converted to accept 64-bit pointers, such sign-extension within the called function is no longer necessary.
Q&A