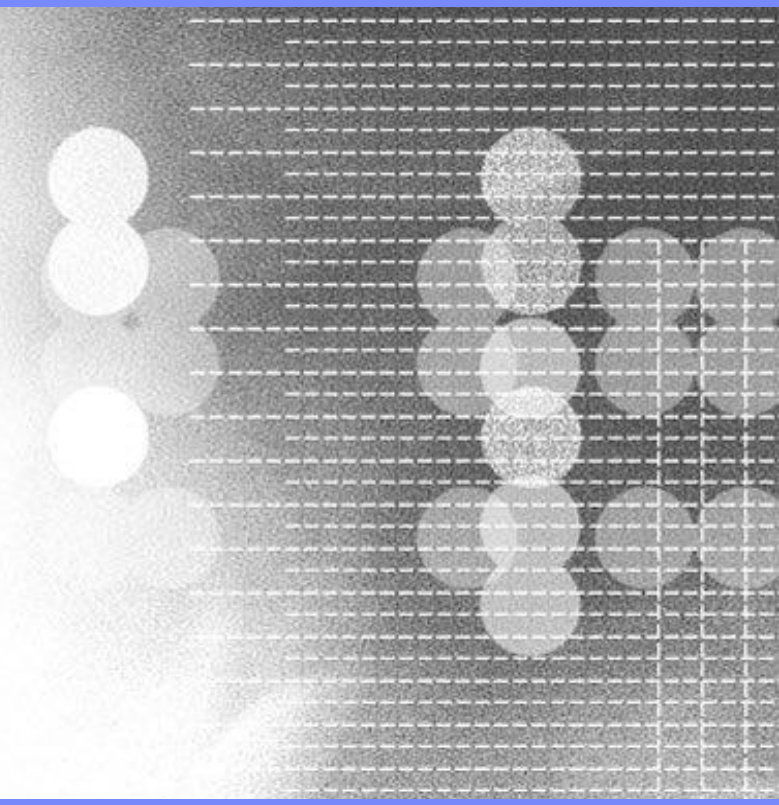
z/TPF EE V1.1
z/TPFDF V1.1
TPF Toolkit for WebSphere® Studio V3
TPF Operations Server V1.2

IBM Software Group

TPF Users Group Fall 2005

z/TPF Programming Model



Name : Michael Shershin
Venue : Main Tent

AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Agenda

- z/TPF Programming Model
- z/TPF PUT 01 Enhancements

What is z/TPF Programming Model?

- Large memory areas
- Large programs
- For assembler
  - Subroutines
  - Stack

How does this help you?

- Reduce cost of doing business
  - Improve productivity
    - Less resources required to develop a product
  - Improve quality
    - Less problems means less customer impact
    - Less resources required to correct problems

Large Memory – ECB Private Area

- ECB Private Area
  - Minimum size is 4 meg
  - Maximum size is 16 meg
- Consider: have ECB cache
  - Keep records in private area longer
    - Use DECBs
    - Don’t need to free up ECB data levels
  - Processes which retrieve the same record multiple times
  - Improve performance – less I/O needed

Large Memory - ECB Usage

- Consider: single ECB rather than multiple ECBs to do a task
  - If multiple ECBs are created due to lack of memory per ECB
  - Use DECBs
  - Use parallel I/O
  - Improve productivity - write less code
  - Improve quality – less code means less chance of errors
- Consider: allow ECB to dynamically move between I-Streams
  - Processes which don't use I-Stream unique fields or records (Globals)
  - Improve I-Stream utilization balance

Large Memory – ECB Usage

- Consider: longer running ECBs
  - More tasks can be synchronous
    - Park ECBs
    - Do not need to exit and create new ECBs when response is received
  - Rather than allowing 500 active ECBs, consider 1000 or 2000 or 5000 active ECBs
  - Improve productivity
    - Write less code
    - Easier to write code
      - Stack is maintained
      - ECB is maintained
      - ECB private area is maintained
  - Improve quality
    - Less code means less chance of errors
    - Less complexity means less chance of errors

Large Memory
Use Large Data Structures

- Use files rather than records
  - One API call gets complete file rather than finding many records
- Improves productivity
  - Less code needed
- Improves quality
  - Less code needed means less chance of errors
  - Less complexity means less chance of errors
- On DASD keep large data structures in the File System
  - Use Memory File System (MFS) for temporary files
  - Use Fixed File System (FFS) or Pool File System (PFS) for better performance
    - Data in FFS and PFS is processor unique

Large Memory
Use Large Data Structures

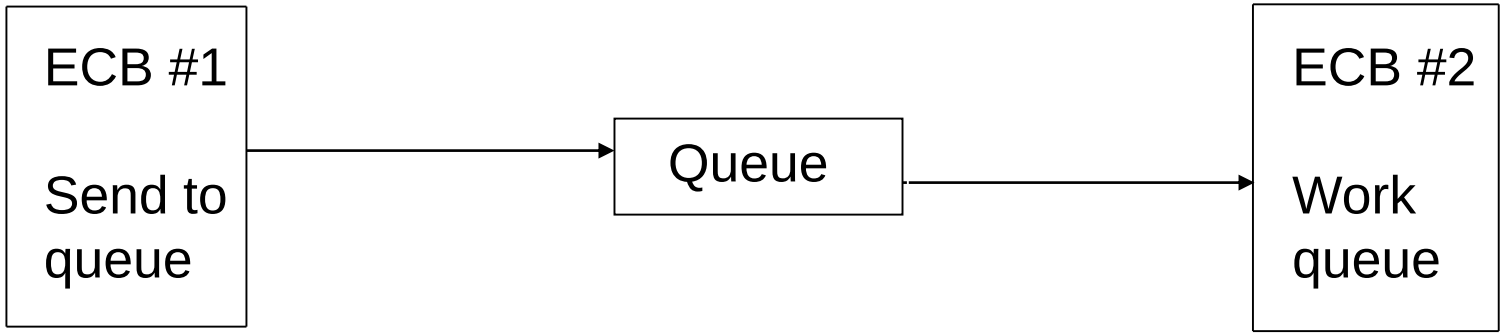
- For ECB unique data structures use ECB Heap
  - Use heap for file system accesses
- Two ECB Heaps
  - 64-bit ECB Heap
    - APIs
      - MALOC, CALOC, RALOC with HEAP=64 in assembler
      - malloc64(), calloc64(), realloc64() in C
  - 31-bit ECB Heap
    - APIs
      - MALOC, CALOC, RALOC with HEAP=31 (default) in assembler
      - malloc(), calloc(), reallloc() in C
    - Consider: 100 meg or 500 meg heap size
    - Consider: many ECBs each using large amounts of Heap
      - 1000 ECBs each using 100 meg = 10 gig

Large Memory
Use Large Data Structures

- For memory resident data use Format 2 Globals
  - Use files rather than 381, 1055, or 4K records
- Format 2 Globals
  - In core data structures
  - Fast access
  - No limits on number and size of globals
  - Easier to use than Format 1 globals in TPF 4.1

Large Memory Trace Log Example

- Trace log writes all ECB trace entries from one ECB to tape or file system.
  - Traced ECB puts trace entries into queue
  - Second ECB works on queue
  - Large memory allows for large amount of data on queue
    - 50 meg / 100 meg / 500 meg
  - Large in core data structure



Large Memory - ECB Trace

- Improve diagnostic capabilities
  - Improve quality – more trace data to help solve problems
- Consider: increase number of trace items per ECB
  - Set to 200, 500, 1000 or more trace items per ECB trace buffer
  - Keep more diagnostic data for every ECB
- Consider: define multiple trace groups
  - Each middleware package to have own trace group
  - For transactions which use services from multiple applications, have one trace group per application

Large Memory ECB Trace Information

- Add free form information to ECB trace
  - Function tpf_trace_info()
  - Up to 80 characters
- Consider: at various points in an application add free form text to trace
  - Can be fixed message – “Routine 1 entered” or “Routine 1 complete”
  - Can be variable message
    - Contents of a register or registers
    - Contents of a work area or a record
      - sprintf() to format line to use
      - WTOPC BUFFA=addr,DISP=NONE to format line in assembler
  - Build into the program design

Large Memory ECB Trace Information

- Always on diagnostics
  - Use in error paths
  - Use to keep track of status
    - In large loop call tpf_trace_info() every 1000 times thru a loop
  - Not good for high frequency mainline paths
    - Performance
    - Usually high frequency routines do not need diagnostics
      - Well tested

Large Programs

- Grow existing assembler programs
  - Keep like function in one program
- No need to split programs
  - Improve productivity
    - Do not need to go through steps to create a new program
- Use:
  - CLINKC
  - Multiple base registers
  - Single base register for first 4K only
    - No base register beyond first 4K
    - Constants in first 4K
  - Baseless program
  - CALLC

Large Programs / Subroutines CLINKC

- Standardized subroutine support for assembler
  - Improve quality – structured linkage reduces errors
- Macros are
  - CLINKC .. Call subroutine
  - SLINKC .. Start subroutine
  - RLINKC .. Return from subroutine
  - ELINKC .. End subroutine
- SLINKC saves registers R0 – R13 on entry
  - R14 is linkage register
  - R15 is pointer to stack
- RLINKC return restores R0 – R13
  - One register can be used to return information
  - Multiple RLNKCs can be coded for each SLINKC

CLINKC - example

```
                    CLINKC RTN=CPSI5000_ECBTRACE    COLLATE it
...
...
...
CPSI5000_ECBTRACE DS 0H
                    SLINKC BASE=R6
...
...
...
                    RLINKC ,                        Return
                    ELINKC
```

Large Programs / Subroutines Use of Relative instructions

- Assembler programs new routines may not need a base register
  - Branch instructions can use:
    - Branch Relative Under Condition - BRU or J
    - Branch Relative and Save - BRAS or JAS
    - Branch Relative Long – BRL
  - Avoid use of literals
    - Use immediate instructions
      - Load halfword immediate – LHI or LGHI
      - Add halfword immediate – AHI or AGHI
      - Multiply halfword immediate – MHI or MGHI
    - Constant handling
      - Use Load Address Relative Long (LARL) to point to constants
      - Put all constants in first 4K of program and use one base register

Branch Relative - example

- Need to insert code, but new code will cause the program to be larger than 4K
  - Instead insert new code at end of the program and jump to it

```
          JAS    R1,CLH1HLDA          GO READ IN KPT A W/ HOLD
...
...
...
CLH1HLDA DS     0H
         GETKC KPT=A,DATA=R7,ERROR=CLH1ER04,HOLD GET KPT A
         BR     R1                            RETURN
```

Large Programs - fork()

- Create a new process or ECB
  - New ECB starts after fork() call
  - New ECB has copy of original ECB:
    - Heap
    - Stack
    - Variables
  - New ECB does not have copy of
    - ECB
    - Traditional ECB private area
- Alternative to traditional TPF create ECB
  - CREMC / CREDC / CREXC / CREEC / SWISC
- Not intended for high frequency mainline paths
  - Copying heap, stack, and variables may be expensive
  - Use in lower frequency routines
- Available in TPF 4.1 and z/TPF
  - Larger memory in z/TPF allows more usage of fork()

fork() – example

```
/*****************************************************/
/* Create a child ECB to calculate inuse count       */
/*****************************************************/
pid = fork();
if ( pid == -1 ) {
   recount_err = TRUE;
   break;
}
else if ( !pid ) {
   /*****************************************************/
   /* It's is the child process here:                   */
   /*****************************************************/
   rc = bbld_recount(ipartOrdinal, lastUtility);
   postc(&eventInf, EVENT_CNT, rc);
   exit(0);
}
else {
   /*****************************************************/
   /* It is the parent process here:                    */
   /*****************************************************/
   evinc(&eventInf);
   ipartOrdinal++;
}  /* end of if-elseif-else */
```

fork() - Benefits

- Keep all code in same program
  - Don’t need to allocate a separate program
- Easy to pass data to child ECB
  - Don’t need to create special interface
    - Don’t need to rely on data in EBW area
    - Don’t need to rely on data level 0
  - Can use existing memory areas in parent ECB to pass data
    - Heap
    - Variables
- Improves productivity
- Improves quality

Large Programs - SWISC IMMEDIATE

- Switch a program to a specified I-Stream
  - Start immediately following the SWISC call
- APIs
  - SWISC TYPE=IMMEDIATE
  - swisc_immediate()
- Benefits
  - Don’t need to call a separate program
  - Keep all of the code in the same program
  - Improve productivity
  - Improve quality
- Consider:
  - fork() followed by swisc_immediate() to load balance new ECBs
  - Use when cycling through all I-Streams

SWISC IMMEDIATE - example

```
            CINFC R,CMMIST,F,REG=R14     I-STREAM STATUS TABLE
            DCTIST REG=R14               MAP I-STREAM TABLE
            XR    R5,R5                  CLEAR THE REGISTER
            ICM   R5,B'0011',ISTACTIS    GET NUMBER OF I-STREAMS
            DROP  R14
            XR    R0,R0                  CLEAR THE REGISTER
            USING DCTPFX,R0              PREFIXED PAGE
CVMNPFX1    DS    0H
            SWISC TYPE=IMMEDIATE,IS=R5   CHANGE I-STREAM
            ALG   R4,PFX2ILCTR           ADD VALUE TO IL CTR
            STG   R4,PFX2DLTHV           STORE NEW THRESHOLD
            BCT   R5,CVMNPFX1            LOOP FOR EACH I-STREAM
```

Large Programs / Subroutines Service Routines in Program

- Move macro service routines into program
  - Inline
  - Call routine in program
- Several IBM service routines are in FINIS
  - ALASC
  - CALLC
  - ENTxC (partial)
  - FLIPC
  - GLOBLC
- Improve performance
  - Reduce linkage costs

FLIPC - example

```
          FLIPC D0,DA                    ENSURE LEVEL 0 AVAILABLE
+         BRAS  R14,FLIPCDAD0
.....
+*-------------------------------------------------*
+*  FLIPC stub code.                               *
+*-------------------------------------------------*
+FLIPCDAD0  DS 0H
+         XC    CE1FAA(CE1FA1-CE1FA0),CE1FA0
+         XC    CE1FA0(CE1FA1-CE1FA0),CE1FAA
+         XC    CE1FAA(CE1FA1-CE1FA0),CE1FA0
+         XC    CE1CRA(CE1CR1-CE1CR0),CE1CR0
+         XC    CE1CR0(CE1CR1-CE1CR0),CE1CRA
+         XC    CE1CRA(CE1CR1-CE1CR0),CE1CR0
+         XC    CE1FXA(CE1FX1-CE1FX0),CE1FX0
+         XC    CE1FX0(CE1FX1-CE1FX0),CE1FXA
+         XC    CE1FXA(CE1FX1-CE1FX0),CE1FX0
+         XC    CE1SUD+10(1),CE1SUD+0
+         XC    CE1SUD+0(1),CE1SUD+10
+         XC    CE1SUD+10(1),CE1SUD+0
+         BR    R14
```

Subroutines - CALLC

- Ability to call a C function from assembler
  - Interface for legacy assembler to use C
- Ability to write a routine once and call regardless of programming language
    - Improves productivity
    - Improves quality
- Infrastructure to allow migration of applications from assembler to C
- Consider:
  - If a program needs to be split, write new support in C and use CALLC
  - If a common routine, write in C and have assembler macro do setup and CALLC
    - IBM GETKC macro does this

CALLC - example

```
         LG    R2,EBW008              Item pointer array
         LG    R4,EBW016              nbr of items to be collated
         LG    R5,EBW000              ptr to COOLL structure
*
*  Define prototype for tpf_collate_trace_items() function
*
         CPROC RETURN=i,tpf_collate_trace_items,(p,p,i)
*
         CALLC tpf_collate_trace_items(R5,R2,R4)
         LGR   R0,R15
```

CALLC – example in macro

```
          MACRO ,
&LABEL    RELKC &KPT=,&PROC=
....

          J     RELKCC&SYSNDX
          DS    0F
RELKCO&SYSNDX DC H'0'
              DC B'00000000'
              DC B'000&O(5)0000'
RELKCK&SYSNDX DC C'&KPT'
RELKCC&SYSNDX DS 0H
          CPROC RETURN=i,tpf_relkc,(c,i,i)
          LARL  R14,RELKCO&SYSNDX
          PUSH  USING
          USING RELKCO&SYSNDX,R14
          CALLC tpf_relkc(RELKCK&SYSNDX,RELKCO&SYSNDX,&PARM1)
```

Shared functions

- Can use in TPF 4.1 and z/TPF
- Easier to use in z/TPF
  - Creation
    - Library interface script is not needed
    - No need to run LIBI
  - Build
    - maketpf provides simple easy to use interface
  - No limits on library functions
    - TPF 4.1 limited to 1024 libraries
    - TPF 4.1 limited to 1024 functions per library
  - Improved debug capability
    - C function trace
    - Messages and trace includes module name, object name, and trace name
  - Improved performance over TPF 4.1 DLLs

Shared Functions - Examples

- Library CVV0 – alter core, display core
  - Collection of C functions used to alter and display core memory
    - acor_proc()
    - dcor_proc()
    - Contains 33 functions
  - Users
    - acor_proc() called by
      - ZACOR / ZADCA / ZACNF / ZAGBL
    - dcor_proc() called by
      - ZDCOR / ZDDCA / ZDCNF / ZDGBL

Shared Functions - Examples

- Library CVBV – disassembler
  - CVBV_Disassemble()
  - Contains 9 functions
- Users
  - Library CVV0
    - All of the users of CVV0
  - Debugger – CUDA
  - ZDPGM - CVBN

Shared Functions – Example makefile

```
#  DESCRIPTION.....  ALTER and DISPLAY package
APP := CVV0
APP_EXPORT := ALL

# External LIB References
LIBS := CVBV
LIBS += CNG0

# Environments needed for build
maketpf_env := base_rt
maketpf_env += system

# C segments
C_SRC := cvv0.c
CXX_SRC := cvv0f1.cpp

# Assembler segments
ASM_SRC := cvv0f2.asm
ASM_SRC += cvv1f1.asm

# Include maketpf build rules
include maketpf.rules
```

Shared Functions – Example makefile

```
##############################################################
# DESCRIPTION.. This is the makefile for ZDCOR               #
##############################################################
APP := CVV1
APP_ENTRY := CVV1
APP_EXPORT := ENTRY

# External LIB References
LIBS := CVV0

# Environments needed for build
maketpf_env := base_rt

# C segments
C_SRC := cvv1.c

# Include maketpf build rules
include maketpf.rules
```

Shared Functions - Benefits

- Write code once, use multiple times
  - Improves productivity
  - Improves quality
  - Reduces total cost to test
- Use of C functions provides well known standard interface which is highly flexible
  - Improves quality
- Migration from 31-bit to 64-bit is easier

Stack

- Available to all programs including assembler
- Ability to dynamically allocate more space on Stack
  - ALLOC in assembler
  - alloca() in C
- Consider:
  - Variables on the stack rather than in ECB
    - Handle variables in assembler like variables in C
    - Easier to make programs iterable
  - Save registers on stack rather than in ECB
  - Improve quality – work area is only used for this invocation of this ECB
    - Strong benefit when changing legacy assembler programs

Storage Protect Override

- Ability to write into Key 1 and Key 9 storage
  - Key 1 storage = ECB private area
  - Key 9 protected areas
    - Globals
    - Newly obtained system heap
- Use when updating protected areas and stack, ECB, or other parts of the private area
- APIs
  - GLMOD OVERRIDE=YES
  - KEYCC OVERRIDE=YES
  - tpf_stpoc()
- Cannot give up control when storage protect override is active
  - SERRC E,06401A
- Improve productivity
  - Less restrictions means less code is needed

tpf_stpoc() - example

```
 tpf_stpoc(TPF_STPOC_ON,NULL);
/*****************************************************************/
/*  Get lock for the queue and get the first entry.              */
/*****************************************************************/
   lockc(&(queue->itrqlock),LOCK_O_SPIN);

   data_blk=ctlg_Get_Next();
/*****************************************************************/
/*  Now that we have the block we unlock the queue.              */
/*****************************************************************/
   unlkc(&(queue->itrqlock));
/*****************************************************************/
/*  Turn on storage protection override.                         */
/*****************************************************************/
   tpf_stpoc(TPF_STPOC_OFF,NULL);
```

C Environment

- C environment exists for every ECB
  - No startup costs on enter to first C program
- Consider: write short lived highly accessed programs in C
  - Improve productivity – use high level language
  - Improve quality – use high level language

Other Enhancements

- ENTRC PRGM,SAVEREGS=YES
  - Saves R0 – R8 before PRGM is entered.
  - Restores R0 – R8 on return
  - Improves quality – data in registers are known

```
          TM     FM_AIB,AIB_UEXT      Is user exit active?
          BNO    NO_EXIT              No, continue
          ENTRC  UMEX,SAVEREGS=YES    Call user exit
NO_EXIT   DS     0H
```

Other Enhancements

- Extended Register Save
  - Ability to save / restore R10, R11, R12, R13 across general macros
    - Use more registers
  - BEGIN EREGSAVE=YES
    - Enables support in entire program
  - DEFBC EREGSAVE=YES
    - Enables support for a part of the program
  - Improves productivity – allows more registers to use

Other Enhancements

- Tag an ECB Heap buffer
  - To tag an ECB Heap buffer
    - tpf_eheap_tag()
    - EHEAPC FUNC=TAG,TAG=,ADDR=
  - Locate a tagged ECB Heap buffer
    - tpf_eheap_locate()
    - EHEAPC FUNC=LOCATE,TAG=,ADDR=
  - Improve productivity – write less code to keep track of ECB Heap buffers

DECB

- Exists in TPF 4.1 and z/TPF
- Must use for I/O with 8-byte file addresses
- Can be used for I/O with 4-byte file addresses
- Use when more than 16 ECB data levels are needed
- Use in commonly accessed utilities that do I/O
  - Alternative to DETAC / ATTAC
  - Do not need to determine whether an ECB data level is in use
    - Allocate a new DECB when needed
- Improve quality
  - Do not need to manage multiple uses of a specific ECB data level
  - Separate DECBs for each function

Performance Concerns

- Not all features are best for use in high frequency mainline routines
  - TPF File System (TFS)
    - FFS / MFS / PFS are reasonable to use in high frequency mainline routines
  - fork()
  - tpf_trace_info()
- In the lab we have experienced:
  - Most programs are not executed with high frequency
    - Consider using these features to improve productivity
  - Most development is not in high frequency routines
    - Commands
    - Error routines
    - Supporting functions
- If true in your environment, consider using these features

z/TPF PUT 01 Enhancements

- PJ30297 Short term pool logging
- PJ30422 Input list bypass
- Available now on the web

z/TPF Diagnostics

- Education session on Wednesday 10/19/2005
  - An Introduction to Dump and Trace Analysis on z/TPF
    - By Allan Feldman
    - Please bring your laptops
    - Please bring USB memory sticks which will be given out at the IBM hospitality suite Monday evening.

Trademarks

IBM and z/TPF are trademarks of International Business Machines Corporation in the United States, other countries, or both.

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

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, Celeron, Intel Centrino, Intel Xeon, Itanium, Pentium and Pentium III Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Notes

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.