LE z/VSE Run-Time Library Add-Ons

Users Guide

Author
Mr Garry Hasler
Australia Development Laboratory (ADL) - Perth, Western Australia
IBM Australia

Revision Date : 29. Mar. 2016

This document's purpose is to describe the use of new functions provided in the “LE z/VSE Add-Ons” package.

LE z/VSE Add-Ons Table of Contents

Disclaimer

Any pointers, references or links in this publication to external Web-sites are provided for convenience only and do not in any manner serve as an endorsement of those Web-sites.

Supplied LE z/VSE Add-On Run-time Library Functions.

Use of the following run-time library functions and their supporting functions requires the use of LE z/VSE 1.4.8 or above running on z/VSE 5.1 or above.

The LE z/VSE Add-on Library comes with some “core” run-time functions that are designed to enhance and further extend the already capable functions available with LE z/VSE. These are known as the “LE z/VSE Add-On Library Functions”. In addition to these extensions are some further library functions that provide optional extensions to these add-on routines.

Unless otherwise stated, all provided run-time library routines in this package support static calls by all LE-enabled high level languages (HLL). Dynamic calls are supported when stated but use of the PL/I “Release” statement is not supported.

Use by LE-enabled HLASM (High-Level Assembler) is also possible so long as the assembler routine only uses direct static calls or CEEFETCH for dynamic loading of the desired routine. Parameters must be provided in the correct sequence, required content and addressing. Use of OS level services (eg CDLOAD/CDDELETE/LOAD etc) or CEELOAD are not supported. CEERELSE can be used to delete the routine from storage if/when required.

1. **gettimeofday()** : The gettimeofday() function will return the current time, expressed as seconds and microseconds since Epoch (00:00:00 UTC (Universal Time Coordinated), January 1, 1970), and store it in the timeval structure pointed to by “tv”. Supports calls from all LE enabled applications.
2. **ftime() :** The ftime() function sets the time and millitm members of the timeb structure pointed to by “tb” to contain the seconds and milliseconds portions, respectively, of the current time in seconds since Epoch (00:00:00 UTC (Universal Time Coordinated), January 1, 1970). Supports calls from all LE enabled applications.
3. **__getsym()** : The non-ANSI __getsym() function will attempt to locate and then return the value for a given VSE symbol name. These symbol names can be set in JCL via the // SETPARM statement. All levels of symbols will be searched until the first matching entry is found. Then the assigned value will be returned to the caller.

Additional Supporting Routines for the LE z/VSE Run-time Library Add-On Functions.

(All LE z/VSE Only)

1. **ftime__()** : Provide an ILC (inter-language Communication) interface routine into the new LE/C run-time library function ftime() that will allow non-C/VSE applications to convert a timeb structure (produced by the new ftime() function) into a locally time adjusted and formatted character string date to the nearest millisecond resolution.

2. **ctime__()** : Provide an ILC (inter-language Communication) interface routine into the LE/C run-time library function ctime() that will allow non-C/VSE applications to convert a timeval structure (produced by the new gettimeofday() function) into a locally time adjusted and formatted character string date to the nearest microsecond.

3. **tmestamp()** : Provide a standardised time-stamp library function that will produce an ISO 8601:2000 conforming date and time representation time stamp string using the timeval structure (produced by the new gettimeofday() function) information adjusted for local time to the nearest microsecond.
Ref : http://en.wikipedia.org/wiki/ISO_8601

Note : Use of the above additional supporting routines is completely optional and independent of the provided LE z/VSE “Add-On” run-time library routines. These additional routines are supplied only as assisting run-time routines that may prove to be useful.

Installation of Add-On Run-Time Library Functions

The “zip” file that was downloaded contains a z/VSE VTAPE image of the LE z/VSE “Add-On” run-time library functions, any dependant and optional supporting routines. These can be installed in the same library that you currently have LE z/VSE resident in though it will be easier from a management and application control perspective to use a separate sub-library.

The provided “aws” (VTAPE) image file can be used as a remote VTAPE image (if your system and infrastructure is setup to use this) or it can be “ftp'd” to a VTAPE VSAM file on z/VSE using the standard VTAPE VSAM file formats (eg binary, quote site lrecl 32758, quote site recfm v).

Once the “aws” image is available a standard “LIBR” sub-library restore job can be run to install the routines into the desired target sub-library. The following is an example using a VSAM VTAPE file and installing the LE z/VSE Add-On library routines into PRD2.LEADDONS.

```
 DVCDN 591
// VTAPE START,UNIT=591,LOC=VSAM,FILE='VTAPE1',WRITE
 DVCUP 591
// ASSGN SYS006,591
// EXEC LIBR,SIZE=700K
 RESTORE S=PRD2.LEADDONS:PRD2.LEADDONS -
         TAPE=SYS006 RESTORE=ONLINE
/*
/&
```

You can change the second (after the colon) “PRD2.LEADDONS” to your own target library.sublibrary of your choice if desired.

Once the above job has completed successfully the additional library routines are now installed. To use them in your own applications you will need to add your installation library.sublibrary to your compilation and linkedit JCL routines as well as your run-time (execution) JCL routines. It may be easier to just add the installation library.sublibrary to your systems permanent LIBDEF search chain.

Many of the following examples in this manual have example compilation, linkedit and execution JCL which will include the add-on library routines installation library. If you have used another library.sublibrary as your installation target remember to change any sample JCL members you use to include your LE z/VSE “Add-Ons” library.sublibrary.

Installation sub-library sizing report :

```
 L091I SUBLIBRARY PRD2.LEADDONS FOUND ON INPUT TAPE
     APPROXIMATE SPACE REQUIREMENT:     291 LIBRARY BLOCKS
     3375 :      13 TRACKS =      1 CYLINDERS  1 TRACKS
     3380 :      10 TRACKS =      0 CYLINDERS 10 TRACKS
     3390 :      10 TRACKS =      0 CYLINDERS 10 TRACKS
     9345 :      11 TRACKS =      0 CYLINDERS 11 TRACKS
     FBA :      602 BLOCKS
```


LE z/VSE Run-Time Library Add-On Functions

gettimeofday() - get time of day to microseconds

Module Name : gtody (external name : @@GTODY)
Used by : Any LE-enabled language applications. Supported with LE z/VSE 1.4.8 and above only.

Longname : gettimeofday

Call Methods

Supports static (included at linkedit time) calls by any LE-enabled application. Dynamic calls are supported by any LE-enabled High Level Language (HLL) and via CEEFETCH for LE-enabled HLASM (High Level Assembler) routines.

Purpose

Obtains the current time, expressed as seconds and microseconds since 00:00:00 January 1, 1970 (UNIX/ANSI-C Epoch), Universal Time Coordinated (UTC/GMT) to the nearest microsecond and stores it in the “timeval” structure pointed to by “tv”.

```
#include <time.h>
   Converts system time in microseconds into an integer
   number of seconds since Epoch + integer number of
   microseconds. If integer seconds value exceeds 2**31-1,
   gettimeofday() returns -1 without updating caller's timeval
   structure. Otherwise, integer seconds and microsecond values
   are stored in caller's timeval structure and gettimeofday()
   returns 0.

 int gettimeofday(
                 struct timeval *tv,
                 void           *tzp)

 struct timeval {
               time_t      tv_sec;     /* seconds */
               suseconds_t tv_usec;    /* microseconds */
 };
```

Dependency: C/VSE applications need to use : #include “leaddons.h”.

Notes: Requires two parameters however the “tz” parameter is now considered obsolete and should be provided as “NULL” but is kept here for any possible future changes that may include the timezone structure.
Ref : http://www.gnu.org/software/libc/manual/html_node/High_002dResolution-Calendar.html

Error Information:
If tz is not null, sets ERRNO=ENOSYS and returns (-1) to the caller.
Returns zero (0) for success or non-zero (-1) if the system time is not available.

C for VSE/ESA Example (filename: EDCGTODY.C)

```
#include <stdio.h>
#include <stdlib.h>
#include "leaddons.h"

int main(int argc, char *argv[]) {
  struct timeval tv;
  void *tz = NULL;
  char date_time[33];
  char *ctime_s;
  int rc;

  rc = gettimeofday(&tv, tz);
  if (rc == 0) {
    printf("Seconds since Epoch are %i.%6i\n",tv.tv_sec,tv.tv_usec);
 /*
    Format returned time into a char string adjusted to local time
 */
    ctime_s = ctime(&tv.tv_sec);
    sprintf(date_time,"%.10s %.4s %.8s.%.6i ", \
            ctime_s,&ctime_s[20],&ctime_s[11],tv.tv_usec);
    printf("Which is %s\n",date_time);
  }
  else {
    printf("*ERROR* received rc=%d from gettimeofday() call\n",rc);
    perror("   ERRNO ");
  }
  return 0;
}
```

C/VSE Example Compile, Linkedit and execution JCL:

```
// SETPARM EXTLIB='PRD2.LEADDONS'   <== Addons Library
// SETPARM CATLIB='user.testlib'    <== User test library
// SETPARM LELIB='PRD2.SCEEBASE'    <== LE library
// SETPARM CVSE='PRD2.PROD'         <== C/VSE compiler
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&CVSE)
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL
   PHASE EDCGTODY,*
/*
* Compile
// EXEC EDCCOMP,SIZE=(EDCCOMP,100K),                                    X
               PARM='NATLANG(ENU)/ SOURCE NOLIST NORENT NOLONGN TEST'
* $$ SLI MEM=EDCGTODY.C,S=PRD2.LEADDONS
/*
// IF $RC GT 4 THEN
// GOTO $EOJ
* Linkedit
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&CATLIB)
// EXEC LNKEDT,SIZE=256K
* Execute Sample
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&CATLIB)
// EXEC EDCGTODY,SIZE=EDCGTODY
/*
/&
```


Example output :

```
Seconds since Epoch are 1449111830.769489
Which is Thu Dec  3 2015 11:03:50.769489
```

PL/I for VSE/ESA Example (filename: IBMGTDY.P)

```
*PROCESS INCLUDE,SYSTEM(VSE),FLAG(W),XREF(SHORT),NOMAP,NOLIST;
*PROCESS LINECOUNT(100),MACRO;
 /*********************************************/
 /* MODULE/FILE NAME: IBMGTDY.P               */
 /*********************************************/
 /**                                          */
 /** In this example, a call is made to @@gtody  */
 /** to extract the current time of day with     */
 /** microseconds precision. Then, the returned  */
 /** time information is passed to ctime@@()     */
 /** function which will then convert the        */
 /** time into a formatted string including      */
 /** microseconds precision.                     */
 /**                                          */
 /** Note : The functions used in this example   */
 /**   are LE/C library run-time functions.      */
 /**   However they are constructed to be        */
 /**   language independent so need to be        */
 /**   treated by a PL/I program as "assembler". */
 /**                                          */
 /******************************************** */
IBMGTDY: PROC OPTIONS(MAIN,REENTRANT);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

      DCL @@GTODY ENTRY Options(ASM);
      DCL CTIME@@ ENTRY Options(ASM);

      DCL PLIRETV        Builtin;
      DCL DATETIME       Builtin;
      DCL Null_value    INT4 Initial(0);
      DCL CHARDATE      Char(38); /* must be at least 32 bytes long! */

      DCL 01 gtody_s ,
            03 secs     PIC '9999999999.V',
            03 usecs    PIC '999999';

      DCL PLI_D_T     Char(17);
      DCL PLI_D_T_d   PIC '99999999999999';
      DCL PLI_D_T_t   PIC '.999';

      DCL PLI_DateTme PIC '99999999999999';
      DCL PLI_Usecs   PIC '.999999';

      Dcl 01 gtody_timeval,
             03 timeval_time fixed bin(31,0),    /* secs since epoch */
             03 timeval_micros fixed bin(31,0);  /* plus curr u-secs */

 /* ******************************************************** */
 /* Initialise Date/Time receiving fields                    */
```



```
/* ********************************************************** */

 CHARDATE = ' ';

 GTODY_TIMEVAL.TIMEVAL_TIME = 0;
 GTODY_TIMEVAL.TIMEVAL_MICROS = 0;

/* ********************************************************** */
/* Later we will demonstrate how to use the added resolution  */
/* available with the gettimeofday() function to compliment   */
/* the date/time information currently provided by the        */
/* PL/I built-in function.                                    */
/* ********************************************************** */

 PLI_D_T = DATETIME();

/* ********************************************************** */
/* Call 'GetTimeOfDay' to get the number of seconds since     */
/* epoch (1/1/1970) UTC with current microseconds.            */
/*                                                            */
/* The time result will be populated into the 'gtody_timeval' */
/* structure.                                                 */
/* ********************************************************** */

 Call GTODY;

 If PLIRETV = 0 Then
     Do;
        gtody_s.secs = gtody_timeval.timeval_time;
        gtody_s.usecs = gtody_timeval.timeval_micros;

        Put Skip list
               ('IBMGTODY : Time = '||gtody_s.secs||gtody_s.usecs);

/* ********************************************************** */
/* Next we call a front-end to the LE/C run-time library      */
/* function ctime() which will take the @@GTODY timeval       */
/* structure and convert it into a local-time formatted       */
/* date and timestring including the current micro-seconds.   */
/* ********************************************************** */

       Call FrmTime;

       Put Skip List
              ('IBMGTODY : Date = '||Substr(CHARDATE,1,31));
/* ********************************************************** */
/* If very high precision time information is required another */
/* call to "GTODY" would go here to update the current        */
/* "gtody_s" structure information.                           */
/* ********************************************************** */

     PLI_D_T_d = Substr(PLI_D_T,1,14);
     PLI_D_T_t = Substr(gtody_s.usecs,1,3);

    Put Skip List
    ('IBMGTODY : PLI Builtin Date-Time = '||PLI_D_T_d||PLI_D_T_t);


/* ********************************************************** */
```



```
/* Now we show how to improve the date/time resolution currently*/
/* provided by the PLI built-in date/time services using the    */
/* new "gettimeofday" function.                                 */
/* ************************************************************ */

      PLI_DateTme = Substr(PLI_D_T,1,14);
      PLI_Usecs = gtody_s.usecs;

      Put Skip List
      ('IBMGTODY : PLI Date-Time with GTODY = '||PLI_DateTme||PLI_Usecs);
     End;
     Else
     Do;
      Put Skip List
          ('IBMGTODY : *ERROR* Call to @@GTODY has failed!');
      Put Skip List
          ('IBMGTODY : @@GTODY Return-code is : '||PLIRETV);
      Leave;
     End;

 Return;

GTODY: Proc;
/* ************************************************************ */
/* Fetch the 'GetTimeOfDay' function routine PHASE. Then call   */
/* it passing the "timeval" structure as a parameter.           */
/* ************************************************************ */

   Fetch @@GTODY;
   Call @@GTODY(gtody_timeval,Null_value);

END GTODY;

FrmTime: Proc;
/* ************************************************************ */
/* Call the front-end routine CTIME@@ to the LE/C run-time      */
/* library ctime() passing the returned "timeval" structure.    */
/* This will return the date and time (including microseconds)  */
/* information formatted into a string using the "timeval"      */
/* structure as input. The receiving string field MUST be at    */
/* least 32 bytes in length to receive the complete formatted   */
/* string.                                                      */
/* ************************************************************ */

   Fetch CTIME@@;
   Call CTIME@@(gtody_timeval, CHARDATE);

END FrmTime;
END IBMGTDY;
```


PL/I VSE/ESA Example Compile, Linkedit and execution JCL:

```
// SETPARM LELIB='PRD2.SCEEBASE'    <== LE run-time
// SETPARM EXTLIB='PRD2.LEADDONS'   <== Addons library
// SETPARM CATLIB='user.testlib'    <== Test library
// SETPARM PLILIB='PRD2.PROD'       <== PL/I Compiler
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&PLILIB,&CATLIB)
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION CATAL
   PHASE IBMGTDY,*
/*
* Compile
// EXEC IEL1AA,SIZE=256K
* $$ SLI MEM=IBMGTDY.P,S=PRD2.LEADDONS
/*
* Linkedit
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&PLILIB,&CATLIB)
// EXEC LNKEDT,SIZE=512K
/*
// IF $RC GT 4 THEN
// GOTO $EOJ
* Execute
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&PLILIB,&CATLIB)
// EXEC IBMGTDY,SIZE=IBMGTDY
/*
/&
```

Note : Linkeditor information message “2199I” is expected and does not indicate a problem when linkediting sample IBMGTDY.P.

Example Output:

```
IBMGTODY : GTODY = 1449118731.237470
IBMGTODY : Date = Thu Dec  3 2015 12:58:51.237470
IBMGTODY : PLI Builtin Date-Time = 20151203125851.237
IBMGTODY : PLI Date-Time with GTODY = 20151203125851.237470
```


COBOL for VSE/ESA Example (filename : COBGTDY.C)

```
CBL TRUNC(BIN),RENT,APOST,NOSEQ,NODYNAM,MAP,LIB
      IDENTIFICATION DIVISION.
      PROGRAM-ID.    COBGTDY.
      ENVIRONMENT DIVISION.
      CONFIGURATION SECTION.
      SPECIAL-NAMES.
      INPUT-OUTPUT SECTION.
      FILE-CONTROL.
      DATA DIVISION.
      File Section.
      WORKING-STORAGE SECTION.
      01  GetTimeOfDay               PIC X(8)  Value '@@GTODY'.
      01  MYCTIME                    PIC X(8)  Value 'CTIME@@'.
      01  Null-Value                 PIC S9(9) BINARY VALUE 0.
      01  CTIME-DATE                 PIC X(80).
      01  GTODY-SECS                 PIC 9(10).
      01  GTODY-USECS                PIC 9(6).
      01  LE-value.
        03  value-char               PIC X(254).
      01  gtody-timeval.
        03  timeval-time             PIC S9(9) Comp.
        03  timeval-micros           PIC S9(9) Comp.
      LINKAGE SECTION.
      PROCEDURE DIVISION.

         Perform 001-USE-GTODY

         Goback.

     001-USE-GTODY.
         Call GetTimeOfDay Using gtody-timeval, Null-value
         If Return-Code NOT Equal Zero
           Display 'Non-zero ret-code from @@GOTDY call!'
           Move 8 to Return-Code
           Stop Run
         Else
           Cancel GetTimeOfDay
           Move timeval-time To GTODY-SECS
           Move timeval-micros To GTODY-USECS
           Display ' GetTimeOfDay : ' GTODY-SECS '.' GTODY-USECS
           Move spaces to value-char
           Call MYCTIME Using gtody-timeval, value-char
           Move value-char(1:31) To ctime-date
           Display ' GetTimeOfDay Formatted Date : ' ctime-date
         End-if
         Exit.
         Exit Program.
```


COBOL for VSE/ESA Example Compile, Linkedit and execution JCL:

```
// SETPARM LELIB='PRD2.SCEEBASE'    <== LE Library
// SETPARM EXTLIB='PRD2.LEADDONS'   <== Addons Library
// SETPARM CATLIB='user.testlib'    <== User test library
// SETPARM COBVSE='PRD2.PROD'       <== COBOL/VSE Compiler
*  Compile COBGTDY
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION CATAL,XREF,ERRS,DECK,NODUMP,NOSYSDUMP,LOG
 PHASE COBGTDY,*
/*
// LIBDEF *,SEARCH=(&LELIB,&COBVSE,&CATLIB)
// EXEC PGM=IGYCRCTL,SIZE=IGYCRCTL
* $$ SLI MEM=COBGTDY.C,S=PRD2.LEADDONS
/*
// IF $RC GT 4 THEN
// GOTO FINISH
*  Linkedit
// LIBDEF *,SEARCH=(&LELIB,&COBVSE,&CATLIB)
// EXEC LNKEDT,SIZE=512K
/*
// IF $RC GT 4 THEN
// GOTO $EOJ
*  Execute
// LIBDEF *,SEARCH=(&LELIB,&COBVSE,&CATLIB)
// EXEC COBGTDY,SIZE=COBGTDY
/*
/&
* $$ EOJ
```

Example Output :

```
 GetTimeOfDay : 1449118659.675454
 GetTimeOfDay Formatted Date : Thu Dec  3 2015 12:57:39.675454
```


ftime() - time to milliseconds

Module Name : ftime (external name : @ftime)

Used by : Any LE-enabled language applications. Supported with LE z/VSE 1.4.8 and above only.

Longname : N/A

Call Methods

Supports static calls (included at linkedit time) by any LE-enabled application. Dynamic calls are supported by any LE-enabled High Level Language (HLL) and via CEEFETCH for LE-enabled HLASM routines.

Purpose

Sets the “time” and “millitm” fields in the “timeb” structure pointed to by “tb” to contain seconds and milliseconds, respectively, of the current time in the number of seconds since 00:00:00 Universal Time Coordinated (UTC), January 1, 1970 (epoch) to the nearest millisecond.

Format : `int ftime( struct timeb *tb)`

Dependency: C/VSE applications need to use :
#include “leaddons.h”
#define _XOPEN_SOURCE_EXTENDED 1

Error Information:

Returns R15 with zero (0) for success and non-zero (-1) for failure.

Notes:

The ftime() function is provided with this package more for completeness and for the easing of any potential ANSI-C application porting from other platforms.

Both COBOL/VSE and PL/1 for VSE/ESA already provide date/time functions (via LE z/VSE date/time callable services) that include millisecond precision. However the ANSI-C standard only provides for one-second resolution by default for C/VSE applications. The ftime() function now provides this functionality without the need for C/VSE applications to make calls to LE z/VSE date/time callable services.

Just for completeness COBOL and PL/1 samples are provided also to show how to use the ftime() function and its supporting formatting function, ftime__(), in those languages.

C/VSE Example (filename: EDCFTIME.C)

```
#include <stdio.h>
#include <stdlib.h>
#include "leaddons.h"

#define _XOPEN_SOURCE_EXTENDED 1
#define DATESZ 30
#define NOSTOR 12
#define NODATE  8

int main(int argc, char *argv[]) {

 struct timeb tb;
 char *fdate_time;
 char *datetime;
 int rc;

  fdate_time = malloc(DATESZ);
  if (fdate_time == NULL) {
    printf("%s: ERROR! Insufficient storage!\n",argv[0]);
    printf("%s: Unable to acquire %d bytes for testcase\n",\
        argv[0],DATESZ);
    return (NOSTOR);
  }

  memset(fdate_time," ",DATESZ);

  rc = ftime(&tb);
  if (rc == 0) {
    printf("ftime returned %i.%.3i seconds since EPOCH \n", \
          tb.time,tb.millitm);

 /*
 // Format returned time information into a char string adjusted
 //   to the system local time using ftime__()
 */
    rc = ftime__(&tb, fdate_time);
    if (rc == 0) {
      printf("Formatted with ftime__() is : %s\n",fdate_time);
    }
    else {
      printf("%s: ERROR! ftime__() call failed!\n",argv[0]);
      perror("ERRNO ");
      return (NODATE);
    }

 /*
 // Or using the LE/C standard ctime() function convert the time
 //  information returned into a char string adjusted to the
 //  system local time with millisecond resolution.
 */
    datetime = ctime(&tb.time);
    printf("Using ctime() is : %.19s.%d %s\n",datetime, \
           tb.millitm,datetime+20);
 }
 else {
    printf("*ERROR* rc=%d received from ftime() call\n",rc);
```



```
    perror("   ERRNO ");
  }
  return 0;
}
```

C/VSE Example Compile, Linkedit and execution JCL:

```
// SETPARM EXTLIB='PRD2.LEADDONS'   <== Addons Library
// SETPARM CATLIB='user.testlib'    <== User test library
// SETPARM LELIB='PRD2.SCEEBASE'    <== LE library
// SETPARM CVSE='PRD2.PROD'         <== C/VSE compiler
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&CVSE)
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL
   PHASE EDCGTODY,*
/*
* Compile
// EXEC EDCCOMP,SIZE=(EDCCOMP,100K),                                    X
               PARM='NATLANG(ENU)/ SOURCE NOLIST NORENT NOLONGN TEST'
* $$ SLI MEM=EDCFTIME.C,S=PRD2.LEADDONS
/*
// IF $RC GT 4 THEN
// GOTO $EOJ
* Linkedit
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&CATLIB)
// EXEC LNKEDT,SIZE=256K
* Execute Sample
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&CATLIB)
// EXEC EDCFTIME,SIZE=EDCGTODY
/*
/&
```

Example Output :

```
ftime returned 1449119539.603 seconds since EPOCH
Formatted with ftime__() is : Thu Dec  3 2015 13:12:19.603
Using ctime() is : Thu Dec  3 13:12:19.603 2015
```


PL/I for VSE/ESA Example (filename: IBMFTME.P)

```
*PROCESS INCLUDE,SYSTEM(VSE),FLAG(W),XREF(SHORT),NOMAP,NOLIST;
*PROCESS MACRO,LINECOUNT(100);

 /*********************************************
 /* Sample File Name: IBMFTME.P                */
 /* Language : PL/I for VSE/ESA                */
 /*                                            */
 /*  Function: IBMFTME - Call ftime() to get   */
 /*            the current time in seconds since*/
 /*            EPOCH to milliseconds resolution.*/
 /*            Then produce a formatted time    */
 /*            string using the ftime__() service. */
  *********************************************/

IBMFTME: PROC OPTIONS(MAIN,REENTRANT);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL @FTIME EXTERNAL ENTRY Options(ASM);
   DCL FTIME@@ EXTERNAL ENTRY Options(ASM);

   DCL CHARDATE      Char(30); /* must be at least 30 bytes long! */

   DCL 01 timeb_s ,
         03 secs       PIC '9999999999.V',
         03 msecs      PIC '999';

   Dcl 01 ftime_timeb,
         03 time fixed bin(31,0),    /* secs since epoch */
         03 millis fixed bin(31,0);  /* plus curr m-secs */

 /* ********************************************************** */
 /* Initialise Date/Time receiving fields                      */
 /* ********************************************************** */

 CHARDATE = ' ';
 ftime_timeb.time = 0;
 ftime_timeb.millis = 0;

 /* ********************************************************** */
 /* Call 'ftime()' to get the number of seconds since          */
 /* epoch (1/1/1970) UTC with current milliseconds.            */
 /*                                                            */
 /* The time result will be populated into the 'ftime_timeb'   */
 /* structure.                                                 */
 /* ********************************************************** */

  Call FTIME;

  timeb_s.secs = ftime_timeb.time;
  timeb_s.msecs = ftime_timeb.millis;

  Put Skip list
          ('IBMFTIME : ftime() = '||timeb_s.secs||timeb_s.msecs);
```



```
 /* ********************************************************** */
 /* Next we call the ftime__() LE/C run-time library function   */
 /* which will take our 'timeb' structure and convert it into   */
 /* a local-time adjusted date and time string including the    */
 /* milliseconds resolution.                                    */
 /* ********************************************************** */

  Call FrmTime;

  Put Skip List
         ('IBMFTIME : Which is '||Substr(CHARDATE,1,29));

Return;

FTIME: Proc;

 /* ********************************************************** */
 /* Fetch the 'ftime()' function routine PHASE. Then call       */
 /* it passing the "timeb" structure as a parameter.            */
 /* ********************************************************** */

   Fetch @FTIME;
   Call @FTIME(ftime_timeb);

END FTIME;

FrmTime: Proc;

 /* ********************************************************** */
 /* Call the ftime() output formatting routine ftime__()        */
 /* passing the returned "timeb" structure.                     */
 /* This will return the date and time (including milliseconds) */
 /* information formatted into a string using the "timeb"       */
 /* structure as input. The receiving string field MUST be at   */
 /* least 30 bytes in length to receive the complete formatted  */
 /* string.                                                     */
 /* ********************************************************** */

   Fetch FTIME@@;
   Call FTIME@@(ftime_timeb, CHARDATE);

END FrmTime;

END IBMFTME;
```


PL/I for VSE/ESA Example Compile, Linkedit and execution JCL:

```
// SETPARM LELIB='PRD2.SCEEBASE'    <== LE run-time
// SETPARM EXTLIB='PRD2.LEADDONS'   <== Addons library
// SETPARM CATLIB='user.testlib'    <== sample catalog library
// SETPARM PLILIB='PRD2.PROD'       <== PL/I Compiler
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&PLILIB,&CATLIB)
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION CATAL
   PHASE IBMFTME,*
/*
* Compile
// EXEC IEL1AA,SIZE=256K
* $$ SLI MEM=IBMFTME.P,S=PRD2.LEADDONS
/*
* Linkedit
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&PLILIB,&CATLIB)
// EXEC LNKEDT,SIZE=512K
/*
// IF $RC GT 4 THEN
// GOTO $EOJ
* Execute
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&PLILIB,&CATLIB)
// EXEC IBMFTME,SIZE=IBMFTME
/*
/&
```

Note : Linkeditor information message “2199I” is expected and does not indicate a problem when linkediting sample IBMFTME.P.

Example Output :

```
IBMFTIME : ftime() = 1449112868.024
IBMFTIME : Which is Thu Dec  3 2015 11:21:08.209.
```


COBOL for VSE/ESA Example (filename : COBFTME.C)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    COBFTME.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  FTIME                       PIC X(8)  Value '@FTIME'.
       01  MYFTIME                     PIC X(8)  Value 'FTIME@@'.
       01  Null-Value                  PIC S9(9) BINARY VALUE 0.
       01  FTIME-DATE                  PIC X(80).
       01  FTIME-SECS                  PIC 9(10).
       01  FTIME-MSECS                 PIC 9(3).
       01  LE-value.
         03  value-char                PIC X(254).
       01  ftime-timeb.
         03  time-secs                 PIC S9(9) Comp.
         03  millitm                   PIC S9(4) Comp.
       LINKAGE SECTION.
       PROCEDURE DIVISION.

           Perform 001-USE-FTIME

           Goback.

       001-USE-FTIME.
           Call ftime Using ftime-timeb.
           If Return-Code NOT Equal Zero
             Display 'Non-zero ret-code from @FTIME call!'
             Move 8 to Return-Code
             Stop Run
           Else
             Cancel ftime
             Move time-secs To FTIME-SECS
             Move millitm To FTIME-MSECS
             Display ' Ftime Time : ' FTIME-SECS '.' FTIME-MSECS
             Move spaces to value-char
             Call MYFTIME Using ftime-timeb, value-char
             Move value-char(1:29) To ftime-date
             Display ' ftime Formatted Date : ' ftime-date
           End-if
           Exit.

           Exit Program.
```

COBOL for VSE/ESA Example Compile, Linkedit and execution JCL:

See the JCL sample for COBGTDY.C

Example Output :

```
 Ftime Time : 1449041033.991
 ftime Formatted Date : Wed Dec  2 2015 15:23:53.991.
```


__getsym() - Get VSE SYSPARM value

Module Name : __getsym (external name : @@getsym)
Used by : Any LE-enabled language applications. Supported with LE z/VSE 1.4.8 and above only.

Longname : __getsym

Call Methods

Supports static (included at linkedit time) calls by any LE-enabled application. Dynamic calls are supported by any LE-enabled High Level Language (HLL) and via CEEFETCH for LE-enabled HLASM (High Level Assembler) routines.

Purpose

Obtains the current value of a provided symbol name that has previously been set via the // SETPARM JCL statement. All VSE symbol levels are searched until the first matching name is found. Then the value assigned to this symbol name is returned to the caller along with the length of the value.

```
#include <stdio.h>
#include <stdlib.h>
#include “leaddons.h”

void __getsym(char           *sym_name,
              char           *sym_value,
              short int      *sym_length);
```

Dependency: C/VSE applications need to use : #include “leaddons.h”.

Notes: The maximum size supported for symbol values by VSE is 50 characters. So the string area provided for sym_value should be either at least the known size of the value +1 or the maximum of 50 characters. If the length determined (by the position of a null (zero) byte in the sym_value string area) is less than the returned value for the named symbol truncation will be performed. The true length will be returned in the sym_length parameter.

Error Information:
If the requested symbol name cannot be located the “sym_length” parameter will be set to 0. For internal service failures the “sym_length” parameter will be set to -1. Otherwise the sym_value and sym_length fields will contain non-zero values.

C/VSE Example (filename: EDCGTSYM.C)

```
#include <stdio.h>
#include <stdlib.h>
#include "leaddons.h"

#define _XOPEN_SOURCE_EXTENDED 1
#define MAX_VALUE_LEN          51          /* max len (50) plus null */
#define NOSTOR                 8

int main(int argc, char *argv[]) {
  char *sym_value;
  short int  sym_len;
  int rc=0;

 /*
   Verify a parameter has been provided
 */

  if (argc <= 1) {
    printf("%s: At least one parameter is required.\n",argv[0]);
    printf("%s: Specify the name of the symbol you ",argv[0]);
    printf("want the value to be returned for.\n\n");
    printf(" Usage : \n");
    printf("         // EXEC EDCGTSYM,PARM='/\MYSYMB' \n\n");
    return 0;
  }

  sym_value = malloc(MAX_VALUE_LEN);
  if (sym_value == NULL) {
    printf("%s: ERROR! Insufficient storage!\n",argv[0]);
    printf("%s: Unable to acquire %d bytes for execution\n",\
        argv[0],MAX_VALUE_LEN);
    return (NOSTOR);
  }

  memset(sym_value," ",MAX_VALUE_LEN-1); /* set to spaces */
  sym_value[MAX_VALUE_LEN] = 0;     /* add trailing “null” */

  __getsym(argv[1], sym_value, &sym_len);
  if ( sym_len > 0 ) {
    printf("%s: found symbol '%s' with value '%s' for length %d.\n",\
          argv[0],argv[1],sym_value,sym_len);
  }
  else {
    printf("*ERROR* unexpected length %d received ",sym_len);
    printf("from __getsym() call.\n");
    perror("   ERRNO ");
  }

  if (sym_value)
    free(sym_value);

  return 0;
}
```


C/VSE Example Compile, Linkedit and execution JCL:

```
// JOB CCGETSYM - COMPILE PROGRAM EDCGTSYM FOR LEADDONS
 ON $RC GT 16 CONTINUE
// SETPARM EXTLIB='PRD2.LEADDONS'   <== Addons library
// SETPARM CATLIB='user.test'       <== Example PHASE library
// SETPARM LELIB='PRD2.SCEEBASE'    <== LE z/VSE library
// SETPARM CVSE='PRD2.PROD'         <== C/VSE Compiler Library
// LIBDEF *,SEARCH=(&LELIB,&CVSE,&EXTLIB,&CATLIB)
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO COMP
// OPTION ERRS,SXREF,SYM,LIST,NODECK,NOSYSDUMP
// GOTO ENDCAT
/. COMP
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL
   PHASE EDCGTSYM,*
/*
/. ENDCAT
* Compile
// EXEC EDCCOMP,SIZE=(EDCCOMP,100K),                                    X
               PARM='NATLANG(ENU)/SOURCE NORENT NOLONGN TEST(ALL,SYM)'
* $$ SLI MEM=EDCGTSYM.C,S=PRD2.LEADDONS
/*
// IF $RC GT 4 THEN
// GOTO ENDJOB
// IF CATALOG NE 1 THEN
// GOTO NOLNK
/* EXEC EDCPRLK,SIZE=EDCPRLK,PARM='NATLANG(ENU)/UPCASE'
/*
* Linkedit
// EXEC LNKEDT,SIZE=256K
/*
// IF $RC GT 2 THEN
// GOTO ENDJOB
/. NOLNK
* $$ LST DISP=D,CLASS=W,PRI=3,DEST=*
* Execute Sample
// OPTION NODUMP,NOSYSDUMP
// SETPARM MYSYMB='My_Symbol_Value'
// EXEC EDCGTSYM,SIZE=EDCGTSYM,PARM='/MYSYMB'
/*
/. ENDJOB
// EXEC LISTLOG
/&
```

Example Output :

```
// SETPARM MYSYMB='My_Symbol_Value'
// EXEC EDCGTSYM,SIZE=EDCGTSYM,PARM='/MYSYMB'
1S54I  PHASE EDCGTSYM IS TO BE FETCHED FROM PRD2.LEADDONS
EDCGTSYM: found symbol 'MYSYMB' with value 'My_Symbol_Value' for length 15.
1S55I  LAST RETURN CODE WAS 0000
```


COBOL for VSE/ESA Example (filename : COBGTSYM.C)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    COBGTSYM.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  GETSYM                     PIC X(8)  Value '@@GETSYM'.
       01  Null-Value                 PIC S9(9) BINARY VALUE 0.
       01  SYM-NAME                   PIC X(7).
       01  SYM-VALUE                  PIC X(50) Value Spaces.
       01  SYM-LEN                    PIC S9(4) Binary Value 0.
       01  SYM-WORK                   PIC X(50).
       LINKAGE SECTION.
       01  JCL-PARM.
         03  JCL-PARM-LEN           PIC S9(4) Binary.
         03  JCL-PARM-DATA          PIC X(300).

       PROCEDURE DIVISION USING JCL-PARM.
           If JCL-PARM-LEN Greater Than 0 And
              JCL-PARM-LEN Less Than or Equal 7 Then
             Perform 001-USE-GETSYM
           Else
             Display 'COBGTSYM: ** ERROR **'
             If JCL-PARM-LEN Greater Than 7 Then
               Display 'COBGTSYM: Symbol name greater than 7 chars.'
             Else
               Display 'COBGTSYM: No Symbol name parameter provided.'
            End-if
             Display '          Please correct and re-execute.'
           End-if.
           Goback.

       001-USE-GETSYM.
           Move spaces to Sym-work.
           Move JCL-PARM-DATA(1:JCL-PARM-LEN) To Sym-Name.
           Move Length of Sym-value to Sym-len.
           Move X'00' to Sym-value(Sym-Len:1).

           Call GETSYM Using Sym-name, Sym-value, Sym-len.
           If Sym-Len Less Than Or Equal Zero then
             Display 'Non-zero length from @@GETSYM call!'
             Move 8 to Return-Code
             Stop Run
           Else
             Move Sym-value to Sym-Work(1:Sym-Len)
             Display ' The value |' Sym-work(1:Sym-len)
      -      '| was returned for symbol name |' Sym-name
      -      '| with a length of ' Sym-len ' bytes.'
             Move 0 to Return-Code
           End-if.

           Exit.
           Exit Program.
```


COBOL for VSE/ESA Example Compile, Linkedit and execution JCL:

```
// JOB COBGTSYM - COMPILE COBOL SAMPLE COBGTSYM
 ON $RC GT 16 CONTINUE
// SETPARM LELIB='PRD2.SCEEBASE'    <== LE run-time library
// SETPARM EXTLIB='PRD2.LEADDONS'   <== addons library
// SETPARM CATLIB='user.testlib'    <== sample output library
// SETPARM COBVSE='PRD2.PROD'       <== COBOL/VSE compiler
// DLBL VSESPUC,'VSESP.USER.CATALOG',,VSAM
*  Compile COBOL/VSE Main (COBGTSYM)
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION CATAL,XREF,ERRS,DECK,NODUMP,NOSYSDUMP,LOG
 PHASE COBGTSYM,*
/*
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&COBVSE,&FIXLIB,&CATLIB)
// EXEC PGM=IGYCRCTL,SIZE=IGYCRCTL,PARM='SD(&CATLIB(COBGTSYM))'
* $$ SLI ICCF=(COBGTSYM),LIB=(0011)
/*
// IF $RC GT 4 THEN
// GOTO FINISH
*  Linkedit COBGTSYM PHASE
 INCLUDE CEESG003
// EXEC LNKEDT,SIZE=512K
/*
// IF $RC GT 2 THEN
// GOTO $EOJ
* $$ LST CLASS=W,DISP=D,DEST=*
*  Execute Sample
// SETPARM MYSYM='This_is_the_value_for_MYSYM'
// LIBDEF *,SEARCH=(&ELIB,&FIXLIB,&EXTLIB,&CATLIB)
// EXEC COBGTSYM,SIZE=COBGTSYM,PARM='MYSYM/'
/*
/. LSTLOG
// EXEC LISTLOG
/&
* $$ EOJ
```

Example Output :

```
// SETPARM MYSYM='This_is_the_value_for_MYSYM'
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.LEADDONS)
// EXEC COBGTSYM,SIZE=COBGTSYM,PARM='MYSYM/'
1S54I  PHASE COBGTSYM IS TO BE FETCHED FROM PRD2.LEADDONS
 The value |This_is_the_value_for_MYSYM| was returned for symbol name |MYSYM  | with a
length of 00027 bytes.
1S55I  LAST RETURN CODE WAS 0000
```


PL/I for VSE/ESA Example (filename: IBMGTSY.P)

```
IBMGTSY: PROC(vse_parm) OPTIONS(MAIN,REENTRANT);

   DCL @@GETSYM EXTERNAL ENTRY Options(ASM);
   DCL SUBSTR BUILTIN;

   DCL vse_parm      Char(300) varying;

   DCL SYMNAME       Char(7);
   DCL SYMVALUE      Char(51); /* must not exceed 50 (+null) long */
   DCL SYMARRAY(51)  CHAR(1) Based(Addr(SYMVALUE));
   DCL SYMLEN        Fixed Bin(15,0);
   DCL PARMLEN       Fixed Bin(15,0);

 /* ********************************************************* */
 /* Initialise Date/Time receiving fields                     */
 /* ********************************************************* */

 SYMVALUE = ' ';          /* initialise receiving fields */
 SYMLEN = 0;
 SYMARRAY(51) = 0;         /* Add 'null' for __getsym service */
 ParmLen = Length(vse_parm);

 if ( ParmLen > 0 ) & ( ParmLen <= 7 )  Then
   SYMNAME = Substr(vse_parm,1,length(vse_parm));
 else Do;
   if ParmLen = 0 then
     put skip list('IBMGTSYM: No parm supplied for symbol name.');
   else
     put skip list('IBMGTSYM: Supplied parm > 7 characters long.');
   put skip list('          Unable to continue.');
   put skip list('          Please correct and re-run.');
   stop;
 end;

 /* ********************************************************* */
 /* Call __getsym to lookup our symbol name and return the    */
 /* value if any.                                             */
 /* ********************************************************* */

  Fetch @@GETSYM;
  Call @@GETSYM(SYMNAME,SYMVALUE,SYMLEN);

  Put Skip list
      ('IBMGTSY : __getsym returned len = '||SYMLEN);

  If SYMLEN = 0 then
    Put Skip list
      ('IBMGTSY : __getsym unable to locate '||SYMNAME);
  Else
    Put Skip list
      ('IBMGTSY : symbol '||SYMNAME||'= "'||Substr(SYMVALUE,1,SYMLEN)||
      '"');
Return;
END IBMGTSY;
```


PL/I for VSE/ESA Example Compile, Linkedit and execution JCL:

```
// JOB IBMGTSYC - COMPILE PL/I PROGRAM IBMGTSY
// SETPARM LELIB='LE148.BASELIB'    <== LE run-time
// SETPARM EXTLIB='PRD2.LEADDONS'   <== addons library
// SETPARM CATLIB='user.testlib'    <== sample catalog lib
// SETPARM PLILIB='PRD2.PROD'       <== PL/1 Compiler
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&PLILIB,&CATLIB)
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO CAT
// OPTION NODECK
// GOTO ENDCAT
/. CAT
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION CATAL
   PHASE IBMGTSYM,*
/. ENDCAT
* Compile
// EXEC IEL1AA,SIZE=256K
* $$ SLI ICCF=(IBMGTSYM),LIB=(0011)
/*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK
* Linkedit
// EXEC LNKEDT,SIZE=256K
/. NOLNK
// IF $RC GT 4 THEN
// GOTO $EOJ
* $$ LST DISP=D,CLASS=W,PRI=3,DEST=*
* Execute
// SETPARM MYSYMB='My_Symbol_Value_for_IBMGTSYM'
// EXEC IBMGTSYM,SIZE=IBMGTSYM,PARM='/MYSYMB'
/*
/&
* $$ EOJ
```

Example Output :

```
* Execute
// SETPARM MYSYMB='My_Symbol_Value_for_IBMGTSYM'
// EXEC IBMGTSYM,SIZE=IBMGTSYM,PARM='/MYSYMB'
1S54I  PHASE IBMGTSYM IS TO BE FETCHED FROM PRD2.LEADDONS
IBMGTSY : __getsym returned len =        28
IBMGTSY : symbol MYSYMB = "My_Symbol_Value_for_IBMGTSYM"
1S55I  LAST RETURN CODE WAS 0000
```


Supporting Routines for LE z/VSE Run-Time Library Add-Ons

ftime__()

Module Name : ftime__ (external name : ftime@@)

Used by : Any LE-enabled language applications. Supported with LE z/VSE 1.4.8 and above only.

Call Methods : Supports static calls by any LE-enabled HLL application. Dynamic calls are supported by any LE-enabled High Level Language (HLL) and via CEEFETCH for LE-enabled HLASM routines.

Purpose: Takes two parameters from a caller (a pointer to a "timeb" structure and a pointer to a return character string area of at least 30 bytes in length) and returns a return-code result and a character string representation of the converted (documented LE/C run-time ctime() library function output) input timeval structure contents.

Format : `int ftime__( struct timeb *tb, char *str)`
Note : Pointer "*str" must point to a character string area of at least 30-bytes in size.

Error Information:

Returns R15 with zero (0) for success.
ERRNO = EINVAL will be set if an insufficiently sized output character string has been provided.
Non-zero (-1) for failure.

Notes :
Intended for use by PL/I for VSE/ESA, COBOL for VSE/ESA or LE-enabled HLASM applications needing to convert the "timeb" structure information from an earlier call to "ftime()" into a human-readable date and time string.
C for VSE/ESA applications should consider using the LE z/VSE provided ctime() run-time library function.

When linkediting an application that will statically include the "ftime@@" run-time library routine with the ftime() function the following messages may be produced by the linkediter immediately after the include is processed for "ftime@@" :

2139I DUPLICATE SECTION DEFINITION: CEESTART. ***** SECTION IGNORED *****

2139I DUPLICATE SECTION DEFINITION: CEEFMAIN. ***** SECTION IGNORED *****

These are expected and can be ignored.

PL/I for VSE/ESA Example

See the earlier ftime() PL/I sample program for an example use of ftime__() function call by PL/I.

COBOL for VSE/ESA Example

See the earlier ftime() COBOL sample program for an example use of ftime__() function call by COBOL.

ctime__()

Module Name : ctime__ (external name : ctime@@)

Used by : Any LE-enabled language applications. Supported with LE z/VSE 1.4.8 and above only.
C for VSE/ESA applications should consider using the LE z/VSE provided ctime() run-time library function.

Call Methods : Supports static calls by any LE-enabled application. Dynamic calls are supported by any LE-enabled High Level Language (HLL) and via CEEFETCH for LE-enabled HLASM routines.

Purpose: Takes two parameters from a caller (a pointer to a "timeval" structure and a pointer to a return character string area of at least 32 bytes in length) and returns a return-code result and a character string representation of the formatted (documented LE/C run-time ctime() library function output) input timeval structure contents.

Format : `int ctime__( struct timeval *tp, char *str)`
Note : Pointer "*str" must point to a character string area of at least 32-bytes in size.

Error Information:

Returns R15 with zero (0) for success.
ERRNO = EINVAL will be set if an insufficiently sized output character string has been provided.
Non-zero (-1) for failure.

Notes :
Intended for use by PL/I for VSE/ESA, COBOL for VSE/ESA or LE-enabled HLASM applications needing to convert the "timeval" structure information from an earlier call to "getimeofday()" into a human-readable formatted date and time string.
C for VSE/ESA applications should use the LE z/VSE provided ctime() run-time library function.

PL/I for VSE/ESA Example

See the earlier gettimeofday() PL/I sample program for use of ctime__() function call by PL/I.

COBOL for VSE/ESA Example

See the earlier gettimeofday() COBOL sample program for use of ctime__() function call by COBOL.

tmestamp()

Module Name : tmestamp

Used by : Any LE-enabled language applications. Supported with LE z/VSE 1.4.8 and above only.

Call Methods : Supports static calls by any LE-enabled application. Dynamic calls are supported by any LE-enabled High Level Language (HLL) and via CEEFETCH for LE-enabled HLASM routines.

Purpose: Takes two parameters from a caller (a pointer to a "timeval" structure and a pointer to a return character string area of at least 33 bytes in length) and returns a return-code result and a character string conforming to the ISO 8601:2000 standard of the converted (documented ctime() LE/C run-time library function) local time including microseconds.

Format : `int tmestamp(struct timeval *tp, char *timestamp);`
Note : Pointer "timestamp" must point to a character string area of at least 33-bytes in size.

Error Information:

Returns R15 with zero (0) for success.
Returns R15 with Non-zero (-1) for failure.
ERRNO = EINVAL will be set if an insufficiently sized output character string has been provided.

Note : When linkediting an application that will statically include the "tmestamp" run-time library routine with the getimeofday() function the following messages may be produced by the linkediter immediately after the include is processed for "tmestamp" :

2139I DUPLICATE SECTION DEFINITION: CEESTART. ***** SECTION IGNORED *****

2139I DUPLICATE SECTION DEFINITION: CEEFMAIN. ***** SECTION IGNORED *****

These are expected and can be ignored.

System Configuration Notes

To get a time-stamp produced in the local-time format with the time-zone information appended (as per the ISO 8601:2000 standard) the LE/C locale time-zone information needs to be defined. Macro EDCLOCI can be used to do this with the provided JCL member EDCLLOCL.Z.

If the EDCLOCI macro configuration information is not available, the function will instead attempt to use the z/VSE time zone information set at IPL time, via the "SET ZONE" statement. If this information is also not available then the current GMT/UTC formatted time will be returned. This is indicated by a "Z" (as per the ISO 8601:2000 standard) appended to the end of the timestamp string.

Daylight savings note : The local-time conversion routine will include any daylight saving time adjustments for the active locale at execution time according to the EDCLOCI macro definitions. If no locale information is available then so long as the currently active z/VSE "SET ZONE" statement actioned at IPL time adjusts the GMT/UTC offset according to the country daylight saving rules then daylight savings adjustments will be performed. No daylight savings adjustments are performed when defaulting to GMT/UTC time.

C/VSE Example (filename: EDCTMSTP.C)

```
#include <stdlib.h>
#include <stdio.h>
#include "leaddons.h"

int main(int argc, char *argv[]) {

 struct timeval tv;
 void *tz = NULL;
 char *ctime_s;
 char date_time[33];
 char timestamp_s[33];
 int rc;

 /*-------------------------------------------------------------------
   Call the new 'gettimeofday' function to get current date and
   time information to the nearest microsecond.
 -------------------------------------------------------------------*/

   rc = gettimeofday(&tv, tz);

 /*-------------------------------------------------------------------
     Check result and if successful convert the returned tv structure
      contents into an ISO 8601:2000 compliant time-stamp string
      adjusted to local time with time-zone offset to the nearest
      microsecond.
 -------------------------------------------------------------------*/

  if (rc == 0) {
    printf("Time since Epoch is : %i.%.6i\n",tv.tv_sec,tv.tv_usec);
    ctime_s = ctime(&tv.tv_sec);
    sprintf(date_time,"Date String : %.10s %.4s %.8s.%.6i ", \
            ctime_s,&ctime_s[20],&ctime_s[11],tv.tv_usec);
    printf("%s\n",date_time);

    rc = tmestamp(&tv, timestamp_s);

    if (rc == 0)
      printf("ISO 8601:2000 Timestamp is : %s\n",timestamp_s);
    else
      printf(" tmestamp() function call failed!\n");
  }
  else {
    printf("*ERROR* rc=%d received from gettimeofday() call\n",rc);
    perror("   ERRNO ");
    exit(12);
  }

  return 0;

 }
```


C/VSE Example Compile, Linkedit and execution JCL:

```
// SETPARM EXTLIB='PRD2.LEADDONS'  <== Addons Library
// SETPARM CATLIB='user.testlib'              <== User test library
// SETPARM LELIB='PRD2.SCEEBASE'         <== LE library
// SETPARM CVSE='PRD2.PROD'              <== C/VSE Compiler
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&CVSE)
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL
   PHASE EDCTMSTP,*
/*
* Compile
// EXEC EDCCOMP,SIZE=(EDCCOMP,100K),                                          X
               PARM='NATLANG(ENU)/ SOURCE NORENT NOLONGN TEST'
* $$ SLI MEM=EDCTMSTP.C,S=PRD2.LEADDONS
/*
// IF $RC GT 4 THEN
// GOTO $EOJ
* Linkedit
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB)
// EXEC LNKEDT,SIZE=256K
/*
// IF $RC GT 2 THEN
// GOTO $EOJ
* Execute
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&CATLIB)
// EXEC EDCTMSTP,SIZE=EDCTMSTP
/*
/&
```

Example Output (in Perth,WA, Australia Time-Zone) :

```
Time since Epoch is : 1449041674.414303
Date String : Wed Dec  2 2015 15:34:34.414303
ISO 8601:2000 Timestamp is : 2015-12-02T15:34:34.414303+08:00
```


COBOL for VSE/ESA (filename: COBTMSTP.C)

```
CBL TRUNC(BIN),NOOPT,RENT,APOST,NOSEQ,NODYNAM,MAP,LIB
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    COBTSTP.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  Null-value      PIC S9(9) BINARY VALUE 0.
       01  Time-Stamp      PIC X(33).
       01  GetTimeOfDay    PIC X(8)  Value '@@GTODY'.
       01  TimeStamp       PIC X(8)  Value 'TMESTAMP'.
       01  GTODY-SECS      PIC 9(10).
       01  GTODY-USECS     PIC 9(6).
       01  gtody-timeval.
           05  timeval-time    PIC S9(9) Binary.
           05  timeval-micros  PIC S9(9) Binary.
      PROCEDURE DIVISION.
          Display 'COBTSTMP : Start tmestamp() function test.'
                 Upon Console.
          Perform 001-USE-GTODY.
          GOBACK.

      001-USE-GTODY.
          Call GetTimeOfDay Using gtody-timeval, Null-value
          If Return-Code NOT Equal Zero
            Display 'Non-zero ret-code from @@GOTDY call!'
            Move 8 to Return-Code
             Goback
          Else
            Move Timeval-Time TO GTODY-Secs
            Move Timeval-Micros TO GTODY-Usecs
            Display ' GetTimeOfDay : ' GTODY-Secs '.' GTODY-Usecs
            Perform 001-USE-TSTAMP
          End-if

      001-USE-TSTAMP.
          Call TimeStamp Using gtody-timeval, Time-Stamp
          If Return-Code NOT Equal Zero
            Display 'Non-zero ret-code from TimeStamp call!'
            Move 12 to Return-Code
            Stop Run
          Else
            Display ' ISO 8601:2000 TimeStamp : ' Time-Stamp(1:32)
          End-if
          Exit.
          Exit Program.
```


COBOL for VSE/ESA Example Compile, Linkedit and execution JCL:

```
// SETPARM LELIB='PRD2.SCEEBASE'          <== LE Library
// SETPARM EXTLIB='PRD2.LEADDONS'   <== Addons Library
// SETPARM CATLIB='user.testlib'               <== User test library
// SETPARM COBVSE='PRD2.PROD'       <== COBOL/VSE Compiler
*  Compile
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION CATAL,XREF,ERRS,DECK,NODUMP,NOSYSDUMP,LOG
 PHASE COBTMSTP,*
/*
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB,&COBVSE)
// EXEC PGM=IGYCRCTL,SIZE=IGYCRCTL
* $$ SLI MEM=COBTMSTP.C,S=PRD2.LEADDONS
/*
// IF $RC GT 4 THEN
// GOTO $EOJ
*  Linkedit
// LIBDEF *,SEARCH=(&EXTLIB,&LELIB)
// EXEC LNKEDT,SIZE=512K
/*
// IF $RC GT 2 THEN
// GOTO $EOJ
*  Execute COBTMSTP Sample
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&CATLIB)
// EXEC COBTMSTP,SIZE=COBTMSTP
/*
/&
```

Example Output (in Perth,WA, Australia Time-Zone (GMT+8)):

```
 GetTimeOfDay : 1449041112.398624
 ISO 8601:2000 TimeStamp : 2015-12-02T15:25:12.398624+08:00
```


PL/I for VSE/ESA Example (filename: IBMTSTP.P)

```
*PROCESS INCLUDE,SYSTEM(VSE),FLAG(W),XREF(SHORT),NOMAP,NOLIST;
*PROCESS SYSTEM(VSE),MACRO,LINECOUNT(100),NOLIST;
 /*********************************************/
 /* MODULE/FILE NAME: IBMTSTP.P               */
 /*********************************************/
 /**                                          */
 /** Function: IBMTSTP - GetTimeOfDay in      */
 /**               microseconds precision and */
 /**           produce an standard timestamp. */
 /**                                          */
 /** In this example, a call is made to @@gtody  */
 /** to extract the current time of day with     */
 /** micro-seconds included.                     */
 /** Then finally the time information is then   */
 /** passed to the tmestamp() function which     */
 /** will return a string containing the         */
 /** current UTC in the ISO 8601:2000 standard   */
 /** to the current microseconds precision.      */
 /**                                          */
 /** Ref : http://en.wikipedia.org/wiki/ISO_8601 */
 /**                                          */
 /** Note : The functions used in this example   */
 /**   are LE/C library run-time functions.      */
 /**   However they are constructed to be        */
 /**   language independent so need to be        */
 /**   treated by a PL/I program as "assembler". */
 /******************************************** */

IBMTSTP: PROC OPTIONS(MAIN,REENTRANT);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

      DCL @@GTODY EXTERNAL ENTRY Options(ASM);
      DCL CTIME@@ EXTERNAL ENTRY Options(ASM);
      DCL TMESTAMP EXTERNAL ENTRY Options(ASM);

      DCL Null_value    INT4 ;
      DCL CharDate      Char(32); /* must be at least 32 bytes long! */
      DCL TimeStamp     Char(33); /* must be at least 33 bytes long! */

      DCL 01 gtody_s ,
            03 secs       PIC '9999999999.V',
            03 usecs      PIC '999999';

      Dcl 01 gtody_timeval,
             03 timeval_time fixed bin(31,0),    /* secs since epoch */
             03 timeval_micros fixed bin(31,0);  /* plus curr u-secs */

 /* ******************************************************** */
 /* Initialise Date/Time receiving fields.                   */
 /* ******************************************************** */

  CharDate  = ' ';
  TimeStamp = ' ';
```



```
  gtody_timeval.timeval_time = 0;
  gtody_timeval.timeval_micros = 0;
  Null_value = 0;

 /* ********************************************************** */
 /* Call procedure to invoke the gettimeofday() function.      */
 /* ********************************************************** */

  Call GTODY;

  If PLIRETV = 0 Then
    Do;

      gtody_s.secs = gtody_timeval .timeval_time;
      gtody_s.usecs = gtody_timeval .timeval_micros;

      Put Skip list
          ('IBMGTODY : Time = '||gtody_s.secs||gtody_s.usecs);
 /* ********************************************************** */
 /* Next we call a front-end to the LE/C run-time library      */
 /* function ctime() which will take the @@GTODY timeval       */
 /* structure and convert it into a local-time formatted       */
 /* date and timestring including the current micro-seconds.   */
 /* ********************************************************** */

      Call FrmTime;

      Put Skip List ('IBMTSTMP : Date = '||Substr(CharDate,1,31));

 /* ********************************************************** */
 /* Next we call a LE/C function that will produce a ISO 8601  */
 /* standard compliant timestamp string from the provided      */
 /* timeval structure.                                         */
 /* ********************************************************** */

      Call T_Stamp;

      Put Skip List ('IBMTSTMP : ISO 8601:2000 TimeStamp = '
    End;
  Else
    Do;
       Put Skip List
           ('IBMGTODY : *ERROR* Call to @@GTODY has failed!');
       Put Skip List
           ('IBMGTODY : @@GTODY Return-code is : '||PLIRETV);
       Leave;
    End;                                  ||Substr(TimeStamp,1,32));

  Return;


GTODY: Proc;

  /* ********************************************************** */
  /* Fetch the 'GetTimeOfDay' function routine PHASE. Then call  */
  /* it passing the "timeval" structure as a parameter.         */
  /* ********************************************************** */

    Fetch @@GTODY;
```



```
   Call @@GTODY(gtody_timeval,Null_value);

END GTODY;

FrmTime: Proc;

/* ********************************************************** */
/* Call the front-end routine CTIME@@ to the LE/C run-time     */
/* library ctime() passing the returned "timeval" structure.   */
/* This will return the date and time (including microseconds) */
/* information formatted into a string using the "timeval"     */
/* structure as input. The recieving string field MUST be at   */
/* 32 bytes in length to receive the complete formatted string.*/
/* ********************************************************** */

   Fetch CTIME@@;
   Call CTIME@@(gtody_timeval, CharDate);

END FrmTime;

T_Stamp: Proc;

/* ********************************************************** */
/* Fetch the 'TmeStamp' function routine PHASE. Then call      */
/* it passing the "timeval" structure as a parameter.          */
/* ********************************************************** */

   Fetch TmeStamp;
   Call TmeStamp(gtody_timeval, TimeStamp);

END T_Stamp;

END IBMTSTP;
```


PL/I for VSE/ESA Example Compile, Linkedit and execution JCL:

```
// SETPARM EXTLIB='PRD2.LEADDONS'  <== Addons Library
// SETPARM CATLIB='user.testlib'              <== User test library
// SETPARM LELIB='PRD2.SCEEBASE'         <== LE Library
// SETPARM PLILIB='PRD2.PROD'      <== PL/1 VSE Compiler
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&PLILIB)
// LIBDEF PHASE,CATALOG=&CATLIB
// OPTION CATAL
   PHASE IBMTSTMP,*
   INCLUDE CEESG003
/* ENDCAT
* Compile
// EXEC IEL1AA,SIZE=256K
* $$ SLI MEM=IBMTSTMP.P,S=PRD2.LEADDONS
/*
// IF $RC GT 4 THEN
// GOTO $EOJ
* Linkedit
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB)
// EXEC LNKEDT,SIZE=256K,PARM='RMODE=24'
/*
// IF $RC GT 2 THEN
// GOTO $EOJ
* Execute
// LIBDEF *,SEARCH=(&LELIB,&EXTLIB,&CATLIB)
// EXEC IBMTSTMP,SIZE=IBMTSTMP
/*
/&
```

Note : Linkeditor information message “2199I” is expected and does not indicate a problem when linkediting sample IBMTSTMP.P.

Example Output (in Perth,WA, Australia Time-Zone (GMT+8)):

```
IBMTSTMP : Time = 1449041092.642088
IBMTSTMP : Date = Wed Dec  2 2015 15:24:52.642088
IBMTSTMP : ISO 8601:2000 TimeStamp = 2015-12-02T15:24:52.642088+08:00
```


General Application Implementation Notes

Batch Environment Use

When calling any of these documented functions from a non-C/VSE application using a dynamic loading method it is recommended that the “main” program of the application manually include the LE/C signature routine – CEESG003 – at link-edit time. This is to facilitate the initialisation of the LE/C run-time support during the other language (eg COBOL, PL/1) initialisation processing. This will eliminate any possible duplication of initialisation processing. This is not mandatory but it will assist in improving execution performance by reducing initialisation processing duplication.

CICS Environment Use

To ensure minimal CPU overhead is experienced when calling any of these documented functions in a CICS environment from a non-C/VSE application in a dynamic way, the “main” CICS program (either the direct/indirect target of a CICS transaction or the target of an EXEC CICS LINK or XCTL command) should manually include the LE/C signature routine – CEESG003 – at link-edit time. This will ensure that LE/C initialisation is performed at the same time as any other language initialisation and reduce initialisation processing duplication. This is not mandatory but is recommended.