Export Compliance Agreement

Export Requirements. You may not export or re-export products subject to this agreement in violation of any applicable laws or regulations.

Without limiting the generality of the foregoing, products subject to this agreement may not be exported, re-exported, otherwise transferred to or within (or to a national or resident of) countries under U.S. economic embargo and/or sanction including the following countries:

Cuba, Iran, North Korea, Sudan and Syria.

This list is subject to change.

In addition, products subject to this agreement may not be exported, re-exported, or otherwise transferred to persons or entities listed on the U.S. Department of Commerce Denied Persons List; U.S. Department of Commerce Entity List (15 CFR 744, Supplement 4); U.S. Treasury Department Designated/Blocked Nationals exclusion list; or U.S. State Department Debarred Parties List; or to parties directly or indirectly involved in the development or production of nuclear, chemical, or biological weapons, missiles, rocket systems, or unmanned air vehicles as specified in the U.S. Export Administration Regulations (15 CFR 744); or to parties directly or indirectly involved in the financing, commission or support of terrorist activities.

By accepting this agreement you confirm that you are not located in (or a national or resident of) any country under U.S. embargo or sanction; not identified on any U.S. Department of Commerce Denied Persons List, Entity List, US State Department Debarred Parties List or Treasury Department Designated Nationals exclusion list; not directly or indirectly involved in the development or production of nuclear, chemical, biological weapons, missiles, rocket systems, or unmanned air vehicles as specified in the U.S. Export Administration Regulations (15 CFR 744), and not directly or indirectly involved in the financing, commission or support of terrorist activities.

Printed in USA

Quick HP Integrity Application Migration
Student guide
October 2011
UC414S D.00
Contents

Course Overview

Course Objectives .................................................................................................. Overview-2
Prerequisites ......................................................................................................... Overview-3
Contents .............................................................................................................. Overview-4

Module 1 - TNS/E Architecture Overview

Objectives .............................................................................................................. 1-2
HP NonStop System Platforms ............................................................................. 1-3
TNS/E Hardware Architectures ............................................................................ 1-5
NonStop Advanced Architecture (NSAA) ......................................................... 1-6
NSAA – Single Logical Processor ..................................................................... 1-7
DMR v TMR ......................................................................................................... 1-8
Terminology for NonStop Advanced Architecture ........................................... 1-9
Dual Modular Redundancy (DMR) — Four Logical Processors/Two Blade Elements 1-10
Triple Modular Redundancy (TMR) — Four Logical Processors/Three Blade Elements 1-11
NSAA Advantages and Features ...................................................................... 1-12
NSMA – Single Logical Processor .................................................................... 1-13
NonStop Multi-core Architecture (NSMA) ....................................................... 1-14
NSMA Terminology .......................................................................................... 1-15
NonStop Operating System for NSMA ............................................................ 1-16
NSVA – Single Logical Processor .................................................................... 1-17
NSVA Features .................................................................................................. 1-19
Integrity NonStop NSVA and NSAA Systems ............................................... 1-20
Integrity NonStop NSMA Systems .................................................................. 1-21
Any NonStop Server Is a Collection of Logical Processors .......................... 1-22
The NonStop Operating System ...................................................................... 1-23
Software Libraries ............................................................................................ 1-24
Dynamic Libraries ............................................................................................. 1-25
DLL Advantages ................................................................................................. 1-26
Native TNS/E Code Generation ...................................................................... 1-27
Execution Modes ............................................................................................... 1-29
TNS/E Process Execution Environment .......................................................... 1-31
Languages Supported ......................................................................................... 1-32
Changed Products and Features ...................................................................... 1-33
New Features and Products .............................................................................. 1-34
Discontinued Products and Features ............................................................... 1-35
Unchanged Products and Features —Partial Listing ...................................... 1-36
Migration Paths .................................................................................................. 1-37
Module 2 - Compilers

Objectives ................................................................. 2-2
Migrating Native Applications — Server-Hosted Development Tools ............ 2-3
TNS/E COBOL Compiler ............................................... 2-4
Obsolete Directives .................................................. 2-5
Changed Directives and Clauses .................................... 2-6
New Features .............................................................. 2-7
File Name Changes ..................................................... 2-8
Other Changes .......................................................... 2-9
Screen COBOL Compiler .............................................. 2-10
TNS/E C/C++ Compilers .............................................. 2-11
C89 Obsolete Pragmas ................................................ 2-12
C89 Changed Pragmas ................................................ 2-13
C89 New Pragmas ....................................................... 2-14
C89 New Features ....................................................... 2-16
C89 Math Functions Changes ...................................... 2-20
C89 Other Changes .................................................... 2-21
Documentation Change .............................................. 2-23
Using Conditional Compilation .................................... 2-24
C99 Compiler ............................................................. 2-25
TNS/E epTAL Compiler ............................................... 2-26
Obsolete Directives .................................................. 2-27
Changed Directives .................................................... 2-28
Other Changes .......................................................... 2-29
Function Changes ....................................................... 2-30
RETURN and RETURNSCC .......................................... 2-31
New epTAL Feature .................................................... 2-32
Compiler Optimization Levels ..................................... 2-33
Optimization level 0 .................................................... 2-34
Optimization level 1 .................................................... 2-35
Optimization level 2 .................................................... 2-36
New Guardian Procedure Calls .................................... 2-37
Changed Guardian Procedures .................................... 2-38
Java 6 .......................................................... 2-39

Module 3 - Linking and Accelerating Applications

Objectives ................................................................. 3-2
eld Basic Operation ................................................... 3-3
Linking a Multi-Module C program ................................ 3-4
Linking a Multi-Module C++ Program ............................. 3-5
Linking a Multi-Module COBOL Program ........................ 3-6
eld and Symbols ....................................................... 3-7
Unresolved References — Default Behavior ..................... 3-8
Unresolved References — Optional Behavior .................... 3-9
Unresolved References — Additional Option ................................................. 3-11
Behavior Change – TNS Executable ......................................................... 3-13
Creating a DLL ......................................................................................... 3-14
Changing Object Attributes ...................................................................... 3-15
General Changes — Public Libraries (SRL—> DLL) .................................... 3-16
Registered DLLs ..................................................................................... 3-17
Examining an Object File ......................................................................... 3-19
enof Examples ........................................................................................ 3-20
Object Code Accelerator (OCA) ................................................................. 3-24
OCA Items to Watch for ......................................................................... 3-25
Object Code Accelerator — Example ...................................................... 3-26
Determining If Program Accelerated ......................................................... 3-28
TNSVU — Accelerated Object Examiner ................................................... 3-30
TNSVU Help ........................................................................................... 3-31
TNSVU Example — FILEINFO ................................................................. 3-33
TNSVU Example — PROCS .................................................................... 3-34
TNSVU Example — EXTERNALS ............................................................. 3-35
TNSVU Example — ICODE ..................................................................... 3-36
BINDER – Disabling Acceleration ............................................................... 3-37

Module 4 - Debuggers
Objectives ............................................................................................... 4-2
Available Debuggers ................................................................................ 4-3
Debugger Selection .................................................................................... 4-4
Visual Inspect 4.0 ..................................................................................... 4-5
Visual Inspect .......................................................................................... 4-6
Native Inspect .......................................................................................... 4-11
Native Inspect — Features ...................................................................... 4-12
Starting Native Inspect .......................................................................... 4-13
Native Inspect - Help ............................................................................... 4-14
Native Inspect – Breakpoints ................................................................... 4-15
Native Inspect – Execution Control ......................................................... 4-16
Commands at a Breakpoint ...................................................................... 4-18
Native Inspect – Data Display ................................................................. 4-19
Native Inspect – Variable Information Display ....................................... 4-21
Native Inspect – Data Modification ......................................................... 4-22
Native Inspect – Source Statements Display .......................................... 4-23
Native Inspect – Source Files .................................................................. 4-24
Native Inspect – Function Names (C) ....................................................... 4-25
Native Inspect – Function Names (pTAL) ............................................... 4-26
Native Inspect – Program Names (COBOL) ............................................ 4-27
Native Inspect – Registers Display ........................................................... 4-28
Native Inspect – Open File Information .................................................... 4-30
Native Inspect – Stack Commands ................................................................. 4-32
Native Inspect – Heap Commands ................................................................. 4-33
Native Inspect – Session Logging ................................................................. 4-36
Miscellaneous Commands ......................................................................... 4-38
Snapshot/Saveabend files ........................................................................ 4-42
Native Inspect — TNS Process ................................................................... 4-43

Module 5 - Workstation Tools

Objectives ................................................................................................. 5-2
PC-Hosted Development Tools .................................................................... 5-3
ETK Changes ............................................................................................... 5-4
New Project Type ........................................................................................ 5-5
Application Wizard .................................................................................... 5-6
New Project ................................................................................................. 5-9
Project Properties ....................................................................................... 5-10
Multiple Platforms ...................................................................................... 5-16
Migrating ETK Projects ............................................................................. 5-17
Eclipse — Open Source Development Platform ......................................... 5-20
NonStop Development Environment for Eclipse (NSDEE) ......................... 5-21
Key Features of NSDEE ............................................................................. 5-22
Support for NonStop Server – Project Types ............................................ 5-24
Support for Native Language Programming .............................................. 5-25
Supplied Skeleton Programs ....................................................................... 5-26
Configurable Compiler Options ................................................................... 5-27
Integrated Help System ............................................................................. 5-28
SQL/MP and SQL/MX Support ................................................................... 5-29
Integrated Debugger ................................................................................... 5-30
ETK to NSDEE Migration Utility ............................................................... 5-31
An ETK Solution ......................................................................................... 5-32
Running the Migration Utility ................................................................... 5-33
The Migrated Solution ................................................................................. 5-34
Configuration Properties ........................................................................... 5-35
Project Properties ....................................................................................... 5-36
The Migrated Application .......................................................................... 5-37

Module 6 - Code Profiling Utilities

Objectives ................................................................................................. 6-2
Code Coverage ........................................................................................... 6-3
Profile Guided Optimization (PGO) ............................................................ 6-4
Applicability ............................................................................................... 6-5
Use ............................................................................................................. 6-6
Code Coverage Process Flow ..................................................................... 6-7
Installing the Code Cover Utility ............................................................... 6-8
Building an Application for Code Coverage ............................................. 6-9
Running the Application .......................................................................... 6-10
Module 7 - Application Migration

Objectives ...................................................... 7-2
NonStop S-Series to Integrity NonStop Migration Paths ........................................ 7-3
NonStop S-Series TNS Mode to Integrity NonStop TNS Mode ............................ 7-4
TNS/R Native Mode to TNS/E Native Mode .................................................... 7-5
Preparing Programs for Migration .................................................................. 7-6
Maintaining Common Source Code ................................................................. 7-7
Alignment of Data .......................................................................................... 7-8
Adjusting for Increased DCT Limits ................................................................. 7-9
TNS/E pTAL Compiler ...................................................................................... 7-11
pTAL Compiler Directives .............................................................................. 7-12
pTAL Functions and Procedures ...................................................................... 7-14
Other pTAL Migration Tasks .......................................................................... 7-15
TNS/E Native C/C++ Compilers Standard and Invocation ................................ 7-17
Standard C++ Run-Time Libraries ................................................................. 7-18
Native C/C++ Library Migration ...................................................................... 7-19
Changed Pragmas and Flags .......................................................................... 7-21
Neutral C++ Dialect ....................................................................................... 7-24
Other C/C++ Migration Tasks ......................................................................... 7-25
TNS/E Native COBOL Compiler ..................................................................... 7-29
COBOL Compiler Directives ......................................................................... 7-30
Other COBOL Migration Tasks ...................................................................... 7-32
User Library Migration Tasks .......................................................................... 7-33
Floating-point Consistency Checks ................................................................. 7-36
Guardian API Migration Tasks ........................................................................ 7-37
Third-Party Tools ........................................................................................... 7-40
HP Evolution Services .................................................................................... 7-41

Appendix – J06.12
QUICK HP INTEGRITY APPLICATION MIGRATION
COURSE OBJECTIVES

- Introduce the major features of the HP NonStop Advanced Architecture, the NonStop Multi-core Architecture, and the NonStop Value Architecture
- Describe the TNS/E process execution environment
- Describe the TNS/E native compilers
- Describe the debugging tools available
- Describe the Code Profiling Utilities
- Describe some new features available
- Describe changes to the Enterprise Toolkit (ETK)
- Describe the Enterprise Plugins for Eclipse (EPE)
- Describe the steps required to migrate applications
PREREQUISITES

Students should have experience in the following areas:

• Guardian environment
• Open System Services (OSS) environment
• Transaction Application Language (TAL)
• Portable Transaction Application Language (ptAL)
• C/C++
• COBOL
• Inspect/Visual Inspect
• Link editors — Binder, nld
• Pathway
• ETK
CONTENTS

- Module 1 — TNS/E Architecture Overview
- Module 2 — Compilers
- Module 3 — Linking and Accelerating Applications
- Module 4 — Debuggers
- Module 5 — Workstation Tools
- Module 6 — Code Profiling Utilities
- Module 7 — Application Migration
OBJECTIVES

- Describe the major features of the HP NonStop Advanced Architecture, including the Dual Modular Redundancy configuration and the Triple Modular Redundancy configuration
- Describe the major features of the HP NonStop Multi-core Architecture
- Describe the major features of the HP NonStop Value Architecture
- Discuss the modes of process execution
- Describe the TNS/E process execution environment
- List some of the products and features that are no longer available
- List some of the changed products and features
- List some of the products and features that have not changed
- Introduce the options available for executing applications

Reference Manuals:
HP Integrity NonStop NS1200 Planning Guide
HP Integrity NonStop NS1400-Series Planning Guide
HP Integrity NonStop NS1600-Series Planning Guide
HP Integrity NonStop BladeSystem Planning Guide
A hardware platform cannot function without the support of an operating system, and vice versa. The H-Series and J-Series in the title of this course refer to the threads of the NonStop Operating System used by the Integrity NonStop and Integrity NonStop BladeSystem models with single-core based processors and multi-core based processors, respectively. The term TNS/E is used in this course to refer to both the Integrity NonStop and the Integrity NonStop BladeSystem platforms.
On the left is the standard NonStop S-series server dual lockstep processors architecture. This is superseded by one of the new architectures. The different designs for the Itanium processor–based servers offer different benefits for customers.
The different architectures for the TNS/E family of systems are shown above. Each one offers something different in terms of performance, availability, and cost.

These are the only system architectures that combine hardware fault tolerance with software fault tolerance. If you only have hardware fault tolerance, a majority of the reasons systems fail are not addressed, because software causes most failures.

The NonStop fundamentals are based on a unique combination of hardware and software fault tolerance. The loosely coupled, process and message–based architecture is a key underpinning.
The NonStop Advanced Architecture (NSAA) was the first architecture introduced with the Integrity NonStop family of systems. NSAA’s design objective was to provide the highest levels of availability and scalability at a significantly improved price/performance – leveraging industry standards with modular and reusable components.
A logical processor in the NonStop Advanced Architecture uses microprocessors resident on different boards.

The two or three microprocessors that constitute one logical processor run the same instruction stream for a single process. The microprocessors in a logical processor normally can be slightly out of sync, but will be resynchronized by the LSU whenever an interprocess communication or I/O operation causes a ServerNet packet to be sent out.

The LSU detects differences in the results presented by the microprocessors, and also front ends the ServerNet interface to perform I/O.
The TMR configuration significantly improves the hardware availability of the Integrity NonStop server,
A logical processor is a processor in the original HP NonStop operating system sense. A processing element is a single microprocessor. For the Itanium systems it's one of the two or four microprocessors in a NonStop Blade Element, which is the whole board with the processing elements, memory, and a memory controller.

The logical synchronization unit is a unit that allows voting between the processing elements that compose the logical processor, and also front ends the ServerNet interface to perform I/O. The NonStop Blade Complex is the entire set of these. In the examples shown, the four logical processors are composed of two or three NonStop Blade Elements plus their associated LSUs.
The basic module building block is called a NonStop Blade Element. This shows the processor connections to the LSUs from each of the four processors. The processor boards reside in the NonStop Blade modules.

The LSUs represent added value from HP. LSUs communicate with ServerNet technology. The LSU is a self-checking module; if the LSU fails, then its logical processor is considered to be “down.”

Each logical processor — what the operating system views as a single processor — is composed of one microprocessor from each of two NonStop Blade Elements.

A logical processor in the NonStop Advanced Architecture uses microprocessors resident on two different boards.

In this configuration, two four-way SMP Itanium processor boards are used to provide a four-processor server.

The pair of microprocessors that constitute one logical processor run the same instruction stream. The microprocessors in a logical processor normally can be slightly out of sync, but will be resynchronized by the LSU whenever an interprocess communication or I/O operation causes a ServerNet packet to be sent out.
This slide shows the connections to the logical synchronization units from each of the four microprocessors. The processor boards reside in the NonStop Blade element modules.

A TMR logical processor in the NonStop Advanced Architecture uses microprocessors resident on three different boards.

The three microprocessors that constitute one logical processor run the same instruction stream. The microprocessors in a logical processor normally can be slightly out of sync, but will be resynchronized by the LSU whenever an interprocess communication or I/O operation causes a ServerNet packet to be sent out.
All of the NonStop system advantages for customers remain unchanged. This is the only system architecture that combines hardware fault tolerance with software fault tolerance. If you only have hardware fault tolerance, a majority of the reasons systems fail are not addressed, because software causes most failures.

This architecture provides transparent recovery from any failures previously recovered from. In fact, it actually increases the overall availability of the system, because, with the NonStop Advanced Architecture, hardware faults are less likely to result in processor failures.

The NonStop fundamentals are based on a unique combination of hardware and software fault tolerance. The loosely coupled, process and message-based architecture is a key underpinning.

This is the same process-pair technology that NonStop systems have always relied upon for transparent takeover after a failure. This technology results in no application downtime, no lost messages, and no lost connections.

Previously offered features such as a single system image (SSI), fault-tolerant parallel database, and scalable application transaction processing monitors with persistent processes remain the same.

There are some changes in the NonStop server, where the software touches the hardware. Everything else remains the same.

The NonStop Advanced Architecture represents a change in approach to physical implementation, not a change to any previously announced Itanium software migration strategies or procedures.
In NSMA, a logical processor is made up of a multi-core Itanium processor with shared memory. The NonStop OS assigns a process to one of the cores (Instruction Processing Unit (IPU)) on the processor. The user has no control over which IPU a process runs in.

NSMA is the only architecture in which more than one process will be executing at a time in a single logical processor (cpu). The Instruction Streams shown above are from different processes.

This slide shows a quad-core microprocessor; some NSMA models use a dual-core microprocessor.
Any IPU can initiate I/O but only the Monarch (IPU 0) receives I/O interrupts. However, Interrupt Processes can run elsewhere.
NSMA TERMINOLOGY

- IPU — “Instruction Processing Unit”, a core
- Monarch (IPU) — the initial IPU upon power on (Intel calls it the boot processor)
- CPU — logical processor, a set of cores sharing the same memory. The traditional NS Logical CPU extended to be a multiprocessor
  - One X and Y 3NNet interface per CPU
  - All IPU’s in CPU share same memory map (except small per-IPU anchor area for low-level software)
- n-Way — traditional indication of IPU’s in a multiprocessor: 4-way means 4 IPU’s per CPU
- Process Scheduler: the new NSK subsystem that distributes and redistributes processes to IPU’s
The Process Scheduler (PS) picks which IPU a process is to run in. Each IPU has a distinct ready list. The PS is composed of sampling logic to see how the system runs and decision logic to rearrange the contents of the ready lists.

The PS uses the recent past history to guide it towards optimizing use of all of the IPUs. A process may have cache affinity with the IPU in which it has been running:
- Instructions currently in its instruction cache
- Data currently in its data cache

If a process that has data or instructions in one IPU’s cache is moved to the other IPU, that data or block of instructions has to be pulled over to the other IPU’s cache as it is accessed.

The Dispatcher is the logic that unloads the current process and loads the process at the top of the ready list.

Prior to the NonStop Multicore Architecture the NonStop OS only needed a dispatcher since the ready list was priority ordered and there was only one IPU to run processes.

Individual IPUs are externalized only selectively:
- Measure’s CPU entity shows IPU level busy/idle
- Measure’s Process entity shows:
  - The current IPU number
  - The number of IPU switches in the interval
  - If a switch was made the previous (or last if there is more than one) IPU number is shown
- PEEK shows the number of IPUs in a CPU
- PROCESSOR_GETINFOLIST_ has an attribute that returns the number of IPUs in a given
In NSVA, a logical processor is made up of a single core Itanium microprocessor and its associated memory.
NSVA incorporates industry-standard, commercial hardware data integrity.
NSVA uses traditional NonStop principles:
  – Fault tolerant immediate "fail-fast" design paradigm.
  – Error detection on all data paths, memory, and disk.
  – Multiple paths to all data and communications.
  – Hot-pluggable components.
    • Online repair.
    • Online addition/upgrade.

The architecture is designed to provide at least 99.999% availability and offers a lower-priced configuration option for critical business applications requiring less scalability and/or power.
NSVA FEATURES

- Industry-standard, commercial hardware data integrity
- Error detection on all data paths, memory, and disk.
- A lower-priced configuration option for critical business applications requiring less scalability and/or power.
The above chart and the one on the following page illustrate the differences between the available systems in performance, architecture, processor speeds and configurations. Note that all the systems can be connected in a network by using either ServerNet or Expand over IP.

<table>
<thead>
<tr>
<th></th>
<th>High End (NS16200)</th>
<th>Mid-Range (NS14200)</th>
<th>Entry Level (NS1200)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSAA (DMR or TMR)</td>
<td>NSAA (DMR or TMR)</td>
<td>NSVA</td>
<td></td>
</tr>
<tr>
<td>1.6 GHz/12MB cycle</td>
<td>1.4 GHz/12MB cycle</td>
<td>1.6 GHz/12MB cycle</td>
<td></td>
</tr>
<tr>
<td>4, 8, 16 GB memory per logical cpu</td>
<td>4, 8 GB memory per logical cpu</td>
<td>4, 8 GB memory per logical cpu</td>
<td></td>
</tr>
<tr>
<td>2:16 processors/node</td>
<td>1:8 processors/node</td>
<td>2:8 processors/node</td>
<td></td>
</tr>
<tr>
<td>Metro Cluster (ServerNet Cluster)</td>
<td>Metro Cluster (ServerNet Cluster)</td>
<td>Not ServerNet clustered, Expand over IP supported</td>
<td></td>
</tr>
<tr>
<td>1.0 performance</td>
<td>0.75</td>
<td>0.65</td>
<td></td>
</tr>
<tr>
<td>H-series operating system</td>
<td>H-series operating system</td>
<td>H-series operating system</td>
<td></td>
</tr>
</tbody>
</table>
## INTEGRITY NONSTOP NSMA SYSTEMS

<table>
<thead>
<tr>
<th>NonStop BladeSystem (NB50000c)</th>
<th>NonStop BladeSystem (NB54000c)</th>
<th>Entry Level (NS2000)</th>
<th>Entry Level (NS22000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSMA</td>
<td>NSMA</td>
<td>NSMA</td>
<td>NSMA</td>
</tr>
<tr>
<td>1.6GHz dual-core/18MB cache</td>
<td>1.6GHz quad-core/20MB cache</td>
<td>1.4GHz dual-core/12MB cache</td>
<td>1.6GHz quad-core/12MB cache</td>
</tr>
<tr>
<td>8, 16, 24, 32, 48 GB memory per logical cpu</td>
<td>16, 24, 32, 48 GB memory per logical cpu</td>
<td>8, 16 GB memory per logical cpu</td>
<td>8, 16, 32 GB memory per logical cpu</td>
</tr>
<tr>
<td>2/16 processors/node</td>
<td>2/16 processors/node</td>
<td>2-4 processors/node</td>
<td>2/4 processors/node</td>
</tr>
<tr>
<td>Metro Cluster (ServerNet Clustered)</td>
<td>Metro Cluster (ServerNet Clustered)</td>
<td>Not ServerNet clustered; Expand over IP supported</td>
<td>Not ServerNet clustered; Expand over IP supported</td>
</tr>
<tr>
<td>1.0 performance</td>
<td>1.8</td>
<td>0.75</td>
<td>0.95</td>
</tr>
</tbody>
</table>

J-series operating system | J-series operating system | J-series operating system | J-series operating system |
As shown above, all NonStop servers feature a cluster architecture. A single server can consist of 2 to 16 logical processors, each of which incorporates 1, 2, or 3 microprocessors. Each logical processor is capable of independent operations and does useful work. Specifically, each has its own memory and copy of the NonStop Operating System.

NonStop servers can be interconnected to produce a massively parallel computing environment consisting of hundreds of processors (the theoretical maximum is actually 4080). The operating system is designed to blur boundaries between processors — that is, to make them appear as a single computing resource. It does this by facilitating rapid and efficient communication between processes running in the various processors. For this reason, the NonStop Operating System is often referred to as a message-based operating system. It provides a foundation for fault tolerance and effectively eliminates the contention that restricts the ability of shared-memory systems (symmetric multiprocessors) to grow.
All TNS/E systems run the HP NonStop Operating System – Mission Critical Edition:
- Transparent recovery from any failure
- Hardware fault tolerance
- Software fault tolerance
- Process pairs
- Message-based operating system
- Transaction support
- Distributed single system
- Fault-tolerant parallel database
- Application server TP monitors
Clients use functions or data from a library.
A library can also be the client of another library.
A linkable library is an object file that is the output of a compiler. It is linked with its client to create a single loadable object.
A loadable library is joined to its client at run time.
Static libraries are embedded in the load module and are loaded as part of the same logical segment as the referencing code. There is a separate copy of the library embedded in each distinct load module.
Archives are like suitcases containing link modules of code.
Dynamic libraries are loaded as a separate logical segment at run-time and one copy can be shared by many processes.

Dynamic Link Libraries (DLLs) have been standard on UNIX and Microsoft Windows systems for a long time. The UNIX and Windows models for DLLs differ somewhat, and the HP NonStop DLL follows the UNIX model:

A public DLL performs the same role as Public SRLs on NonStop G-series systems. “Public” indicates they are preloaded and registered DLLs. Public DLLs are mapped into the process only if they were explicitly requested or required by an explicitly requested library at link-edit time or by means of dynamic load procedures.

For more information on DLLs, see the DLL Programmer’s Guide for TNS/E Systems.
DLL ADVANTAGES

- DLLs offer many advantages:
  - Programs can use many DLLs
  - Different processes can run with different versions of a DLL
    - Even running same program file
  - Symbol resolution does not require writing to disk
  - Additional libraries can be loaded after the program starts
  - DLLs make porting software built with DLLs much easier
The TNS/E compiler output consists of Itanium instructions designed to be executed on an Itanium instruction processing unit (IPU).

The decision to create TNS or TNS/E programs is up to a user. When running on a TNS/E processor, a TNS/E program is said to be “native” to that processor.

TNS/E code cannot be built or run on a TNS/R platform. (TNS/R code can be built, but not run, on a TNS/E platform.)

TNS/E (and TNS/R) code can be built on a PC through the NonStop Development Environment for Eclipse (NSDEE), Enterprise Tool Kit (ETK), or command line mode tools. This is the preferred method.

TNS code is native to a TNS processor and non-native to the TNS/E processor.
The process execution modes available on NonStop H-Series and J-Series systems are similar to those on NonStop S-series systems:

– TNS interpreted mode
  • Programs generated by TNS compilers.
  • Programs use TNS process and memory architecture.
  • Programs consist of TNS object code.
  • Programs consist of TNS instructions. Millicode routines implement TNS instructions on Itanium processors.
– TNS accelerated mode
  • Programs generated by TNS compilers and processed by the Object Code Accelerator (OCA).
  • Programs use TNS process and memory architecture.
  • Programs consist of TNS object code and accelerated object code.
  • Programs consist of TNS instructions and equivalent OCA generated Itanium instructions. Programs execute Itanium instructions directly on Itanium processors. Programs transition to TNS mode when OCA was unable to generate equivalent Itanium instructions.
– Native mode
  • Programs generated by native compilers.
  • Programs use native process and memory architecture.
  • Programs consist of native object code.
  • Programs consist of Itanium instructions. Programs execute Itanium instructions directly on Itanium processors.

TNS interpreted object code, accelerated object code, and native object code cannot be mixed in one program file. A native program can contain only native object code.

There is no support on TNS/E systems for RISC object code; that is, for programs with file code 700.
The diagram above shows the per-process areas of the TNS/E environment.

When initially started, a TNS/E process begins execution of its main procedure in the Main Program space. The code space of a TNS/E process consists of one contiguous range up to 256 MB.

At link edit time, eld is given "-lib <dllname>" options to specify which DLLs (along with the DLLs they depend on) should be loaded when a load module is loaded as part of a process.

The process might have zero or more ordinary DLLs. The DLL names are specified at link edit time and those DLLs, and the DLLs they depend on, are loaded when the program starts. For legacy support, an additional DLL can be specified (either at link edit time or during the process launch call) as the user library.

Public DLLs are registered and preloaded. The Public Library registry is one of the places checked for a matching library name. Programs written in languages other than pTAL require a run-time library as part of the process execution environment. Run-time libraries are provided for these languages in the form of public DLLs. Additional application services are also provided as public DLLs.

All processes have the implicit DLL code spaces mapped. This is the equivalent of the system libraries (System Code and System Library) and millicode on previous platforms. For some applications, additional service code is available in preloaded registered public dynamic-link libraries (DLLs).

Additional libraries can be loaded during process execution under programmer control.

When a TNS/E procedure executes, a local data area is required. This local data area is allocated out of one of two TNS/E stack areas: the main stack or the priv stack. The latter is used when the process runs in privileged mode. The area used for the local data is referred to as a TNS/E stack frame.

Any additional data areas that might be required for the execution of the process are also established at process start time. These include the global areas and the process file segment (PFS). Each load module can have its own global area.
LANGUAGES SUPPORTED

- Native (TNS/E)
  - C/C++
  - PL/AL
  - COBOL
  - Java
- Non-native (TNS)
  - C/C++
  - TAL
  - COBOL/5
  - FORTRAN
- Interpreted
  - Screen COBOL
CHANGED PRODUCTS AND FEATURES

- **OS Builder**
  - SYSGEN.NK is replaced by a set of utilities — OS.Builder
  - Input is CONFAUX and CONPHETX1 (some changes needed)
  - OSIMAGE is now split into OS Fileset and OSIMAGE file
    - OS Fileset includes files for millcode and system processes
    - OSIMAGE file contains data structures

- **Measure**
  - New counters for NSMA
    - IPL.Busy-Time
    - IPL.Qtime
    - IPL.Dispitches

- **Pathway**
  - Default security on coldstart changed to “O” from “N”
The Code Coverage tool reports on which blocks and functions within a program have been executed. It is useful in a QA environment to verify that all code segments in a program have been tested.

With Profile Guided optimization, you use the results of the execution of a program on a representative amount of work as input to the compiler. The compiler uses that information to produce code optimized for that work.
This is a partial list of the discontinued products. For a complete list, refer to the H06.nn or J06.nn Release Compendium.
UNCHANGED PRODUCTS AND FEATURES — PARTIAL LISTING

- HP Tandem Advanced Command Language (TACL)
- HP NonStop SQL/MP
- HP NonStop SQL/MX Version 2.0
  - Version 3.0 available
- HP NonStop TS/MP
  - LINKMON must be manually started
  - User Conversion routines need to be recompiled
  - Screen COROLI supports context-sensitive statements:
    - DIAGNOSTIC, DIAGNOSTIC, DIAGNO bổ, DIAGNO end
- HP NonStop Transaction Management Facility (TMF)
MIGRATION PATH IS

- TNS to TNS
  - Accelerated or Interpreted
  - No program changes needed
- TNS/R to TNS/E
  - Recompile/link all programs:
    - Program changes may be needed
- TNS to TNS/E
  - Recompile/link all programs:
    - Program changes may be needed
OBJECTIVES

- Compile native TNS/E applications in COBOL in Guardian, OSS, and PC environments
- Compile native TNS/E applications in C/C++ in Guardian, OSS, and PC environments
- Compile native pTAL applications in Guardian and PC environments
- List obsolete directives
- Identify changed directives
- Describe additional changes
- Describe new features available
- Describe the use of toggles for conditional compilation
- Describe some of the new Guardian procedures

Reference Manuals:

C/C++ Programmer’s Guide
COBOL Manual for TNS/E Programs
pTAL Conversion Guide
pTAL Reference Manual
<table>
<thead>
<tr>
<th>Tool</th>
<th>TNS/R</th>
<th>TNS/E</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTAL compiler (Guardian)</td>
<td>PTAL</td>
<td>EPTAL</td>
</tr>
<tr>
<td>C compiler (Guardian)</td>
<td>NMC</td>
<td>CCMP</td>
</tr>
<tr>
<td>C++ compiler (Guardian)</td>
<td>NMCPLUS</td>
<td>CPCMP</td>
</tr>
<tr>
<td>COBOL compiler (Guardian)</td>
<td>NMCOBOL</td>
<td>ECOBOL</td>
</tr>
<tr>
<td>C compiler (OSS)</td>
<td>c89</td>
<td>c89</td>
</tr>
<tr>
<td>C++ compiler (OSS)</td>
<td>c89</td>
<td>c89</td>
</tr>
<tr>
<td>COBOL compiler (OSS)</td>
<td>nmcobol</td>
<td>ecobol</td>
</tr>
<tr>
<td>Non-PIC linker</td>
<td>rld</td>
<td>&lt;not-supported&gt;</td>
</tr>
<tr>
<td>PIC linker</td>
<td>ld</td>
<td>eld</td>
</tr>
<tr>
<td>Loader</td>
<td>rld</td>
<td>rld</td>
</tr>
<tr>
<td>Object File Utility</td>
<td>noff</td>
<td>enoff</td>
</tr>
<tr>
<td>Debuggers - command line</td>
<td>Debug, Inspect</td>
<td>Native Inspect (EINSPECT)</td>
</tr>
<tr>
<td>SQL compiler</td>
<td>SQLCOMP, MXCMP</td>
<td>SQLCOMP, MXCMP</td>
</tr>
<tr>
<td>Data Definition Language</td>
<td>DDL</td>
<td>DDL</td>
</tr>
</tbody>
</table>

This table lists the server-hosted development tools. In the left column the server-hosted development tools are listed. The middle column lists the names of the tools for TNS/R, and in the right column lists the names of the equivalents on TNS/E. For example, the pTAL compiler on TNS/R is invoked as pTAL, whereas on TNS/E, the compiler is invoked as EPTAL. There are similar changes for all the server-hosted development tools. Some tools, however, go by the same name. For example, the C/C++ compiler is still C89, but with a different pragma that indicates which will be the target platform.

TNS/R development tools are supported on TNS/E systems. You can compile and link TNS/R native programs on a TNS/E system, and you can debug TNS/R native snapshot files on a TNS/E system.
The TNS/E COBOL compilers are executed in the same way as the TNS/R COBOL compilers. The compiler name for the Guardian environment is ECOBOL; for the OSS environment, it is ecobol. The default object file name is RUNUNIT for the Guardian environment and a.out for the OSS environment.
The NON-SHARED directive was used to generate non-PIC code. All code on the TNS/E system is PIC. LARGEDATA was used to control the placement of data-items in storage. The LD directive was available to pass options to the ld linker. The NLD directive passed options to the nld linker. The linker for TNS/E programs is eld. The LESS-CODE directive was used to control the way in which initialization of some data areas was performed.
Compilers

CHANGED DIRECTIVES AND CLAUSES

- UL — Indicates code for a dynamic-link library (DLL).
- SYNCHRONIZED
  - Applies to 01 and 77 level items.
  - May now align to a 8-byte boundary if item is 4 or more bytes long.
- OPTIMIZE — Level 2 provides more.
- RENAMES — of an 01 item now correctly gives error (use REDEFINES instead)

The UL directive was used to inform the compiler that the resulting object code would be in a native user library. It now indicates that the code will be placed in a DLL.

On previous NonStop systems the SYNCHRONIZED clause would force alignment of a data item on a 2-byte boundary. On TNS/E systems, alignment is to a 8-byte boundary if the data item is 4 bytes or larger.

The RENAMES clause has never been permitted on an 01 level item. However, the TNS/R COBOL compiler did not produce an error; the TNS/E COBOL compiler does produce an error.
NEW FEATURES

- Support for some COBOL-2002 standard features:
  • BASED
    - No storage allocated for item
  • ALLOCATE
    - Obtain dynamic memory
  • FREE
    - Release dynamic memory
  • New reserved words
    - Requires STANDARD 2002 directive
The names of the files containing the Guardian application program interface (API) declarations have changed for the TNS/E systems.
The new names are ECOBEX0, ECOBEX1, and ECOBEXT—from newest to oldest respectively.
OTHER CHANGES

- Floating point:
  - Used for certain intrinsic functions:
    - Trigonometric
    - Logarithmic
  - IEEE format now used
    - Better performance
    - Results difference
  - Output files marked NEUTRAL_FLOAT

- Maximum size of a group or independent item:
  - Now 128MB (was 127.5MB)
SCREEN COBOL COMPILER

- Optional use of Pathcp4
  - Context-sensitive support:
    - DIALOG-BEGIN
      Start a dialog with single server process
      Optionally sends data
    - DIALOG-SEND
      Continue dialog
    - DIALOG-ABORT
      Cancel dialog
    - DIALOG-END
      Perform Clean-up operations
      No data sent to server
  - Allows >32k bytes of data per transaction

- Server:
  - New system message: -121, PATHSEND DIALOG ABORT
The TNS/E C and C++ compilers are executed in the same way as the TNS/R compilers, but the names are different in the Guardian environment.

In the OSS environment, you still use the c89 front end; it selects whichever platform it is running on as the target platform.

The compilers will not overwrite existing code 100 files with the same name.
Compilers

**C89 OBSOLETE PRAGMAS**

- EXTERN_DATA — GP-relative addressing not supported.
- NON_SHARED — Ignored. All code is now position-independent code (PIC).
- LD — Error generated.
- NLD — Replaced by ELD.
- VERSION1 — Not supported; VERSION3 is the default (the MIGRATION_CHECK pragma can be used with the VERSION2 pragma to aid in converting to VERSION3).
- SRLxxxx — SRLs obsoleted by DLLs. TNS/R C/C++ provided a number of SRL-related pragmas. Since TNS/E does not support SRLs, these pragmas are obsolete and not supported by the TNS/E C/C++ compiler.

The EXTERN_DATA pragma specified whether external data references (object declared extern) could use GP-relative addressing.

You used the ld and nld pragmas to pass arguments to those utilities.

You used the VERSION1 directive to direct the TNS/R native C++ compiler to compile according to the D40 version or dialect of C++. It disabled all new features added from the D45 RVU.

C++ version 1 was the first Native C++ compiler supported on the NonStop platform. It represented an early version of the language and did not support many of the newer C++ features now taken for granted. For example, version 1 does not support IEEE floating point. The major feature of the runtime library is stream support. The addition of the Rogue Wave Tools.h++ Version 6.1 library provided significant additional functionality. There are both Guardian and OSS versions of these libraries. Version 1 does not support DLLs or DLL clients (position-independent code [PIC] programs).

As the C++ language evolved, the library specified by the standards committee grew significantly more rich. C++ version 2 provides many of the new features defined in the draft standard of 1996. This includes template support, most of the Standard Template Library, the ability to replace global new and delete, namespaces, and other features. C++ version 2 can also work with Rogue Wave’s Tools.h++ version 7.0. The Guardian and OSS functions were combined into each library.

C++ version 3 is the first NonStop implementation of the 1998 standard ratified by the ISO. The NonStop implementation includes full language and library support, with the exception of unsigned long long integer and the “export” keyword. C++ version 3 also supports all of the other NonStop features provided in C++ version 2. The Rogue Wave Tools.h++ libraries have not been ported to C++ version 3 and cannot be used with the version 3 library or language.

The VERSION3 standard libraries for C++ are a port of the Dinkum™ C++ libraries.
IEEE format floating point is the default for TNS/E programs; previously, Tandem format was the default.

C89 CHANGED PRAGMAS

- IEEE_FLOAT — is now the default
- TANDEM_FLOAT — is still available
- Linkfiles can be flagged as NEUTRAL_FLOAT
  · Usable in programs of either format
A new command line option is used to specify that a module is being compiled into a version-neutral C++ DLL. In the OSS environment and on PCs, the `c89` option is `-Wbuilding_neutral_library`. In the Guardian environment, the `CPPCOMP` option is `building_neutral_library`. When this option is specified, the compiler will issue an error if any exported or imported interfaces depend upon any types that have been marked with `#pragma model_dependent`. This option is used in conjunction with the `version2/version3` option. The `version2/version3` option specifies which version of the model dependent portions of the standard headers should be used.
Interaction of MAXALIGN with FIELDALIGN:
Both pragmas may be applied to class, struct, or union definitions. However, not all combinations are allowed. Some generate only a warning, while others will generate an error. FIELDALIGN does not apply to arrays, so there are no possible interactions between FIELDALIGN and MAXALIGN with respect to array definitions.
FIELDALIGN shared2 and cshared2 structs are designed so that no component has alignment greater than 2 bytes regardless of the component type’s natural alignment. The overall alignment of a shared2 or cshared2 struct is also never greater than 2 bytes. Because of this part of the definition of shared2 and cshared2, an error is issued if a pragma MAXALIGN is applied to a struct with shared2 or cshared2 FIELDALIGN. An error is issued for a component with MAXALIGN declared within a shared2 or cshared2 struct.
The FIELDALIGN shared8 struct is designed so that each component is aligned on its natural alignment. It also has the characteristic that the same definition should have the same layout on all platforms – TNS, TNS/R, and TNS/E. Another characteristic of the shared8 struct is any definition that cannot have the same representation on all three platforms is diagnosed with either a warning or an error. Because MAXALIGN is not supported on TNS or TNS/R, any application of MAXALIGN on a shared8 struct creates a struct whose definition cannot be guaranteed to be identical across all platforms. A warning is issued if pragma MAXALIGN is applied to a struct with shared8 FIELDALIGN. A warning is issued for a component with MAXALIGN declared within a shared8 struct.
C89 NEW FEATURES (1 OF 4)

- New data type
  - unsigned long long
    - TN5E C/C++ supports an unsigned 64 bit integer type, unsigned long long.
    - Range is 0 to 18446744073709551615.
    - The suffixes for unsigned long long literals are ull and ULL.
      unsigned long long var = 18446744073709551615ULL;
  - Supporting functions:
    - strtoll: char string to signed long long
    - strlno: char string to unsigned long long
    - wstrtol: wide-char string to signed long long
    - wstrtoUll: wide-char string to unsigned long long

- C99 support:
  - __func__ Identifier
  - Universal Character Names
    - Usable in identifiers etc. to designate characters that are not in the basic character set.
    - A universal character name has the form:
      '\u'\hexadecimal{nnn}
      where n is a hexadecimal digit.
C89 NEW FEATURES (2 OF 4)

- C99 support (continued):
  - Hexadecimal Floating Point Constants
    - Floating point constants can be expressed using hexadecimal digits.
      ```c
      float f = 0x3F800000;
      ```
  - Digraph Characters
    - Two-character sequences that represent a single character
  - Comments
    - The characters // can be used to introduce a comment. The comment begins with the // and terminates at the end of the line.
  - Implicit Function Declarations Not Allowed
  - Designated Initialization for Structures
    - Aggregate initialization is enhanced to allow designators that initialize the components of an array or struct.
      ```c
      typedef struct div_t {
        int quot;
        int rem;
      } div_t;
      div_t answer = {.quot = 2, .rem = -1};
      ```
    - The components of div_t.quot and rem are initialized to 2 and -1, respectively.
C89 NEW FEATURES (3 OF 4)

- C99 support (continued):
  - Nonconstant Aggregate Component Initializers
    - The initializer expression for an automatic composite variable can contain nonconstant subexpressions.
      
      ```c
      typedef struct whole {
        int part1;
        int part2;
      } whole;
      ```
      
      ```c
      void foo (int i, int j) {
        whole local_val = {i, j}, ...
      }
      ```
  
  - Mixed Statements and Declarations
    - Declarations can appear within a block.
  
  - New Block Scopes for Selection and Iteration Statements
    - All selection statements (if, if/else, and switch statements) and iteration statements (while, do/while, and for statements) and their associated substatements are considered to be blocks.
C89 NEW FEATURES (4 OF 4)

- C99 support (continued):
  - `intmax` for Preprocessor expressions
    - Preprocessor expressions are evaluated using `intmax_t` and `uintmax_t`.
    - `intmax_t` and `uintmax_t` are defined in `<stdint.h>` to be long long and unsigned long long, respectively.
  - Variadic Macros
    - Variadic function like macros with a variable number of arguments.
    - Example
      ```
      #define debug ( ) fprintf (stderr, __VA_ARGS__)
      
      debug ("Flag");
      // expands to: fprintf (stderr, "Flag");
      
      debug ("n = %d\n", n);
      // expands to: fprintf (stderr, "n = %d\n", n);
      ```
C89 MATHEMATICAL FUNCTIONS CHANGES

- IEEE_FLOAT changes only
- C/C++ math routines (pow, sqrt, cos[], sin[], ...)
- Not-a-number (NaN) return values changed
- All NaN bit patterns changed:
  - TNS/R Quiet Nan = TNS/E Signaling Nan
  - TNS/R Signaling Nan = TNS/E Quiet Nan
Compilers

C89 OTHER CHANGES (1 OF 2)

- CEXTDECS file:
  - Uses int data type for 32-bit values
  - Previously used long data type
  - Gives warning during compilation
  - Use:
    - The WARN/NOWARN pragma or flag to suppress the warning message
    - The int type in the calling module
    - The new __int32_t typedef
- New Exception handler architecture (C++)
  - Better performance

In the TNS/E environment, the CEXTDECS file uses the int data type for 32-bit values. This is a change from the TNS and TNS/R environments where CEXTDECS uses the long data type for 32-bit values. Due to this change, you may get a type mismatch warning message when recompiling native TNS/R programs for the TNS/E environment. This warning does not impact the program execution, but can be avoided by changing to coding practices that use:
- The WARN/NOWARN pragma or flag to suppress the warning message
- The int type in lieu of long in the calling module
- The __int32_t typedef introduced by the TNS/E version of CEXTDECS for 32-bit items

The C++ compiler has been modified to produce more efficient code in the area of exception handling; this should provide better performance.
C89 OTHER CHANGES (2 OF 2)

- Compiler order of evaluation
  - INS/t compiler evaluates in different order
  - Incorrect code that worked on INS/X may not work on INS/t

```c
char * ptr;
...
ptr = strpyp(ptr, "string") + strlen(ptr);
```
The OSS Library Calls Reference Manual and the Guardian Native C Library Calls Reference Manual now list the library to be referenced for each function.
USING CONDITIONAL-compilation

- Test macros are available to control conditional compilation:
  - _INS_R_TARGET
  - _INS_R_TARGET
    #ifdef _INS_R_TARGET
    /* TNS/R specific code goes here */
    #else
    /* TNS/R specific code goes here */
    #endif
  - _TANDEM_ARCH_
    - 0: TNS
    - 1: TNS/R
    - 2: TNS/E
    #if _TANDEM_ARCH_ = 2
    /* TNS/R specific code goes here */
    #else
    /* TNS or TNS/R specific code goes here */
    #endif

The macros shown above are automatically set by the compiler.
C99 COMPILER

- Full support is only provided for programs using IEEE floating point format.
  - Complex types and several new math functions and macros are not supported for Tandem floating point format.
  - All other c99 features are available when compiled using the tandem floating point format option.
- Support for embedded SQL/MX, but not SQL/MP is available with c99.
- Guardian:
  
  COOMP/ in source/c99
  
  OSS:
  
  c99 source.c

Refer to the C/C++ Programmer’s Guide for details about the c99 compiler and the library functions and macros that do not support Tandem floating point format.
The TNS/E pTAL compilers are executed in the same way as the TNS/R pTAL compilers. The compiler name for the Guardian environment is EPTAL; there is no compiler for the OSS environment.

```
TNS/E EPTAL COMPILER

- No source code changes are needed in most programs.
- Guardian environment:
  EPTAL/IN <src-file> [.OUT <list-file>] [options]/
  [.OBJ <obj-file>] [.directives,..]
- PC:
  eptal.exe
  Creates code 800 linkfile
```
**OBSCOLE DIRECTIVES**

- `?SAVEGLOBALS` — Gives error
- `?USEGLOBALS` — Gives error
- `?BEGINCOMPIILATION` — Ignored
- `?NOCALL_SHARED` — Ignored
- `?SRL` — Ignored
- `?NOGP_OK` — Ignored
- `?PUSHGP_OK` — Ignored
- `?POPGP_OK` — Ignored

SAVEGLOBALS saves all global data declarations in a file for use in subsequent compilations that specify the USEGLOBALS directive.

BEGINCOMPIILATION marks the point in the source file where compilation is to begin if the USEGLOBALS directive is in effect.

The GP_OK directive controls whether the pTAL compiler uses GP relative addressing.
**CHANGED DIRECTIVES**

- `?NOOVERFLOW_TRAPS` — Now the default
- `?OPTIMIZEFILE` — Some conditions now cause warnings:
  - File does not exist, cannot be opened, is not code 101
  - Some procedure appears multiple times
  - Procedure is unknown
  - Invalid optimize level
OTHER CHANGES

- INTERRUPT procedure attribute not supported
- AUTO and PLATFORM perform same alignment in eptal
This table lists the different function changes between TNS/R and TNS/E versions of pTAL. All of these functions, except $STACK_ALLOCATE, require privileged access.
RETURN AND RETURNSCC

- TNS/E behavior
  - The TN52/T pIAL compiler issues a warning whenever a pIAL procedure returns both a traditional function value and a condition code value.
  - These C and C++ prototypes are not guaranteed to work, and extracting only the traditional function value does not work.
  - Read the migration guide for details on how to migrate such a procedure.
NEW EPTAL FEATURE

- New attribute STRUCTALIGN (MAXALIGN) for 16-byte aligned data
- Certain NonStop H-series functions require data to be aligned on 16-byte boundaries. TNS/E pTAL requires such data to be declared in a structure template that provides this 16-byte alignment. STRUCTALIGN (MAXALIGN) causes data to be aligned on the maximum alignment defined for the platform, which is 16 bytes for the TNS/E platform.

```plaintext
BEGIN
  INT(16) I: --Located at byte-offset 0 as defined by --SHARED8
  FILLER 2;
  INT(32) J: --Located at byte-offset 4 as defined by --SHARED8
  FILLER 0; --Required by optal
END;
STRUCT A1(A): --The base of A1 is guaranteed to be aligned --on a 16-byte boundary
```

Like C/C++, there is a new epTAL feature on TNS/E, STRUCTALIGN for MAXALIGN, which forces 16-byte aligned data. Certain NonStop H-series functions require this data to be aligned on the 16-byte boundaries, and TNS/E pTAL requires such data to be declared in a structure template that provides this 16-byte alignment. The STRUCTALIGN (MAXALIGN) clause causes data to be aligned on the maximum alignment defined for the platform, which is a 16-byte boundary for the TNS/E platform. The code snippet indicates how you can obtain 16-byte alignment using the STRUCTALIGN (MAXALIGN) clause.
A general recommendation is that you should use level 0 when you are developing a program, use level 1 in production, and use level 2 for modules or programs that are not changed for a long time and for programs that are very CPU intensive.
OPTIMIZATION LEVEL 1

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Level</th>
<th>Last error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
#include <stdio.h>

test(char *argv, char *arg2)
{
    int ret;
    int x = 0;
    int y = 0;
    int z = 0;
    int a = 0;
    int b = 0;

    ret = 0;
    return ret;
}

main(int argc, char *argv[])
{
    int x = 0;
    int y = 0;
    int z = 0;
    int a = 0;
    int b = 0;

    ret = 0;
    return ret;
}
```
## OPTIMIZATION LEVEL 2

<table>
<thead>
<tr>
<th>Opt 2</th>
<th>Hold</th>
<th>Last held</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
int main(short ans, char * args[]) {
    int return = 0;
    printf("Hello Students!
");
    return return;
}
```
The PROCESS_DELAY_ procedure performs the same function as DELAY but allows the time to be specified in microseconds rather than hundredths of a second.

The TIMER_START_ and TIMER_STOP_ procedures perform similar functions to SIGNALTIMEOUT and CANCELTIMEOUT but with the time period expressed in microseconds and with longer fields (64-bit) for delivery through the system message (although both get truncated), and with a 32-bit identifier.

The new TS_xxxx procedures allow for the creation, comparison, and manipulation of 128-bit timestamps. A timestamp obtained by TS_UNIQUE_CREATE_ contains a system identifier to ensure uniqueness throughout an Expand network.
It is unlikely that any of these changes will require code changes.
NonStop Server for Java 6.0 is a certified implementation of the Java SE 6.0 Platform, and implements all Application Programming Interfaces (API) and tools that are defined for a “Headless” configuration of Java SE 6.0 Platform. To view a complete list of features and enhancements in Java SE 6.0 Platform, refer to:

http://java.sun.com/javase/6/webnotes/features.html

NonStop Server for Java 6.0 includes a new HotSpot compiler optimized for Java execution on the Intel Itanium processor architecture. The use of the new HotSpot compiler can improve performance of NonStop Java applications by as much as 30 percent over NonStop Server for Java 5.0, based on HP internal benchmark tests.

Unlike previous versions of NonStop Server for Java, NonStop Server for Java 6.0 does not block the JVM process when making OSS file I/O calls, but merely blocks the calling thread. This enables the NonStop Server for Java 6.0 JVM to run other threads while waiting for the OSS file I/O to complete, thus improving application throughput.

NonStop Server for Java 6.0 implements two HP specific Java runtime options—eprof and verbosegc—that enables collection of detailed data related to application profiling and JVM Garbage Collection (GC) activities. The -Xeprof option generates profile data for HPjmeter. The -Xeprof option controls profiling and collects method clock and CPU times, method call count, and call graph. The -Xverbosegc option prints out a detailed information about the spaces within the Java Heap before and after garbage collection. The analysis of profiling data and GC activity data can help you to fine-tune your application prior to production deployment.

NonStop Server for Java 6.0 implements HPjmeter “agent” software that collects profiling and GC activity data and forwards them to a HPjmeter console running on Windows or HP-UX server while the Java application is running on NonStop servers. This allows real-time analysis and visualization of your application profile and GC activity. The HPjmeter console can be downloaded at no cost from http://hp.com/go/hpjmeter
LINKING AND ACCELERATING APPLICATIONS

Module 3
UC414S D.00
OBJECTIVES

- Use eld to create executable applications
- Use enof to display attributes of an object file
- Use the Object Code Accelerator (OCA)
- Use the TNSVU program to examine an accelerated object

Reference Manuals:
Binder Manual
eld Manual
enof Manual
Object Code Accelerator Manual
rld Manual
TNSVU User's Guide
Native object files are in 64-bit Executable and Linking Format (ELF), a standard format used for UNIX object files, with HP extensions. The native object file format is the same in the Guardian and OSS environments and on the PC as part of NonStop Development Environment for Eclipse (NSDEE) or Enterprise Toolkit (ETK).
Note that you may need to refer to additional DLLs, depending on the functions invoked by the program.

In the OSS environment it is easier to use c89 to perform the linking.

When you specify the names of the DLLs you want to use, you can use the short form on the -l (or -lib) as shown in the examples above, or you can specify the complete DLL file name such as ZCREDLL. If you use the short form, then you must use lower case.
LINKING A MULTI-MODULE C++ PROGRAM

- Version 3
  
  `aid -o outfile
  lnmod1 lnmod2 &system.system.app1main
  -lappdxlib -lappdxlib`
Compilation and Linking in the OSS Environment:
$ ecobol -g -o main.o -c mainprog.cob
$ ecobol -g -o call.o -c callprog.cob
$ eld -o myexe call main prog -l cob
Or as a single step instead of three steps:
$ ecobol mainprog.cob callprog.cob -o myexe
ELD AND SYMBOLS

- Missing Symbols
  - Unlike nld, eld knows which procedures exist in system library
  - By default, eld considers it an error if you refer to a symbol and the symbol does not exist.
    - You can change this with the -unres_symbols option:
      - error, warn, ignore
  - By default, your program cannot run if it refers to a symbol that does not exist.
    - You can change this with the -set rlid_unresolved option:
      - error, warn, ignore
UNRESOLVED REFERENCES — DEFAULT BEHAVIOR

```
ld - o mvea maincoho -l cob
ld - TMS/B Native Moda Liner - 70688B01STEM - OBEDC2010
Copyright 1996-2011 Hewlett-Packard Development Company L.P.
This program may be distributed under the terms of the GNU General Public
License.

ld command line:
/venus.5rcsrn.srsvn.ld -o mvea maincoho -l cob

**** INFORMATIONAL MESSAGE **** [1019]:
Using DLL \VENUS.5RCRM.5r51001a,zxob.dll.

**** INFORMATIONAL MESSAGE **** [1539]:
Using the rimepin file \VENUS.5RCRM.CYGT.SIMPIN.

**** ERROR **** [1210]:
maincoho: In function 'CONCEPTS-REQ':
maincoho( test_CONCEPTS-REQ=0x1010): unresolved reference to GET-SECOND-
NUMBER.

No output file created.

1 error reported.
No warnings reported.
2 informational messages reported.
```
UNRESOLVED REFERENCES — OPTIONAL BEHAVIOR (1 OF 2)

```
ld command line:
\venus.$system.systx.\ld -c myexe maincobo -l cob -unres_symbols warn

**** INFORMATIONAL MESSAGE **** [1012]:
Using DLL \VENUS.$SYSTEM.sdll001a.cobd.dll.

**** INFORMATIONAL MESSAGE **** [1530]:
Using the zimpamp file \VENUS.$SYSTEM.SYSZ2.ZIMPIMP.

**** WARNING **** [1254]:
maincobo: In function `CONCEPTS-REQ':
maincobo(text_CONCEPTS-REQ=0x1012): unresolved reference to GET-SECOND-
NUMBER.

Output file: myexe (program file)
Output file timestamp: Jul 29 08:42:38 2011

No user input reported.
1 warning reported.
2 informational messages reported.
```
UNRESOLVED REFERENCES — OPTIONAL BEHAVIOR (2 OF 2)

21> run myexe

*** RLD ERROR ***: Unresolved Text Symbol GET-SECOND-NUMBER in file \VENUS\0FC1\TERRYS.MYDR

*ERROR* PROCESS_CREATE_STATUS: 74
UNRESOLVED REFERENCES — ADDITIONAL OPTION (1 OF 2)

2Do eld -v mxe涔 maincобo -1 coB -unres_symbols warn -set rld unresolved warn
eld - TMS/E Native Mode Linker - T060H501'AAM - UNRECG2010
Copyright 1996-2011 Hewlett-Packard Development Company, L.P.
This program may be distributed under the terms of the GNU General Public License.

eld command line:
\venus.$system system eld -v mxe涔 maincобo -1 coB -unres_symbols warn -set
erl_unresolved warn

**** INFORMATIONAL MESSAGE **** (1012):
Using DLL \VENUS.$SYSTEM.sd11001a.zcobdll.
**** INFORMATIONAL MESSAGE **** [1320]:
Using the simlpimp file \VENUS.$SYSTEM.SYSZ2.ZIMSPIM.
**** WARNING **** [1294]:
maincобo: in function 'CONCEPTS_REQ';
maincобo (text_CONCEPTS_REQ); unresolved reference to
GET-SECOND-RUNNER.

Output file: mainco (program files)
Output file timestamp: Jul 28 08:48:50 2011

no errors reported.
1 warning reported.
2 informational messages reported.
UNRESOLVED REFERENCES — ADDITIONAL OPTION (2 OF 2)

23> run myexe

*** RLD WARNING ***: Unresolved Text Symbol GMT-SECOND-NUMBER in file
\VENUS:SFU1:TERVOS.MYEXE.

Undefined variables:
YOU HAVE JUST STARTED THE PROCCES.

ANYTH FIRST NUMMAK (max. 3 digits) [or U to stop]:
?12
\VENUS:.$1:981:7390904 - "*** Run-time Error 003 ***"
\VENUS:.$1:981:7390904 - Instruction failure
\VENUS:.$1:981:7390904 - Procm ---
\VENUS:.$1:981:7390904 - UNRESOLVED_PROCEDURE_CALLED_ 0x2 (SLr)
\VENUS:.$1:981:7390904 - CONCEPTS-REQ 0x1020 (HCx)
ABENDED: 1,981
CPU time: 0:00:00.003
3: Premature process termination with fatal errors or diagnostics
24>
If there is a missing function at run time, a TNS executable program (code 100) will still produce an “Undefined Externals” message. However, if the call to that missing function is executed, the process will trap rather than go into debug mode.
When you create a DLL, you can specify an internal name (using the –dllname parameter) and an external name (using the –o parameter.) Usually, these will be the same but you can specify different names if you are creating a DLL on one system for use on another system.
You can use the eld program to change certain attributes of an existing code 800 file. The attributes that you can change include:

- pfssize
- systype
- libname
- highpin
- highrequestors
- oklosetype
- runnamed
- saveabend
- inspect
- heap_max
- mainstack_max
- space_guarantee
- floattype
- float_lib_override
- CPlusPlusDialect
All TNS/E libraries are packaged as DLLs. Shared run-time libraries (SRLs) are not supported on TNS/E platforms. The TNS/E libraries have a different name. You will have to change any build procedures that specifically refer to the libraries and replace them with the DLLs. The table gives the general libraries that most applications use, what their names are on the TNS/R system, and what are their corresponding names on TNS/E system.

<table>
<thead>
<tr>
<th>Library</th>
<th>TNS/R</th>
<th>TNS/E</th>
</tr>
</thead>
<tbody>
<tr>
<td>COBOL run-time library</td>
<td>ZCOB5RL</td>
<td>ZCOBDLL</td>
</tr>
<tr>
<td>Common Runtime Environment (CRE)</td>
<td>ZCRESRSL</td>
<td>ZCREDLL</td>
</tr>
<tr>
<td>C run-time library</td>
<td>ZCRTLSRL</td>
<td>ZCRT1DLL</td>
</tr>
<tr>
<td>C++ run-time library</td>
<td>ZCPLG5RL</td>
<td>not supported</td>
</tr>
<tr>
<td>VERSION 1</td>
<td>ZCPP2DLL</td>
<td></td>
</tr>
<tr>
<td>VERSION 2</td>
<td>ZCPPCDLL</td>
<td></td>
</tr>
<tr>
<td>Tools.H++ 7.0</td>
<td>ZTIH7DLL</td>
<td></td>
</tr>
<tr>
<td>VERSION 3 (ANSI/ISO Standard)</td>
<td>ZCPPCDLL</td>
<td></td>
</tr>
</tbody>
</table>

*Names of some DLLs may change. Refer to the latest application migration guide.*
REGISTERED DLLS (1 OF 2)

- Current sysnn subvolume has pointer to DLL subvolume:
  32> fileinfo &system.sysnn.zregptr

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>SYSNN</th>
<th>CODE</th>
<th>NPTR</th>
<th>LAST MODIFIED</th>
<th>OWNER</th>
<th>SET</th>
<th>^HSET</th>
<th>^HSET</th>
</tr>
</thead>
</table>

25> ftp copy &system.sysnn.zregptr
DLLS SUBVOL EDL001A;
1 RECORDS TRANSFERRED

- DLL subvolume has registration file:
  33> fileinfo &system.edl001a.zreg

<table>
<thead>
<tr>
<th>CODE</th>
<th>EOF</th>
<th>LAST MODIFIED</th>
<th>OWNER</th>
<th>BHEF</th>
<th>^HSET</th>
<th>^HSET</th>
</tr>
</thead>
<tbody>
<tr>
<td>zreg</td>
<td>180</td>
<td>7672 27MAY2011 9:46</td>
<td>255/255/255</td>
<td>HNNU</td>
<td>14</td>
<td>42</td>
</tr>
</tbody>
</table>
# REGISTERED DLLS (2 OF 2)

```plaintext
# File: public\#DLLs\registry\file

dll_count: 30

<table>
<thead>
<tr>
<th>dll file</th>
<th>link_timestamp</th>
<th>update_timestamp</th>
<th>mod_timestamp</th>
<th>export_digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>link_timestamp</td>
<td>2009-01-28 21:16:00.000000</td>
<td>2011-05-27 16:45:43.000000</td>
<td>2011-05-27 16:45:43.000000</td>
<td>81dc71bb 0c319706 0e824a6e 669b69c1</td>
</tr>
<tr>
<td>mod_timestamp</td>
<td>2011-05-27 16:45:43.000000</td>
<td>2011-05-27 16:45:43.000000</td>
<td>2011-05-27 16:45:43.000000</td>
<td>81dc71bb 0c319706 0e824a6e 669b69c1</td>
</tr>
<tr>
<td>export_digest</td>
<td>a047ca19 83e6a7c4 a1071dab 396b06dc</td>
<td>a047ca19 83e6a7c4 a1071dab 396b06dc</td>
<td>a047ca19 83e6a7c4 a1071dab 396b06dc</td>
<td>a047ca19 83e6a7c4 a1071dab 396b06dc</td>
</tr>
</tbody>
</table>
```

---

**Linking and Accelerating Applications**

---

3 – UC414S C.00
EXAMINING AN OBJECT FILE

- New tool — enoft
- Many commands identical to noft
  - listproc, listsrce, listcompilers, listattribute, listoptimize, listunresolved, dumpproc, and so forth
- Some commands changed
  - listinfo -> liblist
  - sat out -> log
ENFET EXAMPLES (1 OF 4)

```
enoft
TNG/R Native Object File Tool [T2823 - Mar 1 2010]
Copyright 2010 Hewlett-Packard Development Company, L.P.
enoft> file myexu
Object File:  myexu
Format Type:  ELF64-bit, Big_Endian, IA64, PIC_Program (loadfile)
enoft> listproc *

*********** List of Procedures

Unlinked Proc_Addr  Proc_Name
-------------------------------------------------------------------
 0 0x70000000  Request_address
 1 0x70000100  Concepts_REQ
 2 0x70000200  __Init__D_CONCEPTS-REQ
 3 0x70000300  INIT 1_CONCEPTS-REQ
 4 0x70000700  GET-SECOND-NUMBER
 5 0x70003300  __Init__D_GET-SECOND-NUMBER
 6 0x70003400  end_of_code
```
## ENOFT EXAMPLES (2 OF 4)

### enoft.h10

****** List of Common File Attributes

<table>
<thead>
<tr>
<th>name:</th>
<th>myexe</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Format:</td>
<td>ELF64-hex, BigEndian, EAT64</td>
</tr>
<tr>
<td>Type:</td>
<td>PIC_Program (loadfile)</td>
</tr>
<tr>
<td>Debugging Symbols:</td>
<td>Yes</td>
</tr>
<tr>
<td>Float-Point Type:</td>
<td>NEUTRAL_FLOAT</td>
</tr>
<tr>
<td>Plant override:</td>
<td>No</td>
</tr>
<tr>
<td>System Type:</td>
<td>Guardian</td>
</tr>
<tr>
<td>Creation Timestamp:</td>
<td>2011-07-29 09:02:15</td>
</tr>
<tr>
<td>Process Subtype:</td>
<td>0</td>
</tr>
<tr>
<td>Highrequesters:</td>
<td>Yes</td>
</tr>
<tr>
<td>Nomininal:</td>
<td>No</td>
</tr>
<tr>
<td>Right:</td>
<td>Yes</td>
</tr>
<tr>
<td>Slaveabend:</td>
<td>No</td>
</tr>
<tr>
<td>PRIV or CALLABLE Main:</td>
<td>No</td>
</tr>
<tr>
<td>CALLABLE Proc:</td>
<td>No</td>
</tr>
<tr>
<td>DEFAULT unwrapper:</td>
<td>Visual Inspect</td>
</tr>
<tr>
<td>Language and Dialect:</td>
<td>Amd64 Real Mode C</td>
</tr>
<tr>
<td>Instrumented File:</td>
<td>No</td>
</tr>
</tbody>
</table>
**ENOFT EXAMPLES (3 OF 4)**

Enoft le + detail

******* List of Source Files (Compilation Units) *******

Source: 0 \VENUS\SFCL\TERRIG\MAINCOB
Compiler: Cobol85
Description: HP CMS/E COBOL T0355H01_19JAN2011_08DEC2010_ABY + 71244H01_19JAN2011_U
SNOV2010_PRODUCT_AAP_CRRD_E_10_1
Compiled: 2011-07-29 09:01:42 (Timestamp at Compile site)

File: 1 \VENUS\SFCL\TERRIG\MAINCOB
Size: 1428

Source: 1 \VENUS\SFCL\TERRIG\CALLCOB
Compiler: Cobol85
Description: HP CMS/E COBOL T0355H01_19JAN2011_08DEC2010_ABY + 71244H01_19JAN2011_U
SNOV2010_PRODUCT_AAP_CRRD_E_10_1
Compiled: 2011-07-29 09:01:42 (Timestamp at Compile site)

File: 1 \VENUS\SFCL\TERRIG\CALLCOB
Time: 2009-06-30 13:36:03
Size: 1154
Linking and Accelerating Applications

ENOFT EXAMPLES (4 OF 4)

enofth ls -

******** List of Unresolved (undefined) Symbols

Symbol: Symbol_Name

- COBLIB_ACCEPT
- _SharedMilli_MVFB_FWD_INACT
- COBLIB_INITIALIZER
- COBLIB_ABEND
- COBLIB_DISPLAY
- COBLIB_INITIALIZATION_COMPLETE
- COBLIB_ERROR
- _SharedMilli_COA
- COBLIB_INIT_PROG
- COBLIB_STOP
- COBLIB_SEQ_ALL

Number of symbols matching scope: 11
OBJECT CODE ACCELERATOR (OCA)

- Operates on Guardian-based code 100 files
  - May already be TMS/K accelerated
- Adds Itanium Processor Family (IPF) instructions
- Adds calls to millicode routines
- Uses symbols region, if present
  - Produces more efficient code
  - Can strip after acceleration
- Controllable by Binder options
- Use SQLCOMP after acceleration
OCA ITEMS TO WATCH FOR

- Disabled overflow traps
  - Should test for overflow using 3OVRFLOW or UNSIZE
- Trap handlers
  - Register contents are not precise
- Pre-absolute addresses
  - Cannot be passed as parameters in UC
- References to system global data
  - Instruction Failure exception
- Odd-byte references
  - Various behaviors
- Single-word shift statements with dynamic shift count
  - Shift count > 31 gives undefined result
- May affect debugging ability
The OCA program does not support the TACL IN option; instead, you should use the OCA OBEY command.
DETERMINING IF PROGRAM ACCELERATED (1 OF 2)

97> fileinfo abjcom*

<table>
<thead>
<tr>
<th>OBJCOM</th>
<th>CODE</th>
<th>DOT</th>
<th>LAST MODIFIED</th>
<th>OWNER</th>
<th>MODE</th>
<th>TEST</th>
<th>SGR</th>
<th>GSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJCOM</td>
<td>100</td>
<td>15496 29JUL2011 9:24</td>
<td>34,26</td>
<td>ABCC</td>
<td>14</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJCOM</td>
<td>100</td>
<td>43008 29JUL2011 9:24</td>
<td>34,26</td>
<td>ABCC</td>
<td>56</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJCOM</td>
<td>100</td>
<td>30720 29JUL2011 9:24</td>
<td>34,26</td>
<td>ABCC</td>
<td>14</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJCOM</td>
<td>100</td>
<td>59332 29JUL2011 9:24</td>
<td>34,26</td>
<td>ABCC</td>
<td>56</td>
<td>56</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

98> vproc objcom

VENUS - T981/901 - (01 FEB 2009) SYSTEM \VENUS Date 29 JUL 2011, 09:21:19
Copyright 2009 Hewlett-Packard Development Company, L.P.

$FC1.TERRYG.OBJCOM

Binder timestamp: 29JUL2011 09:24:36
Version procedure: >> NO T9xxx PROC <<
Target CPU: UNSPECIFIED

Not accelerated
DETERMINING IF PROGRAM ACCELERATED (2 OF 2)

099 vproc objects
VPROC - F5617H01 - (01 FEB 2009) SYSTEM \VENUS Date 29 JUL 2011, 09:28:02
Copyright 2004 Hewlett-Packard Development Company, L.P.

SPC1: TERRY'S OBJECTS

Binder timestamp: 29JUL2011 09:24:26
Version procedure: >> NO TRAXX PROC <<
Target CPU: UNSPECIFIED
OCA timestamp: 29JUL2011 09:25:30
1003 vproc objects
VPROC - F5617H01 - (01 FEB 2009) SYSTEM \VENUS Date 29 JUL 2011, 09:28:20
Copyright 2004 Hewlett-Packard Development Company, L.P.

SPC1: TERRY'S OBJECTS

Binder timestamp: 29JUL2011 09:24:36
Version procedure: >> NO TRAXX PROC <<
Target CPU: UNSPECIFIED
EXEC timestamp: 29JUL2011 09:25:07
OCA timestamp: 29JUL2011 09:26:43

TNS/E accelerated
TNS/R accelerated
TNS/L accelerated

3 - 29

UC414S C.00
TNSVU — ACCELERATED OBJECT EXAMINER

Opened file talex

talex>
TNSVU HELP (1 OF 2)

talax> help

TNSVU Command Summary

Commands are not case sensitive; file and procedure names are.
More information is available using "HELP command-name".

LOG — Opens a text file in which TNSVU records all text
LOGAPPEND — Written to the screen (including typed commands).
ENDLOG — Closes a log file.
OPEN — Opens a type-100 TNS object file for viewing.
FILEINFO — Displays general information about the open file.
PROCINFO — Displays general information about a procedure.
PROCS — Displays the names of procedures in the object file. TNS addresses and lengths, TNS addresses and lengths.
LOCAL — Displays the names of local sllilcde routines, and their
Titanium addresses and lengths.
MULTIONS — Displays the names of emulation and shared sllilcde routines.
EXTERNALS — Displays the names of external (system or user) library routines.
IUCV — Displays Titanium and TNS instructions.
HISTORY — Displays previously-issued commands.
TNSVU HELP (2 OF 2)

! - Repeats a previously-issued command.
QUIT - Exits TNSVU.
TNSHEADER - Displays the TNS file header.
OCARHEADER - Displays the Itanium region header.
AXCELHEADER - Displays the Axcel region header.
DUMPMAPS - Displays Pmaps.
OPTIONS - Displays translation options.
TNStoPPF - Converts TNS address to Itanium.
LVTOUPS - Converts Itanium address to TNS.
XREF - Displays Pppu targets of XEP entities.
VERSION - Displays the OCRVersion field of the OCRHeader.
EXCEPTE - Displays the except points in the given procedure.
OBJEY - Reads TNSVU commands from the given text file.
UNIT - Overrides limit on # of TNS instrs printed by TNSVU procsname.
COMPARE - Compares TPF node from the current file with the given file.
LINESUMTOTPS - Converts a source line number to a TNS code location.
PC - Fix command.
TNSPOOLIDNUM - Converts a TNS code location to a source line number.
SOURCE - Display source from files in the given subvol.
SOURCE OFF - Turn off source code display.
SOURCE ON - Turn on source code display (on by default).


```
talex> fileinfo

Examining file talex
Occupies code segment 0 through 0 in User Code space
0x1d1 Itanium bundles
Translated on Jul 29 2011 09:38:32
  using an OCA built on Dec 3 2010 13:11:51

Accelerator region: 0x0 for 0x0 bytes
Itanium region: 0x4000 for 0x3146 bytes
Symbolic region: 0x2000 for 0x1b1a bytes
Binder region: 0x2c00 for 0x108c bytes
```
TNSVU EXAMPLE — PROCS

Procedures:

DISPLAY_INITIAL_SEQ_MESSAGE

CSEG O PED# 03 TNS Base: 08 TNS Size: 071 TNS Entry: 08
Itanium code: 0x70420480 Itanium bundles: 0x3d Itanium entry: 0x70420480
Implemented in TAL in source file \VENUS\$FCI\TERRVU\MAINTAL
line 36.0 to 44.0

GET_SECOND_NUMBER

CSEG O PED# 04 TNS Base: 0345 TNS Size: 0154 TNS Entry: 0345
Itanium code: 0x704201f0 Itanium bundles: 0x72 Itanium entry: 0x704201f0
Implemented in TAL in source file \VENUS\$FCI\TERRVU\CALLTAL
line 31.0 to 39.0

MAIN.PROC

CSEG O PED# 03 TNS Base: 076 TNS Size: 0247 TNS Entry: 076
Itanium code: 0x70420050 Itanium bundles: 0xda Itanium entry: 0x70420050
Implemented in TAL in source file \VENUS\$FCI\TERRVU\MAINTAL
line 46.0 to 53.0
TNSVU EXTERNAL — EXTERNALS

talex> externals

Library procedures external to this code space:

WRITEREAD
WRITE
STOP
PROCESS_GETINFO
PROCESS_DEBUG
NUMOUT
NUMIN
INITIALIZER
FILE_OPEN
FILE_GETINFO
FILE_CLOSE
Linking and Accelerating Applications

TNSVU EXAMPLE — ICODE

talex icode DISPLAY_INITIAL_REQ_MESSAGE
Procedure DISPLAY_INITIAL_REQ_MESSAGE (from source file
\VENUS \FC \ TERRYG MAINTAL
Itanium virtual address 0x70420480 for 0x3d Itanium bundles
CSEG U THE offset 0b for U/V THE instructions
...
37:0            write_count := 0p '1' '0' '0' '0';
016:       0x70420570
    LDR   G0106,1
    LDR   G0105,1
    LDR   r45 = [r35]
    SUB   r45 = r45, r46
    ADDS  r46 = 01ah, r0
    NOP.1 00h
    ...
    NOP.0 00h
    EXTZ  r45 = r43
    ___

Binder commands allow you to control whether acceleration is permissible on a code 100 executable. Binder commands also allow you to remove the accelerated region from an executable.
DEBUGGERS

Module 4
UC414S D.00

© Copyright 2011 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice.
OBJECTIVES

- Describe the debuggers available for the TNS/E environment
- Introduce the features and commands of Native Inspect
- Describe the changes to Visual Inspect

Reference Manuals:
Native Inspect Manual
Native Inspect Quick Reference Card
Migrating from Inspect to Native Inspect
Visual Inspect is the preferred debugger for all application program debugging. If Visual Inspect is not available (or if the Visual Inspect client is not running), then, for TNS/E processes, Native Inspect is used instead.

You can use the Inspect debugger for Screen COBOL programs and for TNS programs. You can also use it to examine saveabend (snapshot) files from TNS processes.

The debug debugger is not available for use on TNS/E systems.

The Eclipse/NSDEE debugger is not covered in this course.
DEBUGGER SELECTION

TNS Process

INSPECT ON

Visual Inspect

Inspect

Native Inspect

INSPECT OFF

TNS/E Process

INSPECT ON

Visual Inspect

VISUAL INSPECT UNAVAILABLE

Native Inspect

INSPECT OFF
The Visual Inspect client has been changed to support TNS/E processes. When logging on to Visual Inspect, you can choose to encrypt the username and password using 3DES-based encryption to comply with the Payment Card Industry (PCI) data security standard. Other changes have been made to it to provide additional capability. One function that is not available for TNS/E applications is the Call Hierarchy.
The two new buttons on the Program Control View toolbar are Step Instruction and Step In Instruction.

For a native process (TNS/R or TNS/E) the Step Instruction executes a single native instruction. For a TNS process it will execute a single TNS instruction, unless the program is in accelerated mode in which case the button is disabled.

The Step In Instruction performs the same function as the Step Instruction unless the instruction in the native process is a call. In that case execution continues to the start of the called scope where the program stops in a Hold state.

The Execution Mode Indicator shows one of the following:

- Optimize: The optimization level of the program (0, 1, or 2). Optimize 2 is shown in red text and indicates that full symbolic debugging is not supported for the program.
- Interpreted: The program is a TNS program currently executing in interpreted mode.
- Accelerated: The program is a TNS program currently executing in accelerated mode.
- Type Unknown: The optimization level of the program is unknown, and the program is not interpreted or accelerated. The program might have been built with an unsupported compiler, and full symbolic debugging is not available. This indicator is shown in red text.
The Switch to System debugger command switches the current live program from Visual Inspect to the system debugger. This command is supported only on TNS/E systems, on which Native Inspect is the system debugger. The program being transferred must be in a Hold state.

Debugger terminal name:
Specify a home terminal for the target debugger, which will be the new default debugger terminal for the target process. You can also either enter a terminal name or select one from the drop-down list. By default, Visual Inspect uses the home terminal of the process you are transferring. (To verify a terminal name, enter the STATUS *, TERM command at the TACL prompt on the terminal.)

About the target debugger:
– If Native Inspect is not running on the terminal you specify, a new Native Inspect session is started on that terminal, and Native Inspect now owns the process.
– If Native Inspect is already running on the terminal, the process is given to that Native Inspect session. However, if the process and Native Inspect are running in different processors, a new Native Inspect session is started on that terminal, and the process is given to the Native Inspect session running in the same processor as the process being debugged.
Checking the “Enable access to privileged code and data” box is not restricted but has effect only for a debugging session by the super ID.
Modified values not displayed in red.
The Native Inspect product is based on the HP wdb debugger, which in turn is based on the Open Source product, gdb.

You control Native Inspect with UNIX-style commands.

Most of the commands from Inspect have corresponding commands in Native Inspect.
NATIVE INSPECT — FEATURES

- Automatically displays next source statement
  - "source on" equivalent
- Automatically displays function arguments
  - Value if data item
  - Address if pointer
- Automatically displays return values from functions
  - When function is "finished"
  - Including void functions
- Breakpoint display shows number of times triggered
- Supports TCL scripts
You can start Native Inspect in the same way that you started Inspect on previous systems. Remember that Visual Inspect will be the selected debugger, if it is available.
NATIVE INSPECT - HELP

-help [<topic>]
(eInspect 2,372):help
Native Inspect help

Welcome to Native Inspect Online help!

The following are major topics from which you can choose:

all       -- List all Native Inspect commands.
help      -- Help on help, what options you can have.
source    -- Source in a Tcl script file.
target process -- Commands that support process debugging.
target snapshot -- Commands that support snapshot analysis.

Type "help" followed by the keyword of the topic or Native Inspect command you want to browse.
**NATIVE INSPECT – BREAKPOINTS**

**Setting:**
- `break [function | paragraph | [source-file:]line-number] [if cond-exp]`
- `tbreak [function | paragraph | [source-file:]line-number] [if cond-exp]

```
(einspect 2,372):break get_second_number
Breakpoint 1 at 0A700000:0.1: file \DAGGER\DOTA00.TERRYG.CALLC, line 4.
(einspect 7,279):n call h if second_numbers == 99
Breakpoint 3 at 0A700000:0:1: file \DAGGER\DOTA00.TERRYG.CALLC, line 8.
```

**Displaying:**
- `info break`

```
(einspect 2,372):info b
Num Type      Disp     Orig    Hits     Where
2 breakpoint  keep y n 0X700000 in get_second_number
   at \DAGGER\DOTA00.TERRYG.CALLC:6

3 breakpoint  keep y n 0X700000 in get_second_number
   at \DAGGER\DOTA00.TERRYG.CALLC:8
```

**Clearing:**
- `delete [breakpoint-number]`
  - Deletes all breakpoints if no number given

The `tbreak` command sets a temporary breakpoint which is automatically deleted after one use. The `info break` command displays information about each breakpoint including the location, the number of times triggered, and a breakpoint number. You use the breakpoint number as an argument to the `delete` command to remove the breakpoint. If you do not specify an argument to the `delete` command, all breakpoints are removed.

Note that for C/C++ programs, the first breakpoint you set will be number 2.
The continue command causes execution of the process to continue normally until a breakpoint is reached, at which point the process will go into a Hold state and Native Inspect will prompt for the next command.

The next and step commands are similar – each one executes the next source statement. However, if the next statement is a call to a function (or program in COBOL), the step command causes the process to go into a Hold state at the start of the function, while the next command will execute the complete called function, including any nested function calls, and the process will go into a Hold state only on return from the called function.
The until command executes the program as normal, going into a Hold state when it reaches the specified location. However, if a breakpoint is triggered before the program reaches the specified location, the until command is cancelled.

In C/C++/pTAL, you use the finish command to complete execution of the current function. The process suspends when it returns to the calling function.

In COBOL, you cannot use the finish command to step execution out of a PERFORM. To step out of a PERFORM, you can set a breakpoint at the return location and then continue execution until that breakpoint is encountered.
You can use the commands command to specify a series of commands to be executed each time the process triggers a breakpoint.

You can use the display command (covered later in this module) to display variables any time the process goes into a Hold state.
For C/C++/pTAL, the info locals command displays the names and contents of all the current function’s local variables; for COBOL, the command displays the names and contents of all the current program’s Working Storage.

You use the print command to display the content of any variable or Working Storage field that is currently in scope. For COBOL, you must qualify the name of a field that is not unique by using the OF or IN clause.
In pTAL programs, you must explicitly dereference a pointer, using C/C++ style syntax, to access the data.
NATIVE INSPECT – VARIABLE INFORMATION DISPLAY

- Data type
  - whatis <variable-name>

```
  eInspect 2,372):what is total
type = short *
  (eInspect 1,494):what is total
type = INT *
  (eInspect 0.779):what is total
type = PIC S9999
```

You can use the whatis command to display the data type and length of a variable.
Debuggers

NATIVE INSPECT – DATA MODIFICATION

- Modifying variable or pointer
  - set [variable] <variable-name>=<new-value>
  - (eInspect 2,372): set +total=999

You must specify the keyword “variable” on the set command if the name of the variable you are changing might be interpreted as one of the other options of the set command, for example, “print” or “heap-check”.

4 – 22

UC414S D.00
The list command with no parameter either lists 10 lines around the current program location or continues from where the last list command finished. If the list command does not show code from the current location, use the backtrace (bt) command to display the current line, then use the list command specifying that line number.

The next line that will execute is indicated with an asterisk (“*”).
The info sources command shows the names of the source files that were used at compilation. If a source file has moved to a different location since compilation, and it also has a different filename, use the map command.
NATIVE INSPECT – FUNCTION NAMES (C)

- info functions [<wildcard-string>]
  (einspect 12, 372): info functions
  All defined functions:

  File \ATOM\\.\RLSB\\.T0432PAM\\.VER32RNC:
  void T0432PAM2_3IMAY2008_CCDMAIN();

  File \SPEEDY\\.\RLSE\\.T0432H02\\.CDDMAIN:
  void _MAIN();
  void _INIT__1_C();

  File \DAGGER\\.\DATA00\\.TERRYG.CALLC:
  short get_second_number(short, short *);

  File \DAGGER\\.\DATA00\\.TERRYG.MAINC:
  void display_initial_req_message();
  int main();

  Non-debugging symbols:
  0x700008e0 .plt
NATIVE INSPECT – FUNCTION NAMES (PTAL)

{cInspect 1,404): info functions
All defined functions:

File \DAGGER \$DATAOO.TERRYG.CALLP:
void GET_SECOND_NUMBER(int, int *);

File \DAGGER \$DATAUU.TERRYG.MAINP:
void DISPLAY_INITIAL_REQ_MESSAGE();
void MAIN_PROG();

Non-debugging symbols:
  0x700007c0 .plt
Current language: auto; currently ptal
NATIVE INSPECT – PROGRAM NAMES (COBOL)

(eInspect 0,779).info functions
All defined functions:

File \DAGGER\$DATA00.TERRYG.CALLCOB:
GET-SECOND-NUMBER PROGRAM-UNIT;

File \DAGGER\$DATA00.TERRYG.MAINCOB:
CALCULATOR PROGRAM-UNIT;

Non-debugging symbols:
0x70000a00 .plt
0x70004780 __INIT_0_CALCULATOR_
0x70004900 __INIT_1_CALCULATOR_
Current language: auto; currently COBOL
If you specify all-registers, then the floating-point registers display as well as all the general purpose and other registers.
### Native Inspect - Registers Display (2 of 2)

**Floating point registers f0..f127:**
- f0: 0x00000000 00000000 00000000 0
- f1: 0x00000000 80000000 00000000 1
- ...
- f126: 0x00000000 00000000 00000000 0
- f127: 0x00000000 00000000 00000000 0

- ap: 0x7000e70:1
- bsp: 0x6e000006
- tps: 0x99a0c3a/0030f
- lc: 0x0
- wc: 0x0
- cfm: 0x309 (sox:0, sol:6, sof:9)
- psp: 0x60000030
- ra: 0x70000a20

---

**UC414S D.00**
The `fopen` command with no arguments displays, for each file the process has open, the filenumber, the last error on the file, and the name of the file.

If you specify a file number with the `-d` option, you will get a display of all the relevant file open attributes.
Native Inspect – Open File Information (2 of 2)

```
(eInspect 0,779): fopen
FileNum  LastErr  Name
 1        0       /
 3        0       /G/data00/terryg
(eInspect 0,779): fopen 3 -d
Name      /G/data00/terryg
FileNum 3

OSF File Information:
 Mode 16384
 Error 0
 Error Detail 0
```
The backtrace command displays the current program location and how the program got there.
For more information on memory debugging with Native Inspect, see the *Debugging Dynamic Memory Usage Errors Using HP WDB* white paper and the *Debugging with GDB* manual at the HP WDB Documentation webpage: http://www.hp.com/go/WDB. Native Inspect's implementation of memory debugging is similar to that of WDB.
NATIVE INSPECT – I HEAP COMMANDS (2 OF 3)

Leak detection
- info leaks

```
(leinspect 2.88):1 0.1
  0.010  #include <stdlib.h>
  0.11
  0.111     void loseit(char *x);
  0.21
  0.211     char * p;
  0.31
  0.311     p = malloc (100);
  0.41
  0.411     strcpy(p,x);
  0.51
  0.511 }
  0.61    int main (void)
  0.71    {
  0.81     loseit("This is just a string.");
  0.91     loseit("Try again.");
  1.01     loseit("Third time.");
  1.11 }
```

200 bytes leaked in 2 blocks

<table>
<thead>
<tr>
<th>No.</th>
<th>Total bytes</th>
<th>Blocks</th>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>1</td>
<td>0x08429d08</td>
<td>loseit()</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>1</td>
<td>0x0042f000</td>
<td>loseit()</td>
</tr>
</tbody>
</table>
Heap display

```
(cinepect 2,90):info heap
Analyzing heap ...
```

**Actual Heap Usage:**
- Heap Start = 0x08001000
- Heap End = 0x08491000
- Heap Size = 4390912 bytes

**Outstanding Allocations:**

<table>
<thead>
<tr>
<th>No.</th>
<th>Total bytes</th>
<th>Blocks</th>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1024</td>
<td>1</td>
<td>0x08404570</td>
<td>&lt;system/unknown&gt;()</td>
</tr>
<tr>
<td>1</td>
<td>247</td>
<td>1</td>
<td>0x08429b80</td>
<td>&lt;system/unknown&gt;()</td>
</tr>
<tr>
<td>2</td>
<td>247</td>
<td>1</td>
<td>0x08429a30</td>
<td>&lt;system/unknown&gt;()</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>1</td>
<td>0x08429de0</td>
<td>loseit()</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1</td>
<td>0x08429d00</td>
<td>loseit()</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>1</td>
<td>0x08429d70</td>
<td>loseit()</td>
</tr>
<tr>
<td>6</td>
<td>66</td>
<td>1</td>
<td>0x08429b30</td>
<td>&lt;system/unknown&gt;()</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>1</td>
<td>0x08429c80</td>
<td>&lt;system/unknown&gt;()</td>
</tr>
</tbody>
</table>
You use the log command to turn on and turn off logging; in addition, you can use it to display the current log file name, if any.
NATIVE INSPECT – SESSION LOGGING (2 OF 2)

Log file \DAGGER$DATA00.TERRYG.junklog opened on Fri Nov 7 08:47:33 2008
(aInspect 0.855): log -d
The log file in use is \DAGGER$DATA00.TERRYG.junklog.
(aInspect 0.855): b get_second_number
Breakpoint 2 at 0x70000000:1: file \DAGGER$DATA00.TERRYG.CALLC: line 6.
(aInspect 0.855): c
Continuing.

Breakpoint 2, get_second_number (first_number=12, totals=0x60000000)
    at \DAGGER$DATA00.TERRYG.CALLC:6
        6     printf ("ENTER THE SECOND NUMBER: ");
(aInspect 0.855): log
Log file \DAGGER$DATA00.TERRYG.junklog closed on Fri Nov 7 00:40:10 2000
13 RECORDS TRANSFERRED

17 for more junking
MISCELLANEOUS COMMANDS (1 OF 4)

– Recall and modify a previous command
  • fc
– Display contents of current subvolume or directory
  • Is | files |<wildcard-pattern>
– Change to a different debugger
  • switch
    – Changes TNS/C process to Visual Impact
    – Changes TNS process to Impact
– Stop current process
  • kill
– History
  • show commands
MISCELLANEOUS COMMANDS (2 OF 4)

- Automatically display a variable at HOLD status
  
  display [ [/<format-letter>] ] <variable-name>

  (eInspect 1.911):display /x first_number
  1: /x first_number = 0xc
  (eInspect 1.911):c

  Continuing.

  Breakpoint 2, get_second_number (first_number=567, total=0xfffff44)
  at \DAGGER$.DATA00.TERRYG.CALLC:6
  * 6  printf("ENTER THE SECOND NUMBER: ");
  1: /x first_number = 0x237
  (eInspect 1.911):n
  * 7  scanf("%hi", &second_number);
  1: /x first_number = 0x237
The print command, in addition to displaying a variable, can be used to invoke a function contained in the program.
MISCELLANEOUS COMMANDS (4 OF 4)

- Define a command
  · Up to 1U arguments can be passed
  · define <command-name>
  
  eInspect 1,919):define mycmd
  
  Type commands for definition of "mycmd".
  End with a line saving just "end".

>> a
>> $arg0
>> $arg1
>> $arg0 + $arg1
>> end

{eInspect 1,742):mycmd 12 56
$2 = 9
$3 = 12
$4 = 56
$5 = 68
SNAPSHOT/SAVEABEND FILES

- Create a snapshot/saveabend file
  - `save <filename>`,
  - (eInspect 3.433): `save snaptest`
  - Successful switch to snapshot server
  - Switching process (3,433) to eInspect from SFS
  - Created snapshot file named `\$DATA00.TERRYG.snaptest`.
  - Process (3,433) forced into DEBUG.
    - [switching to process (3,433)]
    - get_second_number (first_number=12, total=0x00ffffff44)
    - at `\$DATA00.TERRYG.CALLC`: 6
    - printf("ENTER THE SECOND NUMBER: ");

- Open a snapshot/saveabend file
  - `snapshot <filename>`
    - (eInspect 3.433): `snapshot snaptest`
    - Loaded snapshot file `\$DATA00.TERRYG.snaptest` for program `\$DATA00.TERRYG.CEX`
    - Created by unknown at time 2010-11-05 14:37:24
    - Symbols read in for program loadfile `\$DATA00.TERRYG.CEX`.
    - Type 'info target' for additional details.
    - (eInspect 3.433):

---

The `save` command creates a snapshot or saveabend file of the current process. You can input this file, or any other snapshot or saveabend file, into Native Inspect with the `snapshot` command.
NATIVE INSPECT — TNS PROCESS

– Debugger of last resort
  • Limited commands:
    – continue — Resume execution
    – kill — Terminate process
    – bt — Stack trace
    – save — Create snapshot file
    – switch — Switch to different debugger
OBJECTIVES

- List the PC-based development tools available
- Discuss the new TNS/E project type in ETK
- Describe using the Enterprise Toolkit to create different applications using one set of source files
- Describe the use of NonStop Development Environment for Eclipse (NSDEE)

Reference Manuals:
C/C++ Programmer’s Guide
COBOL Manual for TNS/E Programs
pTAL Reference Manual
NonStop Development Environment for Eclipse User Guide
Workstation Tools

<table>
<thead>
<tr>
<th>Tools</th>
<th>TNS/R</th>
<th>TNS/E</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ compiler (Guardian, OSS)</td>
<td>c89.exe -Wtarget=tns/r</td>
<td>c89.exe -Wtarget=tns/e</td>
</tr>
<tr>
<td>pTAL compiler (Guardian)</td>
<td>ptal.exe</td>
<td>spial.exe</td>
</tr>
<tr>
<td>COBOL compiler (Guardian, OSS)</td>
<td>nrcobol.exe</td>
<td>acobol.exe</td>
</tr>
<tr>
<td>Linker</td>
<td>ld.exe</td>
<td>ndt.exe</td>
</tr>
<tr>
<td>Object File Utility</td>
<td>lc.exe</td>
<td>ld.exe</td>
</tr>
<tr>
<td>Tandem Dev Suite (TDS)</td>
<td>Available</td>
<td>&lt;not-supported&gt;</td>
</tr>
<tr>
<td>Enterprise toolkit (ETK)</td>
<td>R2.x, R3.0* or later</td>
<td>R3.0* or later</td>
</tr>
<tr>
<td>NonStop Dev. Env. for Eclipse (NSDEE)</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>Visual Inspect (GUI Debugger)</td>
<td>R2.0, R3.0* or later</td>
<td>R3.0* or later</td>
</tr>
</tbody>
</table>

* Supports both TNS/R as well as TNS/E platform.

In the left column, the PC-based tools are listed. The middle column lists the tools that target TNS/R, and the right column lists the tools that target TNS/E. All the tools have either the same name or equivalent names, going from TNS/R to TNS/E, and perhaps a small change in how to select that using a target flag. For example in c89.exe, if you want to build a TNS/R program, you select –Wtarget=tns/r and for TNS/E, the target is tns/e.

The HP Tandem Development Suite (TDS), which is the predecessor of the HP Enterprise Toolkit – NonStop Edition (ETK), is not supported for Itanium. This product has been superseded by ETK and R3.0 supports both TNS/R and TNS/E.

HP NonStop Development Environment for Eclipse (NSDEE) supersedes ETK. It also supports both TNS/R and TNS/E.
ETK CHANGES

- New project types:
  - Main
  - With HP NonStop SQL/MP
- New/changed compiler and linker properties
Welcome to the HP NonStop Server Application Wizard

This wizard generates a THSE NonStop Server Application Project based on information recorded through the wizard.

There are the current project settings:
- NonStop Server Crawler Executable Project
- HPQSTI is the target release variant (THSE)
- Initial Source File Language is C++
- THSE Native objects will be generated

Click Finish from any window to accept the current settings.

After you create the project, use the project's readme.txt file for information about the project features and files that are generated.
MULTIPLE PLATFORMS

- Use two projects in same solution
  - Same source files
  - Different properties
    - Target platform
    - Unlink
This slide illustrates building an application targeting the MIPS platform. You start with ETK and when you perform a build solution, ETK uses the MIPS native compilers, and you generate a MIPS object library. Then you use a deploy solution button to deploy it on a target platform.
The slide shows the steps for the Itanium systems, essentially the same thing, except rather than ETK using MIPS-based compilers, ETK uses the compilers for Itanium-based HP NonStop servers and generates Itanium object code that you then deploy. The methodology and the process by which you build and deploy application between TNS/R and TNS/E has not changed at all.
Migrating ETK projects is very simple. First, you copy your existing project. The virtual links point to the same source file. Since there is source code compatibility, while you are creating a copy of the project, all the files continue to point to the same physical files that you have on your PC workstation or on a remote server. On the upper left box, you can see that the project, NSProject4, was copied as NSProject4E for Itanium, and it created the same file hierarchy. The same virtual links were copied. The bottom window shows how the target platform for NSProject4 can be changed to TNS/R or TNS/E. Then you build and deploy the solution.
Eclipse was originally developed by IBM but is now managed by the Eclipse Foundation, an independent not-for-profit consortium of software industry vendors, including HP.

The Eclipse platform is designed for building integrated development environments (IDEs), and arbitrary tools. Eclipse includes C/C++ Development Tooling (CDT), Java Development Tools and vendor plug-ins. The HP NonStop Development Environment for Eclipse provides a new development tool based on the Eclipse environment.

With these plug-ins, you can leverage the ease of use and cost benefits of developing on a Microsoft® Windows® system–based PC to create applications for NonStop servers using Java™, native C/C++, COBOL, and pTAL languages. NSDEE streamlines development and, as a result, increases productivity.

The Eclipse development platform is popular among developers and is fast becoming the dominant IDE of choice in many organizations. Programmers familiar with Eclipse can be trained easily to use NSDEE and become productive quickly. Consequently, you can more easily tap skills in the job market. Furthermore, leading-edge technology increases job satisfaction among programmers developing applications for NonStop servers.
HP NonStop Development Environment for Eclipse (NSDEE) adds platform-specific enhancements to the open-source Eclipse environment while preserving the extensibility of the Eclipse integrated development environment (IDE). A range of tools from HP as well as some third-party add-ons can be used with these plug-ins.

NSDEE increases productivity and reduces development costs. This one-stop, state-of-the-art Integrated Development Environment (IDE) streamlines application development. PC-based compilations run faster, helping to boost productivity, and greater productivity translates into reduced development costs. The use of inexpensive PC hardware for development leads to better utilization of business critical NonStop server resources.
The NonStop Development Environment for Eclipse is tightly integrated into the Eclipse development environment and provides the same benefits provided by the underlying Eclipse environment. It leverages open-source technologies and shares the overall look and feel of Eclipse. This is achieved by developing plug-ins built on top of the Eclipse and C/C++ Development Tooling (CDT) projects and by using integrated toolbars, buttons, dialog boxes, drop-down menus, pop-up context-sensitive menus, and wizards to incorporate NonStop server application development features into the Eclipse environment.

As an added benefit, the NonStop Development Environment for Eclipse offers cross-platform development support similar to its predecessor—the Enterprise Toolkit. With the number of open-source initiatives being built on top of the Eclipse IDE, it is becoming an ideal development environment for Web, client/server, and service-oriented architecture (SOA)-enabled applications. Sample programs in different languages are supplied showing various aspects of NSDEE.
KEY FEATURES OF NSDEE (2 OF 2)

- Launch feature
  - Start NonStop applications and view terminal output
- Remote Projects
  - Build (compile/link) performed on NonStop
  - Integrity NonStop systems only
  - Secure connection support
  - All source and object files stored on NonStop
    - Files copied to workstation for editing
    - Simultaneous updates detection
- NSDEE debugger
- Complementary tools:
  - File transfer
  - Visual Inspect
- Integrated help system
- Sample Programs
  - A functional alternative to Enterprise Toolkit (ETK)
  - Tool to migrate ETK-generated NonStop projects to Eclipse
NSDEE supports the creation of various NonStop server-specific projects. Each project type comes with a wizard that helps you set up key project properties. Using a wizard, you can select the target platform (TNS/R or TNS/E, and the HP Guardian or Open System Services environment), the Release Version Update (RVU), and the type of object you want to create (for example, an executable program). You also can initialize compilation and deployment options and properties. The software preserves persistence of the project properties between sessions. At the highest level, the NonStop project types that are supported include:

- Managed Make—wizard-driven make files are generated for projects.
- Standard Make—make files are created by the user, or reused from existing projects.
- Remote Standard Make—all the files and build operations are on the NonStop system.
You can use the NonStop Development Environment for Eclipse to write programs in native languages that are supported on the NonStop server, including
- Native C and Native C++
- Native COBOL (Tandem COBOL as well as ANSI COBOL)
- pTAL
- Java
  - The ability to develop Java-based applications using Eclipse provides a significant advantage over the Enterprise Toolkit product.

Source files written in different supported languages can belong to the same project, provided that the appropriate cross compilers have been installed. The NonStop Development Environment for Eclipse is upwards compatible with future cross compiler releases (that is, NSDEE will allow programmers to use a compiler that has been installed more recently than itself.) Each language compiler is ordered separately.
Language compiler options are user configurable.
NSDEE comes with built-in documentation and product-specific help functions that are integrated into the Eclipse help system. This dynamic, context-sensitive help system is comprehensive and covers a wide range of topics. You can search for topics or use an index without leaving the environment.
The new project wizard lets you select a skeleton containing embedded SQL statements. SQL/MX preprocessing is performed entirely on the workstation. MXCMP is used on the NonStop server side to complete the SQL/MX compile.
The integrated Eclipse debugger supports applications written in Java. The NSDEE debugger supports native Integrity NonStop applications developed locally or remotely, written in C/C++ or Cobol. For other system types and other languages, Visual Inspect is available.
ETK TO NSDEE MIGRATION UTILITY

- Migrates a complete solution from ETK to NSDEE.
- Only NonStop Server, SQL/MP, SQL/MX and Archive projects can be migrated.
  - Other project types generate warnings.
- The migration utility enables the user to:
  - Browse to the location of the ETK solution file (.sln).
  - Select an existing workspace or create a new one for the project being migrated.
  - Create an NSDEE project with a similar configuration to the ETK project.
  - Choose the make type for the migrated projects:
    - The default make type is Managed make.
  - The project profile in the ETK nsproj XML file is mapped to the .project file and the persistent properties in Eclipse.
THE MIGRATED APPLICATION

4> fileinfo hptfexe

$FCI.TESEY3

<table>
<thead>
<tr>
<th>CODE</th>
<th>EDF</th>
<th>LAST MODIFIED</th>
<th>OWNER</th>
<th>RMEF</th>
<th>PEXT</th>
<th>SEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>hptfexe</td>
<td>000</td>
<td>21568 17MAR2011 12:25</td>
<td>34,26</td>
<td>AAGO</td>
<td>14</td>
<td>28</td>
</tr>
</tbody>
</table>

5> run hptfexe
A demo for HPTF2008.
The time is 12:25:59.

6> fileinfo hptfexe

$FCI.TESEY3

<table>
<thead>
<tr>
<th>CODE</th>
<th>EDF</th>
<th>LAST MODIFIED</th>
<th>OWNER</th>
<th>RMEF</th>
<th>PEXT</th>
<th>SEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>hptfexe</td>
<td>000</td>
<td>21416 17MAR2011 12:37</td>
<td>34,26</td>
<td>AAGO</td>
<td>14</td>
<td>28</td>
</tr>
</tbody>
</table>

7> run hptfexe
A demo for HPTF2008.
The time is 12:38:40.
CODE PROFILING UTILITIES

Module 6
UC4145 D.00
OBJECTIVES

- Describe the installation and use of the Code Cover utility
- Describe the use of Profile Guided Optimization

Reference Manuals:
Code Profiling Utilities Manual
The code generator used by COBOL, pTAL, and C/C++ compilers on NS-series servers has the capability to create instrumented object files. Such object files contain extra code that records which functions and blocks are executed, and how many times each is executed. The Code Coverage Tool uses this information to produce a report indicating what code in a program file or DLL was actually executed during one or more invocations. The code coverage report is a set of HTML files that you can view with any standard HTML browser.

No source code changes are needed to instrument an application. The only required changes are in the commands used to compile and link the application. If you choose to instrument only a subset of your application, you specify code-coverage compiler options for only that subset of your source files.

Compilation can occur on the Guardian, NonStop Open System Services (OSS), or Windows platform. Execution must occur on either the Guardian or the OSS platform.

You can instrument all or part of any type of application, for instance:
- OSS processes
- Guardian processes
- Active and passive process pairs
- Mixed language processes
- Processes with embedded NonStop SQL/MP or NonStop SQL/MX
With Profile Guided Optimization, the compiler takes as input the data generated from an instrumented program and uses it to perform optimization of the most frequently executed code segments.
APPLICABILITY

- Can be used with:
  - UJS's processes
  - Guardian processes
  - Active and passive process pairs
  - Mixed language processes
  - Processes with embedded NonStop SQL/MP or NonStop SQL/MX
The use of instrumented code is not recommended for production environments. Applications compiled with code profiling instrumentation will experience greatly reduced performance.
You can run the profmrg program on a workstation instead of a NonStop server.
INSTALLING THE CODE COVER UTILITY

- Program codecov runs on a Windows workstation
- codecov:
  - Copy $SYSTEM\ZCODECOV.T0746SET to workstation
  - Change the name to setup.exe
  - Run setup.exe
  - Create \program files\Hewlett-Packard\CodeCoverage\codecov.exe

Add \program files\Hewlett-Packard\CodeCoverage to PATH
BUILDING AN APPLICATION FOR CODE COVERAGE

- Compiler option:
  - Guardian: LUDLCUV
  - OSS: -Wcodecov
  - Windows:
    - c89, eccbo: -Wcodecov
    - sptsl: -Wcodecov

- Creates or updates the Static Profiling Information (SPI) file:
  - Guardian: PGOSPI, code 1803
  - OSS/Windows: pgopti.spi

- If link editing manually:
  - Use -l pgsl to reference the pgsl library
RUNNING THE APPLICATION

- Execute as normal
- Process creates a new raw data file for each execution:
  - Guardian: ZZPFxxxx, code 180, in current default subvolume
  - OSS: ZZPFxxxx.dvn in current working directory
- Creates ZZPLOG if any errors during execution
CODE COVERAGE SAMPLE 1 (1 OF 8)

101> comp/in/main/incl/ccode, optimize 1
102> comp/in/incl/cal/calclcallco/ccode, optimize 1
103> acl -o calculator mainco calcol %system%system%copmain -L c-1 -cc
104> run calculator

YOU HAVE JUST STARTED THE PROCESS.

ENTER FIRST NUMBER [or 0 to stop]: 12
ENTER THE SECOND NUMBER: 24

THE NUMBERS ENTERED WERE: 12 & 24
THE TOTAL IS: 46
ENTER FIRST NUMBER [or 0 to stop]: 0

JVC17089420

Created files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date/Time</th>
<th>User</th>
<th>Group</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCOSS</td>
<td>190</td>
<td>29-Jul-2011 19:49</td>
<td>34:26</td>
<td>43:26</td>
<td>34:26</td>
<td>14:42</td>
</tr>
<tr>
<td>FDOCSS</td>
<td>180</td>
<td>29-Jul-2011 13:47</td>
<td>34:26</td>
<td>43:26</td>
<td>34:26</td>
<td>14:42</td>
</tr>
<tr>
<td>ZERFS</td>
<td>180</td>
<td>29-Jul-2011 13:46</td>
<td>34:26</td>
<td>43:26</td>
<td>34:26</td>
<td>14:42</td>
</tr>
</tbody>
</table>
Even if there is only one run of the application process, you must run profmrg to create the DPI file. You can run profmrg on a workstation, instead of on a NonStop system. To do this, you must download the file $SYSTEM.ZCPS.T0747SET, rename it as an executable such as setup.exe, and execute it. This will install the program as Program Files\Hewlett-Packard\CodeCoverage\profmrg.exe. You might want to add the \Program Files\Hewlett-Packard\CodeCoverage directory to your PATH variable.

The options for profmrg can be any from the following list. Options must be in lowercase.

-a dpi_file_list
  - specifies the DPI files to use as input. The list can consist of any number of file names, separated by spaces. There is no rule for the format of a filename; for example, it need not contain the string .dpi. If the -prof_dir option is also present in the command, then each name specified for the -a option is concatenated with the name given in the -prof_dir option. The -a option must be the last option on the command line. Any options specified after it are ignored.

-dump
  - produces a text dump of the contents of the input files. The profmrg utility writes the dump to the standard output file (stdout). You can redirect it to any convenient location. The -dump option produces text output even if profmrg does not re-create the DPI file. Therefore, this option is useful for gathering information about an existing DPI file.

-help
  - displays brief descriptions of syntax options. Do not specify any other options on the command line. When you specify the -help option, any other options on the command line are either ignored or produce command syntax error messages.

-nologo
  - suppresses the banner that profmrg would otherwise display.

-prof_dir directory
  - specifies the name of a folder relative to which profmrg looks for its input files and creates its output file. By default, profmrg uses the current folder. Filenames specified in the -prof_dpi and -a options are concatenated with the directory name specified in this option.

-prof_dpi filename
  - specifies the name of the DPI file that profmrg creates, overriding the default name pgopti.dpi. There is no rule for the format of the filename; for example, it need not contain the string .dpi. If the -prof_dir option is also present in the command, the filename is concatenated with the name given in the -prof_dir option.

Note. A filename specified in a profmrg command option must not begin with a slash(/).
The options for the codecov program can be any from the following list. Options must be in lowercase.

-bcolor color
  - specifies the name or hexadecimal code of the HTML color used in reports to show uncovered basic blocks within a function for which some basic blocks were covered and some were not. The default value is #ff99, which is yellow.

-ccolor color
  - specifies the name or hexadecimal code of the HTML color used in reports to show the basic blocks that were covered (that is, executed during the test run). The default value is #ffffff, which is white (no color).

-counts
  - causes execution counts to be included in the code coverage report. A basic block executed once has a count of 1, a basic block executed twice has a count of 2, and so on.

-dpi filename
  - specifies the name of the DPI file, overriding the default name pgopti.dpi.

-fcolor color
  - specifies the name or hexadecimal code of the HTML color used in reports to show functions that were uncovered (never called). The default is #fccc, which is pink.

-h or -help
  - causes codecov to stop processing the command line, print out a syntax description of all options that it supports, and terminate.

-host string
  - provides a host address to use for access to a NonStop server. The codecov utility uses the name when fetching source files from the NonStop server. The string could be a DNS name- or an IP address in dotted decimal format. You must specify this option if the application includes source files on NonStop servers.

Continued on next page.
-login string
  – provides a login name for access to a NonStop server. The codecov utility uses this name when
  fetching source files from the NonStop server. You must specify this option if the application
  includes source files on NonStop servers.

-maddr email
  – specifies a destination for email sent from the code coverage report. The codecov utility places a
  link at the bottom of each screen of the report. When you click the link, a window for sending
  email appears, with the address specified by the -maddr option. If you omit this option, the mail is
  sent to nobody.

-mname message
  – specifies the text of the link used to invoke the mail window. If you omit the -mname option but
  include the -maddr option, the text of the link is the same as the address specified in the -maddr
  option. If neither option is present, so that -maddr defaults to nobody, then -mname defaults to
  Nobody.

-nopartial
  – specifies that, if multiple basic blocks are generated for a single source position, codecov should
  consider them all to be covered if any one of them was covered. In the report, such code appears
  in the color for covered code rather than partially covered code.

-nopmeter
  – suppresses the progress meter, which codecov would normally write to the standard output file
during its operation. The progress meter reports the percentage of functions analyzed so far. For
example, if a program had only four functions, codecov would print 25%, 50%, 75%, and finally
100%. If a program contains ten or more functions, codecov prints the percentage each time it
completes analysis of one-tenth of the functions, so the progress meter is updated at most ten
times.

Continued on next page.
- `passwd string`
  - Specifies the password for the user name given in the `-login` option. If `codecov` must fetch source files from the NonStop server and the `-passwd` option is not present on the command line, `codecov` prompts for the password.

- `pcolor color`
  - Specifies the name or hexadecimal code of the HTML color used in reports to show partially covered code. If the `-nopartial` option is present on the command line, the `-pcolor` option is meaningless. The default value is `#fafad2`, which is light brown.

- `prj title`
  - Specifies a title to be included at the top of the top-level HTML file in the code coverage report. For example, if you specified the value `CallDistribution`, the full title printed at the top of the report would read “Coverage Summary of CallDistribution.” If you omit this option, the top-level HTML file bears the title Coverage Summary.

- `spi filename`
  - Specifies the name of the SPI file, overriding the default name. The default name is:
    - `pgopti.spi` if the program was compiled on Windows
    - `pgopti.spi` if the program was compiled in the OSS environment and the current directory is an OSS directory
    - `pgospi` if the program was compiled in the Guardian environment
    - `pgospi` if the program was compiled in the OSS environment and the current directory is a Guardian subvolume

- `ucolor color`
  - Specifies the name or hexadecimal code of the HTML color used in reports to show source for which no code was generated. Examples are comments, statements that include header files, and variable declarations. The default value is `#ffffff`, which is white (no color).
## Code Coverage Sample 1 (6 of 8)

### Coverage Summary of Sample Run

<table>
<thead>
<tr>
<th>File</th>
<th>Function</th>
<th>Block</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>2</td>
<td>100.00%</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>3</td>
<td>100.00%</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>12</td>
<td>100.00%</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>10</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

### Covered Files in Sample Run

#### File A

- **Total Blocks**: 1
- **Total Coverage**: 100.00%

#### File B

- **Total Blocks**: 1
- **Total Coverage**: 100.00%

#### File C

- **Total Blocks**: 2
- **Total Coverage**: 100.00%

#### File D

- **Total Blocks**: 11
- **Total Coverage**: 100.00%
Before you run the application to produce a different report, you must delete any raw data files and the dynamic profiling information file.
CODE COVERAGE SAMPLE 2 (2 OF 5)
You supply the –counts option when you run the codecov program.
When executed, the instrumented application creates a raw data file that contains information about the execution path of the application. You can run the instrumented application multiple times, perhaps using different input data, representing typical workloads, for each run. Each execution creates a unique raw data file.

The profmrg program merges the raw data files into a Dynamic Profiling Information file which the compiler then uses.
BUILDING AN APPLICATION FOR PGO

- Available only for eptAL, C/C++
- Compiler option:
  - Guardian: PROFGEN
  - OSS: -Warmeem
  - Windows:
    - c89: -Warmeem
    - eptal: -profgem
  - Must use level 2 optimization.
- If link editing manually:
  - Use -l pgo to reference the pgodll library
USING PGO

- Compiler option:
  - Guardian: MKUPUSB
  - Unix: -Wprofuse
  - Windows:
    - /SG: -Wprofuse
    - gptal: -profuse
  - profuse option may specify the name of the LPM file if not default name or location
  - Use basename option if source files moved from original location
  - Must also use level 2 optimization
PGO SAMPLE RUN (1 OF 2)

6D> ccomp/bin/main/mainc/protgen_optimize 2
66> ccomp/bin/calle/calle/prefgen_optimize
66> eld -o calcatr1 mainc calc2 $sys/main.c compiling -l crtl -l cre
66> pgo
6D> fileinfo calc*

$FCI.TOCPU

<table>
<thead>
<tr>
<th>Code</th>
<th>EOF</th>
<th>LAST MODIFIED</th>
<th>OWNER</th>
<th>RPM</th>
<th>DWT</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALCATR1</td>
<td>800</td>
<td>21376</td>
<td>28 JUL 2011 14:43</td>
<td>34.26</td>
<td>A000</td>
<td>28</td>
</tr>
</tbody>
</table>

6D> run calcatr1

PID: 2954 \VERUN $FCI.TOCPU.CALCATR1

You have just started the process.

STOPPED: 2954
CPU time: 0:00:00.000

$FCI.TOCPU

<table>
<thead>
<tr>
<th>Code</th>
<th>EOF</th>
<th>LAST MODIFIED</th>
<th>OWNER</th>
<th>RPM</th>
<th>DWT</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZRDP279</td>
<td>180</td>
<td>984</td>
<td>29 JUL 2011 14:46</td>
<td>34.26</td>
<td>A000</td>
<td>14</td>
</tr>
</tbody>
</table>
Note that the eld step does not specify the pgo DLL. If you do accidentally specify the DLL, then at the end of the execution of the application you receive the message: “No segs collected - did you build with profgen option?”
APPLICATION MIGRATION

Module 7
UC414S D.00
OBJECTIVES

- Describe the possible HP NonStop S-Series to Integrity NonStop migration paths
- Describe how to prepare programs for migration
- Describe how to maintain common source code for TNS/R native and TNS/E native compilers
- Describe how to handle misaligned data
- Describe how to adjust for increased UCL limits
- Describe the tasks for migrating pTAL programs from TNS/R to TNS/E
- Describe the tasks for migrating C/C++ programs from TNS/R to TNS/E
- Describe the tasks for migrating COBOL programs from TNS/R to TNS/E
- Describe the tasks for migrating user libraries
- Describe the tasks for migrating Guardian APIs

Reference Manuals:
H-Series Application Migration Guide
NONSTOP S-SERIES TO INTEGRITY NONSTOP MIGRATION PATHS

- Both the S-series and the Integrity NonStop Release Version Updates (RVUs) support TNS execution modes (interpreted and accelerated) and native execution mode. This results in two possible migration paths:
  - S-series TNS Mode to Integrity NonStop TNS Mode
  - TNS/R Native Mode to TNS/F Native Mode
NONSTOP S-SERIES TNS MODE TO INTEGRITY NONSTOP TNS MODE

- Any TNS Guardian program that runs on an S-series platform should run on an Integrity NonStop with no changes
- TNS Guardian object code can be emulated by an interpreter
- TNS Guardian object code accelerated in TNS/R system can be accelerated again in TNS/E system
  - Object file can contain both accelerated versions
- TNS execution is not supported in the HP NonStop Open System Services (OSS) environment
TNS/R NATIVE MODE TO TNS/E NATIVE MODE

- Migrating object code running in TNS/R native mode to run in TNS/E native mode usually requires only a recompilation using a TNS/E native compiler, except:
  - In a few cases, source code needs changes
  - Changes in the scripts/macos to compile and link programs:
    - the compiler names
    - linker name
    - SRL to DLL name changes
    - some compiler and link options
  - To incorporate some of the new H- or J-series features into the application for performance improvement and other benefits
  - Some features considered obsolete in the TNS/R native programming languages are not supported in the TNS/E native version of the language
PREPARING PROGRAMS FOR MIGRATION

– Be sure the program runs successfully on the G06.20 or later G-series Release Version Update (RVU)
– If the program runs on an earlier RVU, see the following documents and upgrade:
  • G06 xx Software Installation and Upgrade Guide
  • G06 xx Release Version Update Compendium
  • Interactive Upgrade Guide and Interactive Upgrade Guide 2
MAINTAINING COMMON SOURCE CODE

- Most TNS/R native programs will require no changes to migrate them to an Integrity NonStop system.
- When maintaining a common source code for TNS/R and TNS/E native compilers:
  - Do not use any of the new H- or J-series features.
  - Avoid using those few features in the TNS/R programs that are not supported in the H- or J-series RVUs.
ALIGNMENT OF DATA

- Different compilers align data items differently:
  - TNS compilers align data on even-byte boundaries
  - TNS/R and TNS/E compilers align 4-byte data items on 4-byte boundaries
  - TNS/E compilers also align 8-byte data items on 8-byte boundaries
  - TNS/E native POOL, GETSPACE, GETPOOL, and malloc procedures allocate buffers aligned on 16-byte boundaries

- Programming errors or run-time events can cause a data item to violate the data alignment rules and can change the behavior of the program. Some programs, while functioning correctly, with misaligned data can experience a performance loss due to compatibility traps.
ADJUSTING FOR INCREASED DCT LIMITS (1 OF 2)

- The maximum size of the Destination Control Table (DCT) can be optionally increased from 32,767 to 65,376 as of G06.23 RVU and H-series RVU.
  - The default setting for the extended DCT limit is:
    - OFF for G06.23 or later RVU
    - OFF for H-series RVU (originally proposed to be on)
- The affected programs are those that call the following C-series procedures:
  - FILEINFO
  - GETDEVNAME
  - GETPPDENTRY
  - GETSYSTEMNAME
  - LOCATFSYSTEM
  - LOOKUPPROCESSNAME
ADJUSTING FOR INCREASED DCT LIMITS (2 OF 2)

- To migrate the affected programs:
  - Use XLC to reset the DCT limits extension default to O+h
  - Change programs to use replacement procedures:

<table>
<thead>
<tr>
<th>C-Series Procedure</th>
<th>Replacement Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILEINFO</td>
<td>FILE_GETINFO_LIST</td>
</tr>
<tr>
<td>GETDEVNAME</td>
<td>DEVICE_GETINFO_BYDEV_, CONFIG_GETINFO_BYDEV_, CONFIG_GETINFO_BYDEV2_, FILENAME_FINDSTART_, FILENAME_FINDNEXT_</td>
</tr>
<tr>
<td>GETPPDENTRY</td>
<td>PROCESS_GETPAIRINFO</td>
</tr>
<tr>
<td>GETSYSTEMNAME</td>
<td>NODENUMBER_TO_NODENAME_</td>
</tr>
<tr>
<td>LOCATESYSTEM</td>
<td>NODENAME_TO_NODENUMBER_</td>
</tr>
<tr>
<td>LOOKUPPROCESSNAME</td>
<td>PROCESS_GETPAIRINFO_</td>
</tr>
</tbody>
</table>
The following few slides outline some Portable Transaction Application Language (pTAL) migration tasks. For more detailed information in a particular migration situation, refer to the latest publications on pTAL compiler and programming.

<table>
<thead>
<tr>
<th>TNS/E PTAL COMPILER</th>
</tr>
</thead>
<tbody>
<tr>
<td>- All TNS/R pTAL programs must be recompiled with the TNS/E pTAL compiler.</td>
</tr>
<tr>
<td>- Most of the features and constructs of TNS/R pTAL are supported in TNS/E pTAL, except:</td>
</tr>
<tr>
<td>- Some privileged operations</td>
</tr>
<tr>
<td>- Some features, such as SRI, that are not supported in Hseries RVUs</td>
</tr>
<tr>
<td>- Features considered obsolete in TNS/E pTAL</td>
</tr>
<tr>
<td>- To invoke pTAL compiler:</td>
</tr>
<tr>
<td>- In Guardian environment: Run ptal command</td>
</tr>
<tr>
<td>- In Microsoft Windows: Run eptal command or eptal.exe through B1K</td>
</tr>
<tr>
<td>- All code produced by the TNS/E compilers is position-independent code (PIC), which can be used to create DLLs.</td>
</tr>
</tbody>
</table>
After removing either the SAVEGLOBALS or USEGLOBALS directive, the program should compile without errors and no further action should be required. However, if compilation errors occur as a result of removing the directive, further source code change is needed. The recommended changes are described in detail in the pTAL Reference Manual.
PTAL COMPILER DIRECTIVES (2 OF 2)

- Compiler directives that have different behavior and may require source code changes:
  - OPTIMIZEFILE: warnings are issued in certain conditions
  - OVERFLOW_TRAPS and NOOVERFLOW_TRAPS: In TNS/R pTAL, OVERFLOW_TRAPS is the default, but in TNS/E pTAL, NOOVERFLOW_TRAPS is the default
PTAL FUNCTIONS AND PROCEDURES

- Functions and procedures that are not supported and should be replaced with H- or J-series directly-called privileged interface:
  - $EXECUTIO, $INTERROGATEIO, $INTERROGATEHIO, $LOCATESPTHDR, $LOCKPAGE,
    $READBASELIMIT, $READSPT, $UNLOCKPAGE, and $WRITEPTE

- Functions and procedures that are changed:
  - $AXADR: Replace with standard interface
  - $FREEZE: Replace with the STRIGGER function.
  - $HALT: Replace with the STRIGGER function.
  - $STACK_ALLOCATE: Perform different rounding and alignment.
    - Address of allocated area aligned to 16-byte boundary
    - Size of allocated area rounded up to the multiple of 16 bytes
OTHER PTAL MIGRATION TASKS (1 OF 2)

- Procedure attributes
  - TNS/E PTAL does not support the INITIALIZE procedure attribute
- Address types
  - CBADDR and CWADDR address types can be used only to access PRelative (read-only) arrays. They are considered obsolete, and their use is not recommended.
- 16-byte-aligned data
  - The TNS/E PTAL compiler provides STRUCTALIGN (MAXALIGN) attribute. This attribute causes data to be aligned on the maximum alignment defined for the platform, which is 16 bytes for TNS/E platform.
OTHER PTAL MIGRATION TASKS (2 OF 2)

- FIELDALIGN directive in mixed language programming
  - FIELDALIGN [PLATFORM] causes the PTAL compiler to use the same data alignment as other compilers on the same platform
  - FIELDALIGN [AUTO] allows the PTAL compiler to determine data alignment based on what is most efficient

- Functions that return condition code and value:
  - TNS/E PTAL compiler issues warning
  - Not callable from C/C++
    - TNS/R PTAL 64-bit workaround no longer works
  - Write jacket procedure
The following slides outline some TNS/E native C/C++ migration tasks. For more detailed information about a particular migration situation, refer to the latest publications on TNS/E native C/C++ compiler and programming.
STANDARD C++ RUN-TIME LIBRARIES

- Three versions of the standard C++ library are available on NonStop systems:
  - VERSION1
    - Data structures and algorithm libraries
    - String, complex, and numeric limits classes
    - Not supported on Integrity NonStop systems
    - Tools.h++ version 0.1 library
  - VERSION2
    - STL (Standard Template Library)
    - Memory management, language support, locale and exception handling features
    - Tools.h++ version 7 library
  - VERSION3
    - Conforms to ISO/IEC 14882:1998 (C) standard for C++
    - Does not support any version of Tools.h++ library
NATIVE C/C++ LIBRARY MIGRATION (1 OF 2)

- Native C run-time library
  - Native C run-time library on S-series and on Integrity NonStop are nearly identical
- Native C++ run-time library
  - Migrating from C++ VERSION2 and VERSION3 on S-series to the respective version on Integrity NonStop should require no additional source code changes
  - Migrating from C++ VERSION1 on G-series to C++ VERSION2 on Integrity NonStop requires source code changes
    - If linking the program as a separate step, you must also specify the ZCPPCDLL and ZCPP2DIL on the \edl command
    - Tools.h++ version 0.3, if any, should be migrated to Tools.h++ version 7
    - Recommendation is to migrate to VERSION3
NATIVE C/C++ LIBRARY MIGRATION (2 OF 2)

- Native C++ run-time library (Continued)
  - Migrating from C++ VERSION1 on S-series to C++ VERSION3 on Integrity NonStop requires source code changes
    - The two versions are source code compatible; the only differences are in the names of header files
    - Requires source code changes to remove any use of Tools.h++ library
  - Migrating from C++ VERSION2 on S-series to C++ VERSION3 on Integrity NonStop requires source code changes
    - It is recommended that you convert to C++ VERSION2 on S-series, then to C++ VERSION3 on Integrity NonStop
    - Requires source code changes to remove any use of Tools.h++ library
CHANGED PRAGMAS AND FLAGS (1 OF 3)

- The following pragmas and flags are either obsolete or have different behavior:
  - EXTERN_DATA GP_OK allows GP-relative addressing for external data references. This pragma is ignored and a warning is issued in TNS/E systems.
  - NUN_SHARED pragma is ignored and a warning is issued
  - Skl pragma or -Wsr flag is ignored and a warning is issued
  - SRLExports or -Wsrlexport is ignored and a warning is issued
  - SRLExportMembers or -Wsrlexportmembers is ignored and a warning is issued
  - SRLName or -Wsrname is ignored and a warning is issued
CHANGED PRAGMAS AND FLAGS (2 OF 3)

- The following pragmas and flags are either obsolete or have different behavior (Continued):
  - **INSPECT**
    - Controls the selection of the debugging tools. The order of precedence follows:
      - for S-series non-PC: Visual Inspect, Inspect, DEBBUG
      - for S-series PC: Visual Inspect, DEBUG
      - for Integrity NonStop: Visual Inspect, Native Inspect
  - **NOINSPECT**
    - Controls the selection of the debugging tools. The order of precedence follows:
      - For S-series, DEBUG
      - For Integrity NonStop: Native Inspect
CHANGED PRAGMAS AND FLAGS (3 OF 3)

- The following pragmas and flags are either obsolete or have different behavior (Continued):
  - NLd or -Wnl d is replaced with ELd or -Weld respectively
  - LD or -Wld is replaced with ELd or -Weld respectively
  - VERSION1 replaced with VERSION2 or VERSION3
  - -Wversion1 replaced with -Wversion2 or -Wversion3
**NEUTRAL C++ DIALECT**

- The neutral C++ dialect consists of library components that are sharable between VERSION2 and VERSION3 of the C++ standard libraries.
- Permits middleware to support VERSION2 and VERSION3 clients.
- Two pragmas are provided in support of the neutral C++ dialect:
  - `NEUTRAL` Marks a class definition as being sharable (allowed only in standard header files).
  - `BUILD_NEUTRAL_LIBRARY` Used to create DLLs that use the neutral dialect. If this pragma is specified and the program references an object not marked with `NEUTRAL`, a compiler error is issued.

An object is neutral when the library and the application using it conform with the following rules:
- The interface that the library provides uses only C linkage, or
- The interface that the library provides uses C++ linkage and all of the parameters in these interfaces are marked as neutral or are strictly user-defined class types.
- Only the following C++ standard library interfaces are shared directly or indirectly between the library and the user program:
  - global new operators
  - global delete operators
  - stdin, stdout, and stderr
  - errno
  - class std::exception
  - class std::bad_alloc

For example:

To create and link a library that uses the neutral C++ dialect and can be used by either a VERSION2 or VERSION3 program, enter:
```
CPPCOMP / IN ZCPPCDLL, OUT $S.#LIST / mydll; SHARED, &
VERSION3, BUILD_NEUTRAL_LIBRARY
```

where ZCPPCDLL contains only interfaces common to both a VERSION2 and VERSION3 program; this is the low-level Standard C++ Library. The dynamic-link library mydll has its CPPNEUTRAL flag set for linker or loader checking.
OTHER C/C++ MIGRATION TASKS (1 OF 4)

- New C and C++ features
  - Pragmas
    - NEUTRAL: Marks an object as sharable between VERSION2 and VERSION3 of the C++ standard library
    - BUILD_NEUTRAL_LIBRARY: Build a DLL using the neutral C++ dialect
    - MAXALIGN: Aligns composite objects to the 16-byte alignment, which is the maximum alignment supported by the compiler for some functions to work. Most programs will not need to declare the MAXALIGN pragma, as it is included in the standard header file that supplies the data type declaration.
OTI IER C/C++ MIGRATION TASKS (2 OF 4)

- New C and C++ features (Continued)
  - New 64-bit unsigned integer data type
    - Type declaration is unsigned long long
  - New functions for conversion between character strings and the 64-bit integer type:
    - strtoll — Converts a character string to a signed long long integer
    - strtoull — Converts a character string to an unsigned long long integer
    - wcstoll — Converts a wide-character string to a signed long long integer
    - wcstoull — Converts a wide-character string to an unsigned long long integer
OTHER C/C++ MIGRATION TASKS (3 OF 4)

- Data type for 32-bit values in CEXTDECS file changes from long to int
  - When recompiling native INs/K programs for the INs/K environment, if
    - the parameter is passed by value: automatic type conversion is done and no warning is issued
    - The parameter is passed by reference: correct code is generated and the warning of mismatched
data types is issued
  - One of the following actions may be taken, but is not required:
    - Use the WARN/NOWARN pragma or flag to suppress the warning message
    - Use the int type instead of the long type in the calling program
    - Use __int128_t typedef for 32-bit items

OTHER C/C++ MIGRATION TASKS (4 OF 4)

- TNS/R and TNS/E systems use opposite conventions for returning quiet and signaling NaN (not-a-number) values
- Programs that depend on the TNS floating point format must specify the TANDEM_FLOAT pragma because the default floating-point format is IEEE
- Replace _TNS_R_TARGET feature test macro with _TNS_E_TARGET
- Use _TANDEM_AKCH_ feature test macro to identify the platform
  - 0 for TNS, 1 for TNS/R and 2 for TNS/E
The following slides outline some TNS/E native COBOL migration tasks. For more detailed information about a particular migration situation, refer to the latest publications on TNS/E native COBOL compiler and programming.
COBOL COMPILER DIRECTIVES (1 OF 2)

- The following obsolete compiler directives cause a warning to be issued if they are not removed. No code change is required.
  - NON-SHARED: In a TNS/E system, all code is sharable PIC; therefore, this directive is ignored.
  - LARGEA: In a TNS/E system, there is no distinction of data areas so this directive is ignored.
  - LESS-CODE: In a TNS/E system, native COBOL does not support the option to use a system call to initialize the Extended Storage Section and this directive has no effect.

- Additional remarks:...
COBOL COMPILER DIRECTIVES (2 OF 2)

- The following compiler directives have different meanings and changes may be required:
  - **UI**: Has the same effect as the **SHARED** directive. If used with CALLSHARED or SHARED, can cause an error. If used with RUNNABLE, causes the compiler to link the object file into a DLL.
  - **SEARCH** and **CONSULT**: These must specify TNS/E object files
  - **OPTIMIZE**:
    - 0: no optimization
    - 1: produces optimization for use with debugging
    - 2: produces full optimization
  - **INSPECT**: If available, selects Visual Inspect over Native Inspect as the debugging tool
  - **NOINSPECT**: Selects Native Inspect as the debugging tool
OTHER COBOL MIGRATION TASKS

- Use of floating-point data
  - IN$ native COBOL uses the $BB format, not HP proprietary $NS format

- Alignment of Level-01 and Level-77 data items
  - are aligned on an 8-byte boundary
  - can be aligned on an 4-byte boundary by adding the SYNCHRONIZED clause to the declarations

- File names referenced in the following contexts are changed from NMCOBEX<n> to ECOBEX<n> where <n> is 0, 1 or T:
  - CONSULT, LIBRARY, and SEARCH directives
  - File-systemic clause of the SPECIAL NAMES paragraph
  - OP or IN clause of the CALL or ENTER statement
USER LIBRARY MIGRATION TASKS (1 OF 3)

- The user library is an object file that the operating system links to a program file at run time
- TNS/R native user libraries:
  - Non-PIC user library is a special form of shared run-time library (SRL)
  - PIC user library is a dynamic link library (DLL)
- TNS/E supports PIC user libraries only
- Migrating either type of TNS/R native user library requires recompiling and relinking
USER LIBRARY MIGRATION TASKS (2 OF 3)

- Migrating SRL-type of TNS/R native user library
  - for PL/AL programs
    - Guardian environment: Use EPIAL with CALL SHARED directive on the command line
    - PC: Use epial with -shared flag on the command line
  - For C/C++ programs
    - Guardian environment: Use CCMP or CPPCOMP
    - OSS environment or PC: Use CC with -Wl,--shared flag. Replace SRL with SHARED pragma
  - For COBOL programs
    - Guardian environment: Use ECOBOL with SHARED, ELD, and RUNNABLE directives on the command line (or in the source code)
    - OSS environment or PC: Use cobol
USER LIBRARY MIGRATION TASKS (3 OF 3)

- Migrating DLL-type of TNS/R native user library
  - for all programs
    - DLLs on TNS/E systems are functionally identical to DLLs on TNS/R systems
    - Recompile DLLs with TNS/E native compiler and link the object codes with TNS/E eld linker using some compiler directives and same linker flags
Each linkfile and loadfile has a floating-point attribute which can contain one of the values:
- IEEE_FLOAT
- TANDEM_FLOAT
- NEUTRAL_FLOAT

The value is set either by the compiler or linker. Consistency checks are performed by the linker program, euld, and at run-time by the loader. These checks do not allow IEEE_FORMAT and TANDEM_FORMAT files to be used together, unless the floattype or float_lib_overrule attribute is set.
GUARDIAN API MIGRATION TASKS (1 OF 3)

- Procedures having different behavior
  - Different buffer alignment for GETPOOL and POOL_GETSPACE procedures
    - For TNS/R native callers, the above procedures return pointers to 4-byte aligned buffers.
    - For TNS/E native callers, the above procedures return pointers to 16-byte aligned buffers by default
    - Results may differ when the pool is already near capacity. It may not be possible to allocate as many data blocks on a TNS/E system as on a TNS/R system
  - DEBUG and PROCESS_DEBUG procedure invokes Hseries debuggers
    - Visual Inspect, Native Inspect, or Inspect
GUARDIAN API MIGRATION TASKS (2 OF 3)

- Procedures having new return values
  - PROCESSOR_UPROC procedure returns:
    - 10 indicating an EPIC processor
  - PROCESSOR_GETINFO procedure returns:
    - For name parameter: NSEx
    - For unit-type-out parameter: 10
    - For cpu-model-out parameter: 1 to 92
  - PROCESSOR_GETINFOOP procedure returns:
    - For processor type: 10
    - For processor model number: 1 to 92
    - For processor name: NSEx
    - For processor full name: HP NonStop System EPIC Model x CPU
    - Number of IPU's for NSMA
  - TOSVERSION and REMOTETOSVERSION procedures return the value of R06

* Refer to the Guardian Procedure Calls Reference Manual for specific values.
GUARDIAN API MIGRATION TASKS (3 OF 3)

- Superseded procedures
  - The UDELAY procedure is deprecated; it is superseded by the MKUL.TS.UDELAY procedure
- New procedures to support process management enhancements
  - PROCESS_DELAY
  - TIMER.START
  - TIMER.STOP
  - TS_NANO.SFC'S
  - TS.UNIQUE_COMPARE
  - TS.UNIQUE_CONVERT_TO_JULIAN
  - TS.UNIQUE_CREATE
THIRD-PARTY TOOLS

- Ensure that any third-party tools you use are available for TNS/E systems:
  - Development tools
  - Code libraries
  - Operations tools
    - Schedulers
    - Performance monitors
    - Backups
HP EVOLUTION SERVICES

- HP Evolution Services for Itanium processor-based NonStop servers:
  - Evolution Assessment
    - Identify areas of the current environment that need to be addressed, including operations, hardware, and software
  - Evolution Infrastructure Planning and Design
    - Evolution Strategy Report outlining the proposed NonStop environment and recommended evolution scenario
    - Project Plan identifying the specific tasks and resources needed to deploy the final-state solution
    - Proposal for the Evolution Infrastructure Implementation service
  - Evolution Pilot
    - Configure and test hardware and software
  - NonStop Service Solutions
    - Ongoing support
The NonStop Operating System (OS) schedules processes on cores as if each of the cores were an individual processor.

The OS pre-assigns interrupt processing for a CPU to occur in IPU 0. Key interrupt processes in IPU 0 include:

- TSMSGIP - Interrupt process responsible for handling inter-processor message system interrupts
- TSSTRIP - Interrupt process responsible for handling storage interrupts
- TSCOMIP - Interrupt process responsible for handling networking interrupts

The OS pre-assigns the disk processes (DP2s) configured in a CPU to the IPUs in a round-robin manner. DP2 process groups (the set of processes supporting a given disk volume) can be migrated by the OS to lower loaded IPUs during the course of operation of the system.

The OS with the help of the processor scheduler (PS), starts user processes in the IPUs in a round-robin manner – each successive launch of a user process takes place in the next successive IPU. The PS attempts to keep each process in the IPU in which it was started for the duration of execution of the process, but will migrate the process to another IPU for either of the following reasons:

- The process IPU is busy with higher priority processes, and this process has not had time to execute on the IPU for 100ms while there are available cycles in another IPU
- To try and keep the IPUs evenly utilized by moving some processes from busy IPUs to less-busy IPUs

The PS schedules processes in each of the IPUs based on the traditional NonStop processor scheduling algorithm using process priorities and time slices. The PS maintains an individual ready list per IPU, and processes for a given IPU are scheduled on the IPU, based on the priorities of processes in that IPU.

The PS attempts to keep all the IPUs in the CPU evenly balanced, though, in general, IPU 0 is the busiest IPU in the CPU, since it handles the interrupt processing for a CPU. This IPU usage might be about 10-20% higher than the other IPUs in the processor, particularly for I/O intensive workloads.

With NonStop OS version J06.12, users can assign processes to specific IPUs using a utility called IPUCOM or an associated set of APIs. The users are thus able to influence the performance of multi-core servers either manually or programmatically. In general, users do not have to use this utility.

Some circumstances where this utility is useful are:

- A critical user process starts in IPU 0 of a processor and is being pre-empted by higher-priority interrupt processes. A process being pre-empted by higher-priority processes will tend to show a very high ready-time to cpu-busy-time ratio. In this case, it would be beneficial to assign this process to another IPU.
- The TMF transaction monitor ($TMP) starts in IPU 0 and is being pre-empted by higher priority interrupt processes. In this case, it would be beneficial to assign $TMP to another IPU thereby increasing transaction throughput for the system.
- Workloads require placement of processes on IPUs before any load has been applied to the system.
- Certain heavyweight processes need to be bound to IPUs to not conflict with each other.

Using this utility can preclude the Process Scheduler from addressing balancing issues, so it must be used with great care.

Appendix – 2
In multi-core CPUs, a process runs in a specific IPU of the CPU. The IPU to which a process is assigned is referred to as the IPU affinity of the process. By default the IPU affinity can be dynamically changed by the process scheduler. You can override this for a specific process, binding it to a specific IPU via the IPUAFFINITY_SET_ procedure. The current IPU affinity of a process can be obtained via the IPUAFFINITY_GET_ procedure.

The IPUAFFINITY_CONTROL_ procedure can be used to override the process scheduler controls more generally to turn off dynamic load-balancing on all soft-affinity processes (see the definition of "soft affinity" below) or on all DP2 processes in the specified CPU.

The binding between IPUs and processes can only be done after the process is created.

A process has one of the following types of IPU affinity, known as its IPU affinity class:

- **Hard**: the process can only be run on a specific IPU and is not subject to any kind of movement.
- **Group**: only applies to DP2 process groups (the one to eight processes that compose a disk volume). The group as a whole can be moved from IPU to IPU but only as the whole group.
- **Dynamic**: only applies to system processes known as Interrupt Processes (IPs) and Auxiliary Processes (APs). All of these processes other than the TSMMSGIP, TSCOMP, and TSSTRIP can run on any IPU as selected by the low level software for optimum response time, and are not subject to user control of the IPU placement. The TSMMSGIP, TSCOMP, and TSSTRIP processes are the only dynamic affinity processes that are subject to user control of their IPU affinity.
- **Soft**: all user processes and any other processes which do not fall into one of the other categories.
- **Soft Bind**: Soft Affinity processes whose IPU affinity has been set via the IPUAFFINITY_SET_ procedure.

The IPU affinity class of a process can be obtained via PROCESS_GETINFOLIST_attribute_136.

The IPUAFFINITY_SET_ procedure is used to bind a process to an IPU. Once set, the process will only run on the associated IPU. This procedure can be used on all user processes and many system processes. In particular it can be used on the interrupt processes TSMMSGIP, TSCOMP, and TSSTRIP, $TMP and DP2 process groups. The procedure is also used to disassociate a process from an IPU, thereby allowing the process scheduler to control any subsequent IPU assignments of the process.

The IPUAFFINITY_GET_ procedure can be used to get the current IPU a process associated with, as well as an indicator if the process can be the target of an IPUAFFINITY_SET call.

The IPUAFFINITY_CONTROL_ procedure is used to control Process Scheduler characteristics. It can cause the process scheduler to stop changing all processes with soft affinity or all DP2 processes (group affinity processes). It can also be used to disassociate all bindings that were created via IPUAFFINITY_SET_.

Using IPUAFFINITY_SET_ and IPUAFFINITY_CONTROL_ restricts the function of the process scheduler to keep the CPU (that is all its IPUs) in balance. Much care should be exercised when utilizing these functions as an inappropriate choice can cause severe performance problems.

References:
Guardian Procedure Calls Manual
Guardian Programmer’s Guide
You use the IPUCOM program as a command line interface which can be used to set, reset, or display an IPU number associated with a process. It can also be used to set or display CPU-wide controls. Since the primary and backup processes of a process pair are on different CPUs, setting an association for each of them requires two invocations of IPUCOM. For processes that are not a part of a process pair, use either the -pname or -bname option to specify the process name. When -pname or -bname is used with the -cpu option, the process specified is the process named <name> in the specified CPU regardless of its primary or backup status.

The specified <name> or <pin> and <cpu-number> are all relative to the system where IPUCOM is running.

Specifying a system name in <name> (for example: \MYSYS.$XYZ) is not allowed and results in an error from IPUCOM. IPUCOM expects only a local process name in <name>.

The following definition apply to the syntax shown above:

- **-pin <pin>**: specifies the PIN of the process whose association is to be set.
- **-pname <name>**: specifies the primary process named <name> for a process pair.
- **-bname <name>**: specifies the backup process named <name> for a process pair.
- **-balanceSA**: indicates the process scheduler to enable (on, default) or disable (off) the balancing of Soft Affinity processes.
- **-balanceDP2**: indicates the process scheduler to enable (on, default) or disable (off) the balancing of DP2 processes.
- **-initialState**: indicates the process scheduler to put the balancing back to the initial state of the scheduler, and to clear all associations established via IPUAFFINITY_SET_.
- **<cpu-number>**: indicates the CPU on which to apply the change. A value of -1 indicates that the change should be applied to all CPUs.
Take control of your future with training from HP

Visit: www.hp.com/learn

Follow us on Facebook:
www.facebook.com/HPEnterpriseBusiness

Follow us on Twitter:
www.twitter.com/TSatHP