

C/TAL Non Stop Environment/Multi-Threading

- **NSE/MT Offers Rapid Fault Tolerant Application Development:**
 - Powerful and easy to use C and TAL application program interface
 - TAL programmers — expensive, if they can be found — are not required
 - Guardian/NSK skills and experience of course helps, but is not required
 - Transparent and automatic backup and requester (OCB) management
 - Fault tolerance testing requirements are dramatically reduced
 - Extension of CRE, supports mixed-language application development
 - **Bottom line: Significant Development Project Savings (Time == \$\$\$)**

- **Unprecedented Primary/Backup NSE/MT Process Pair Performance:**
 - Very CPU efficient active backup (like most Guardian system processes)
 - Automatic user checkpointing greatly simplifies “NonStop” user coding
 - Support for multi-threading of I/O, TCPI/IP socket, and requester operations
 - Innovative “checklisting” scheme yields very low checkpointing overhead

- **Secure, Uniform Run-Time and Operations NSE/MT Process Access:**
 - Run-time NSE/MT process pair control via NSECOM command interpreter
 - Only authorized processes (requesters) can open an NSE/MT process
 - Only authorized NSECOM users have run-time access to NSE/MT processes
 - Examine collected NSE/MT run-time process statistics at any time

- **Flexible and Efficient NSE/MT Process “Attribute Configuration”:**
 - Preferred and alternate backup CPU may be configured
 - Maintains original CPU assignments for proper load balancing after failures
 - User exits (C/TAL) enable NSE/MT functionality extensions and customizing
 - Backup co-processing capability for **true parallel processing**

- **Significantly Lower NonStop Server Application Development Costs:**
 - **Dramatically shortened application development cycle**
 - **C developers—with no prior TNS experience—quickly become productive**
 - **TNS platform-specific issues transparently handled by NSE/MT, therefore:**
 - **Developers focus on the business logic implementation only, and:**
 - **NSE/MT process pair testing requirements drastically reduced!**

The C/TAL Non Stop Environment (NSE/MT) Library:

C (and TAL) TNS Application Development Made Easy!

Ever since the **NonStop/I**, implementing process pair fault tolerance has been considered difficult. Knowledge of Guardian/NSK internals and many HP NonStop Server platform-specific capabilities is needed, and a large amount of highly specialized code — not application related — is required, even for minimal fault tolerance. Further, the Guardian checkpointing facilities used in passive backup implementations are **not available** to C programmers: writing fault tolerant applications in C requires even more deeply specialized skills, due to the added complexities of handling an active backup. And last, but not least: adequate testing of fault tolerant applications has been a complex and protracted activity.

The MicroTech **C/TAL Non Stop Environment (NSE/MT) Library** software favorably changes all this: the NSE, coexisting with the common runtime environment (CRE) to support mixed language programming, automatically and transparently handles all requester and server fault tolerance issues, turning fault tolerant program development into solely an application logic design and implementation task. No longer do “NonStop” application developers need years of TAL and Guardian experience: coding with the NSE, even junior C programmers quickly become productive!

NSE/MT programmers are not concerned with the complexities of handling TNS platform-specific things like failed and reloaded CPUs, network status changes, backup failures, requester failures (all of which may affect a fault tolerant process), misdirected, obsolete, and duplicate messages, and much more. Except for user checkpointing, this is **automatically** handled, invisible to user code — which is of course notified, if so desired, when events occur that may require some user code action.

NSE/MT consists of **(1)** a public API with NSE specific and general utility functions; **(2)** an NSE private API used by its “Non Stop Engine”; **(3)** C and TAL macros and structure templates assisting “NSE style” programming; **(4)** the NSE command interpreter (NSECOM); plus **(5)** a number of functional NSE applications to help getting up to speed with NSE/MT. **NSE developers are free to focus on their application logic, with no concerns about the fault tolerant NSE infrastructure.**

With no added user code, all NSE processes automatically share these characteristics and attributes:

- **Languages supported: C and TAL (TNS mode, MIPS and Itanium native mode)**
- **Transparent and fault tolerant “non stop engine”:** user code is only **application specific**
- **Transparent backup management:** user code is notified of backup state changes
- **Value-added security:** only authorized requesters can open an NSE process
- **Active backup:** Passive Guardian checkpointing is **not** used, enabling “Non Stop C”
- **Very CPU efficient:** low overhead, very short NSE execution path lengths
- **Multi-threading support:** Guardian file I/O, TCP/IP socket, and user requests
- **User exits:** Extend NSE functionality with user written code (C/nmc or TAL/pTAL)
- **Automatic and transparent** handling of “background” processing such as:
 - **OCB management** (“opener control blocks,” keeping track of requesters)
 - **System messages** (CPU up/down, network changes, backup down, etc, etc)
 - **Rejection** of misdirected, invalid, or obsolete application messages
 - **Saving request results** of the last (sync depth) requests for each requester
 - **Duplicate requests** (gets an immediate reply with the original response)

- **NSE/MT Process control via startup keyword parameters**, all optional, including:
 - **Requester authorization** (user specifies who is allowed to open the server)
 - **Backup CPU** number (if fault tolerance is desired)
 - **Alternate backup** in case preferred backup is not available when needed
 - **Automatic switchback** to preferred CPU when it is reloaded (after a failure)
 - **Server type** (determines I/O requesters may use (read, write, or writeread))
 - **Complete requester control, simply specify:**
 - Max number of openers the server will handle (requesters, up to 256)
 - Max number of request replies to save (“sync depth,” up to 15)
 - Max requester reply size to save (up to 16Kb)
 - **Message queue ordering:** FIFO (default), or by requester priority
 - **Automatic open** of **in** and **out** files (in primary and backup, if backed up)
 - **Waited/nowait I/O** option for **in** and **out** files
 - **Pathway server** option: terminate when last requester closes server
 - **User space** option: no need for user code to allocate segment space
 - **Memory pool** option: ready to use with standard Guardian pool functions
 - **Debug** option: very useful during development and test
 - **Trace on/off** option: very useful during development and test
 - User can supply “private” parameters intermixed with NSE keyword parameters
 - Runs as **single process** if no **backup** parameter is specified
- **Home terminal automatically opened** and ready to use
- **NSECOM command interpreter** for run-time NSE process pair control/management
- **Request service time running average** computed (display using NSECOM)
- **Automatic collection of NSE statistics** (system/user messages, failures, etc)
- **Simple checkpointing strategies** (alas, user code is on its own here... **as always**)
- **Enlightened:** runs with high or low PINs, accepts high or low PIN requesters
- **Programmer Productivity Plus (PPP) tools:**
 - **C and TAL “parametric defines,”** a collection of development time savers
 - **General utility functions**, for use with or without the NSE specific functions
 - **MIP** (MicroTech Interactive Program), a versatile test and development tool

To present a graphical idea of what NSE/MT provides, listed below is a complete “NSE style” C version of a **fully fault tolerant** “sequence number distributor” (useful when it is important to assign a sequence number from a central source to enforce “FIFO” handling and assign a unique request id): **all the above described server characteristics, in less than 50 lines of user written C code!**

For more information about **MicroTech Consulting** and our **HP NonStop Server** products and services, please visit our website at www.microtechnonstop.com, contact us by phone +1 (408) 786-5735, or send email to sales@microtechnonstop.com. From our website, you may download *NSE/MT User’s Guide* (Microsoft Word document) — of course with no obligation — and an “NSE/MT Web Seminar” (Microsoft PowerPoint presentation), and find out more about this groundbreaking new TNS development tool.

NSE/MT Example: Fault Tolerant “Sequence Number Distributor”

This program listing is the **complete** “NSE style” C source for a **fully functional** and **fault tolerant** “sequence number distributor,” with **all the server characteristics described above**. This server, coded and tested in **less than one hour** (50 non-comment C source lines), handles up to 256 requesters, easily handling a throughput of 1,000+ requests per second (S74K, Guardian G06). **NSE/MT interface code highlighted.**

```

1  #include "nseCall" nolist
2  struct _seq
3      SEQ;          /* Sequence record structure */
4  dALLOCNSEVECTOR; /* Allocate NSE address vector and standard variables */
5  short main ( void ) {
6      long long
7          trigTime; /* For "trigger time" timestamp */
8      short
9          iTime[ 8 ]; /* To compute "trigger time" */
10 /* Initialize NSE/MT structures: */
11 dINITIALIZENSE ( LNOUSERSPACE, LCMAIN );
12 /* Initialize the sequence number: */
13 SEQ.sequence = 0L; /* Start with 1 (change to -1 to start with 0) */
14 /* Compute "trigger time" timestamp (09:00:00): */
15 TIME( iTime );
16 iTime[ 3 ] = 9; /* Set trigger hour, and */
17 for ( size = 4; size > 7; iTime[ size++ ] = 0 ); /* Clear the rest... */
18 trigTime = COMPUTETIMESTAMP( iTime );
19 /* Add SEQ structure to "auto checkpoint item" (ACI list): */
20 NSEADDAUTOCHECK( SEQ.sequence, sizeof( struct _seq ) );
21 /* Main loop: */
22 while ( PIB->fRunning ) {
23     /* Extract, then process, one NSE/MT event: */
24     switch ( NSEBEGINCYCLE( PIB ) ) {
25         case LNSEUSERMSG: /* Requester READ() request received: */
26             /* Takeover point selection: */
27             switch ( PIB->TOP.function ) {
28                 case 0: /* No takeover occurred: */
29                     /* Set request time and trigger flag, increment sequence: */
30                     SEQ.timestamp = CONVERTTIMESTAMP( JULIANTIMESTAMP() );
31                     SEQ.trigger = ( SEQ.timestamp > trigTime );
32                     SEQ.sequence++;
33                     /* Set and checkpoint takeover point: */
34                     PIB->TOP.function = 1;
35                     dCHECKITEMF( PIB->TOP, sizeof( PIB->TOP ) );
36                 case 1: /* Takeover point 1: */
37                     /* Reply with SEQ structure (14 bytes): */
38                     RIB->replyError = LNSEOK;
39                     RIB->replyData = (char _far *) &SEQ;
40                     RIB->replyLength = (short) sizeof( SEQ );
41                     break;
42                 default: /* Should never occur, but safety first... */
43                     dDEBUGABORT;
44             }
45             break;
46         case LNSEBACKUP: /* Backup state change: No action needed */
47         case LNSESYSMSG: /* System message received: No action needed */
48         case LNSEIOCOMPL: /* No action needed (in this application) */
49         case LNSETIMEOUT: /* No action needed (in this application) */
50         case LNSECONTINUE: /* NSE/MT internal event, no action needed */
51             break;
52         default: /* User program logic problem... */
53             /* Show diagnostic text and abort: */
54             dFATALERROR( "User code internal error", 1, 54 );
55     }
56     NSEENDCYCLE( PIB );
57 }
58 /* Show statistics and quit: */
59 NSESHOWSTATS( PIB );
60 }

```

To implement the equivalent functionality without the NSE/MT library, the programmer would have to write **several thousand lines** of complex C code, requiring “guru level” Guardian/NSK experience!

To demonstrate and verify the reliability and functionality of NSE/MT software, it comes bundled with a complete NSE Q/A and Functional Verification application in NSE source form, as outlined in the server and requester pseudocode logic flow below.

Server Logic

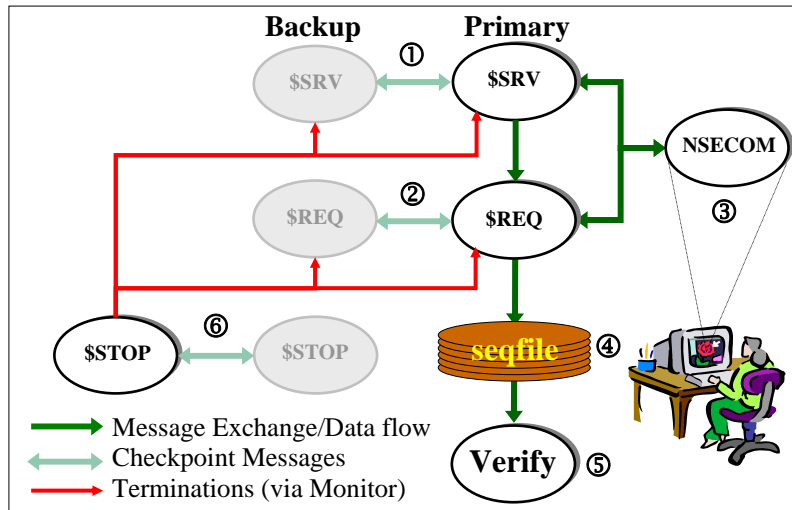
```

BEGIN
Initialize
WHILE running:
  Read $receive request
  Increment sequence number
  Checkpoint "take over point" and sequence record
  Reply with sequence record
  "Idle" checkpoint (TOP 0)
ENDWHILE
Terminate
END
  
```

Requester Logic

```

BEGIN
Initialize
WHILE running:
  Read IN file record (process $SRV)
  Checklist IN file sync info
  Checklist OUT file sync info
  Checklist "take over point" (TOP 1)
  Checkpoint sequence record buffer and checklisted data
  Write sequence record to OUT file
  "Idle" checkpoint (TOP 0)
ENDWHILE
Terminate
EN
  
```



NSE Q/A Test/Verification Application Configuration

① **Server** process pair. Opened by the \$REQ process pair (and optionally, by NSECOM). At intervals specified when starting \$STOP, either the primary or the backup process is terminated by \$STOP.

② **Requester** process pair. Opens the \$SRV process pair (via its **in** file) and the output sequence record file (via its **out** file). Also optionally opened by NSECOM. At intervals specified when starting \$STOP, either the primary or the backup process is terminated by \$STOP.

③ **NSECOM command interpreter.** Use to switch roles primary/backup target process, to examine collected run time statistics, and to stop any NSE process gracefully. **Note:** to run NSECOM, the target process must run with parameter **AUTHORIZE**, and the mapped authorization file must include object file name `\<sysname>.$system.system.nsecom`.

④ **Sequence record file.** Enscribe entry-sequenced file. The 14-byte records hold a Julian timestamp, a doubleword (32 bits) sequence number, and a Boolean “trigger time passed” flag word.

⑤ **Verification process.** Reads the sequence record file and verifies ascending sequence numbers, with no duplicates, and no numbers in the generated sequence missing.

⑥ **“Stopper” process.** In user specified intervals, this process randomly selects a primary or backup process to stop, from both \$REQ and \$SRV process pairs. The default delay time range between termination cycles is from two to seven seconds. ***This NSE based fault tolerant “stopper” program can be used to test any NSE/MT application, as a general fault tolerance testing tool.***

All the above described programs are completely self-contained, with in-source documentation.

Comparison: C Active Backup, TAL Passive Backup, and NSE/MT

Coding Task	Active Backup (C)	Passive Backup (TAL)	NSE/MT (C/TAL)
Start backup process	<code>_ns_start_backup()</code> (user code)	<code>PROCESS_CREATE()</code> (user code)	<code>NSEINITIALIZER()</code>
Establish primary and backup process communication	<code>FILE_OPEN()</code> (protocol is application specific, using Guardian file system functions)	<code>CHECKMONITOR()</code> in backup, <code>CHECK*()</code> functions in primary	
Monitor primary CPU by backup process	<code>MONITORCPUS()</code> (backup explicitly coded to check for CPU or process failure messages) (user code)	<code>MONITORCPUS()</code> , <code>CHECKMONITOR()</code> (additional user code needed)	
Monitor backup CPU by primary process	<code>PROCESS_GETPAIRINFO()</code> (or check for backup communication failures)	Same as C active backup (additional user code needed)	
Backup management	Additional user code	Additional user code	<code>NSEBEGINCYCLE()</code> and its "companion" <code>NSEENDCYCLE()</code>
Requester management	Additional user code	Additional user code	
Remote requesters	Additional user code	Additional user code	
OCB management	Additional user code	Additional user code	
OCB checkpointing	Additional user code	Additional user code	
Duplicate messages	Additional user code	Additional user code	
Misdirected messages	Additional user code	Additional user code	
Obsolete msg syncids	Additional user code	Additional user code	
Sending file sync state info to backup	<code>_ns_fget_file_state()</code> , then send sync information to backup	<code>CHECKPOINT*()</code> file sync block	<code>NSECHECKITEM()</code> <code>NSEADDAUTOCHECK()</code>
Sending process state information to backup	Primary sends data to backup, which updates its own memory	<code>CHECKPOINT*()</code> user data block	
Setting file sync state info in backup	<code>_ns_fset_file_state()</code> after receiving primary file state info	<code>CHECKMONITOR()</code> automatically updates process state	
Checkpoint item list	Additional user code	Not available	
Implementing backup memory updates	Additional user code	Automatic, <code>CHECKMONITOR()</code> updates memory in backup	<code>NSECHECKPOINT()</code>
Defining backup take-over ("restart") points	Primary sends control state information to backup	Restart point is return from most recent <code>CHECKPOINT*()</code>	
Paired file open	<code>_ns_fget_file_open_state()</code> , send open state to backup, which calls <code>_ns_backup_fopen()</code>	<code>FILE_OPEN()</code> (primary), <code>FILE_OPEN_CHKPT()</code> (backup)	<code>NSEFILEOPEN()</code>
Paired file close	Not available (additional user code)	<code>FILE_CLOSE()</code> (primary) <code>FILE_FLOSE_CHKPT()</code> (backup)	<code>NSEFILECLOSE()</code>
Paired setmode	Not available (additional user code)	<code>CHECKSETMODE()</code>	<code>NSESETMODE()</code>
Paired file control	Not available (additional user code)	Not available (additional user code)	<code>NSECONTROL()</code>
Primary/backup switch	Additional user code	<code>CHECKSWITCH()</code>	<code>NSECHECKSWITCH()</code>
Coexist with CRE	Additional user code	Additional user code	NSE/MT standard
Run time parameters	Additional user code	Additional user code	
Run-time statistics	Additional user code	Additional user code	
Authorized requesters	Additional user code	Additional user code	Parameter <code>AUTHORIZE</code>
Original CPU config	Additional user code	Additional user code	Parameter <code>CONFIG</code>
Receive queue order	Additional user code	Additional user code	Parameter <code>FIFO</code>
Number of openers	Additional user code	Additional user code	Parameter <code>OCB</code>
Define memory pool	Additional user code	Additional user code	Parameter <code>POOLSIZE</code>
Requester sync depth	Additional user code	Additional user code	Parameter <code>SYNCDEPTH</code>
Program action trace	Additional user code	Additional user code	Parameter <code>TRACE</code>
Alternate backup CPU	Additional user code	Additional user code	Parameter <code>ALTBACKUP</code>
Auto checkpointing	User code does all checkpointing	User code does all checkpointing	Parameter <code>AUTOCHECK</code>
Requester I/O type	Additional user code	Additional user code	Parameter <code>SERVER</code>
Runtime appl control	Additional user code	Additional user code	NSE/MT standard
Test requirements	All code	All code	Application logic only!



1333 Colonia, Yap Islands, Federated States of Micronesia

Web: www.microtechnonstop.com

E-mail: sales@microtechnonstop.com

Services

TNS systems **performance analysis and tuning**

Pathway systems configuration **analysis and tuning**

Custom TNS **software development** (pTAL/TAL, nmc/C, nmcobol/COBOL85)

On-site or remote application design reviews, technical support, problem analysis and resolution

Software

NSE/MT (Non Stop Environment, Multi-Threading): Develop fully fault tolerant multi-threaded process pairs in nmc/C or pTAL/TAL, as easy as coding a batch program. The parameter driven active backup “Non Stop Engine” transparently and automatically provides **complete fault tolerance** (user defines application specific takeover points). Supports single- as well as multi-threaded code. Preferred and alternate backup can be specified. *HP NonStop S-series Servers or later only.*

QMP (Queue Manager Process): A fault tolerant (NSE/MT based), flexible, efficient, and easy to use memory based multi-queue manager. Dynamically create up to 63 named queues, or use the “general” unnamed queue, for any application that needs to queue data—FIFO, LIFO or “push-pop” stack—for later processing. QMP queues are very easy to use, accessed using standard file system **read**, **write**, and **writeread** calls. *HP NonStop S-series Servers or later only.*

Training

Programming With NSE/MT: A 3-day course, teaching students all they need to become proficient with the MicroTech NSE product. Students must have a working knowledge of either nmc/C or pTAL/TAL (both is also ©K). Lecture sessions before lunch, afternoons dedicated to labs, implementing a number of complete and useful NSE applications. This course is conducted at the customer site, accommodating both C and TAL programmers (*customer provides access to development system*).

Guardian Architecture Made Easy (GAME): Guardian/NSK internals course developed specifically for TNS systems programmers and application developers, praised by students as “*the best Tandem course I ever attended.*” Conducted at the customer site, or you are invited to visit us at MicroTech in **Colonia** on tropical **Yap Islands** for the ten-day all-inclusive “**Yap GAME Special**” package: GAME course, airport transportation, hotel, all meals, and one reef diving tour included.

(The GAME material is current as of HP NonStop Server S Series, Guardian/NSK release G06.24)