Topics to be Discussed

1. EVALUATE verb
2. Scope terminators
3. NOT phrases.
4. COBOL-89 Functions
5. Calling MPE Intrinsics
6. Implementing Control-Y trapping
7. File Organization & Access – Sequential /Relative/Indexed
8. Appendix of Sample COBOL Code and Demo programs
   - Sample 1 - Sample code to show use of File Status code in READ statement.
   - Sample 3 - Sample code to show use of Date Validation Intrinsics.
   - Sample 4 - Sample code to show use of a separately compiled module to remove data from 'global' working storage to provide a form of encapsulation.
   - DEMO1-14 - Demo program to demonstrate 14 different COBOL features
   - DEMO16 - Demo program to demonstrate the use of circular files.
   - DEMO17 - Demo programs to demonstrate the use of message files.
   - DEMO18 - Demo program to demonstrate Data Base Access and KSAM Files.
   - DEMO19 - Demo program to demonstrate use of TZ variable and GMT by printing the current date in a number of different time zones.

Section 1 - EVALUATE verb

The EVALUATE statement is a multicondition, multibranch case statement. It evaluates sets of conditions. The first time all the conditions in a set are true, it executes the associated group of statements.

You can always write an EVALUATE statement that is equivalent to a nested IF statement, but you cannot always write a nested IF statement that is equivalent to an EVALUATE statement. This is because there is a limit to the depth that IF statements can be nested, but an EVALUATE statement can specify any number of conditions.
HP COBOLII/IX evaluates the clauses in an EVALUATE statement in order therefore, for fastest execution, order the clauses from most frequent value to least frequent value.

Example 1:
```
EVALUATE INPUT-FLAG
    WHEN "Y"       MOVE PROD-NO     TO OUTPUT-REC
    WHEN  "N"      MOVE SPACES      TO OUTPUT-REC
    WHEN "Q"       PERFORM TERMINATION-ROUTINE
    WHEN "C"       CONTINUE
    WHEN OTHER     PERFORM GET-INPUT
END-EVALUATE
```

Example 2:
```
EVALUATE TRUE
    WHEN FLAG-Y      MOVE PROD-NO     TO OUTPUT-REC
    WHEN FLAG-N      MOVE SPACES      TO OUTPUT-REC
    WHEN FLAG-Q      PERFORM TERMINATION-ROUTINE
    WHEN FLAG-C      CONTINUE
    WHEN OTHER       PERFORM GET-INPUT
END-EVALUATE
```

Example 3:
```
EVALUATE NUMBER-OF-THINGS
    WHEN 1
    WHEN 2         DISPLAY "The value is 1 or 2"
    WHEN 3         STOP RUN
    WHEN OTHER     DISPLAY "Input Again"
END-EVALUATE
```

Example 4:
```
EVALUATE HOURS-WORKED   ALSO EXEMPT
    WHEN 0                  ALSO  ANY    PERFORM NO-PAY
    WHEN NOT 0              ALSO  "Y"    PERFORM SALARIED
    WHEN 1 THRU 40          ALSO  "N"    PERFORM HOURLY
    WHEN NOT 1 THRU 40      ALSO  "N"    PERFORM OVERTIME
    WHEN OTHER       DISPLAY HOURS-WORKED " &  " EXEMPT
                     MOVE 0 TO HOURS-WORKED
END-EVALUATE
```

Section 2 - Scope Terminators

The **Explicit Scope Terminators**, END-*verb*, terminate the scope of the last instance of the –*verb*.

END-ACCEPT	END-IF	END-START
END-ADD	END-MULTIPLY	END-STRING
END-CALL	END-PERFORM	END-SUBTRACT
END-COMPUTE	END-READ	END-UNSTRING
END-DELETE	END-RETURN	END-WRITE
END-DIVIDE	END-REWRITE	
END-EVALUATE	END-SEARCH	

Without the END-*verb* statement, the above *verbs* would be conditional statements and therefore could not be used wherever an imperative statement is required.

By always using an END-*verb* statement, where applicable, the period is only needed to terminate a paragraph in the PROCEDURE DIVISION.

The **Implicit Scope Terminators** are –

- The separator period at the end of a Sentence, Paragraph or Section
- The ELSE, WHEN, AT END, ON EXCEPTION, ON SIZE ERROR, ON OVERFLOW, ON INPUT ERROR, INVALID KEY are examples of implicit terminators.

Section 3 - NOT Phrases

A NOT phrase specifies a set of statements to be executed if an exception condition does not exist. The NOT phrases are –

NOT AT END	NOT ON EXCEPTION	NOT ON SIZE ERROR
NOT AT END-OF-PAGE	NOT ON INPUT ERROR	
NOT INVALID KEY	NOT ON OVERFLOW	

Using NOT phrases can make code more readable and sometimes more efficient.
The following 2 examples are functionally equivalent but the 2nd is easier to read and is more efficient (only one test for every record read).

Example 1:
```
READ IN-FILE
     AT END      SET IN-FILE-EOF TO TRUE
END-READ
IF   NOT IN-FILE-EOF
     ADD 1 TO IN-CNT
END-IF
```

Example 2:
```
READ IN-FILE
     AT END          SET IN-FILE-EOF TO TRUE
     NOT AT END      ADD 1 TO IN-CNT
END-READ
```

Section 4 - COBOL-89 Functions

The 1989 addendum to the ANSI COBOL 85 standard added built-in functions within the COBOL language. These predefined functions (called Intrinsic Functions in the standard) provide the capability to reference a data item whose value is derived automatically at the time of reference during the execution of the program.

$CONTROL POST85 is required in any program that calls a COBOL function and the ANSI85 entry point of the HP COBOLII/IX compiler. The keyword FUNCTION becomes a reserved word and must precede any reference to a COBOL function.
Functions are treated like temporary, elementary data items so they may be used wherever an elementary data item is valid except as a receiving operand.
They return alphanumeric, numeric or integer values depending on the function type.

Alphanumeric Functions

Date Functions:
CURRENT-DATE, WHEN-COMPILED
String Functions:
CHAR, LOWER-CASE, REVERSE, UPPER-CASE
General Functions:
MAX, MIN (if all parameters are alphanumeric or alphabetic)

Integer Functions

Date Functions:
DATE-OF-INTEGER, DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY
String Functions:
LENGTH, ORD
General Functions:
MAX, MIN (if all parameters are integer), ORD-MAX, ORD-MIN
Arithmetic Functions:
INTEGER, INTEGER-PART, MOD, SUM (type depends on parameters)
Financial and Statistical Functions:
FACTORIAL, RANGE (type depends on parameters)

Numeric Functions

String Functions:
NUMVAL, NUMVAL-C
Arithmetic Functions:
LOG, LOG10, RANDOM, REM, SQRT, SUM (type depends on parameters)
Financial, Statistical and Trigonometric Functions:
ACOS, ASIN, ATAN, ANNUITY, COS, MEAN, MEDIAN, MIDRANGE, PRESENT-VALUE, RANGE (type depends on parameters), SIN, STANDARD-DEVIATION, TAN, VARIANCE
General Functions
MAX, MIN (if some or all parameters are numeric)

The following Rules apply to the use of COBOL Functions -

- Alphanumeric Functions have an implicit usage of display.
- Numeric Functions always have an operational sign, can only be used in arithmetic expressions (such as COMPUTE, relational conditions or reference modification) and cannot be used where an integer operand is required, even if the function call might yield an integer value.
- Integer Functions always have an operational sign with all zero digits to the right of the decimal point, and can only be used in arithmetic expressions.

```
MOVE FUNCTION LENGTH(CITY) TO A          is not allowed
COMPUTE A = FUNCTION LENGTH(CITY)        is allowed
IF  FUNCTION LENGTH(CITY) > X-VALUE      is allowed
```

- A Function may be used as an argument to another function as long as the function type matches the argument requirements of the function in which it is being used.
- Reference modification may be used with Functions of type alphanumeric.

```
DISPLAY FUNCTION UPPER-CASE(FULL-NAME)(1:5)
```

- The program must specify the number of arguments required for the specific function being used, and they must be in the correct sequence.
- Literal or arithmetic expressions may be used as arguments, but they must conform to the requirements specific to the function being used.

Some of the numeric functions convert arguments to intermediate floating point values to calculate the function result. The precision of these functions is limited to 15 significant digits. The fractional values may have rounding errors even if the total size of the argument is <= 15 digits.

Use of the ROUNDED phrase is recommended for all Numeric Functions when precision of the resultant value is important.

Some of the functions are implemented as calls to run-time libraries while the rest are implemented as in-line code. As In-line functions will be generally faster than functions in the run-time library, it might be more efficient to code your own routines for these run-time library functions.

Using ALL as a table subscript. Some functions allow a variable number of arguments (such as MAX, MEAN and SUM). You can pass all the elements of a table to one of these functions by specifying the table as an argument with ALL as the table subscript.

```
Example: COMPUTE GRAND-TOTAL = FUNCTION SUM (SUB-TOTAL(ALL))
                where SUB-TOTAL is defined by an OCCURS clause.
```

Date Functions: These functions all deal with dates having a full four-digit year.
See Appendix Section 8 – SAMPLE 3 – for use of validation intrinsics to validate date parameters before using the Date Functions.

CURRENT-DATE

The function returns a 21 character alphanumeric string containing the current date (YYYYMMDD), current time (HHMMSS00) and GMT offset (+HHMM or -HHMM). There are no arguments. To get the correct time differential from Greenwich Mean Time, you need to set the TZ environment variable to your local time zone. If TZ is not set the function assumes Eastern Standard Time (EST5EDT). The time differential is automatically adjusted for daylight savings time according to the values in the time and zone adjustment table (in file TZTAB.LIB.SYS).
Note – If you set the TZ variable via a program make sure you provide the correct length parameter. The variable with extra spaces will not match what appears to be the same entry on the TZTAB.LIB.SYS file.

This function uses the hardware clock plus the TZ variable to calculate the current date and time. This means the hardware clock **must** be set to Greenwich Mean Time.
Note - The old (pre 85) CURRENT-DATE special register returns the date and time directly from the Software Clock therefore these 2 'CURRENT-DATE' statements can return different values at the same time if the clocks have not been set correctly.
Sample Program: See the appendix Section 8 for a sample program designed to print the current date in a number of different time zones (DEMO19).

WHEN-COMPILED

The function returns a 21 character alphanumeric string containing the date (YYYYMMDD), time (HHMMSS00) and GMT offset (+HHMM or -HHMM) when the source program was last compiled. There are no arguments. See above for the GMT offset definition.

INTEGER-OF-DATE (parameter-1)

The function converts a date in the form (YYYYMMDD) to an integer that is the number of days past the date December 31, 1600.
Because this is an integer function, it can only be used where an arithmetic expression is allowed. A good use of this function is to create date arithmetic or comparison routines.
Note: The date **must** be a valid date otherwise the program will abort.

DATE-OF-INTEGER (parameter-1)

The function converts an integer value into a date of the form (YYYYMMDD) as the inverse of the INTEGER-OF-DATE function.
Because this is an integer function, it can only be used where an arithmetic expression is allowed. A good use of this function is to create date arithmetic or comparison routines.

Example 1: Add 15 days to MY-DATE

```
COMPUTE MY-DATE =  FUNCTION DATE-OF-INTEGER
        (FUNCTION INTEGER-OF-DATE (MY-DATE) + 15)
```

Example 2: Example 1 rewritten to use a Macro

```
*   !1 = Date
*   !2 = Days to Add (+ or -)
$DEFINE     %ADDTODATE=
        COMPUTE !1 =
            FUNCTION DATE-OF-INTEGER
            (FUNCTION INTEGER-OF-DATE (!1)
            + !2)#
%ADDTODATE(MY-DATE#, 15#)
%ADDTODATE(MY-DATE#,-30#)
```

INTEGER-OF-DAY (parameter-1)

The function converts a Julian date in the form (YYYYDDD) to an integer that is the number of days past the date December 31, 1600.
Because this is an integer function, it can only be used where an arithmetic expression is allowed.
Note: The date **must** be a valid date otherwise the program will abort.

DAY-OF-INTEGER (parameter-1)

The function converts an integer value into a Julian date of the form (YYYYDDD) as the inverse of the INTEGER-OF-DAY function.
Because this is an integer function, it can only be used where an arithmetic expression is allowed.

Arithmetic Functions:

REM (parameter-1 parameter-2)

The function returns a numeric value that is the remainder of parameter-1 divided by parameter-2. Both parameters must be numeric and the 2nd must not be zero.

Example: A good use of this function is to obtain the day of the week in conjunction with the INTEGER-OF-DATE function. This works because the ANSI committee chose January 1, 1601 as day 1 and this is a Monday. The Remainder after dividing 7 into the integer value of a date, and then adding one, will be the numeric day of the week starting from Sunday.

```
01  WEEK-DAY-DATA.
    03  WEEK-DAY-SUB      PIC 9.
    03  WEEK-DAYS.
        05                 PIC X(9) VALUE "Sunday".
        05                 PIC X(9) VALUE "Monday".
        05                 PIC X(9) VALUE "Tuesday".
        05                 PIC X(9) VALUE "Wednesday".
        05                 PIC X(9) VALUE "Thursday".
        05                 PIC X(9) VALUE "Friday".
        05                 PIC X(9) VALUE "Saturday".
    03  REDEFINES WEEK-DAYS.
        05  WEEK-DAY      PIC X(9) OCCURS 7.

*   !1 = Date
*   !2 = Result field for Day of Week
$DEFINE  %DAYOFWEEK=
    COMPUTE WEEK-DAY-SUB = FUNCTION REM
                (FUNCTION INTEGER-OF-DATE (!1), 7) + 1
    MOVE WEEK-DAY(WEEK-DAY-SUB) TO !2#
```

See Appendix Section 8 – SAMPLE 4 – for the same code as above in a separately compiled module to remove data from ‘global’ working storage to provide a form of encapsulation.

SUM (parameter-1 … parameter-n)

The function returns a numeric value that is the sum of the parameters. The function will be integer if all the parameters are integer otherwise the function will be numeric. Every parameter must contain a numeric or integer value.
If all the elements of a table are to be totaled the word ‘ALL’ may be used in place of the list of elements.

Example: Sum the elements (TAB-ITEM) in a table plus field REJECT-VAL

```
COMPUTE TOTAL-VALUE =
            FUNCTION SUM (TAB-ITEM(ALL)   REJECT-VAL)
```

RANDOM [(parameter-1)]

The function returns a numeric value, between 0 and 1, that is a pseudo-random number. The optional parameter, if used, becomes the seed value to generate a sequence of pseudo-random numbers, and must be between 0 and 999999999 inclusive. For a given seed value the sequence of pseudo-random numbers is always the same. To produce a different set of pseudo-random numbers every time a program is run the seed parameter is usually taken from data like the current date and time

MEAN, MEDIAN, RANGE, MIDRANGE (parameter-1 … parameter-n)

These functions all have the same characteristics as the SUM function.
MEAN – Average or Statistical mean of all the parameters
MEDIAN – The Middle value of the list of parameters.
RANGE – The difference between the largest and smallest parameters.
MIDRANGE – The average of the largest and the smallest parameters.

String Functions:

UPPER-CASE (parameter-1)

The function returns a character string, the same length as the parameter, with each lowercase letter replaced by the corresponding uppercase letter.

LOWER-CASE (parameter-1)

The function returns a character string, the same length as the parameter, with each uppercase letter replaced by the corresponding lowercase letter.

Example: To ensure first letter is uppercase and rest is lowercase.

```
MOVE FUNCTION UPPERCASE (NAME(1:1) TO NAME(1:1)
MOVE FUNCTION LOWERCASE (NAME(2:) TO NAME(2:)
```

CHAR (parameter-1)

The function returns a single character having a value corresponding to its ordinal position in the ASCII character set. The parameter must be an integer between 1 and 256 inclusive.

Note: Because the ASCII character set starts from decimal 0 but the Ordinal value start from 1, the parameter must be the decimal value of the character + 1.

Example: To display the character "A" (decimal value 65)

```
DISPLAY FUNCTION CHAR (66)
```

ORD (parameter-1)

The function returns an integer value between 1 and 256 corresponding to the ordinal position in the ASCII character set of the character in the parameter. Because this is an integer function the COMPUTE verb must be used, not a MOVE.

Example: To move the ordinal value of 'A' (66) to NUM

```
COMPUTE NUM = FUNCTION ORD ("A")
```

NUMVAL (parameter-1)

The function returns a numeric value equivalent to the character string in the parameter. Leading and trailing spaces are ignored.

Example:
```
FUNCTION NUMVAL (“  +  25.90   “)      = 25.9
FUNCTION NUMVAL ( “  25.90   +  “)     = 25.9
FUNCTION NUMVAL ( “  25.90  CR”)       = - 25.9
FUNCTION NUMVAL ( “   -.36  “)         = - 0.36
```

NUMVAL-C (parameter-1 {parameter-2})

The function returns a numeric value equivalent to the character string in the parameter. Leading and trailing spaces are ignored. Any optional currency sign specified by the 2nd parameter and any optional commas preceding the decimal point will be ignored.

Example:
```
FUNCTION NUMVAL-C ( “+USD25.90 “ “USD”)  = 25.9
FUNCTION NUMVAL-C ( “  2,425.90   +  “)  = 2425.9
FUNCTION NUMVAL-C ( “  $25.90  CR” “$”)  = - 25.9
FUNCTION NUMVAL-C ( “-$567,123  “ “$”)   = -567123
```

MAX, MIN (parameter-1 … parameter-n)

The function returns the contents of the parameter that is the maximum or minimum, respectively. The function type depends on the type of the parameters which must all be of the same type.

Example:
```
DISPLAY FUNCTION MAX (FIELD-1 FIELD-2 FIELD-3)
COMPUTE MIN-VALUE = FUNCTION MIN (ITEM (ALL) )
```

ORD-MAX, ORD-MIN (parameter-1 … parameter-n)

The function returns an integer value that is the ordinal number of the parameter that contains the maximum or minimum value, respectively.

Example:

COMPUTE NUM-1 = FUNCTION ORD-MAX (“M” “C” “Z” “A”)
NUM-1 will contain 3 as the “Z” is in the 3rd position.

COMPUTE NUM-2 = FUNCTION ORD-MIN(“M” “C” “Z” “A”)
NUM-2 will contain 4 as the “A” is in the 4th position.

Section 5 – Calling MPE Intrinsics

An HP extension to the 1985 ANSI COBOL standard allows the keyword INTRINSIC to be used with the CALL verb to specify that an operating system intrinsic is to be called rather than a user subprogram. These MPE Intrinsics are special procedures whose declarations reside in the intrinsic file SYSINTR.PUB.SYS which enables the types and bounds of the parameter values to be checked before the call is actioned.

There are some special rules for defining parameters for MPE Intrinsics.

- **MPE Intrinsics** are referenced by their name within a literal.
  e.g. CALL INTRINSIC "RESETCONTROL"

- **Optional Parameters**: If a parameter is optional and you have no value for it the parameter must be specified as \\.
  e.g. See WHO intrinsic below.

- **Parameter Type**: The Intrinsic Manual defines the Parameter Type as a mnemonic. The following table shows the commonest parameters and the equivalent COBOL type that must be used.

Type	Name of Type	Size in Bytes	COBOL Type
A	Array	n	X(n)
C	Char	1	X
I16	16 bit signed integer	2	S9(1) COMP – S9(4) COMP
I32	32 bit signed integer	4	S9(5) COMP – S9(9) COMP
I64	64 bit signed integer	8	S9(10) COMP – S9(18) COMP
U16	16 bit unsigned integer	2	S9(1) COMP – S9(4) COMP
U32	32 bit unsigned integer	4	S9(9) COMP
U64	64 bit unsigned integer	8	S9(18) COMP
@32	32 bit address	4	S9(9) COMP
@64	64 bit address	8	S9(18) COMP

- **Binary (COMP) Fields**: An intrinsic may return a value outside the range of valid numbers for the COBOL type. The full value can be referenced by MOVEing the resultant field to a larger Binary field before any further reference. One example is the WHO intrinsic which can return a logical device number up to 16385. Because the parameter must be defined as COBOL type S9(4) COMP (2 bytes) the resultant value needs to be moved to a COBOL type S9(5) COMP (4 bytes) to reference any value greater then 9999.

- **Real Numbers**: Passing Real Numbers to intrinsics requires some manipulation to get a real number in a COBOL program. The number must be converted from a string (display) form into a floating point form by using the "HPEXTIN" intrinsic. See the PAUSE intrinsic below.

- **Intrinsic Errors**: A special relation operator, defined under SPECIAL-NAMES in the program, can be used to check the condition code returned by an intrinsic. If this code is not zero, then an error occurred.

```
SPECIAL-NAMES.
    CONDITION-CODE IS C-C.

CALL INTRINSIC "HPCICOMMAND" USING
                MPE-COMMAND   MPE-ERROR-CODE   MPE-ERROR-TYPE
IF   C-C <> 0
    Command not executed
END-IF
```

Useful Intrinsics

1. **DATELINE** – Returns the current date and time formatted into a 27 character field, including the day of week, month, day, year, hours and minutes.

```
CALL INTRINSIC "DATELINE" USING DATE-BUFFER
DATE-BUFFER = WED, AUG 22, 2001,  2:30 PM
```

2. **FCONTROL –** Performs various control operations on a file or on the device where the file resides. They include
   - Supplying a printer carriage control directive
   - Verifying I/O
   - Reading the hardware status word for the device
   - Setting a terminal's time-out interval
   - Repositioning a file at it beginning
   - Writing an end-of-file marker
   - Enable/disable echo facility
   - Enable/disable System or subsystem Break function
   - Enable/disable Extended Wait for message file

   e.g. CALL INTRINSIC "FCONTROL" USING FD-FILE-NAME 45 1

3. **FINDJCW** – Searches the Job Control Word table for the specified JCW and returns its value (which must be an unsigned integer). The JCW's can be user defined, System defined (e.g. CIERROR) or System Reserved (e.g. HPDATE).

```
CALL INTRINSIC "FINDJCW"    USING
                JCW-NAME      JCW-VALUE     JCW-STATUS
```

4. **GETINFO** – Returns user supplied information that was passed to the program when it was created.
   The INFO-STRING is passed via the ";INFO=" parameter to the RUN command.
   The INFO-PARM is passed via the ";PARM=" parameter to the RUN command.

```
CALL INTRINSIC "GETINFO"
    USING     INFO-STRING   INFO-LENGTH     INFO-PARM
    GIVING    INFO-RESULT
```

   The ";PARM=" parameter to the RUN command also sets the Software Switches (SW0 to SW15). The switches are defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION and are associated with an 'on' and an 'off' condition name.

```
SPECIAL-NAMES.
    SW0 ON STATUS PHASE-1   OFF STATUS PHASE-2.
```

   If I am logged on as MGR.SYS and my session number is #S1234 then
   :RUN MYPROG;INFO="!HPUSER,!HPACCOUNT";PARM=!HPJOBNUM

   INFO-STRING = "MGR,SYS" INFO-LENGTH = 7
   INFO-PARM = 1234 INFO-RESULT = 0 (if no errors)
   1234 = %10011010010 which will set SW5,SW8,SW9,SW11,SW14 to 'ON'.

5. **HPCICOMMAND** – Enables the program to execute an MPE Command, including UDC"s, Command files and implied Run commands. The last character in the command field must be a carriage return (%15).

```
CALL INTRINSIC "HPCICOMMAND" USING
            MPE-COMMAND   MPE-ERROR-CODE   MPE-ERROR-TYPE
```

6. **HPCIGETVAR** – Retrieves a valid variable name from the session-level variable table and returns the current value and/or attributes. The intrinsic can reference up to six items at a time. Item number 1 return Integer value, Item number 2 returns string value, Item number 11 returns string length, Item number 13 returns type of variable (1 = Integer, 2 = String).

```
CALL INTRINSIC "HPCIGETVAR"   USING
    VAR-NAME      VAR-STATUS
    2       VAR-ITEM-STRING     11      VAR-ITEM-LGTH
    1       VAR-ITEM-INTEGER    13      VAR-ITEM-TYPE
```

7. **HPCIPUTVAR** – Sets the value of a session-level variable. The first Call references a string variable and the second Call references an integer variable.

```
CALL INTRINSIC "HPCIPUTVAR"  USING
    VAR-NAME      VAR-STATUS
    2         VAR-ITEM-STRING      11        VAR-ITEM-LGTH

CALL INTRINSIC "HPCIPUTVAR"  USING
    VAR-NAME      VAR-STATUS
    1         VAR-ITEM-NUMBER
```

8. **PAUSE** – Suspends the program for the specified number of seconds. The parameter must be a Real (floating point) number which cannot normally be defined in a COBOL program. The number of seconds (defined as 9(..)) must be converted to REAL-SECONDS (defined as a S9(9) COMP) via the "HPEXTIN" intrinsic. This field can only be used as a real number passed to an intrinsic as the field no longer contains a normal binary value.

```
COMPUTE NUM-LEN = FUNCTION LENGTH (NUM-SECONDS)
CALL INTRINSIC "HPEXTIN"
    USING  NUM-SECONDS      NUM-LEN   0 1 0 0
           REAL-SECONDS     ERROR-CODE
CALL INTRINSIC "PAUSE"  USING  REAL-SECONDS
```

9. **PUTJCW** – Assigns a value to the specified JCW.

```
CALL INTRINSIC "PUTJCW"      USING
           JCW-NAME      JCW-VALUE     JCW-STATUS
```

10. **RESETCONTROL** – Re-enables the subsystem break trap which allows a process to accept other subsystem break signals. Used to reset a trap when a trap handler routine has been invoked.

```
CALL INTRINSIC "RESETCONTROL"
```

11. **WHO** - Returns the access mode (Session or Job)and attributes of the user running the program including User's Name, User's Logon Group, User's Logon Account, User's Home Group, User's Logical Device. If WHO-MODE is < 8 the program is running in a Session otherwise it is running in a Job.

```
CALL INTRINSIC "WHO" USING
    WHO-MODE     \\    \\    WHO-USER      WHO-GROUP
    WHO-ACCOUNT          WHO-HOME      WHO-LDEV
```

Section 6 – Implementing Control-Y Trapping

Many COBOL programs need to provide the user with control to either stop the program before it has normally completed or temporarily suspend the program for some reason. The main mechanism to do this from a terminal or terminal emulator is by pressing Control & Y together. This is called Control-Y trapping. The procedure to trap the Control-Y can be written as a COBOL subprogram and put into an XL library or linked directly with the program requiring this facility.

The subprogram must arm the Control-Y trap once by calling the intrinsic **XCONTRAP** and then a second entry point is used to set a Control-Y indicator and reset the trap (intrinsic **RESETCONTROL**) ready for another Control-Y.

Beware of the name of the entry point. If you have used hyphens, these will be translated to underscores when the external name is created by the compile. This means the "Procname" parameter to the "HPGETPROCPLABEL" intrinsic must be preset with the entry point name using underscores instead of hyphens.

The main program calls the subprogram once in the initialization and then tests the indicator at strategic points in the program. The indicator must be defined as EXTERNAL in both main program and subprogram, and have the same definition.

The appendix shows a Control-Y subprogram and how it is referenced from the demo program **DEMO1-14** (Test 14).

Section 7 – File Organization & Access

- In HP COBOL, Logical files can be organized in 4 ways – **Sequential**, **Random**, **Relative** and **Indexed**.

- They can be accessed in 3 ways
  - **Sequential Access**
    This means that existing records are accessed in ascending order. The Relative Key is used for relative files and a prime or alternate key is used for indexed files. Random access files may not be accessed sequentially.
  - **Random Access**
    This means that the records are accessed directly by using a record key data item (for indexed files) or by using the relative record numbers of records (for relative and random access files).
  - **Dynamic Access**
    This means that the program may alternate between sequential and random access modes by selectively using different forms of various input-output statements. This type of access may only be used for relative and indexed files.

- **Sequential Organization Files**
  - Because of their simplicity they are the most portable file type.

  - Can be opened for INPUT (reading), OUTPUT (writing), I-O (reading and writing), EXTEND (append records to the end of the file).

  - Can be accessed sequentially only, with update allowed in place. Records cannot be deleted or inserted. Space is required only for actual records written.

  - Can reside on any device.

  - Special MPE Sequential file types – Circular, Message.
    - **Circular**
      A circular file is a special MPE file type that is organized like a sequential file except that it has no "last" record. It is most appropriate for a history file where the last 'n' transactions would always be recorded in a 'n' record file.
      The file can be built with an MPE command – BUILD filename;CIR
      Or an MPE file command can be used - FILE filename;CIR.
      See Appendix Section 8 for sample program (DEMO16).

    - **Message**
      A Message file is a special MPE file type that is organized like a sequential file, that is open for input and output access at the same time. Programs can use message files to communicate with each other.
      The file can be built with an MPE command – BUILD filename;MSG
      Or an MPE file command can be used - FILE filename;MSG.

The normal use is for one, or more, programs to open the file for write access. One, and only one, other program opens the file for read access to process the records. If more than one program reads the file, records can appear to be lost as every record is deleted once it is read.

The use of the FCONTROL intrinsic (parameter 45 and 1) enables an extended wait to be set by the 'reading' program so it will wait on an empty file that is not currently opened by a writer. FCONTROL intrinsic (parameter 45 and 0) resets the extended wait so that a read on an empty file with no writers will return an 'end-of-file' condition.
See Appendix Section 8 for sample program (DEMO17).

- **Sequential Organization File Access**
  - The ORGANIZATION and ACCESS clauses are not required for the SELECT The USING phrase provides the name of the file at run time.

```
SELECT      SEQ-FILE        ASSIGN TO "file info"
```

  - **Read** next sequential record

```
READ SEQ-FILE               [INTO Work-data-item]
    [AT END                 Imperative Statement]
    [NOT AT END             Imperative Statement]
[END-READ]
```

  - **Write** next sequential record at end of file only.

```
WRITE SEQ-FILE              [FROM Work-data-item]
    [BEFORE ADVANCING n LINES]
    [AT END-OF-PAGE         Imperative Statement]
    [NOT AT END-OF-PAGE     Imperative Statement]
[END-WRITE]

WRITE SEQ-FILE              [FROM Work-data-item]
    [AFTER ADVANCING n LINES]
    [AT END-OF-PAGE         Imperative Statement]
    [NOT AT END-OF-PAGE     Imperative Statement]
[END-WRITE]
```

  - **Rewrite** by replacing last record read

```
REWRITE SEQ-FILE            [FROM Work-data-item]
[END-REWRITE]
```

- **Sequential Organization Random Access Files (HP Extension)**
  - Have one numeric unique key. The first key is number zero (0).
  - Can be opened for INPUT (reading), OUTPUT (writing), I-O (reading and writing). When records are written to the file, and previous record areas have had no data written to them, the empty records are filled with blanks (ASCII file) or binary zeros (Binary file). When records are sequentially read, the blank (empty) records will be accessed. This capability does not exist for any other type of file.
  - Can only be accessed randomly by the key, which corresponds to the record number minus 1, but by setting the key to zero then reading records with the NEXT phrase, the file can be read sequentially.
  - Records can be inserted, updated or appended (not deleted).
  - Space is required for maximum possible records plus one due to the first key being zero.
  - Can reside on disc devices only.
  - A hashing algorithm may be required if the key can have a larger value than the maximum number of records.
  - The MPE operating system does not distinguish between Random Access and Sequential Organization files. The distinction is made by the HP COBOL compiler, which generates different code for random access files. This means a file can be created as sequential organization by one program and treated as a random access file by another program. The file appears as a sequential file, therefore it is portable to other MPE systems but the program will not be portable to other non-MPE systems as the random access file is defined with an ACTUAL KEY clause (an HP extension to the ANSI standard) in the SELECT statement.

- **Sequential Organization Random Access**
  - The clauses ACCESS IS RANDOM and ACTUAL KEY are required. The actual key must be defined in working storage, preferably as an S9(9) COMP field.

```
SELECT    RAND-FILE    ASSIGN TO "RAND-FILE"
                       ACCESS IS RANDOM
                       ACTUAL KEY IS RAND-KEY
```

  - To **Read** records **Sequentially** the NEXT phrase must be used

```
READ RAND-FILE  NEXT  [INTO Work-data-item]
    [AT END           Imperative Statement]
    [NOT AT END       Imperative Statement]
[END-READ]
```

- To **Read** a **Random** record the ACTUAL KEY data item must be preset with the required record number.

```
READ RAND-FILE          [INTO Work-data-item]
    [INVALID KEY        Imperative Statement]
    [NOT INVALID KEY    Imperative Statement]
[END-READ]
```

- To **Write** a record, the INVALID KEY phrase is required. The ACTUAL KEY data item is used in an implicit Seek to find the record into which the data is to be written.

```
WRITE RAND-FILE         [FROM Work-data-item]
     INVALID KEY        Imperative Statement
    [NOT INVALID KEY    Imperative Statement]
[END-WRITE]
```

- To **Rewrite** a record, the record number must be specified in the ACTUAL KEY data item and a record must already exist.

```
REWRITE RAND-FILE       [FROM Work-data-item]
     INVALID KEY        Imperative Statement
    [NOT INVALID KEY    Imperative Statement]
[END-REWRITE]
```

- **Relative Organization Files**
  - Has one numeric unique key. The first key is number one (1). This is defined as a RELATIVE KEY in the SELECT statement
  - Can be opened for INPUT (reading), OUTPUT (writing), I-O (reading and writing), EXTEND (write to the end of the file if access is sequential). When reading sequentially, only records that have been written will be accessed. This means empty space in the file is skipped (unlike Random Access files).
  - Can be accessed sequentially, randomly or dynamically and Records can be deleted, inserted, updated or appended.
  - Space is required for maximum possible records plus one tag per record. The tag indicates whether the associated record has been deleted.
  - Can reside on disc devices only.
  - A hashing algorithm may be required if the key can have a larger value than the maximum number of records.
  - The file has a special MPE file type, RIO, therefore it is not portable except to other MPE systems but Relative Organization files are an ANSI standard file type therefore the program will be portable.
  - Relative Organization files are useful for storing records with numeric keys when random or directed access is required and the key value is much larger than the expected maximum number of records.

- **Relative Organization File Access**
  - The clause ORGANIZATION RELATIVE is required. The ACCESS clause is required to define the RELATIVE KEY and may be SEQUENTIAL, RANDOM or DYNAMIC. The Relative key data item must be defined in working-storage, preferably as an S9(9) COMP field (or S9(9) BINARY).

```
SELECT    REL-FILE    ASSIGN TO "REL-FILE"
                      ACCESS IS SEQUENTIAL
                      RELATIVE KEY IS REL-KEY

SELECT    REL-FILE    ASSIGN TO "REL-FILE"
                      ACCESS IS RANDOM
                      RELATIVE KEY IS REL-KEY

SELECT    REL-FILE    ASSIGN TO "REL-FILE"
                      ACCESS IS DYNAMIC
                      RELATIVE KEY IS REL-KEY
```

- To **Read** records **Sequentially** the NEXT phrase must be used. The actual position may be made available via a START statement.

```
READ REL-FILE          NEXT   [INTO Work-data-item]
    [AT END            Imperative Statement]
    [NOT AT END        Imperative Statement]
[END-READ]
```

- To **Read** a **Random** record the RELATIVE KEY data item must be preset with the required record number.

```
READ REL-FILE          [INTO Work-data-item]
    [INVALID KEY       Imperative Statement]
    [NOT INVALID KEY   Imperative Statement]
[END-READ]
```

- To **Write** a new record, the INVALID KEY phrase is required unless a USE statement has been issued for the referenced file.
  The Relative Key data item is updated as records are written in Sequential access mode. The first record written is record number 1 then 2, 3, 4 etc.
  If the file is open in random or dynamic access mode the Relative Key data item must be set to specify where the record is to be written (and the record number must not already exist).

```
WRITE REL-FILE         [FROM Work-data-item]
    [INVALID KEY       Imperative Statement]
    [NOT INVALID KEY   Imperative Statement]
[END-WRITE]
```

- To **Rewrite** a record in **Sequential** access - the last I-O statement must have been a READ statement. The REWRITE replaces the last record read. In this mode the INVALID KEY clause must not be used.

```
REWRITE REL-FILE       [FROM Work-data-item]
[END-REWRITE]
```

- To **Rewrite** a record in **Random** or **Dynamic** access - The record to be logically replaced is specified by the Relative Key data item and must already exist.

```
REWRITE REL-FILE       [FROM Work-data-item]
    [INVALID KEY       Imperative Statement]
    [NOT INVALID KEY   Imperative Statement]
[END-REWRITE]
```

- To Define the **Starting Record** for Sequential or Dynamic access the required record number must be preset in the RELATIVE KEY data item. The file must be opened in INPUT or I-O mode.

```
START REL-FILE
    [INVALID KEY       Imperative Statement]
    [NOT INVALID KEY   Imperative Statement]
[END-START]
```

- To **Delete** a record in **Sequential** access - the INVALID KEY phrase must not be used. The record to be deleted is the last record read.
  The file must be open in I-O mode.

```
DELETE REL-FILE
[END-DELETE]
```

- To **Delete** a record in **Random** or **Dynamic** access – the INVALID KEY phrase must be used unless a USE statement has been issued for the referenced file. The record removed is the one referenced by the Relative Key data item.
  The file must be open in I-O mode.

```
DELETE REL-FILE
    [INVALID KEY          Imperative Statement]
    [NOT INVALID KEY      Imperative Statement]
[END-DELETE]
```

- **Indexed Organization Files (KSAM)**
  - Up to 16 alphanumeric keys. The primary key must be written in ascending order if access mode is sequential. The first key can be any value. Keys do not have to be unique although it is recommended that the primary keys be unique.

  - Can be opened for INPUT (reading), OUTPUT (writing), I-O (reading and writing), EXTEND (write to the end of the file if access is Sequential).

  - Can be accessed sequentially, randomly or dynamically and Records can be deleted, inserted, updated or appended.

  - In Compatibility Mode, two files are created: one for data and one for the keys. In Native Mode, only one file is created.

  - The file has a special MPE file type, KSAM or KSAMXL, therefore it is not portable except to other MPE systems but Indexed Organization files are an ANSI standard file type therefore the program will be portable.

  - The same COBOL program can access a CM KSAM file or an NM KSAM file.

  - To build a CM KSAM file

```
:KSAMUTIL
>BUILD MYFILE;REC=-56,1,F,ASCII;DISC=5000;KEY=B,1,40;KEYFILE=MYFILEK
>EXIT
```

  - To build an NM KSAM file

```
:BUILD MYFILE;REC=-56,1,F,ASCII;DISC=5000;KSAMXL;KEY=(B,1,40)
```

  - See Appendix Section 8 for sample program reading a database and creating a KSAM file (DEMO18).

- **Indexed Organization File Access**
  - The clause ORGANIZATION INDEXED is required. The ACCESS clause is not required if the access is to be Sequential otherwise it must be RANDOM or DYNAMIC.
    A RECORD KEY clause must be defined. This data item must be described as alphanumeric within the record description.
    Up to 16 alternate keys can be defined with the clause ALTERNATE RECORD IS Dataitem-1, Dataitem-2 etc.

```
SELECT    INDEX-FILE    ASSIGN TO "INDEX-FILE"
                        [ACCESS IS SEQUENTIAL]
                        RECORD KEY IS REC-KEY

SELECT    INDEX-FILE    ASSIGN TO "INDEX-FILE"
                        ACCESS IS RANDOM
                        RECORD KEY IS REC-KEY

SELECT    INDEX-FILE    ASSIGN TO "INDEX-FILE"
                        ACCESS IS DYNAMIC
                        RECORD KEY IS REC-KEY
```

  - To **Read** records **Sequentially** the NEXT phrase must be used. The actual position may be made available via a START statement.

```
READ INDEX-FILE      NEXT   [INTO Work-data-item]
    [AT END                 Imperative Statement]
    [NOT AT END             Imperative Statement]
[END-READ]
```

  - To **Read** a **Random** record the RELATIVE KEY data item must be preset with the required record number.

```
READ INDEX-FILE          [INTO Work-data-item]
    [KEY IS              data-name]
    [INVALID KEY         Imperative Statement]
    [NOT INVALID KEY     Imperative Statement]
[END-READ]
```

  - To **Write** a new record the INVALID KEY phrase is required unless a USE statement has been issued for the referenced file.
    If the file is open in Sequential access mode the records must be written in ascending order of primary key values.
    If the file is open in random or dynamic access mode the records will be written based on the keys in the record.

```
WRITE INDEX-FILE         [FROM Work-data-item]
    [INVALID KEY         Imperative Statement]
    [NOT INVALID KEY     Imperative Statement]
[END-WRITE]
```

– To **Rewrite** a record: For **Sequential** access - the last I-O statement must have been a READ statement. The REWRITE replaces the last record read and must have the same primary key.
For **Random** or **Dynamic** access - The record to be replaced is specified by the Primary record key data item and must already exist.

```
REWRITE INDEX-FILE      [FROM Work-data-item]
    [INVALID KEY        Imperative Statement]
    [NOT INVALID KEY    Imperative Statement]
[END-REWRITE]
```

– To Define the **Starting Record** for Sequential or Dynamic access the required record number must be preset in the RELATIVE KEY data item.
The "relation" can only be =, >, NOT<, >=.
The file must be opened in INPUT or I-O mode.
If the KEY phrase is not used the value in the Primary key data field is used otherwise the data name must reference a primary or alternate key data item or the first part of one of the key data items.

```
START INDEX-FILE        [KEY  relation  data-name]
    [INVALID KEY        Imperative Statement]
    [NOT INVALID KEY    Imperative Statement]
[END-START]
```

– To **Delete** an existing record: For **Sequential** access – the INVALID KEY phrase must not be used. The record to be deleted is the last record read.
For **Random** or **Dynamic** access – the INVALID KEY phrase must be used unless a USE statement has been issued for the referenced file. The record removed is the one referenced by the Primary Key data item.
The file must be open in I-O mode.

```
DELETE INDEX-FILE
    [INVALID KEY        Imperative Statement]
    [NOT INVALID KEY    Imperative Statement]
[END-DELETE]
```

- **Variable Length Records**
  - Variable length records are allowed in every logical file organization
  - For Relative Organization files, HP COBOL simulates variable length records by using fixed length records therefore no space is saved.
  - Specify variable length records with the RECORD IS VARYING clause in the FD definition.
  - Variable length records are not allowed when using the REWRITE statement.
  - Example

```
FD  IFILE
    RECORD IS VARYING FROM 10 TO 50 DEPENDING ON LEN.
01  IREC.
    03  FILLER      PIC X  OCCURS 10 TO 50 DEPENDING ON LEN.
```

    To write a record to IFILE, LEN must be set to a valid value between 10 and 50. When a record is read from IFILE, LEN will contain the number of characters in the record after it has been read. If IREC is displayed, only the LEN number of characters will be displayed.

- **Dynamic File Names**
  - The ASSIGN clause normally assigns a static name to the physical file using the TO phrase.

```
SELECT IFILE   ASSIGN      TO “IFILE1”
```

  - With the addition of the USING phrase, the ASSIGN clause can specify a data name to contain the name of the physical file. The data name can be changed before the file is opened to enable a different file to be opened each time, as required.

```
SELECT IFILE   ASSIGN      USING FILE-NAME
```

- **Optional Files**
  - The OPTIONAL phrase can be used in the SELECT statement. It will cause the AT END statement, on the first Read, to be actioned if the file does not exist.

```
SELECT IFILE   OPTIONAL  ASSIGN TO “IFILE1”
```

- **ASSIGN clause File Info**
  - A new temporary file can be created with more than the default number of records

```
SELECT OFILE      ASSIGN   “OUTFILE1,,A,,50000”
```

  - A Print file output can be defined

```
SELECT OFILE      ASSIGN “PRINTLP,,,LP(CCTL),,LOAD A4 STATIONERY.”)
```

- **File Status Codes**
  - The optional **FILE STATUS** clause on the SELECT statement can specify a 2 character data name that will contain a file status code after any I-O statements have been actioned.

  - The **first digit** of the code indicates one of the following
    - 0: The I-O operation was successful
    - 1: An AT END condition occurred
    - 2: An INVALID KEY condition occurred
    - 3: A permanent error occurred
    - 4: A logical error occurred
    - 9: An implementation-defined condition occurred

  - The **second digit** of the code gives further information about the condition.

  - The **FILE STATUS data name** can be checked immediately after the I-O statement or via a USE procedure.

  - If the **I-O operation is successful**, the file status code will be returned and the program will continue.

  - If the **I-O operation is unsuccessful** and receives an **AT END** or **INVALID KEY** condition, the program will execute the AT END or INVALID KEY statement, if present. Otherwise the USE procedure will be executed, if present, and the file status code will be returned. The program will continue after any error procedure has been executed.

  - If the **I-O operation is unsuccessful** and receives a **permanent** or **logical** or **implementation-defined** error, and the file **does have a FILE STATUS** clause, the file status code will be returned and the program will continue to execute, after any applicable USE procedure has been executed.

  - If the **I-O operation is unsuccessful** and receives a **permanent** or **logical** or **implementation-defined** error, and the file **does NOT have a FILE STATUS** clause, the program will abort if no applicable USE procedure has been defined. The program will continue to execute only after a USE procedure is executed.

  - See Appendix Section 8 – SAMPLE1 – for use of File Status Codes without declaratives.

– Example Code segment showing use of Declaratives

```
FILE-CONTROL.
     SELECT IFILE   ASSIGN USING FILE-NAME
                    FILE STATUS I-STATUS.
…..
01   I-STATUS.
     03   I-STATUS1          PIC X.
     03                      PIC X.
…..
PROCEDURE DIVISION.
DECLARATIVES.
D-PARA-1 SECTION.
     USE AFTER STANDARD EXCEPTION PROCEDURE ON IFILE.
TEST-STATUS.
     IF   I-STATUS1 = "1"
          SET IFILE-EOF TO TRUE
     ELSE
          Provide error message
     END-IF
     .
EXIT-PARA.
     EXIT.
END-DECLARATIVES.
```

- **Posix File Access**
  - MPE files and **Posix files** (**Hierarchical file system – HFS files**) can be read from and written to by the same COBOL program using FILE commands to define the specific file type.

  - The following 3 examples show the file commands for reading INFILE and writing OUTFILE

    - Read MPE "TEST1.PUB.SYS" and write HFS "/tmp/hfs1"

      ```
      FILE INFILE=TESTIN.PUB.SYS
      FILE OUTFILE=/tmp/hfs1;SAVE
      ```

    - Read HFS "/tmp/hfs1" and write HFS "/tmp/hfs2"

      ```
      FILE INFILE=/tmp/hfs1
      FILE OUTFILE=/tmp/hfs2;SAVE
      ```

    - Read HFS "/tmp/hfs2" and write MPE "TEST2.PUB.SYS"

      ```
      FILE INFILE=/tmp/hfs2
      FILE OUTFILE= TEST2.PUB.SYS;SAVE
      ```

Section 8 - Appendix – Sample Programs

SAMPLE 1

Sample code to show use of File Status code with the READ statement.

```
 FILE-CONTROL.
   SELECT ORDER-FILE ASSIGN USING FILE-NAME
                     FILE STATUS ORDER-FILE-STATUS-CODE.
...
 01   MISCELLANEOUS.
   05 ORDER-FILE-STATUS-CODE     PIC X(2).
      88 ORDER-FILE-STATUS-OK          VALUE ZERO "10".
      88 ORDER-FILE-STATUS-END         VALUE “10”.
...
   OPEN INPUT ORDER-FILE

   PERFORM UNTIL ORDER-FILE-STATUS-CODE <> ZERO
      READ ORDER-FILE
         NOT AT END ADD 1 TO ORDER-READ-COUNT
      END-READ
   END-PERFORM

   IF ORDER-FILE-STATUS-OK
      CLOSE ORDER-FILE
   ELSE
...
   END-IF
...
```

SAMPLE 3

Sample code to show use of Date Validation Intrinsics.

HP support a number of standard Date formats with a number of intrinsics to manipulate these formats. The main format of interest to COBOL programmers is format 38, which is the ASCII representation of an 8 byte date in format CCYYMMDD.
The new intrinsics are

- HPDATECONVERT – convert dates from one format to another
- HPDATEFORMAT – to convert dates into display strings with a number of options
- HPDATEDIFF – to determine the number of days between 2 dates
- HPDATEOFFSET – increment/decrement a date with a given offset
- HPDATEVALIDATE – check the validity of a given date to a specified format

The Date Validate intrinsic should be used before any COBOL Date function if the data could be invalid. The intrinsic returns a 32-bit signed integer to the second macro parameter. Zero indicates the first parameter contains a valid date. Anything else indicates an invalid date or an execution error occurred.

```
*    !1 = Date
*    !2 = Days to Add (+ or -)
$DEFINE  %ADDTODATE=
   COMPUTE !1 = FUNCTION DATE-OF-INTEGER
                (FUNCTION INTEGER-OF-DATE(!1) + !2)#
*-------------------------------------------------------------
*  !1 = Date to be validated
*  !2 = Error indicator (Zero = no error)
$DEFINE     %VALIDATECCYYMMDD=
$CONTROL LOCOFF
   CALL INTRINSIC "HPDATEVALIDATE"
      USING    38    !1
      GIVING   !2
$CONTROL LOCON#
*-------------------------------------------------------------
...
   %VALIDATECCYYMMDD(MY-DATE#,VALIDATE-STATUS-CODE#)
   IF VALIDATE-STATUS-CODE = ZERO
      %ADDTODATE(MY-DATE#,15#)
   ELSE
      Perform error processing
   END-IF
...
```

SAMPLE 4

Sample code to show use of a separately compiled module to remove data from 'global' working storage to provide a form of encapsulation.

```
$CONTROL DYNAMIC
 IDENTIFICATION DIVISION.
 PROGRAM-ID.   DAYOFWEEK.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   CONSTANTS.
      03 WEEK-DAYS.
         05       VALUE "Sunday"    PIC X(9).
         05       VALUE "Monday"    PIC X(9).
         05       VALUE "Tuesday"   PIC X(9).
         05       VALUE "Wednesday" PIC X(9).
         05       VALUE "Thursday"  PIC X(9).
         05       VALUE "Friday"    PIC X(9).
         05       VALUE "Saturday"  PIC X(9).
      03 REDEFINES   WEEK-DAYS.
         05 WEEK-DAY                PIC X(9) OCCURS 7.
*
 LINKAGE SECTION.
 01   INPUT-DATE                    PIC 9(8).
*
 01   WEEK-DAY                      PIC X(9).
*
 PROCEDURE DIVISION USING  INPUT-DATE
                           WEEK-DAY.
 DAYOFWEEK.
      CALL INTRINSIC "HPDATEVALIDATE"
         USING    38    INPUT-DATE
         GIVING   TALLY
      IF  TALLY <> 0
         MOVE SPACES TO WEEK-DAY
      ELSE
         COMPUTE TALLY = FUNCTION REM
            (FUNCTION INTEGER-OF-DATE(INPUT-DATE), 7) + 1
         MOVE WEEK-DAY(TALLY) TO WEEK-DAY
      END-IF
      GOBACK
      .
 END PROGRAM   DAYOFWEEK.

To obtain the day of the week in the main program -
   CALL "DAYOFWEEK" USING MY-DATE MY-WEEK-DAY
```

DEMO1-14

Demo program to demonstrate 14 different COBOL features

- Test 1 Show Date addition using Function DATE-OF-INTEGER & INTEGER-OF-DATE
- Test 2 Same as Test 1 but using a Macro
- Test 3 Show Day of the Week using Function REM & INTEGER-OF-DATE
- Test 4 Show Ordinal values of characters using Function ORD & CHAR
- Test 5 Show use of Function NUMVAL
- Test 6 Show use of Function NUMVAL-C
- Test 7 Show use of Intrinsic "HPCICOMMAND"
- Test 8 Show use of Intrinsic "DATELINE"
- Test 9 Show use of Intrinsic "HPCIPUTVAR"
- Test 10 Show use of Intrinsic "HPCIGETVAR"
- Test 11 Show use of Intrinsic "GETINFO"
- Test 12 Show use of Intrinsic "WHO"
- Test 13 Show use of Intrinsic "PAUSE"
- Test 14 Show how to test for a Control Y trap

```
$CONTROL USLINIT,POST85
*    To Compile and Link  -  COB85XLK CJEN,PJEN
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DEMO1-14.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.    HP-3000.
 OBJECT-COMPUTER.    HP-3000.
 SPECIAL-NAMES.
     CONDITION-CODE IS C-C
     SYMBOLIC CHARACTERS NULL 1, BELL 8, CR 14, ESCAPE 28
     .

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  WORKERS.
     03  TEST-CURRENT-DATE.
         05  TEST-DATE           PIC 9(8).
         05  TEST-TIME           PIC 9(6).
         05                      PIC 99.
         05  TEST-GMT-OFFSET     PIC X.
         05  TEST-GMT            PIC 9(4).
     03  TEST-WEEK-DAY           PIC X(10).
     03  TEST-CHAR               PIC X.
     03  TEST-NUM                PIC 9(4).
     03  TEST-VAL                PIC S9(8)V99.
     03  EDIT-NUM                PIC Z(4).
     03  EDIT-NUM8               PIC Z(8).
     03  EDIT-VAL-1              PIC ++++,+++,+++.99.
     03  EDIT-VAL-2              PIC ZZZ,ZZZ,ZZZ.99CR.
     03  EDIT-VAL-3              PIC +$$$,$$$,$$$.99.
     03  EDIT-VAL-4              PIC $$$,$$$,$$$.99CR.
     03  DATE-BUFFER             PIC X(27).
```

```
01  INFO-VARIABLES.
    03  INFO-LENGTH             PIC S9(4) COMP.
    03  INFO-PARM               PIC S9(4) COMP.
    03  INFO-RESULT             PIC S9(4) COMP.
    03  REAL-SECONDS            PIC S9(9) COMP.
    03  INFO-STRING             PIC X(50).

01  CI-VARIABLES.
    03  VAR-ITEM-2              PIC S9(9) COMP  VALUE 2.
    03  VAR-ITEM-11             PIC S9(9) COMP  VALUE 11.
    03  VAR-ITEM-14             PIC S9(9) COMP  VALUE 14.
    03  VAR-ITEM-TYPE           PIC S9(9) COMP  VALUE 1.
    03  VAR-ITEM-LGTH           PIC S9(9) COMP.
    03  VAR-STATUS              PIC S9(9) COMP.
    03  VAR-ITEM-NUM            PIC S9(9) COMP.
    03  VAR-ITEM-VALUE          PIC X(100).
    03  VAR-ITEM-NAME           PIC X(20).

01  WHO-VARIABLES.
    03  WHO-MODE                PIC S9(4) COMP.
    03  WHO-LDEV                PIC S9(4) COMP.
    03  WHO-USER                PIC X(8).
    03  WHO-GROUP               PIC X(8).
    03  WHO-ACCOUNT             PIC X(8).
    03  WHO-HOME                PIC X(8).

01  WEEK-DAY-DATA.
    03  WEEK-DAY-SUB            PIC 9.
    03  WEEK-DAYS.
        05                       PIC X(9) VALUE "Sunday".
        05                       PIC X(9) VALUE "Monday".
        05                       PIC X(9) VALUE "Tuesday".
        05                       PIC X(9) VALUE "Wednesday".
        05                       PIC X(9) VALUE "Thursday".
        05                       PIC X(9) VALUE "Friday".
        05                       PIC X(9) VALUE "Saturday".
    03  REDEFINES WEEK-DAYS.
        05  WEEK-DAY             PIC X(9) OCCURS 7.

01  CONTROL-Y       EXTERNAL    PIC S9(4) COMP.
    88  CONTROL-Y-HIT                    VALUE 1.
    88  CONTROL-Y-OFF                    VALUE 0.

01  DISPLAY-BUFFER   EXTERNAL   PIC X(40).

01  INDICATORS.
    03  TEST-COMPLETE-IND       PIC X.
        88  TEST-COMPLETE                VALUE "Y".
        88  TEST-NOT-COMPLETE            VALUE "N".

01  MPE-PARMS.
    03  MPE-ERROR-CODE          PIC S9(4) COMP.
    03  MPE-ERROR-TYPE          PIC S9(4) COMP.
    03  MPE-COMMAND             PIC X(80).
    03  MPE-MAX                 PIC 999   VALUE 80.
```

```
*            Issue an MPE Command !1 (Max 79 chars)
*
*            Variables defined above
$DEFINE  %MPECOMMAND=
*START    MPECOMMAND
     MOVE !1 TO MPE-COMMAND
     MOVE CR TO MPE-COMMAND(80:1)
     CALL INTRINSIC "HPCICOMMAND" USING MPE-COMMAND
                    MPE-ERROR-CODE MPE-ERROR-TYPE
*END      MPECOMMAND#

*    !1 = DATE
*    !2 = DAYS TO ADD (+ OR -)
$DEFINE  %ADDTODATE=
     COMPUTE !1 = FUNCTION DATE-OF-INTEGER
                    (FUNCTION INTEGER-OF-DATE (!1)
                    + !2)
#
*    !1 = DATE
*    !2 = RESULT FIELD FOR DAY OF WEEK
$DEFINE  %DAYOFWEEK=
     COMPUTE WEEK-DAY-SUB = FUNCTION REM
                 (FUNCTION INTEGER-OF-DATE (!1), 7) + 1
     MOVE WEEK-DAY(WEEK-DAY-SUB) TO !2
#

 PROCEDURE DIVISION.

 PROGRAM-START.
     MOVE FUNCTION CURRENT-DATE TO TEST-CURRENT-DATE
     PERFORM TEST1
     PERFORM TEST2
     PERFORM TEST3
     PERFORM TEST4
     PERFORM TEST5
     PERFORM TEST6
     PERFORM TEST7
     PERFORM TEST8
     PERFORM TEST9
     PERFORM TEST10
     PERFORM TEST11
     PERFORM TEST12
     PERFORM TEST13
     PERFORM TEST14
     DISPLAY " "
     STOP RUN.

 TEST1.
     DISPLAY " "
     DISPLAY "**** TEST 1 ****"
     DISPLAY "Show CURRENT-DATE + Adjustment"
     DISPLAY "DATE       = " TEST-DATE
     COMPUTE TEST-DATE = FUNCTION DATE-OF-INTEGER
                         (FUNCTION INTEGER-OF-DATE (TEST-DATE)
                                  + 15)
     DISPLAY " + 15 DAYS = " TEST-DATE
     .
```

```
 TEST2.
     DISPLAY " "
     DISPLAY "**** TEST 2 ****"
     DISPLAY "Show TEST-1 Date + Adjustment using a Macro"
     DISPLAY "DATE       = " TEST-DATE
     %ADDTODATE(TEST-DATE#, 15#)
     DISPLAY " + 15 DAYS = " TEST-DATE
     %ADDTODATE(TEST-DATE#,-30#)
     DISPLAY " - 30 DAYS = " TEST-DATE
     .
TEST3.
     DISPLAY " "
     DISPLAY "**** TEST 3 ****"
     DISPLAY "Show Current Day of the Week"
     DISPLAY "DATE       = " TEST-DATE
     %DAYOFWEEK(TEST-DATE#,TEST-WEEK-DAY#)
     DISPLAY "Day of Week= " TEST-WEEK-DAY
     .

 TEST4.
     DISPLAY " "
     DISPLAY "**** TEST 4 ****"
     DISPLAY "Show Ord values of Chars"
     COMPUTE EDIT-NUM = FUNCTION ORD ("A")
     DISPLAY " A = " EDIT-NUM
     DISPLAY "70 = " FUNCTION CHAR (70)
     DISPLAY " 8 = " FUNCTION CHAR (8)
     DISPLAY "256= " FUNCTION CHAR (256)
     .

 TEST5.
     DISPLAY " "
     DISPLAY "**** TEST 5 ****"
     DISPLAY "Show NUMVAL"
     COMPUTE TEST-VAL = FUNCTION NUMVAL ("  +  25.90   ")
     MOVE TEST-VAL TO EDIT-VAL-1
     DISPLAY "  +  25.90     = " TEST-VAL(1:10)
     COMPUTE TEST-VAL = FUNCTION NUMVAL ( "  25.90   +  ")
     MOVE TEST-VAL TO EDIT-VAL-1
     DISPLAY "  25.90   +    = " TEST-VAL(1:10)
     COMPUTE TEST-VAL = FUNCTION NUMVAL ( "  25.90  CR")
     MOVE TEST-VAL TO EDIT-VAL-2
     DISPLAY "  25.90  CR    = " TEST-VAL(1:10)
     COMPUTE TEST-VAL = FUNCTION NUMVAL ( "   -.35  ")
     MOVE TEST-VAL TO EDIT-VAL-1
     DISPLAY "   -.35        = " TEST-VAL(1:10)
     .
```

```
TEST6.
    DISPLAY " "
    DISPLAY "**** TEST 6 ****"
    DISPLAY "Show NUMVAL-C"
    COMPUTE TEST-VAL = FUNCTION NUMVAL-C ( "+USD25.90 " "USD")
    MOVE TEST-VAL TO EDIT-VAL-3
    DISPLAY "+USD25.90      = " TEST-VAL(1:10)
    COMPUTE TEST-VAL = FUNCTION NUMVAL-C ( "  2,425.90   +  ")
    MOVE TEST-VAL TO EDIT-VAL-1
    DISPLAY "  2,425.90   + = " TEST-VAL(1:10)
    COMPUTE TEST-VAL = FUNCTION NUMVAL-C ( "  $25.90  CR" "$")
    MOVE TEST-VAL TO EDIT-VAL-4
    DISPLAY "  $25.90  CR   = " TEST-VAL(1:10)
    COMPUTE TEST-VAL = FUNCTION NUMVAL-C ( "-$567,123  " "$")
    MOVE TEST-VAL TO EDIT-VAL-3
    DISPLAY "-$567,123      = " TEST-VAL(1:10)
    .

TEST7.
    DISPLAY " "
    DISPLAY "**** TEST 7 ****"
    DISPLAY "Show MPE Command"
    %MPECOMMAND("SHOWME"#)
    .

TEST8.
    DISPLAY " "
    DISPLAY "**** TEST 8 ****"
    DISPLAY "Show DATELINE Intrinsic"
    CALL INTRINSIC "DATELINE" USING DATE-BUFFER
    DISPLAY DATE-BUFFER
    .

TEST9.
    DISPLAY " "
    DISPLAY "**** TEST 9 ****"
    DISPLAY "Show HPCIPUTVAR Intrinsic"
    MOVE "JEN"          TO VAR-ITEM-NAME
    MOVE DATE-BUFFER    TO VAR-ITEM-VALUE
    MOVE 27             TO VAR-ITEM-LGTH
    CALL INTRINSIC "HPCIPUTVAR" USING
        VAR-ITEM-NAME, VAR-STATUS,
        2, VAR-ITEM-VALUE, 11, VAR-ITEM-LGTH

    MOVE "JEN1"         TO VAR-ITEM-NAME
    MOVE 123456         TO VAR-ITEM-NUM
    CALL INTRINSIC "HPCIPUTVAR" USING
        VAR-ITEM-NAME, VAR-STATUS, 1, VAR-ITEM-NUM
    .

TEST10.
    DISPLAY " "
    DISPLAY "**** TEST 10 ****"
    DISPLAY "Show HPCIGETVAR Intrinsic"
    MOVE "JEN"          TO VAR-ITEM-NAME
    MOVE SPACES         TO VAR-ITEM-VALUE
    MOVE 0              TO VAR-ITEM-NUM
```

```
    CALL INTRINSIC "HPCIGETVAR" USING
        VAR-ITEM-NAME, VAR-STATUS,
        2, VAR-ITEM-VALUE, 11, VAR-ITEM-LGTH,
        1, VAR-ITEM-NUM, 13 VAR-ITEM-TYPE
    IF  VAR-ITEM-TYPE = 1
       MOVE VAR-ITEM-NUM        TO EDIT-VAL-2
       DISPLAY VAR-ITEM-NAME ": " EDIT-VAL-2
    ELSE
       DISPLAY VAR-ITEM-NAME ": " VAR-ITEM-VALUE(1:VAR-ITEM-LGTH)
    END-IF
    MOVE "JEN1"         TO VAR-ITEM-NAME
    MOVE SPACES         TO VAR-ITEM-VALUE
    MOVE 0              TO VAR-ITEM-NUM
    CALL INTRINSIC "HPCIGETVAR" USING
        VAR-ITEM-NAME, VAR-STATUS,
        2, VAR-ITEM-VALUE, 11, VAR-ITEM-LGTH,
        1, VAR-ITEM-NUM, 13 VAR-ITEM-TYPE
    IF  VAR-ITEM-TYPE = 1
       MOVE VAR-ITEM-NUM        TO EDIT-NUM8
       DISPLAY VAR-ITEM-NAME ": " EDIT-NUM8
    ELSE
       DISPLAY VAR-ITEM-NAME ": " VAR-ITEM-VALUE(1:VAR-ITEM-LGTH)
    END-IF
    .
TEST11.
    DISPLAY " "
    DISPLAY "**** TEST 11 ****"
    DISPLAY "Show GETINFO Intrinsic"
    MOVE SPACES             TO INFO-STRING
    MOVE 50                 TO INFO-LENGTH
    MOVE 0                  TO INFO-PARM
                               INFO-RESULT
    CALL INTRINSIC "GETINFO"
        USING  INFO-STRING INFO-LENGTH INFO-PARM
        GIVING INFO-RESULT
    DISPLAY "INFO = " INFO-STRING(1:INFO-LENGTH)
    MOVE INFO-PARM          TO EDIT-NUM8
    DISPLAY "PARM = " EDIT-NUM8
    .
TEST12.
    DISPLAY " "
    DISPLAY "**** TEST 12 ****"
    DISPLAY "Show WHO Intrinsic"
    CALL INTRINSIC "WHO" USING WHO-MODE \\ \\ WHO-USER WHO-GROUP
                               WHO-ACCOUNT WHO-HOME WHO-LDEV
    DISPLAY "USER       " WHO-USER
    DISPLAY "GROUP      " WHO-GROUP
    DISPLAY "ACCOUNT    " WHO-ACCOUNT
    DISPLAY "HOME       " WHO-HOME
    MOVE WHO-LDEV           TO EDIT-NUM
    DISPLAY "LDEV       " EDIT-NUM
    IF  WHO-MODE < 8
        DISPLAY "SESSION"
    ELSE
        DISPLAY "JOB"
    END-IF
    .
```

```
 TEST13.
     DISPLAY " "
     DISPLAY "**** TEST 13 ****"
     DISPLAY "Show PAUSE Intrinsic"
     DISPLAY "ENTER NUMBER OF SECONDS TO PAUSE"
     ACCEPT TEST-NUM FREE
     COMPUTE INFO-LENGTH = FUNCTION LENGTH(TEST-NUM)
     CALL INTRINSIC "HPEXTIN" USING TEST-NUM INFO-LENGTH
                          0 1 0 0 REAL-SECONDS INFO-RESULT
     IF  INFO-RESULT = 0
         CALL INTRINSIC "PAUSE" USING REAL-SECONDS
         MOVE TEST-NUM TO EDIT-NUM8
         DISPLAY "PAUSED FOR " EDIT-NUM8 " SECONDS"
     ELSE
         DISPLAY "INVALID PARAMETER FOR PAUSE INTRINSIC"
     END-IF
     .

 TEST14.
     DISPLAY " "
     DISPLAY "**** TEST 14 ****"
     DISPLAY "Show Control-Y Trap"
     CALL "ARM-CONTROL-Y"
     SET CONTROL-Y-OFF TO TRUE

     SET TEST-NOT-COMPLETE   TO TRUE

     PERFORM UNTIL TEST-COMPLETE
         MOVE 0                  TO REAL-SECONDS
         MOVE ALL "*"            TO DISPLAY-BUFFER

         PERFORM UNTIL CONTROL-Y-HIT
             DISPLAY "HI " WITH NO ADVANCING
             ADD 1 TO REAL-SECONDS
         END-PERFORM

         DISPLAY BELL
         MOVE REAL-SECONDS       TO EDIT-NUM8
         DISPLAY "Control-Y Hit after " EDIT-NUM8 " times"
         DISPLAY DISPLAY-BUFFER
         DISPLAY " "
         DISPLAY "PRESS RETURN TO CONTINUE, 'Z' TO STOP "
                 WITH NO ADVANCING
         ACCEPT TEST-CHAR
         IF  TEST-CHAR = "Z" OR "z"
             SET TEST-COMPLETE   TO TRUE
         ELSE
             SET CONTROL-Y-OFF TO TRUE
         END-IF
     END-PERFORM
     .
 end program DEMO1-14.
```

```
$CONTROL BOUNDS,DYNAMIC
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ARM-CONTROL-Y.
*
*            This program sets the Control Y Trap
*            With grateful thanks to Stan Sieler for recognising
*            the PROCNAME needed underscores instead of hyphens
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01  TOTAL                       PIC S9(9) COMP  VALUE 0.
*                                MUST use underscore not hyphens
 01  PROCNAME                PIC X(20) VALUE "!control_y_trap!".

 01  PLABEL                      PIC S9(9) COMP.
 01  OLDPLABEL                   PIC S9(9) COMP.

 01  PROGFILE                    PIC X(40) VALUE SPACES.
 01  PROGFILE-LENGTH             PIC S9(9) COMP.
 01  I-STATUS                    PIC S9(9) COMP.
 01  CONTROL-Y       EXTERNAL    PIC S9(4) COMP.
     88  CONTROL-Y-HIT                   VALUE 1.
     88  CONTROL-Y-OFF                   VALUE 0.
 01  DISPLAY-BUFFER EXTERNAL     PIC X(40).


 PROCEDURE DIVISION.

 P1.
*        Get Plabel from HPGETPROCPLABEL
     CALL INTRINSIC "HPMYPROGRAM"
                 USING PROGFILE \\ PROGFILE-LENGTH
     DISPLAY "PROGRAM NAME = " PROGFILE(1:PROGFILE-LENGTH)
     CALL INTRINSIC "HPGETPROCPLABEL"
             USING PROCNAME PLABEL I-STATUS PROGFILE
     IF  I-STATUS <> 0
         CALL INTRINSIC "HPERRMSG" USING 2 1 \\ I-STATUS
         STOP RUN
     END-IF
*        Call XCONTRAP
     DISPLAY "CALL XCONTRAP"
     CALL INTRINSIC "XCONTRAP" USING PLABEL OLDPLABEL
     EXIT PROGRAM
     .
*            Can use underscores OR hyphens for the Entry name
 ENTRY "CONTROL_Y_TRAP".

     SET CONTROL-Y-HIT TO TRUE
     DISPLAY "Control Y"
     MOVE "Trap routine re enables again" TO DISPLAY-BUFFER

*        Re enable for next time
     CALL INTRINSIC "RESETCONTROL"
     .
 end program ARM-CONTROL-Y.
```

DEMO16

Demo program to demonstrate the use of circular files.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.     CJEN16.

*    Reads MPE commands from variable record file requesting the
*    name of the file which could be $stdin to provide input from
*    terminal.
*    Requests name of output file and size. Will build circular
*    file if size is not 999999.
*    Writes to the output file after executing the command.
*    If output file is built as a circular file with 20 records,
*    the last 20 commands read will be in the circular file.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     SYMBOLIC CHARACTERS CR IS 14.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
*        Define input file as optional to get end-of-file on first
*        Read if no file exists
     SELECT OPTIONAL IFILE   ASSIGN USING IFILE-NAME
                             FILE STATUS IFILE-STATUS.
     SELECT OFILE            ASSIGN USING OFILE-NAME
                             FILE STATUS OFILE-STATUS.

 DATA DIVISION.
 FILE SECTION.
 FD  IFILE
     RECORD IS VARYING DEPENDING ON ILEN.
 01  IREC.
     03  ICHARS              PIC X   OCCURS 0 TO 80
                                     DEPENDING ON ILEN.

 FD  OFILE.
 01  OREC                    PIC X(80).

 WORKING-STORAGE SECTION.
 01  ILEN                    PIC S9(4) COMP.
 01  ERROR-CODE              PIC S9(4) COMP.
 01  PARM                    PIC S9(4) COMP.
 01  IFILE-STATUS            PIC XX.
 01  OFILE-STATUS            PIC XX.
 01  IFILE-NAME              PIC X(16).
 01  OFILE-NAME              PIC X(16).

 01  END-DATA-IND            PIC X.
     88  END-DATA                        VALUE "E".
     88  DATA-AVAILABLE                  VALUE "A".
```

```
01  BUILD-OFILE.
     03                     PIC X(6)    VALUE "BUILD ".
     03  BUILD-NAME         PIC X(16).
     03                     PIC X(4)    VALUE ";CIR".
     03                     PIC X(16)   VALUE ";REC=-80,,,ASCII".
     03                     PIC X(6)    VALUE ";DISC=".
     03  BUILD-SIZE         PIC 9(6).
     03  BUILD-END          PIC X.

 PROCEDURE DIVISION.
 PARA-1.
*            get name of input file ($stdin for terminal input)
     MOVE SPACES TO IFILE-NAME
     SET DATA-AVAILABLE TO TRUE

     PERFORM UNTIL IFILE-NAME <> SPACES
         DISPLAY "Enter name of input file (// to exit)  ? "
                 WITH NO ADVANCING
         ACCEPT IFILE-NAME FREE
             ON INPUT ERROR
                 DISPLAY "invalid - name too long"
                 MOVE SPACES     TO IFILE-NAME
         END-ACCEPT
         EVALUATE IFILE-NAME
           WHEN SPACES
             DISPLAY "Input File name required or '//'"
           WHEN "//"
             SET END-DATA TO TRUE
         END-EVALUATE
     END-PERFORM

     IF  DATA-AVAILABLE
*            get name of output file
         MOVE SPACES             TO OFILE-NAME
         PERFORM UNTIL OFILE-NAME <> SPACES
             DISPLAY "Enter name of output file (// to exit) ? "
                     WITH NO ADVANCING
             ACCEPT OFILE-NAME FREE
                 ON INPUT ERROR
                     DISPLAY "invalid - name too long"
                     MOVE SPACES TO OFILE-NAME
             END-ACCEPT
             EVALUATE OFILE-NAME
               WHEN SPACES
                 DISPLAY "Output File name required or '//'"
               WHEN "//"
                 SET END-DATA TO TRUE
             END-EVALUATE
         END-PERFORM
     END-IF
```

```
     IF  DATA-AVAILABLE
*            get size of output file
         MOVE 0 TO BUILD-SIZE
         PERFORM UNTIL BUILD-SIZE <> 0
             DISPLAY "Enter size of output file "
                     "(enter 999999 for old file) ? "
                     WITH NO ADVANCING
             ACCEPT BUILD-SIZE FREE
                 ON INPUT ERROR
                     DISPLAY "invalid size "
                     MOVE 0 TO BUILD-SIZE
             END-ACCEPT
         END-PERFORM
         IF  BUILD-SIZE <> 999999
*            Build file if file size is not 999999
             MOVE OFILE-NAME         TO BUILD-NAME
             MOVE CR                 TO BUILD-END
             CALL INTRINSIC "COMMAND"
                 USING BUILD-OFILE ERROR-CODE PARM
             IF  ERROR-CODE <> 0
                 DISPLAY "Cannot build output file - "
                         "Command = " BUILD-OFILE
                         " - Error code = " ERROR-CODE
                 SET END-DATA TO TRUE
             END-IF
         END-IF
     END-IF
*            Open Input file
     IF  DATA-AVAILABLE
         OPEN    INPUT   IFILE
         IF  IFILE-STATUS(1:1) <> "0"
             DISPLAY "Cannot open input file - Status = "
                     IFILE-STATUS
             SET END-DATA TO TRUE
         ELSE
*            Open Output file
             OPEN    OUTPUT  OFILE
             IF  OFILE-STATUS(1:1) <> "0"
                 DISPLAY "Cannot open input file - Status = "
                     IFILE-STATUS
                 SET END-DATA TO TRUE
             END-IF
         END-IF
     END-IF
```

```
*            Process input file
     IF  DATA-AVAILABLE
         PERFORM WITH TEST AFTER
                 UNTIL END-DATA
             IF  IFILE-NAME = "$STDIN"
                 DISPLAY "? " WITH NO ADVANCING
             END-IF
             READ IFILE
                 AT END
                     SET END-DATA TO TRUE
                 NOT AT END
                     IF  IREC = "//"
                         SET END-DATA TO TRUE
                     ELSE
                         PERFORM VARYING ILEN FROM ILEN BY -1
                                 UNTIL ILEN = 1
                                 OR    IREC(ILEN:1) <> " "
                             CONTINUE
                         END-PERFORM
                         IF  IFILE-NAME <> "$STDIN"
                             DISPLAY "========================="
                             DISPLAY "*****" IREC(1:ILEN) "*****"
                             DISPLAY "========================="
                         END-IF
                         MOVE IREC(1:ILEN) TO OREC
                         ADD 1   TO ILEN
                         MOVE CR TO ICHARS(ILEN)
                         CALL INTRINSIC "HPCICOMMAND"
                             USING IREC ERROR-CODE PARM
                         IF  ERROR-CODE = 0
                             WRITE OREC
                         ELSE
                             DISPLAY "Error in Command - "
                                     IREC(1:ILEN - 1)
                                     " - Error code = " ERROR-CODE
                         END-IF
                     END-IF
             END-READ
         END-PERFORM
         CLOSE IFILE OFILE
     END-IF
     STOP RUN.
```

DEMO17

Demo programs to demonstrate the use of message files.

Program CJEN17 – Program to read a message file and continue main processing once all 5 other programs have been processed.
Programs CJEN17A – CJEN17E are programs which must be run before CJEN17.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.     CJEN17.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     SYMBOLIC CHARACTERS     CR 14.

 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT IFILE        ASSIGN "FJEN17".

 DATA DIVISION.
 FILE SECTION.
 FD  IFILE.
 01  IREC                PIC X(8).

 WORKING-STORAGE SECTION.
 01  MPE-ERROR-CODE      PIC S9(4)   BINARY.
 01  MPE-ERROR-TYPE      PIC S9(4)   BINARY.
 01  MPE-COMMAND.
     03                  PIC X(19)   VALUE "TELL TUTOR.TUCOBOL ".
     03  MPE-MESSAGE     PIC X(58).
     03  MPE-END         PIC X.

$DEFINE  %COMMAND=
     MOVE CR             TO MPE-END
     MOVE SPACES         TO MPE-MESSAGE
     STRING !1
            !2
            DELIMITED BY SIZE
            INTO MPE-MESSAGE
     CALL INTRINSIC "HPCICOMMAND" USING MPE-COMMAND
                    MPE-ERROR-CODE MPE-ERROR-TYPE
 #
 01  TRUE-VALUE          PIC S9(4)   BINARY  VALUE 1.
 01  PROGRAM-COUNT       PIC S9(4)   BINARY  VALUE 0.

 01  PROGRAM-TABLE.
     03  PROGRAM-NAME    PIC X(8)    OCCURS 5
                         INDEXED BY PROG-INDEX.
```

```
PROCEDURE DIVISION.
WAITING-TO-GO.
    MOVE SPACES         TO PROGRAM-TABLE
    OPEN INPUT IFILE
    CALL INTRINSIC "FCONTROL" USING IFILE, 45, TRUE-VALUE
    PERFORM UNTIL PROGRAM-COUNT = 5
        READ IFILE
            AT END
                DISPLAY "AT END should not occur"
                STOP RUN
            NOT AT END
                SET PROG-INDEX TO 1
                SEARCH PROGRAM-NAME
                    AT END
                        %COMMAND(IREC#," NO ROOM IN TABLE"#)
                    WHEN PROGRAM-NAME(PROG-INDEX) = SPACE
                        MOVE IREC   TO PROGRAM-NAME(PROG-INDEX)
                        %COMMAND(PROGRAM-NAME(PROG-INDEX)#," "#)
                        ADD 1       TO PROGRAM-COUNT
                    WHEN PROGRAM-NAME(PROG-INDEX) = IREC
                        %COMMAND(IREC#," PROCESSED ALREADY"#)
                END-SEARCH
        END-READ
    END-PERFORM
    CLOSE IFILE
    .
MAIN-PROCESS.
    %COMMAND("CJEN17A THRU CJEN17E NOW COMPLETE"#," "#)
    STOP RUN
    .
```

Program CJEN17A – program to write a message record when it has completed.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     CJEN17A.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OFILE        ASSIGN "FJEN17".
DATA DIVISION.
FILE SECTION.
FD  OFILE.
01  OREC               PIC X(8).

PROCEDURE DIVISION.
MAIN-PROCESS. {Normal program processing }
END-PROCESS.
    OPEN EXTEND OFILE
    MOVE "CJEN17A"      TO OREC
    WRITE OREC
    CLOSE OFILE
    STOP RUN
    .
```

Programs CJEN17B to CJEN17E have similar code to CJEN17A to write a message record when they have completed. Each program will write a unique message record so the main program CJEN17 can recognise which programs have completed.

DEMO18

Demo program to demonstrate Data Base Access and KSAM Files.

```
$CONTROL SOURCE,POST85
*
*   DEMO18
*        1. Request name of Music data base
*        2. Request a 3 character user code to prefix the name of
*           the KSAM file.
*        3. create file containing Album Titles and Composer's
*           Surnames with the appropriate Database key.
*           a. For each entry in the Albums set, write an "A" type
*              record to the KSAM file.
*           b. For each entry in the Composers set, write a "C" type
*              record to the KSAM file.
*           c. Display total number of Album records and total number
*              of Composer records written.
*           d. Save the Music KSAM file as a permanent file.

 IDENTIFICATION DIVISION.
 PROGRAM-ID.     DEMO18.
 AUTHOR.         JEANETTE NUTSFORD - FEBRUARY 2001.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.    HP-3000.
 OBJECT-COMPUTER.    HP-3000.
 SPECIAL-NAMES.
     SYMBOLIC CHARACTERS NULL 1, BELL 8, CR 14, ESCAPE 28
     CLASS VALID-TYPE "A" "C" "E"
     .

 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT MUSIC-INDEX
         ASSIGN  TO "MUSIC,,,,10000"
                 USING MUSIC-INDEX-NAME
         ORGANIZATION IS INDEXED
         ACCESS MODE IS DYNAMIC
         RECORD KEY IS MUSIC-KEY WITH DUPLICATES.

 DATA DIVISION.
 FILE SECTION.
 FD  MUSIC-INDEX.
 01  MUSIC-IREC.
     03  MUSIC-KEY.
         05  MUSIC-TYPE              PIC X.
             88  MUSIC-ALBUM                        VALUE "A".
             88  MUSIC-COMPOSER                     VALUE "C".
         05  MUSIC-NAME              PIC X(39).
     03  MUSIC-DB-KEY.
         05  MUSIC-A-KEY             PIC S9(9) BINARY.
         05                          PIC X(12).
```

```
 WORKING-STORAGE SECTION.
*    Albums set in MUSIC data base
 01  ALBUMS.
     03  ALBUM-CODE                  PIC S9(9) BINARY.
     03  ALBUM-TITLE                 PIC X(40).
     03  MEDIUM                      PIC XX.
     03  ALBUM-COST                  PIC S9(5)V99 PACKED-DECIMAL.
     03  RECORDING-CO                PIC X(16).
     03  DATE-RECORDED               PIC X(16).
     03  MFG-CODE                    PIC X(40).
     03  COMMENT                     PIC X(80).

 01  COMMON-INDICATORS.
     03  END-OF-KEY-IND              PIC X     VALUE "N".
         88  END-OF-KEY                          VALUE "Y".
         88  NOT-END-OF-KEY                      VALUE "N".

 01  COMMON-WORKERS.
     03  CHARS-A                     PIC X(40).
     03  CHARS-B                     PIC X(40).
     03  CHAR-80                     PIC X(80).
     03  COUNT-A                     PIC 99    VALUE 0.
     03  COUNT-B                     PIC 99    VALUE 0.
     03  EDIT-3                      PIC Z(3).
     03  EDIT-4                      PIC Z(4).
     03  EDIT-5                      PIC Z(5).
     03  EDIT-6                      PIC Z(6).
     03  EDIT-DOLLAR                 PIC $(5)9.99.
     03  HOLD-FIRST-NAME             PIC X(16).
     03  MOD1                        PIC 9(4).
     03  MUSIC-INDEX-NAME            PIC X(16).
     03  MUSIC-DB-NAME               PIC X(24).
     03  REQUEST-DATA.
         05  REQUEST-TYPE            PIC X     VALUE SPACE.
             88  REQUEST-ALBUM                   VALUE "A".
             88  REQUEST-COMPOSER                VALUE "C".
             88  REQUEST-END                     VALUE "E".
             88  REQUEST-TYPE-REQD               VALUE SPACE.
         05  REQUEST-NAME            PIC X(40).
     03  USER-INITIALS               PIC XXX.

 01  COUNTERS.
     03  A-COUNT                     PIC S9(4) BINARY.
     03  C-COUNT                     PIC S9(4) BINARY.

*    Composers set in MUSIC data base
 01  COMPOSERS.
     03  COMPOSER-NAME               PIC X(16).
     03  BIRTH                       PIC X(16).
     03  DEATH                       PIC X(16).
     03  BIRTH-PLACE                 PIC X(40).
     03  COMMENT                     PIC X(80).
```

```
 01  DB-WORKERS.
*****       DATA BASE WORKING FIELDS *****
*
  02  FILLER.
     03  DB-MODE1              PIC S9999 COMP VALUE 1.
     03  DB-MODE2              PIC S9999 COMP VALUE 2.
     03  DB-MODE3              PIC S9999 COMP VALUE 3.
     03  DB-MODE4              PIC S9999 COMP VALUE 4.
     03  DB-MODE5              PIC S9999 COMP VALUE 5.
     03  DB-MODE6              PIC S9999 COMP VALUE 6.
     03  DB-MODE7              PIC S9999 COMP VALUE 7.
     03  DB-MODE10             PIC S9999 COMP VALUE 10.
     03  DB-MODE102            PIC S9999 COMP VALUE 102.
     03  DB-MODE201            PIC S9999 COMP VALUE 201.
     03  DB-MODE202            PIC S9999 COMP VALUE 202.
     03  DB-MODE302            PIC S9999 COMP VALUE 302.
     03  DB-OPEN-MODE          PIC S9999 COMP VALUE 5.
  02 IMAGE-FIELDS.
     03  DB-BASE.
         05  DB-BASE-ID       PIC XX     VALUE SPACES.
         05  DB-BASE-NAME     PIC X(24)  VALUE SPACES.
         05  FILLER           PIC XX     VALUE SPACES.
     03  DB-PASSWORD           PIC X(8)  VALUE ";".
     03  DB-READ-PASSWORD      PIC X(8)  VALUE "MGR".
     03  DB-STATUS.
         05  DB-EXCEPT         PIC  S9999 COMP.
             88  DB-SUCCESSFUL           VALUE  0.
             88  START-FILE              VALUE 10.
             88  END-FILE                VALUE 11.
             88  DIRECT-START            VALUE 12.
             88  DIRECT-END              VALUE 13.
             88  START-CHAIN             VALUE 14.
             88  END-CHAIN               VALUE 15.
             88  FULL-SET                VALUE 16.
             88  NO-ENTRY                VALUE 17.
             88  BROKEN-CHAIN            VALUE 18.
             88  NO-UPDATE               VALUE 41.
             88  READ-ONLY               VALUE 42.
             88  DUPLICATE               VALUE 43.
             88  CHAIN-NOT-EMPTY         VALUE 44.
             88  BUFFER-TOO-SMALL        VALUE 50.
             88  DBCB-FULL               VALUE 62.
             88  ILLEGAL-PASSWORD        VALUE -21.
             88  DB-IN-USE               VALUE -32.
             88  DATA-LOCKED             VALUE 20 22 23 24 25.
             88  ODX-ERROR                VALUE 888.
             88  IMSAM-ERROR              VALUE 999.
         05  STAT2             PIC  S9999 COMP.
         05  STAT3-4           PIC  S9(9) COMP.
         05  STAT5-6           PIC  S9(9) COMP.
             88  NO-ENTRY-IN-CHAIN       VALUE 0.
         05  STAT7-8           PIC  S9(9) COMP.
         05  STAT9-10          PIC  S9(9) COMP.
     03  DB-DUMMY              PIC 9(4) COMP VALUE 0.
```

```
     03  DB-SEARCH             PIC X(40).
     03  FILLER REDEFINES DB-SEARCH.
         05  DB-SEARCH-NO      PIC S9(4) COMP.
         05  FILLER            PIC X(38).
     03  DB-QUALIFIER          PIC X(16).
     03  DB-LIST-NAME          PIC X(16).
     03  DB-ARGUMENT           PIC S9(9) COMP VALUE 0.
     03  DB-CONDITION          PIC 9(4).
     03  DB-CONDITION-S  REDEFINES  DB-CONDITION  PIC S9(4).
     03  XDB-CONDITION  REDEFINES DB-CONDITION.
         05  DBPUT-ERROR       PIC 99.
             88  MISSING-CHAINHEAD      VALUE 1.
             88  FULL-CHAIN             VALUE 2.
             88  FULL-AUTOMAST          VALUE 3.
         05  DB-PATH           PIC 99.
     03  DB-DSET-NAME          PIC X(16).
     03  FILLER REDEFINES DB-DSET-NAME.
         05  DB-DSET-NUMBER    PIC S9(4) COMP.
         05  FILLER            PIC X(14).
     03  DB-ALL-ENTRIES        PIC X(4)  VALUE "@;".
     03  DB-CURRENT-LIST       PIC X(4)  VALUE "*;".
     03  DB-EMPTY-LIST         PIC X(4)  VALUE SPACE.
*
  02 DBINFO-BUFFER.
     03  DBINFO-SET-NAME.
         05  DBINFO-SET-NO   PIC S9(4) COMP.
         05  FILLER          PIC X(14).
     03  DBINFO-SET-TYPE     PIC XX.
     03  DBINFO-ENTRY-LENGTH PIC S9(4) COMP.
     03  DBINFO-BLOCK        PIC S9(4) COMP.
     03  FILLER              PIC X(4).
     03  DBINFO-NO-ENTRIES   PIC S9(9) COMP.
     03  DBINFO-CAPACITY     PIC S9(9) COMP.
*
  02  DB-LOCK-ARRAY.
     03  DB-NUMBER-LOCKS     PIC S9(4) COMP.
     03  DB-LOCK-DESC.
       05  DB-LENGTH         PIC S9(4) COMP.
       05  DB-DSET           PIC X(16).
       05  DB-DITEM          PIC X(16).
       05  DB-RELOP          PIC XX.
       05  DB-VALUE          PIC X(20).
  02  DB-LOCK-MODE           PIC S9(4) COMP VALUE 3.
*

*    Selections set in MUSIC data base
 01  SELECTIONS.
     03  ALBUM-CODE                  PIC S9(9) BINARY.
     03  SELECTION-NAME              PIC X(40).
     03  COMPOSER-NAME               PIC X(16).
     03  TIMING.
         05  MM                      PIC 99.
         05                          PIC X.
         05  SS                      PIC 99.
         05                          PIC X(11).
     03  PERFORMERS                  PIC X(40).
     03  COMMENT                     PIC X(80).
```

```
*-----------------------MACROS-----------------------------------

 01  MPE-PARMS.
     03  MPE-ERROR-CODE              PIC S9(4) COMP.
     03  MPE-ERROR-TYPE              PIC S9(4) COMP.
     03  MPE-COMMAND                 PIC X(80).
     03  MPE-MAX                     PIC 999   VALUE 80.

$DEFINE  %CHECKFILEEXISTS=
*START    CHECKFILEEXISTS
     MOVE "LISTF" TO MPE-COMMAND
     MOVE !1 TO MPE-COMMAND(7:)
     MOVE ";$NULL" TO MPE-COMMAND(24:6)
     MOVE CR TO MPE-COMMAND(30:1)
     CALL INTRINSIC "COMMAND" USING MPE-COMMAND MPE-ERROR-CODE
                                    MPE-ERROR-TYPE
*END      CHECKFILEEXISTS#
*-------------------------------------------------------------
*        All variables held in copy library DBWORKS
$DEFINE %DBCLOSE=
*START   DBCLOSE
     IF  DB-BASE-ID <> SPACES
         CALL "DBCLOSE" USING DB-BASE, DB-DSET-NAME, DB-MODE1
                              DB-STATUS
         IF  NOT DB-SUCCESSFUL
             CALL "DBEXPLAIN" USING DB-STATUS
             GOBACK
         END-IF
     END-IF
*END     DBCLOSE#
*-----------------------------------------------------------------
*        !1=Open Mode   !2=DB Base Name
*
*        All variables held in copy library DBWORKS
$DEFINE  %DBOPEN=
*START    DBOPEN
     MOVE !2 TO DB-BASE-NAME
     CALL "DBOPEN" USING DB-BASE, DB-READ-PASSWORD, DB-MODE!1,
                         DB-STATUS
     IF  NOT DB-SUCCESSFUL
         CALL "DBEXPLAIN" USING DB-STATUS
*         GOBACK
     END-IF
*END      DBOPEN#
*---------------------------------------------------------------
*        Serial read Set
*        !1=Set Name/Record Name
*        All variables held in copy library DBWORKS
$DEFINE  %DBSEREAD=
*START    DBSEREAD
     MOVE "!1" TO DB-DSET-NAME
     MOVE SPACES TO !1
     CALL "DBGET" USING DB-BASE, DB-DSET-NAME, DB-MODE2,
         DB-STATUS, DB-CURRENT-LIST, !1, DB-SEARCH
```

```
     IF  !1 = SPACES
     AND NOT END-FILE
         CALL "DBGET" USING DB-BASE, DB-DSET-NAME, DB-MODE1,
             DB-STATUS, DB-ALL-ENTRIES, !1, DB-SEARCH
     END-IF
     IF  NOT (DB-SUCCESSFUL OR  END-FILE)
         CALL "DBEXPLAIN" USING DB-STATUS
     END-IF
*END      DBSEREAD#
*-------------------------------------------------------------
*        !1=ERROR MESSAGE (WITHOUT QUOTES)
*
$DEFINE  %GOBACK=
*START    GOBACK
     DISPLAY "END-OF-PROGRAM - " "!1"
     STOP RUN
*END      GOBACK#
*-------------------------------------------------------------
*            Issue an MPE Command !1 (Max 79 chars)
*            Variables defined above
$DEFINE  %MPECOMMAND=
*START    MPECOMMAND
     MOVE !1 TO MPE-COMMAND
     MOVE CR TO MPE-COMMAND(80:1)
     CALL INTRINSIC "COMMAND" USING MPE-COMMAND
                     MPE-ERROR-CODE MPE-ERROR-TYPE
*END      MPECOMMAND#
*-------------------------------------------------------------
*        !1 = Name of Job Control Variable (literal or variable)
*        !2 = Numeric Value of JCW
*        !3 = text to display (without quotes)
*        !4 = variable to display (for JCW = 10)
 01  JCW-FIELDS.
     03  JCW-NAME                    PIC X(8)  VALUE SPACES.
     03  JCW-VALUE                   PIC S9(4) COMP VALUE 9999.
     03  JCW-STATUS                  PIC S9(4) COMP VALUE 0.
$DEFINE  %TESTJCW=
*START    TESTJCW
     IF  JCW-VALUE = 9999
     OR  !1 <> JCW-NAME
         MOVE 0 TO JCW-VALUE
         MOVE !1 TO JCW-NAME
         CALL INTRINSIC "FINDJCW"
             USING JCW-NAME, JCW-VALUE, JCW-STATUS
     END-IF
     IF  JCW-VALUE = !2
         MOVE !2 TO MOD1
         EVALUATE TRUE
           WHEN MOD1 = 1
             DISPLAY "!3"
           WHEN MOD1 = 10
             DISPLAY "!3" !4
         END-EVALUATE
     END-IF
*END      TESTJCW#
*-------------------------------------------------------------------
```

```
 PROCEDURE DIVISION.
 100-MAIN.
     %TESTJCW("DEMO18"#,1#,1/MAIN#)
     PERFORM 150-MUSICDB-NAME

     IF  MUSIC-DB-NAME = "END"
         MOVE "END" TO USER-INITIALS
     ELSE
         PERFORM 160-CHECK-FOR-INDEX-FILE
     END-IF

     IF  USER-INITIALS = "END"
         DISPLAY BELL "No Index File to be Created"
     ELSE
         IF  MPE-ERROR-CODE = 907
             PERFORM 200-CREATE-INDEX
         ELSE
             DISPLAY BELL "Index File Already Exists"
         END-IF
     END-IF

     STOP RUN
     .
*
*---------------------------------------------------------------
*
 150-MUSICDB-NAME.
     %TESTJCW("DEMO18"#,1#,1/MUSICDB-NAME#)
     MOVE SPACES TO MUSIC-DB-NAME
     MOVE 999 TO DB-STATUS

     PERFORM UNTIL DB-SUCCESSFUL
             OR    MUSIC-DB-NAME = "END"
         MOVE SPACES TO MUSIC-DB-NAME
         PERFORM UNTIL MUSIC-DB-NAME <> SPACES
             DISPLAY "Enter Full name of Music DB (or END)   ?"
                     WITH NO ADVANCING
             ACCEPT MUSIC-DB-NAME FREE
             IF  MUSIC-DB-NAME = SPACES
                 DISPLAY BELL "No Name Entered"
             END-IF
         END-PERFORM

         MOVE FUNCTION UPPER-CASE (MUSIC-DB-NAME) TO MUSIC-DB-NAME
         IF  MUSIC-DB-NAME <> "END"
             %DBOPEN(5#,MUSIC-DB-NAME#)
         END-IF
     END-PERFORM
     .
```

```
*
*-------------------------------------------------------------
*
 160-CHECK-FOR-INDEX-FILE.
     %TESTJCW("DEMO18"#,1#,1/CHECK-FOR-INDEX-FILE#)
     MOVE SPACES TO USER-INITIALS

     PERFORM UNTIL USER-INITIALS <> SPACES
         DISPLAY "Enter your initials (3 chars) (or END) ?"
             WITH NO ADVANCING
         ACCEPT USER-INITIALS FREE
         IF  USER-INITIALS = SPACES
             DISPLAY BELL "No Initials Entered"
         ELSE
             IF  SPACE = USER-INITIALS(1:1) OR
                         USER-INITIALS(2:1) OR
                         USER-INITIALS(3:1)
                 DISPLAY BELL "NO SPACES ALLOWED IN USER INITIALS"
                 MOVE SPACES TO USER-INITIALS
             END-IF
         END-IF
     END-PERFORM

     MOVE FUNCTION UPPER-CASE (USER-INITIALS) TO USER-INITIALS
     IF  USER-INITIALS <> "END"
         MOVE USER-INITIALS TO MUSIC-INDEX-NAME
         MOVE "MUSIX" TO MUSIC-INDEX-NAME(4:5)

         %CHECKFILEEXISTS(MUSIC-INDEX-NAME(1:8)#)
     END-IF
     .
*
*-------------------------------------------------------------
*
 200-CREATE-INDEX.
     %TESTJCW("DEMO18"#,1#,1/CREATE-INDEX#)

     INITIALIZE COUNTERS

     OPEN OUTPUT MUSIC-INDEX

     %DBSEREAD(ALBUMS#)
     MOVE "A" TO MUSIC-IREC

     PERFORM UNTIL END-FILE
         MOVE ALBUM-CODE OF ALBUMS TO MUSIC-A-KEY
         MOVE ALBUM-TITLE OF ALBUMS TO MUSIC-NAME
         WRITE MUSIC-IREC
             INVALID KEY
                 DISPLAY BELL "END OF KSAM FILE"
                 %GOBACK(ERROR CONDITION#)
         END-WRITE
         ADD 1 TO A-COUNT

         %DBSEREAD(ALBUMS#)
     END-PERFORM
```

```
    %DBSEREAD(COMPOSERS#)
    MOVE "C" TO MUSIC-IREC

    PERFORM UNTIL END-FILE
        MOVE COMPOSER-NAME OF COMPOSERS TO MUSIC-DB-KEY
        UNSTRING MUSIC-DB-KEY
            DELIMITED BY SPACE
            INTO HOLD-FIRST-NAME
                 MUSIC-NAME
        END-UNSTRING
        WRITE MUSIC-IREC
            INVALID KEY
                DISPLAY BELL "END OF KSAM FILE"
                %GOBACK(ERROR CONDITION#)
        END-WRITE
        ADD 1 TO C-COUNT

        %DBSEREAD(COMPOSERS#)
    END-PERFORM

    DISPLAY "Music Index Created"
    DISPLAY " "
    MOVE A-COUNT TO EDIT-4
    MOVE C-COUNT TO EDIT-5
    DISPLAY "Albums: " EDIT-4 "   Composers: " EDIT-5

    %DBCLOSE
    CLOSE MUSIC-INDEX

    MOVE "SAVE" TO CHAR-80
    MOVE MUSIC-INDEX-NAME TO CHAR-80(6:16)
    %MPECOMMAND(CHAR-80#)

    .

END PROGRAM DEMO18.
```

DEMO19

Demo program to demonstrate use of TZ variable and GMT by printing the current date in a number of different time zones.

```
$CONTROL SOURCE,POST85
 IDENTIFICATION DIVISION.
 PROGRAM-ID.    SHOWTIME.
 AUTHOR.        Jeanette Nutsford - COMPUTOMETRIC SYSTEMS LTD.
 DATE-WRITTEN.  JANUARY 2001.
*
 DATE-COMPILED.

*Session Compile
*:COB85XLK CJEN19,PJEN19

*----------------------------------------------------------------
*              This program will display the date and time in a
*              number of different time zones.

*              An optional file, called TIMEZONE, can provide up
*              to 100 time zones. Each record (40 CHARS) will
*              contain one time zone followed by a space and time
*              zone description.
*              Default to Baltimore (EST5EDT), London (GMT0BST),
*              Auckland (NZST-12NZDT) and Hawaii (HST10).
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. HP-3000.
 OBJECT-COMPUTER. HP-3000.

 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT OPTIONAL TIMEZONES ASSIGN TO "TIMEZONE"
     .
*
DATA  DIVISION.

 FILE SECTION.

 FD  TIMEZONES.
 01  TIMEZONE-REC               PIC X(40).

 WORKING-STORAGE    SECTION.

 01  FULL-CURRENT-DATE.
     05  C-DATE.
         10  YY                 PIC 9(04) VALUE 0.
         10  MM                 PIC 9(02) VALUE 0.
         10  DD                 PIC 9(02) VALUE 0.
     05  C-TIME.
         10  HOUR               PIC 9(02) VALUE 0.
         10  MINUTE             PIC 9(02) VALUE 0.
         10  SECONDS            PIC 9(02) VALUE 0.
         10  SEC-HUND           PIC 9(02) VALUE 0.
```

```
    05  C-TIME-DIFF.
        10  GMT-DIR            PIC X(01) VALUE SPACE.
        10  HOUR               PIC 9(02) VALUE 0.
        10  MINUTES            PIC 9(02) VALUE 0.

01  SYS-DATE.
        10  YY                 PIC 9(04) VALUE 0.
        10  MM                 PIC 9(02) VALUE 0.
        10  DD                 PIC 9(02) VALUE 0.

01  WK-DATE.
    05  MM                     PIC 9(02) VALUE 0.
    05  DD                     PIC 9(02) VALUE 0.
    05  YY                     PIC 9(04) VALUE 0.

01  WK-TIME.
    05  HOUR                   PIC 9(02) VALUE 0.
    05  MINUTE                 PIC 9(02) VALUE 0.

01  DISP-DATE.
    05  DAYS-NAME              PIC X(03) VALUE SPACES.
    05                         PIC X(02) VALUE ", ".
    05  MONTH                  PIC X(03) VALUE SPACES.
    05                         PIC X(01) VALUE " ".
    05  DD                     PIC Z9    VALUE SPACES.
    05                         PIC X(02) VALUE ", ".
    05  YY                     PIC X(04) VALUE SPACES.

01  DISP-TIME.
    05  HOUR                   PIC Z9    VALUE SPACES.
    05                         PIC X(01) VALUE ":".
    05  MINUTE                 PIC 9(02) VALUE 0.
    05                         PIC X(01) VALUE SPACES.
    05  AM-PM                  PIC X(02) VALUE SPACES.

01  TZ-COUNT                   PIC 9(03).
01  TZ-MAX                     PIC 9(03)   VALUE 100.
01  TZ-TABLE.
    05  TZ-ENTRY               OCCURS 100 INDEXED BY TZ-INDEX.
        10  TZ-VARIABLE        PIC X(12).
        10  TZ-DESCRIPTION     PIC X(30).

01  MONTH-SUB                  PIC 9(02).
01  MONTH-TABLE.
    05  MONTH-VALUES.
        10                     PIC X(06) VALUE "JAN 31".
        10                     PIC X(06) VALUE "FEB 28".
        10                     PIC X(06) VALUE "MAR 31".
        10                     PIC X(06) VALUE "APR 30".
        10                     PIC X(06) VALUE "MAY 31".
        10                     PIC X(06) VALUE "JUN 30".
        10                     PIC X(06) VALUE "JUL 31".
        10                     PIC X(06) VALUE "AUG 31".
        10                     PIC X(06) VALUE "SEP 30".
        10                     PIC X(06) VALUE "OCT 31".
        10                     PIC X(06) VALUE "NOV 30".
        10                     PIC X(06) VALUE "DEC 31".
```

```
    05  MONTH-TABLE-ELEMENTS    REDEFINES MONTH-VALUES.
        10  MONTH-TAB-VALUES    OCCURS 12 TIMES.
            15  MONTH-NAME      PIC X(03).
            15                  PIC X(01).
            15  DAYS-IN-MONTH   PIC 9(02).

01  DAY-SUB                     PIC 9(02).
01  DAY-TABLE.
    05  DAY-VALUES.
        10                      PIC X(03) VALUE "SUN".
        10                      PIC X(03) VALUE "MON".
        10                      PIC X(03) VALUE "TUE".
        10                      PIC X(03) VALUE "WED".
        10                      PIC X(03) VALUE "THU".
        10                      PIC X(03) VALUE "FRI".
        10                      PIC X(03) VALUE "SAT".
    05  DAY-TABLE-ELEMENTS      REDEFINES DAY-VALUES.
        10  DAY-TAB-VALUES      OCCURS 7 TIMES.
            15  DAY-NAME            PIC X(03).

01  TZ-VAR-NAME                 PIC  X(02) VALUE "TZ".
01  TZ-TYPE                     PIC  9(09) COMP SYNC VALUE 2.
01  TZ-VAR-STATUS.
    05  TZ-STATUS-INFO          PIC S9(04) COMP SYNC VALUE 0.
    05  TZ-STATUS-SUBSYS        PIC S9(04) COMP SYNC VALUE 0.
01  TZ-SIZE                     PIC  9(09) COMP SYNC VALUE 12.
01  TZ-VAR-VALUE                PIC  X(12) VALUE SPACES.
01  TZ-TW                       PIC  9(09) COMP SYNC VALUE 2.
01  TZ-ITEM                     PIC S9(09) VALUE 0.

01  HOLD-TZVAR                  PIC  X(12) VALUE SPACES.

01  VAR-NAME                    PIC X(20).
01  TW                          PIC  9(9) COMP SYNC VALUE 1.
01  VAR-STATUS                  PIC S9(9) COMP SYNC VALUE 0.
01  HPDAY-VAR                   PIC 9(09) COMP SYNC VALUE 0.
01  HPDATE-VAR                  PIC 9(09) COMP SYNC VALUE 0.
01  HPDATE-YYYYMMDD.
    03  HPDATE-YYYY             PIC 9(4).
    03  HPDATE-MM               PIC 99.
    03  HPDATE-DD               PIC 99.

01  DISP-STATUS-INFO            PIC 9(04) VALUE 0.

01  INDICATORS.
    03                           PIC X    VALUE SPACE.
        88  TZ-END-OF-FILE               VALUE "Y".

01  POINTERS.
    03  TZ-PTR                  PIC 9(04).
```

```
 PROCEDURE DIVISION.
*****************************************************************
 100-MAINLINE SECTION.
*    --------
 100-START.
*
*    Get variable HPDAY
*
     INITIALIZE HPDAY-VAR
     MOVE "HPDAY"            TO VAR-NAME
     CALL INTRINSIC
          "HPCIGETVAR" USING VAR-NAME,
                             VAR-STATUS,
                             TW,
                             HPDAY-VAR
*
*    Create SYSTEM DATE from HPYYYY + HPMONTH + HPDATE
*
     INITIALIZE HPDATE-YYYYMMDD
     MOVE "HPYYYY"           TO VAR-NAME
     CALL INTRINSIC
          "HPCIGETVAR" USING VAR-NAME
                             VAR-STATUS
                             TW
                             HPDATE-VAR
     MOVE HPDATE-VAR         TO HPDATE-YYYY
*
     MOVE "HPMONTH"          TO VAR-NAME
     CALL INTRINSIC
          "HPCIGETVAR" USING VAR-NAME
                             VAR-STATUS
                             TW
                             HPDATE-VAR
     MOVE HPDATE-VAR         TO HPDATE-MM
*
     MOVE "HPDATE"           TO VAR-NAME
     CALL INTRINSIC
          "HPCIGETVAR" USING VAR-NAME
                             VAR-STATUS
                             TW
                             HPDATE-VAR
     MOVE HPDATE-VAR         TO HPDATE-DD
     MOVE HPDATE-YYYYMMDD    TO SYS-DATE
*
*   Get the value of the TZ variable, if it exists and save it
*
     CALL INTRINSIC
          "HPCIGETVAR" USING TZ-VAR-NAME
                             TZ-VAR-STATUS
                             TZ-TW
                             TZ-VAR-VALUE
     IF  TZ-STATUS-INFO = 0
         MOVE TZ-VAR-VALUE   TO HOLD-TZVAR
     ELSE
         MOVE TZ-STATUS-INFO TO DISP-STATUS-INFO
         DISPLAY "STATUS-INFO = " DISP-STATUS-INFO
     END-IF
```

```
*
*    Load required TZ variables from TIMEZONE file into TZ-TABLE
*
     MOVE SPACES             TO TZ-TABLE
     MOVE 0                  TO TZ-COUNT
     SET TZ-INDEX            TO 1

     OPEN INPUT TIMEZONES
     READ TIMEZONES
         AT END SET TZ-END-OF-FILE   TO TRUE
*             Default to Baltimore (EST5EDT), London (GMT0BST),
*             Auckland (NZST-12NZDT), Hawaii (HST10)
              MOVE "HST10"         TO TZ-VARIABLE(1)
              MOVE "Hawaii"        TO TZ-DESCRIPTION(1)
              MOVE "EST5EDT"       TO TZ-VARIABLE(2)
              MOVE "Baltimore"     TO TZ-DESCRIPTION(2)
              MOVE "GMT0BST"       TO TZ-VARIABLE(3)
              MOVE "London"        TO TZ-DESCRIPTION(3)
              MOVE "NZST-12NZDT"   TO TZ-VARIABLE(4)
              MOVE "Auckland"      TO TZ-DESCRIPTION(4)
              MOVE 4               TO TZ-COUNT
              IF  NOT (HOLD-TZVAR = "HST10" OR "EST5EDT" OR
                                    "GMT0BST" OR "NZST-12NZDT")
                  MOVE HOLD-TZVAR      TO TZ-VARIABLE(5)
                  MOVE "Local Time"    TO TZ-DESCRIPTION(5)
                  MOVE 5               TO TZ-COUNT
              END-IF
     END-READ

     PERFORM UNTIL TZ-END-OF-FILE
             OR    TZ-COUNT > TZ-MAX
         MOVE 1                  TO TZ-PTR
         UNSTRING TIMEZONE-REC
             DELIMITED BY ALL SPACE
             INTO TZ-VARIABLE(TZ-INDEX)
             POINTER TZ-PTR
         MOVE TIMEZONE-REC(TZ-PTR:)  TO TZ-DESCRIPTION(TZ-INDEX)
         SET TZ-INDEX            UP BY 1
         ADD 1                   TO TZ-COUNT
         READ TIMEZONES
             AT END SET TZ-END-OF-FILE TO TRUE
         END-READ
     END-PERFORM

     DISPLAY " "
     PERFORM VARYING TZ-INDEX FROM 1 BY 1
             UNTIL TZ-INDEX > TZ-COUNT
*
*   Now we have to set the TZ variable to TZ-INDEX entry
*
         MOVE "TZ"               TO TZ-VAR-NAME
         MOVE 2                  TO TZ-TYPE
         MOVE 0                  TO TZ-SIZE
         INSPECT TZ-VARIABLE(TZ-INDEX) TALLYING TZ-SIZE
             FOR CHARACTERS BEFORE SPACE
         MOVE TZ-VARIABLE(TZ-INDEX)  TO TZ-VAR-VALUE
```

```
        CALL INTRINSIC
             "HPCIPUTVAR" USING TZ-VAR-NAME
                                TZ-VAR-STATUS
                                2
                                TZ-VAR-VALUE
                                11
                                TZ-SIZE
                                14
                                TZ-ITEM

        MOVE TZ-STATUS-INFO     TO  DISP-STATUS-INFO

        MOVE FUNCTION CURRENT-DATE  TO FULL-CURRENT-DATE

        MOVE HPDAY-VAR          TO DAY-SUB
        MOVE CORR C-DATE        TO WK-DATE
        MOVE CORR C-TIME        TO WK-TIME

        MOVE MM OF WK-DATE      TO MONTH-SUB

        IF HOUR OF WK-TIME >= 12
            SUBTRACT 12 FROM HOUR OF WK-TIME
            MOVE "PM"           TO AM-PM
        ELSE
            MOVE "AM"           TO AM-PM
        END-IF

        MOVE CORR WK-DATE       TO DISP-DATE
        MOVE CORR WK-TIME       TO DISP-TIME

        IF  DD OF WK-DATE > DD OF SYS-DATE
        OR  MM OF WK-DATE > MM OF SYS-DATE
            ADD 1               TO DAY-SUB
            IF  DAY-SUB > 7
                MOVE 1          TO DAY-SUB
            END-IF
        ELSE
            IF  DD OF WK-DATE < DD OF SYS-DATE
            OR  MM OF WK-DATE < MM OF SYS-DATE
                SUBTRACT 1      FROM DAY-SUB
                IF  DAY-SUB < 1
                    MOVE 7      TO DAY-SUB
                END-IF
            END-IF
        END-IF

        MOVE DAY-NAME(DAY-SUB)  TO DAYS-NAME
        MOVE MONTH-NAME(MONTH-SUB) TO MONTH OF DISP-DATE

        DISPLAY TZ-VARIABLE(TZ-INDEX) "  "
                DISP-DATE " "
                DISP-TIME "  "
                TZ-DESCRIPTION(TZ-INDEX)

    END-PERFORM
```

```
*
*   Reset the TZ variable to the original value as found
*   when this program started.  This value was saved as
*   HOLD-TZVAR.
*

     MOVE "TZ"                   TO TZ-VAR-NAME
     MOVE 2                      TO TZ-TYPE
     MOVE 0                      TO TZ-SIZE
     INSPECT HOLD-TZVAR TALLYING TZ-SIZE
         FOR CHARACTERS BEFORE SPACE
     MOVE HOLD-TZVAR             TO TZ-VAR-VALUE

     CALL INTRINSIC
          "HPCIPUTVAR" USING TZ-VAR-NAME
                             TZ-VAR-STATUS
                             2
                             TZ-VAR-VALUE
                             11
                             TZ-SIZE
                             14
                             TZ-ITEM

     DISPLAY " "
     GOBACK.
```