ILE CEE APIs (V5R2)

Table of Contents

ILE CEE APIs

The Integrated Language Environment(R) (ILE) architecture on the OS/400(R) operating system provides a set of bindable application programming interfaces (APIs) known as ILE CEE APIs. In some cases, they provide additional function beyond that provided by a specific high-level language. For example, not all high-level languages (HLL) offer intrinsic means to manipulate dynamic storage. In these cases, you can supplement an HLL function by using appropriate ILE CEE APIs. If your HLL provides the same function as a particular ILE CEE API, use the HLL-specific one.

The ILE CEE APIs are useful for mixed-language applications because they are HLL independent. For example, if you use only condition management ILE CEE APIs with a mixed-language application, you will have uniform condition handling semantics for that application. This uniformity can make condition management easier than when using multiple HLL-specific condition handling models.

The ILE CEE APIs provide a wide-range of functional areas including:

- Activation Group and Control Flow APIs
- Condition Management APIs
- Date and Time APIs
- Math APIs
- Message Services APIs
- Program or Procedure Call APIs
- Storage Management APIs

For more information about using ILE CEE APIs, see the following sections:

- ILE CEE API Calling and Naming Conventions
- Data Type Definitions of ILE CEE
- Omitting Parameters in ILE CEE
- OS/400 Messages and the ILE CEE API Feedback Code

Top | APIs by category

ILE CEE API Calling and Naming Conventions

You access ILE CEE APIs using the same call mechanisms currently used by HLLs to support calls in general.

If you use ILE C, you can use an ILE CEE API with the syntax shown in the following example.

```
#include <leawi.h>
main ()
{
 CEExxxx(&parm1, &parm2, ... &parmn, &fc);
}
```

If you use ILE COBOL, you can call most of the ILE CEE APIs using the following syntax.

```
CALL PROCEDURE 'CEExxxx'
                USING parm1, parm2, ... parmn, fc.
```

The following APIs require a different calling convention in ILE COBOL than that shown above.

- Get String Information (CEEGSI)
- Register a User Written Condition Handler (CEEHDLR)
- Retrieve Operational Descriptor Information (CEEDOD)
- Test for Omitted Argument (CEETSTA)
- Register Call Stack Entry Termination User Exit Procedure (CEERTX)
- Unregister a User Written Condition Handler (CEEHDLU)
- Unregister Call Stack Entry Termination User Exit Procedure (CEEUTX)

In the SPECIAL-NAMES paragraph specify:

```
LINKAGE TYPE
    SYS FOR 'CEExxxx' 'CEEyyyy' ...
```

for the APIs that are to be called. In the PROCEDURE DIVISION, the APIs are called using the following syntax.

```
CALL      'CEExxxx' USING parm1, parm2, ... parmn, fc.
 .
 .
 .
CALL      'CEEyyyy' USING parm1, parm2, ... parmn, fc.
```

If you use ILE RPG, you call an ILE CEE API using the following syntax.

```
C                   CALLB   'CEExxxx'
C                   PARM                    parm1
C                   PARM                    parm2
                    ...
C                   PARM                    parmn
```

```
C                   PARM                 fc
```

Note: Operational descriptors are used for some of the ILE CEE APIs, for example, CEEDATE. Refer to the section on #pragma descriptors in the WebSphere Development Studio: C/C++ Language Reference book for information on how to properly pass arguments to these ILE CEE APIs from inside a C program.

If you use ILE COBOL, you can call functions that require operational descriptors using the following syntax.

```
SPECIAL NAMES.
    LINKAGE PROCEDURE FOR 'CEExxxx' USING ALL DESCRIBED.

PROCEDURE DIVISION.
    CALL PROCEDURE 'CEExxxx'
                USING parm1, parm2, ... parmn, fc.
```

If you use ILE RPG, you can call functions that require operational descriptors using either of the following approaches.

```
C                   CALLB(D)   'CEExxxx'
C                   PARM                 parm1
C                   PARM                 parm2
                    ...
C                   PARM                 parmn
C                   PARM                 fc
```

or

```
D CEExxxx         PR                   OPDESC
D   parm1
D   parm2
D   ...
D   parmn
D   fc

C                   CALLP     CEExxxx(parm1 : parm2 :
C                                     ... : parmn : fc)
```

Note: If you use ILE RPG to call ILE CEE APIs, the following restriction applies:

- APIs cannot be used if DFTACTGRP(*YES) is specified on the CRTBNDRPG CL command. ILE static binding is not available when a program is created with DFTACTGRP(*YES). This means that your program cannot contain a CALLB operation. (Also, you cannot use the BNDDIR or ACTGRP parameters when creating this program.) Specifying DFTACTGRP(*YES) during the creation of an ILE RPG program means that the program will always run in the default activation group. See the WebSphere Development Studio: ILE RPG Programmer's Guide book for more information about the CRTBNDRPG CL command and the DFTACTGRP parameter.

The following descriptions apply to the call syntax shown in the ILE language examples above:

leawi.h

The name of the header file for C prototypes

Note: Header files used with ILE CEE APIs are in library QSYSINC. To ensure that these header files are found during compilation, specify *YES on the SYSINC parameter of the CRTCMOD or CRTBNDC CL commands. Also ensure that the Openness Includes can be selected as an option under GO LICPGM during the installation of the OS/400 operating system.

CEExxxx

The name of the ILE CEE API

parm1, parm2, ... parm*n*

Omissible or required parameters passed to or returned from the called ILE CEE API. The & character in the syntax for ILE C indicates that the parameters are explicitly passed by reference.

fc

An omissible feedback code that indicates the result of the ILE CEE API. The & character in the syntax for ILE C indicates that the parameter is explicitly passed by reference.

The order in which the parameters are listed in the parameter table for each ILE CEE API, is the order in which the parameters must be specified. For example, the following table shows the parameters for the CEEHDLR API:

Required Parameter Group:			
1	procedure	Input	HDLR_ENTRY
2	token	Input	POINTER
Omissible Parameter:			
3	fc	Output	FEEDBACK

When you call the CEEHDLR API, the first parameter you specify must be *procedure* (number 1 in the table), followed by *token* (number 2 in the table), and then *fc* (number 3 in the table).

Naming Conventions of the ILE CEE APIs

Most ILE CEE APIs are available to any HLL that ILE supports. Naming conventions of the ILE CEE APIs are as follows:

- Bindable API names starting with CEE are intended to be consistent across the IBM SAA systems.
- Bindable API names starting with CEE4 are specific to the iSeries system.

For more information about using ILE CEE APIs, see the following sections:

- Data Type Definitions of ILE CEE
- Omitting Parameters in ILE CEE
- OS/400 Messages and the ILE CEE API Feedback Code

Data Type Definitions of ILE CEE APIs

The data types that are used in the parameter tables for each ILE CEE API are defined in Data Type Definitions across ILE Languages. The information in the ILE RPG column assumes RPG D-Specification coding.

Data Type Definitions across ILE Languages

Data Type	Description	ILE C	ILE COBOL	ILE RPG

CHAR	A 1-byte unsigned character	typedef unsigned char _CHAR;	PIC X	`blank or A in data type column`
				`To/L of 1`

UCHAR	A 1-byte unsigned character	typedef unsigned char _UCHAR;	PIC X	`blank or A in data type column`
				`To/L of 1`

SCHAR	A 1-byte signed character	typedef signed char _SCHAR;	PIC X	`blank or A in data type column`
				`To/L of 1`

INT2	A 2-byte signed integer	typedef signed short _INT2;	PIC S9(4) BINARY	`I in data type column`
				`To/L of 5`
				`decimal positions = 0`

UINT2	A 2-byte unsigned integer	typedef unsigned short _UINT2;	PIC 9(4) BINARY	`U in data type column`
				`To/L of 5`
				`decimal positions = 0`

INT4	A 4-byte signed integer	typedef signed int _INT4;	PIC S9(9) BINARY	`I in data type column`
				`To/L of 10`
				`decimal positions = 0`

UINT4	A 4-byte unsigned integer	typedef unsigned int _UINT4;	PIC 9(9) BINARY	`U in data type column`
				`To/L of 10`
				`decimal positions = 0`

FLOAT4	A 4-byte single-precision floating-point number	typedef float _FLOAT4;	COMP-1	`F in data type column`
				`To/L of 4`

FLOAT8	An 8-byte double-precision floating-point number	typedef double _FLOAT8;	COMP-2	`F in data type column`
				`To/L of 8`

COMPLEX8	An 8-byte complex number, whose real and imaginary parts are each 4-byte single-precision floating-point numbers. Used only by ILE math routines.	`typedef struct {`
		`   float  real,`
		`          imaginary;`
		`} _COMPLEX8;`
			`01 complex8`
			`  02 real comp-1`
			`  02 imag comp-1`
				`  Name        To/L`
				`  Entry       Entry`
				`complex8  DS`
				`   real          4F`
				`   imaginary     4F`

COMPLEX16	A 16-byte complex number whose real and imaginary parts are each 8-byte double-precision floating-point numbers. Used only by ILE math routines.	`typedef struct {`
		`   double real,`
		`          imaginary;`
		`} _COMPLEX16;`
			`01 complex16`
			`  02 real comp-2`
			`  02 imag comp-2`
				`  Name        To/L`
				`  Entry       Entry`
				`complex16 DS`
				`   real          8F`
				`   imaginary     8F`

BITS	A set of adjacent bits within a single storage unit. The notation is `_BITS: x`, where x is the field width in bits. (BITS may also be used to define unsigned integers.)	typedef unsigned int _BITS;	Not applicable	Not applicable

POINTER	A platform-dependent address pointer	typedef void * _POINTER;	USAGE IS POINTER	`* in data type column`
				`procedure pointer = ProcPtr`
				`basing pointer is the default if ProcPtr is not defined in the keyword section.`

INVPTR	An invocation pointer	`typedef void * _INVPTR;`
		`#pragma pointer (_INVPTR, INVPTR)`
			Not applicable	Not applicable

LBLPTR	A label pointer	`typedef void * _LBLPTR;`
		`#pragma pointer (_LBLPTR, LBLPTR)`
			Not applicable	Not applicable

CHAR*n*	A string (character array) of length n	typedef char[n] _CHAR[n];	PIC X(n)	`blank or A in data type column`
				`To/L >= 1`

VFLOAT	An ILE variable-length floating-point number used for polymorphic parameter declarations. The length may be any one of 4, 8, or 16 bytes corresponding to single, double, and extended precision.	`typedef union {`
		`   float TypeFloat4;`
		`   double TypeFloat8;`
		`   long double TypeFloat16;`
		`} _VFLOAT;`
			Not applicable	Not applicable

VSTRING	An ILE string of arbitrary length used for polymorphic string parameter declarations. The string may be any one of a fixed-length string, a null-terminated varying string (known as an "ASCIIZ") or a length-prefixed string.	(See note 1)
		`typedef union {`
		`    struct {`
		`       _INT2    length;`
		`       _CHAR255 string;`
		`    } l2pstring;`
		`    struct {`
		`       _INT4    length;`
		`       _CHAR255 string;`
		`    } l4pstring;`
		`    _CHAR1 stringz;`
		`} _VSTRING;`
			`01 string4`
			`  02 len pic 9(9) binary`
			`  02 txt pic x(n)`
			`01 string2`
			`   REDEFINES string4`
			`  02 len pic 9(4) binary`
			`  02 txt pic x(n)`
				`   Name          To/L`
				`   Entry         Entry`
				`vstring    DS`
				`   len         1   2I 0`
				`   txt         3   n`
				`   len2        1   4I 0`
				`   txt2        5   n`

FEEDBACK	A mapping of the feedback (condition) token (fc)	`typedef volatile struct {`
		`   _UINT2 MsgSev;`
		`   _UINT2 MsgNo;`
		`   _BITS Case     :2;`
		`   _BITS Severity :3;`
		`   _BITS Control  :3;`
		`   _CHAR Facility_ID[3];`
		`   _UINT4 I_S_Info;`
		`} _FEEDBACK;`
			`01 fc`
			`  02 sev   pic 9(4) binary`
			`  02 msgno pic 9(4) binary`
			`  02 flgs  pic x(1)`
			`  02 facid pic x(3)`
			`  02 isi   pic 9(9) binary`
				`   Name          To/L`
				`   Entry         Entry`
				`fc         DS`
				`   sev             5U`
				`   msgno           5U`
				`   flags           1`
				`   facid           3`
				`   isi            10U`

CEE-ENTRY	A generic entry constant	`struct {`
		`_POINTER address;`
		`_POINTER nesting;`
		`}`
			`01 STRUC-NAME.`
			`   05 STRUC-ADDRESS POINTER.`
			`   05 STRUC-NESTING POINTER.`
				`DCEE_ENTRY    DS`
				`D Address_Ptr    *   ProcPtr`
				`D Nesting_Ptr    *   ProcPtr`

HDLR_ENTRY	A procedure pointer used on the CEEHDLR and CEEHDLU APIs.	`typedef void (*_HDLR_ENTRY)`
		`           ( _FEEDBACK *,`
		`             _POINTER *,`
		`             _INT4 *,`
		`             _FEEDBACK * );`
			`77 HDLR_ENTRY`
			`     PROCEDURE-POINTER`
				`DHDLR_ENTRY      *   ProcPtr`

RTX_ENTRY	A procedure pointer used on the CEERTX and CEEUTX APIs.	`typedef void (*_RTX_ENTRY)`
		`             ( _POINTER *);`
			`77 RTX_ENTRY`
			`     PROCEDURE-POINTER`
				`DRTX_ENTRY       *   ProcPtr`

RAGE_ENTRY	A procedure pointer used on the CEE4RAGE API.	`typedef void (*_RAGE_ENTRY)`
		`             ( _UINT4 *,`
		`               _UINT4 *,`
		`               _UINT4 *,`
		`               _UINT4 * );`
			`77 RAGE_ENTRY`
			`     PROCEDURE-POINTER`
				`DRAGE_ENTRY      *   ProcPtr`

CEELABEL	A target label to a code point within a call stack entry.	`typedef volatile struct {`
		`    _INVPTR invocation;`
		`    _LBLPTR label;`
		`} _CEELABEL;`
			Not applicable	Not applicable

Note: 1 The typedef for VSTRING is only an indication of the variable string. For ILE C purposes this should be coded as *char **.				

Strong alignment is assumed in all data structures. Each item is aligned on the proper boundary for its type, with padding if necessary.

For more information about using ILE CEE APIs, see the following sections:

- ILE CEE API Calling and Naming Conventions

- Omitting Parameters in ILE CEE
- OS/400 Messages and the ILE CEE API Feedback Code

Top | ILE CEE APIs | APIs by category

Omitting Parameters in ILE CEE APIs

The ILE CEE APIs have parameters that can be omitted. The parameter table for each ILE CEE API uses the term omissible for these parameters.

Warning: It is essential to pass the correct number of parameters, including omitted parameters. You need to use the language-specific syntax for omitted parameters; failure to do so may result in unpredictable results, including a system failure.

For ILE C, you can omit a parameter by passing a null pointer in place of the parameter. The following example in ILE C omits the fc parameter in the call to the API.

```
#include <leawi.h>
main ()
{
 CEExxxx(&parm1, &parm2, ... &parmn, NULL);
}                        /* NULL is used instead  */
                         /* of the omitted parameter */
```

For ILE COBOL, you can omit a parameter by specifying the reserved word OMITTED in place of the parameter. The following example in ILE COBOL omits the fc parameter in the call to the API.

```
CALL PROCEDURE 'CEExxxx'
                USING parm1, parm2, ... parmn, OMITTED.
```

For ILE RPG, you can omit a parameter by specifying *OMIT in the result field of a PARM opcode. The following example omits the fc parameter in the call to the API.

```
C                   CALLB   'CEExxxx'
C                   PARM                 parm1
C                   PARM                 parm2
                    ...
C                   PARM                 parmn
C                   PARM                 *OMIT
```

For ILE RPG if you use a prototyped call when you code the prototype, you specify OPTIONS(*OMIT) in the keywords area of the omissible parameter. Then you can specify *OMIT for that parameter on the call.

```
D CEExxxx         PR                   OPDESC
D   parm1
D   parm2
D   ...
D   parmn
D   fc                                 OPTIONS(*OMIT)

C                   CALLP     CEExxxx(parm1 : parm2 :
C                                      ... : parmn : *OMIT)
```

For more information about using ILE CEE APIs, see the following sections:

- ILE CEE API Calling and Naming Conventions
- Data Type Definitions of ILE CEE
- OS/400 Messages and the ILE CEE API Feedback Code

OS/400 Messages and the ILE CEE API Feedback Code

As input to an ILE CEE API, you have the option of coding a feedback code, and using the feedback code as a return (or feedback) code check in a procedure. The feedback code is a condition token value that is provided for flexibility in checking returns from calls to other procedures. You can then use the feedback code as input to a condition token. For information on the condition tokens, see Condition Management APIs. For more information on condition handling, see **Condition Handling** in the ILE Concepts book.

If you code the feedback code parameter in your application to receive feedback information from an ILE CEE API, the following sequence of events occurs when a condition is raised:

1. An informational message is sent to the caller of the API, communicating the message associated with the condition.
2. The ILE CEE API in which the condition occurred builds a condition token for the condition. The ILE CEE API places information into the instance specific information area. The instance specific information of the condition token is the message reference key of the informational message. This is used by the system to react to the condition.
3. If a detected condition is critical (severity is 4), the system sends an exception message to the caller of the ILE CEE API.
4. If a detected condition is not critical (severity less than 4), the condition token is returned to the routine that called the ILE CEE API.
5. When the condition token is returned to your application, you have the following options:
   - Ignore it and continue processing.
   - Signal the condition using the Signal a Condition (CEESGL) bindable API.
   - Get, format, and dispatch the message for display using the Get, Format, and Dispatch a Message (CEEMSG) bindable API.
   - Store the message in a storage area using the Get a Message (CEEMGET) bindable API.
   - Use the Dispatch a Message (CEEMOUT) bindable API to dispatch a user-defined message to a destination that you specify.
   - When the caller of the API regains control, the informational message is removed and does not appear in the job log.

If you omit the feedback code parameter when you are calling an ILE CEE API, the ILE CEE API sends an exception message to the caller of the bindable API.

Activation Group and Control Flow APIs

ILE CEE APIs are provided to manage activation groups and to determine the control flow of procedures. The activation group and control flow APIs are:

- Abnormal End (CEE4ABN) abnormally ends the activation group containing the nearest control boundary.
- Find a Control Boundary (CEE4FCB) searches the call stack for the nearest call stack entry that is a control boundary.
- Normal End (CEETREC) is used to do a normal ending of the activation group containing the nearest control boundary.
- Register Activation Group Exit Procedure (CEE4RAGE) is used to register procedures that are called when an activation group ends.
- Register Call Stack Entry Termination User Exit Procedure (CEERTX) registers a user-defined procedure that runs when the call stack entry for which it is registered is ended by anything other than a return to the caller.
- Unregister Call Stack Entry Termination User Exit Procedure (CEEUTX) is used to unregister a user-defined procedure that was previously registered by the Register Call Stack Entry Termination User Exit Procedure (CEERTX) API.

Abnormal End (CEE4ABN) API

Omissible Parameter Group:

1	raise_TI	Input	INT4
2	cel_rc_mod	Input	INT4
3	user_rc	Input	INT4

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Abnormal End (CEE4ABN) API abnormally ends the activation group containing the nearest control boundary. The termination-imminent condition can be sent first to give all intervening call stack entries a chance to clean up or stop the abnormal end. This is optional. All call stack entries to the nearest control boundary are ended, unless the resume cursor is moved while handling the terminate-imminent condition. If the call stack entry for the control boundary is also the oldest call stack entry in the activation group, the activation group ends. The exception message CEE9901 (application error) is sent to the caller of the control boundary, whether or not the activation group ended, provided that call stack entries were ended.

Omissible Parameter Group

raise_TI (input)

Whether or not the terminate-imminent condition should be raised before the end operation.

0 The terminate-imminent condition is not raised; the abnormal end operation starts immediately. This value is the default if *raise_TI* is omitted.

1 The terminate-imminent condition is raised. The abnormal end operation occurs only if the handling of the terminate-imminent condition results in the resume cursor not being moved after a resume operation was requested, or, the terminate-imminent condition is not handled.

cel_rc_mod (input)

A language-specific return code passed from one ILE language to another ILE language. The value and meaning is language-specific.

user_rc (input)

A number representing the user portion of the activation group return code. If this parameter is not supplied, the CEE4ABN API uses the current contents of the activation group return code. If it is supplied, it takes precedence over previously set values.

Conditions

CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- High-level language statements that implement abnormal ending of the activation group do so by calling the CEE4ABN API.
- The job-level return codes are updated whether or not the activation group ended, provided that call stack entries were canceled.
- This API cannot end the default activation group.

API Introduced: V2R3

Find a Control Boundary (CEE4FCB) API

Omissible Parameter Group:

1	ctlbdy_inv	Output	INT4
2	ctlbdy_type	Output	INT4
3	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Find a Control Boundary (CEE4FCB) API searches the call stack for the nearest call stack entry that is a control boundary. Beginning with the caller of the CEE4FCB API, a search for a control boundary starts from the call stack entry of the caller and progresses to older call stack entries.

Omissible Parameter Group

ctlbdy_inv (output)

A positive number indicating the control boundary call relative to the caller of the CEE4FCB API.

ctlbdy_type (output)

The type of the control boundary found.

0	The control boundary found is the oldest call stack entry in the activation group.
1	The control boundary found is not the oldest call stack entry in the activation group.

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

API Introduced: V2R3

Top | ILE CEE APIs | APIs by category

Normal End (CEETREC) API

Omissible Parameter Group:

1	cel_rc_mod	Input	INT4
2	user_rc	Input	INT4

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Normal End (CEETREC) API is used to do a normal ending of the activation group containing the nearest control boundary. First, the terminate-imminent condition is sent to give all intervening call stack entries a chance to clean up, or stop the end operation. All call stack entries to the nearest control boundary end unless the resume cursor is moved while handling the terminate-imminent condition. If the call stack entry for the control boundary is also the oldest call stack entry in the activation group, the activation group ends, provided that call stack entries were ended.

Omissible Parameter Group

cel_rc_mod (input)

A language-specific return code passed from one ILE language to another ILE language. The value and meaning is language-specific.

user_rc (input)

A number representing the user portion of the activation group return code. If this parameter is not supplied, the CEETREC API uses the current contents of the activation group return code. If it is supplied, it takes precedence over previously set values.

Conditions

CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- A normal end operation by high-level language exit statements is implemented by calling the CEETREC API. The termination-imminent condition is sent to the control boundary.

- The activation group and job-level return codes are updated whether or not the activation group ended, provided call stack entries were canceled.

- This API cannot end the default activation group.

API Introduced: V2R3

Register Activation Group Exit Procedure (CEE4RAGE) API

Required Parameter:			
1	procedure	Input	RAGE_ENTRY
Omissible Parameter:			
2	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Register Activation Group Exit Procedure (CEE4RAGE) API is used to register procedures that are called when an activation group ends. Activation group exit procedures, registered by CEE4RAGE, are called after HLL user exit procedures, but before any system level activation group resource clean up takes place. The procedures are called in the reverse order of their registration. If a procedure fails, subsequent procedures will not be called.

There is no practical limit to the number of procedures that can be registered. If the same procedure is registered multiple times, it is called multiple times.

Note: This API cannot be called from code running in the default activation group.

Required Parameter

procedure (input)

An entry variable or constant for the procedure that is to be called at activation group termination.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0257	The procedure provided for &1 is not valid
Severity: 30	
CEE3101	&1 cannot be called in the default activation group
Severity: 30	
CEE3103	Cannot allocate storage in &1
Severity: 30	
CEE3111	&1 cannot be called at this time
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The message CEE0257 occurs if *procedure* is not a procedure pointer, or if the procedure identified by *procedure* is not in either the current activation group or the default activation group.
- Once the activation group exit procedures start to run, the CEE4RAGE API cannot be called.

Interface to the Activation Group Exit Procedure

An activation group exit procedure is called when the activation group is ended. The procedure is coded as **activation_group_exit** with the following parameters.

Required Parameter Group:

1	ag_mark	Input	UINT4
2	reason	Input	UINT4
3	result_code	I/O	UINT4
4	user_rc	I/O	UINT4

Required Parameter Group

ag_mark (input)

The activation group mark that uniquely identifies the activation group within the job.

reason (input)

The reason for the activation group being ended. See Common Reason Codes for Ending Activation Groups and Call Stack Entries for a description of the reason codes.

result_code (I/O)

The value passed as input is the action to be taken as specified by a previous exit procedure. The value passed to the first exit procedure is 0. The output value can specify an action to be taken. If the result code does not match any of the following actions, the output value is ignored and the previous action remains unchanged.

No action

0 Do not change the action.

Recover

10 Do not perform any pending error requests. This is used if a previous exit procedure specified a result code of 20 and a subsequent procedure recovers from the error. The message CEE9901, indicating an application error, is not sent.

Failure

20 Send message CEE9901 to the caller of the control boundary after the remaining exit procedures are called.

21 Send message CEE9901 to the caller of the control boundary. The remaining exit procedures registered by the CEE4RAGE API are not called. This is used if an unrecoverable error occurs in the exit procedure requesting this action.

Note: The application error message CEE9901 is sent after the activation group resources of the system are taken down and the activation group has ended.

user_rc (I/O)

The value passed as input is the *user_rc* returned as output from the previous exit procedure. The value passed to the first exit procedure is 0.

Common Reason Codes for Ending Activation Groups and Call Stack Entries.

Bit	Description
Bits 0	Reserved
Bits 1	Call stack entry is canceled because an exception message was sent.
Bits 2-15	Reserved
Bit 16	0 - normal end 1 - abnormal end

Bit 17	Activation Group is ending.
Bit 18	Initiated by the Reclaim Activation Group (RCLACTGRP) command.
Bit 19	Initiated as a result of the job ending.
Bit 20	Initiated by an exit verb, for example exit() in C, or the CEETREC API.
Bit 21	Initiated by an unhandled function check.
Bit 22	Call stack entry canceled because of an out-of-scope jump, for example *longjmp()* in C.
Bits 23-31	Reserved (0)

API Introduced: V2R3

Register Call Stack Entry Termination User Exit Procedure (CEERTX) API

Required Parameter:			
1	procedure	Input	RTX_ENTRY
Omissible Parameter:			
2	token	Input	POINTER
3	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Register Call Stack Entry Termination User Exit Procedure (CEERTX) API registers a user-defined procedure that runs when the call stack entry for which it is registered is ended by anything other than a return to the caller.

Required Parameter

procedure (input)

An entry variable or constant for the procedure that is called if the call stack entry is abnormally ended.

Omissible Parameter

token (input)

A pointer that is passed to *procedure*. If *token* is omitted, a null pointer is passed to *procedure* when *procedure* is called.

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0256	Procedure already registered, registered again
Severity: 10	
CEE0257	The procedure provided for &1 is not valid
Severity: 30	
CEE3103	Cannot allocate storage in &1
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- Multiple user call stack entry termination user exit procedures can be registered for each call stack entry.
- Call stack entry termination user exit procedures run in first in/first out (FIFO) order. If a procedure causes any of the following, the remaining procedures are not run:
    - The call to a call stack entry termination user exit procedure fails.
    - An exception is not handled.
    - Control has moved past the invocation of the call stack entry termination user exit procedure. For example, the resume cursor has been moved, or an out-of-scope jump, such as *longjmp()* in ILE C, has been used.
- The message CEE0257 occurs only if the pointer contained in the procedure parameter is not a procedure pointer. This does not verify that a call through that pointer will be successful. For example, the call may fail because the activation group containing the procedure no longer exists.

Interface to the Call Stack Entry Termination User Exit Procedure

The following is the interface to the call stack entry termination user exit procedure registered by the CEERTX API. The procedure is coded as **termination_procedure** with the following parameter.

Required Parameter:		
1	token	Input POINTER

Required Parameter

token (input)

The user-supplied pointer passed on the call to CEERTX that registered the call stack entry termination user exit procedure.

API Introduced: V2R3

Top | ILE CEE APIs | APIs by category

Unregister Call Stack Entry Termination User Exit Procedure (CEEUTX) API

Required Parameter:			
1	procedure	Input	RTX_ENTRY
Omissible Parameter:			
2	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Unregister Call Stack Entry Termination User Exit Procedure (CEEUTX) API is used to unregister a user-defined procedure that was previously registered by the Register Call Stack Entry Termination User Exit Procedure (CEERTX) API. The CEEUTX API operates on the call stack entry termination user exits that are registered for the call stack entry from which the CEEUTX API is called.

Required Parameter

procedure (input)
: An entry variable or constant for the procedure that is to be unregistered for the call stack entry.

Omissible Parameter

fc (output)
: A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0252	&1 is unable to find the procedure
Severity: 30	
CEE0253	Additional registrations of the procedure remain in the queue

Severity: 10	
CEE0257	The procedure provided for &1 is not valid
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- Registered call stack entry termination user exit procedures are automatically unregistered upon the return from the call stack entry for which they are registered.
- If the same procedure is registered for the call stack entry more than once, the CEEUTX API processes the registrations in LIFO order.
- The message CEE0257 occurs if the pointer contained in *procedure* is not a procedure pointer.

API Introduced: V2R3

Condition Management APIs

ILE condition management APIs allow you to handle errors independent of high-level language-specific error handling.

See Using Condition Management APIs for information on:

- how conditions are represented.
- condition token layout.
- condition token testing.

The condition management APIs are:

- Construct a Condition Token (CEENCOD) is used to dynamically construct a 12-byte condition token.
- Decompose a Condition Token (CEEDCOD) returns the individual elements of an existing condition token.
- Handle a Condition (CEE4HC) handles a specified condition and, optionally, logs the condition.
- Move the Resume Cursor to a Return Point (CEEMRCR) moves the resume cursor to a return point relative to the current handle cursor.
- Register a User-Written Condition Handler (CEEHDLR) registers a user-written condition handler for the current call stack entry.
- Retrieve ILE Version and Platform ID (CEEGPID) retrieves the ILE version ID and the platform (that is, operating system) ID. The IDs are those currently in use for processing the active condition.
- Return the Relative Invocation Number (CEE4RIN) retrieves the relative invocation number of an invocation pointer, and returns it in rel_inv.
- Signal a Condition (CEESGL) signals or resignals a condition to the ILE condition manager.
- Unregister a User-Written Condition Handler (CEEHDLU) unregisters a user-written condition handler for the current call stack entry.

Using Condition Management APIs

How Conditions Are Represented

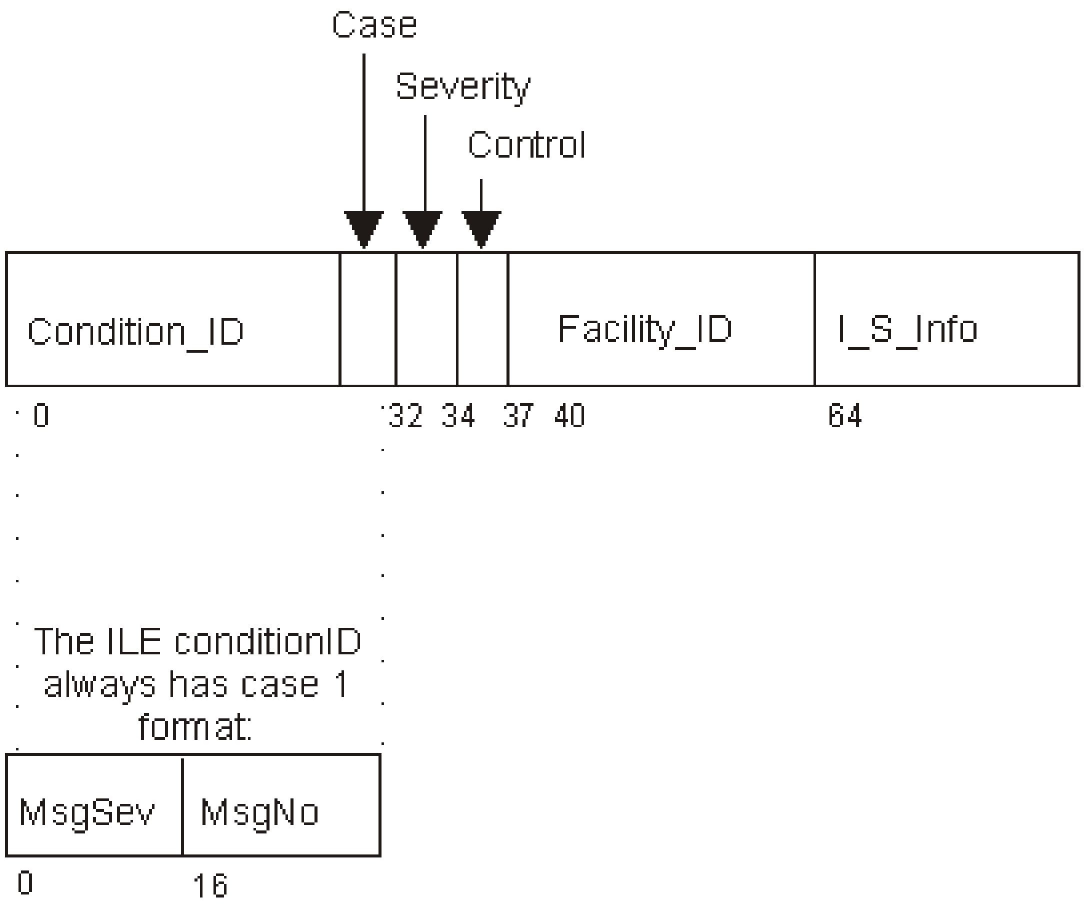
A condition token is used to communicate information about a condition to ILE CEE APIs. It is also used to communicate information to the condition manager, to message services, and to procedures.

The ILE **condition token** is a 12-byte compound data type that contains structured fields to convey aspects of a condition. It conveys aspects such as the severity, the associated message number, and information that is specific to the given instance of the condition.

Condition Token Layout

Figure 1 displays a map of the condition token.

Figure 1. ILE Condition Token Layout



Every condition token contains the components indicated in the figure above.

Condition_ID

A 4-byte identifier that, with the Facility_ID, describes the condition that the token communicates. ILE CEE APIs and most applications produce Case 1 conditions.

Case

A 2-bit field that defines the format of the Condition_ID portion of the token. ILE conditions are always Case 1.

Severity

A 3-bit binary integer that indicates the condition's severity. Severity and MsgSev contain the same information. See Table 1 for a list of ILE condition severities. See Table 2 and Table 3 for the corresponding OS/400 message severities.

Control

A 3-bit field containing flags that describe or control various aspects of condition handling. The third bit specifies whether the Facility--ID has been assigned by IBM.

Facility--ID

A 3-character alphanumeric string that identifies the facility that generated the condition. Although all ILE languages use ILE message and condition handling ILE CEE APIs, the actual run-time messages generated under ILE still carry the HLL identification in the Facility--ID.

I--S--Info

A 4-byte field that identifies the instance specific information associated with a given instance of the condition. The instant specific information contains the reference key to the instance of the message associated with the condition token. If the message reference key is zero, there is no associated message.

MsgSev

A 2-byte binary integer that indicates the condition's severity. MsgSev and Severity contain the same information. See Table 1 for a list of ILE condition severities. See Table 2 and Table 3 for the corresponding OS/400 message severities.

MsgNo

A 2-byte, hexadecimal number that identifies the message associated with the condition. The combination of Facility_ID and MsgNo uniquely identifies a condition.

Table 1 contains default responses that the condition manager takes when a handler attempts to percolate a function check across a control boundary.

Table 1. Default Responses to Unhandled Exceptions

Message Type	Severity of Condition	Condition Raised by the Signal a Condition (CEESGL) Bindable API	Exception Originated from Any Other Source
Status	0 (Informative message)	Return the unhandled condition.	Resume without logging the message.

Status	1 (Warning)	Return the unhandled condition.	Resume without logging the message.
Notify	0 (Informative message)	Not applicable.	Log the notify message and send the default reply.
Notify	1 (Warning)	Not applicable.	Log the notify message and send the default reply.
Escape	2 (Error)	Return the unhandled condition.	Log the escape message and send a function check message to the call stack entry of the current resume point.
Escape	3 (Severe error)	Return the unhandled condition.	Log the escape message and send a function check message to the call stack entry of the current resume point.
Escape	4 (Critical ILE error)	Log the escape message and send a function check message to the call stack entry of the current resume point.	Log the escape message and send a function check message to the call stack entry of the current resume point.
Function check	4 (Critical ILE error)	Not applicable	End the application, and send the CEE9901 message to the caller of the control boundary.

Note: When the application is ended by an unhandled function check, the activation group is deleted if the control boundary is the oldest call stack entry in the activation group.

Table 2 and Table 3 show how ILE condition severity maps to OS/400 message severity.

Table 2. Mapping OS/400 *ESCAPE Message Severities to ILE Condition Severities

From OS/400 Message Severity	To ILE Condition Severity	To OS/400 Message Severity
0-29	2	20
30-39	3	30
40-99	4	40

Table 3. Mapping OS/400 *STATUS and *NOTIFY Message Severities to ILE Condition Severities

From OS/400 Message Severity	To ILE Condition Severity	To OS/400 Message Severity
0	0	0
1-99	1	10

Condition Token Testing

You can test a condition token that is returned from an ILE CEE API for the following:

Success

To test for success, determine if the first 4 bytes are 0. If the first 4 bytes are 0, the remainder of the condition token is 0, indicating a successful call was made to the ILE CEE API.

Equivalent Tokens

To determine whether two condition tokens are equivalent (that is, the same *type* of condition token, but not the same *instance* of the condition token), compare the first 8 bytes of each condition token with one another. These bytes are static and do not change depending upon the given instance when the condition occurs.

Equal Tokens

To determine whether two condition tokens are equal (that is, they represent the same *instance* of a condition), compare all 12 bytes of each condition token with one another. The last 4 bytes can change from instance to instance of a condition.

Construct a Condition Token (CEENCOD) API

Required Parameter Group:

1	c_1	Input	INT2
2	c_2	Input	INT2
3	case	Input	INT2
4	severity	Input	INT2
5	control	Input	INT2
6	facility_id	Input	CHAR3
7	i_s_info	Input	INT4
8	cond_token	Output	FEEDBACK

Omissible Parameter:

9	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Construct a Condition Token (CEENCOD) API is used to dynamically construct a 12-byte condition token.

Required Parameter Group

c-1 (input)

A 2-byte binary integer representing the value of the first part of the Condition_ID. See MsgSev.

c-2 (input)

A 2-byte binary integer representing of the value of the second part of the Condition_ID. See the description of MsgNo.

case (input)

A 2-byte binary integer that defines the format of the Condition_ID portion of the token. See the description of Case.

severity (input)

A 2-byte unsigned binary integer that indicates the condition's severity. The value of this field is the same as in C-1.

control (input)

A 2-byte binary number containing the control information of the condition. See the description of Control.

facility_id (input)

A 3-character field containing three alphanumeric characters that identify the product generating this condition or feedback information. See the description of Facility-ID.

i_s_info (input)

A 4-byte handle identifying the instance specific information. See the description of I-95 S-Info.

cond_token (output)

The 12-byte representation of the constructed condition or feedback information. See How Conditions Are Represented.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0401	The Case parameter for &1 is not valid
Severity: 30	
CEE0402	Unsupported control code &2 passed to procedure &1
Severity: 30	
CEE0403	Severity passed to &1 is not valid
Severity: 30	

Usage Notes

- If the severity in C_1 does not match the severity in the Severity parameter, message CEE0403 is raised.

API Introduced: V2R3

Decompose a Condition Token (CEEDCOD) API

Required Parameter Group:

1	cond_token	Input	FEEDBACK
2	c_1	Output	INT2
3	c_2	Output	INT2
4	case	Output	INT2
5	severity	Output	INT2
6	control	Output	INT2
7	facility_id	Output	CHAR3
8	i_s_info	Output	INT4

Omissible Parameter:

9	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Decompose a Condition Token (CEEDCOD) API returns the individual elements of an existing condition token.

Required Parameter Group

Cond_Token (input)

A condition token that represents the current condition or feedback information. See How Conditions Are Represented

C--1 (output)

A 2-byte binary integer representing the value of the first part of the Condition_ID. See the description of MsgSev.

C--2 (output)

A 2-byte binary integer representing of the value of the second part of the Condition_ID. See the description of MsgNo.

Case (output)

A 2-byte binary integer that defines the format of the Condition_ID portion of the token. See the description of Case.

Severity (output)

A 2-byte unsigned binary integer that indicates the severity of the condition. The value of this field is the same as in C--1.

Control (output)

A 16-bit field containing flags that describe aspects of the state of the condition. See the description of Control.

Facility_ID (output)

A 3-character field containing three alphanumeric characters that identify the product generating this condition or feedback information. See the description of Facility--ID.

I_S_Info (output)

A 4-byte handle identifying the instance specific information. See the description of I--S--Info.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0102	The condition token passed to &1 is not valid
Severity: 30	

API Introduced: V2R3

Handle a Condition (CEE4HC) API

Required Parameter Group:

1	isi	Input	UINT4
2	inv	Input	INVPTR

Omissible Parameter Group:

3	option	Input	UINT4
4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Handle a Condition (CEE4HC) API handles a specified condition and, optionally, logs the condition.

Required Parameter Group

isi (input)

The instance specific information number for the condition. This is the message reference key of the underlying server exception.

inv (input)

The call stack entry that the handle cursor points to for the condition. This is the target call stack entry of the underlying server exception.

Omissible Parameter Group

option (input)

The options are:

0	Take the default log action for the message. *ESCAPE and function check messages are logged. *STATUS messages are never logged.
1	Do not log the condition.
2	Log the condition. Only conditions with a severity greater than 1 can be logged.

If the option parameter is omitted, the value 0 is used.

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE3104	The pointer type in &1 is not valid
Severity: 30	
CEE3105	The call stack entry given to &1 no longer exists
Severity: 30	
CEE3107	The message specified for &1 is not an exception message
Severity: 30	
CEE3108	The option specified for &1 is not valid
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The call stack entry that *inv* points to must appear in the call stack for the job issuing the call to the CEE4HC API.
- The CEE4HC API can be used to take a previously handled message out of the job log, but only if that message is enqueued on *inv*.
- The CEE4HC API can be used to handle *ESCAPE, *STATUS, *NOTIFY, and function check exceptions. A default reply is sent for *NOTIFY messages if no reply has been sent previously.

API Introduced: V2R3

Move the Resume Cursor to a Return Point (CEEMRCR) API

Required Parameter:			
1	type_of_move	Input	INT4
Omissible Parameter:			
2	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Move the Resume Cursor to a Return Point (CEEMRCR) API moves the resume cursor to a return point relative to the current handle cursor.

Initially, the resume cursor is positioned after the machine instruction that caused the condition to be raised. The direction of movement is always toward older call stack entries.

Required Parameter

type_of_move (input)

The type of movement of the resume cursor relative to the current position of the handle cursor. The values are:

0 Move the resume cursor from a later call stack entry to the call stack entry that is currently associated with the handle cursor.

Chose this option if resumption in the call stack entry at which the resume cursor is pointing is not possible, but may be possible in the call stack entry at which the handle cursor is pointing.

1 Move the resume cursor to the *call return* point (immediately following the *call* statement) for the call stack entry one prior to the position of the handle cursor. Also move the handle cursor to the first handler of the call stack entry the resume cursor is being moved to. This action exits the current call stack entry and skips all condition handlers still to be called for the call stack entry.

Chose this option if resumption is impossible at the current location, but may be possible in the caller of the current procedure.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0254	The type of move for &1 is not valid
Severity: 10	
CEE0260	No active condition for call to &1
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- If you attempt to move the resume cursor with a type_of_move of 0 and the resume cursor and the handle cursor are at the same call stack entry, the move is not valid.
- The actual movement of the resume cursor occurs only after the condition handler returns to the condition manager. If two or more calls from a given user-written condition handler set the resume cursor to different places, the most recent call will be used.
- When a return operation is made to the condition manager after the resume cursor is moved, any associated exit procedures are called for each call stack entry that is passed. Moving a resume cursor past a call stack entry also cancels any associated user-written condition handlers.
- Ensure that the CEEMRCR API is called before handling the condition. A call to the CEEMRCR API is not valid if there is no active condition. For example, the condition may not be active if the condition handler uses the message handler API, Change Exception Message (QMHCHGEM), to handle the condition.
- When doing a type 1 move, the exception message is immediately moved to the call message queue of the call stack entry one older than that indicated by the handle cursor. Therefore, when using APIs to perform some action on the exception (for example, QMHCHGEM), you must indicate a target invocation of one older than the original handle cursor.

API Introduced: V2R3

Register a User-Written Condition Handler (CEEHDLR) API

Required Parameter Group:

1	procedure	Input	HDLR_ENTRY
2	token	Input	POINTER

Omissible Parameter:

3	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe:

The Register a User-Written Condition Handler (CEEHDLR) API registers a user-written condition handler for the current call stack entry.

Required Parameter Group

procedure (input)

An entry variable or constant for the procedure that is to be called to process the conditions.

token (input)

A pointer passed to the user-written condition handler at the time the procedure is called.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0256	Procedure already registered, registered again
Severity: 10	

CEE0257	The procedure provided for &1 is not valid
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- CEEHDLR is implemented as a builtin and therefore cannot have its address taken or be called through a procedure pointer.
- A queue of handlers is maintained for each call stack entry. The handlers are given control in LIFO order. That is, the handler most recently registered is the first one used for the call stack entry from which the call to CEEHDLR was made.
- Any registered user-written condition handlers that were not unregistered by the Unregister a User-Written Condition Handler (CEEHDLU) API, are unregistered automatically by ILE upon removal of the associated call stack entry from the call stack. For more information on unregistering user-written condition handlers, see Unregister a User-Written Condition Handler (CEEHDLU) API.
- The message CEE0257 occurs if the pointer contained in *procedure* is not a procedure pointer.

ILE Condition Handler Interface

Following is a description of the interface that the system uses to communicate with ILE condition handlers.

Required Parameter Group:

1	C_CTOK	Input	FEEDBACK
2	token	Input	POINTER
3	result_code	Output	INT4
4	new_condition	Output	FEEDBACK

Required Parameter Group

C_CTOK (input)

Identifies the current condition being processed.

token (input)

The token that was passed to the system with the call to CEEHDLR that registered this condition handler.

result_code (output)

This field contains the instructions from the condition handler to the system regarding the actions that the system should take.

ILE condition handlers get control for all *ESCAPE, *STATUS, *NOTIFY, and function check messages. Not all result code actions are valid for all types of messages.

If the message is handled by the ILE condition handler, the result-code action is not performed.

If a result code is returned that is not valid, the following message occurs:

CEE0265	The result code received from a condition handler is not valid
Severity:30	

Valid result codes are:

Resume

This result code can be used for all exception types.

10 Resume at the resume cursor, and handle the condition, as follows:

Function Check (severity 4)	The message appears in the job log.
**ESCAPE (severity 2-4)*	The message appears in the job log.
**STATUS (severity 1)*	The message does not appear in the job log.
**NOTIFY*	The default reply is sent and the message appears in the job log.

Percolate

These result codes can be used for all exception types.

20 Percolate to the next condition handler.

21 Percolate to the next call stack entry. This can skip a high-level language condition handler for this call stack entry. Any remaining user handlers in the queue for this call stack entry also can be skipped.

This handle cursor movement is in addition to any other handle cursor movement done explicitly in the handler, for example by the CEEMRCR API.

The handle cursor is not moved past a call stack entry for a control boundary. The default condition handling actions are applied to the message. Default Responses to Unhandled Exceptions summarizes the default condition handling actions.

Promote

Only *ESCAPE and *STATUS messages may be promoted.

30 Promote to the next condition handler.

31 Promote to the next call stack entry. This may skip a high-level language condition handler for this call stack entry. Any remaining user handlers in the queue for this call stack entry also can be skipped.

This handle cursor movement is in addition to any other handle cursor movement done explicitly in the handler, for example by the CEEMRCR API.

The handle cursor is not moved past a call stack entry for a control boundary. The default condition handling actions are applied to the new message. Default Responses to Unhandled Exceptions summarizes the default condition handling actions.

32 Promote and restart condition handling with the first condition handler for the call stack entry at which the handle cursor currently points.

Note: It is not valid to promote a condition without returning a new condition token. If the original condition is returned in *new_condition*, the following message occurs:

CEE0262	The condition being promoted is not valid
Severity:30	

new_condition (output)

The condition token representing the promoted condition. This field is used only for *result_code* values of 30, 31, and 32 denoting *promote* or *fix-up and resume*.

API Introduced: V2R3

Retrieve ILE Version and Platform ID (CEEGPID) API

Required Parameter Group:

1	CEE_Version	Output	INT4
2	Plat_ID	Output	INT4

Omissible Parameter:

3	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe:

The Retrieve ILE Version and Platform ID (CEEGPID) API retrieves the ILE version ID and the platform (that is, operating system) ID. The IDs are those currently in use for processing the active condition.

Required Parameter Group

CEE_Version (output)

A 32-bit numeric representation of the version of ILE that created this condition. For example, if 230 is returned as the version, it represents Version 2 Release 3 Modification 0.

Plat_ID (output)

A 32-bit numeric representation of the operating system on which this condition was created. The possible values are:

2	OS/2
3	MVS/VM/370
4	OS/400

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

API Introduced: V2R3

Return the Relative Invocation Number (CEE4RIN) API

Required Parameter Group:

1	rel_inv	Output	INT4
2	inv	Input	INVPTR

Omissible Parameter:

3	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Return the Relative Invocation Number (CEE4RIN) API retrieves the relative invocation number of an invocation pointer, and returns it in *rel_inv*.

Required Parameter Group

rel_inv (output)

The relative invocation number. This is a positive number indicating the number of invocations back from the caller that *inv* occurs.

inv (input)

An invocation pointer.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE3104	The pointer type in &1 is not valid

Severity: 30	
CEE3105	The call stack entry given to &1 no longer exists
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

The invocation specified by *inv* must appear in the call stack for the job issuing the CEE4RIN API.

API Introduced: V2R3

Signal a Condition (CEESGL) API

Required Parameter:			
1	cond_rep	I/O	FEEDBACK
Omissible Parameter Group:			
2	q_data_token	Input	INT4
3	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Signal a Condition (CEESGL) API signals or resignals a condition to the ILE condition manager.

Required Parameter

cond_rep (I/O)

A condition token defining the condition to be raised.

The CEESGL API always uses the facility_ID to retrieve a message, whether or not instance specific information (ISI) is provided. If ISI is provided, the message data from the message is used as insert data for the condition that is to be raised.

Omissible Parameter Group

q_data_token (input)

A 32-bit data object to be placed in the ISI field for use in accessing the qualifying data associated with the given instance of the condition. This parameter is provided for compatibility purposes only. It is ignored on the system.

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0201	The condition sent by &1 was not handled

Severity: 00

CEE0258	The condition token passed to &1 is not valid

Severity: 30

CEE9902	Unexpected user error occurred in &1

Severity: 30

Usage Notes

- If the condition is unhandled and a feedback code is provided, a message number is returned in the fc parameter. The message number returned is dependent on the severity of the condition when it reaches the control boundary.

  If the severity of the condition is 0-1, message number CEE0000 is returned. The message is classified as a *STATUS message.

  If the severity of the condition is 2-4, message number CEE0201 is returned. The message is classified as an *ESCAPE message.

  See Mapping OS/400 *ESCAPE Message Severities to ILE Condition Severities and Mapping OS/400 *STATUS and *NOTIFY Message Severities to ILE Condition Severities for the relationship of condition severities and message severities.

API Introduced: V2R3

Unregister a User-Written Condition Handler (CEEHDLU) API

Required Parameter:			
1	procedure	Input	HDLR_ENTRY
Omissible Parameter:			
2	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe:			

The Unregister a User Condition Handler (CEEHDLU) API unregisters a user-written condition handler for the current call stack entry.

Required Parameter

procedure (input)

An entry variable or constant for a user-written condition handler that was previously registered for the current call stack entry using the Register a User-Written Condition Handler (CEEHDLR) API. For more information on registering user-written condition handlers, see Register a User-Written Condition Handler (CEEHDLR) API.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0252	&1 is unable to find the procedure
Severity: 10	

CEE0253	Additional registrations of the procedure remain in the queue
Severity: 10	
CEE0257	The procedure provided for &1 is not valid
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- Any registered user-written condition handlers not unregistered by CEEHDLU, are unregistered automatically by the system upon removal of the associated call stack entry from the call stack.
- If the specified procedure is registered more than once, the most recent registration is removed. Earlier registrations remain in the queue for the call stack entry.
- The message CEE0257 occurs if the pointer contained in *procedure* is not a procedure pointer.

API Introduced: V2R3

Top | ILE CEE APIs | APIs by category

Date and Time APIs

For information on using the date and time APIs, see Date and Time Notation and Limits.

The Convert Date and Time Format (QWCCVTDT) API converts date and time values from one format to another format. For more information about this API, see Miscellaneous APIs .

The date and time APIs are:

- Calculate Day of Week from Lilian Date (CEEDYWK) returns the day of the week as a number between 1 and 7.
- Convert Date to Lilian Format (CEEDAYS) converts a string representing a date into a number representing the number of days since 14 October 1582.
- Convert Integers to Seconds (CEEISEC) converts separate binary integers representing year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 14 October 1582.
- Convert Lilian Date to Character Format (CEEDATE) formats a number representing a Lilian date.
- Convert Seconds to Character Timestamp (CEEDATM) formats a number representing the number of seconds since 00:00:00 14 October 1582.
- Convert Seconds to Integers (CEESECI) converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond.
- Convert Timestamp to Number of Seconds (CEESECS) converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582.
- Get Current Greenwich Mean Time (CEEGMT) is an alias of CEEUTC.
- Get Current Local Time (CEELOCT) returns the current local time in three formats: Lilian date (the number of days since 14 October 1582), Lilian timestamp (the number of seconds since 00:00:00 14 October 1582), and Gregorian character string (in the form YYYYMMDDHHMISS999').
- Get Offset from Universal Time Coordinated to Local Time (CEEUTCO) provides three values representing the current offset from Universal Time Coordinated (UTC) to local system time.
- Get Universal Time Coordinated (CEEUTC) returns the current Universal Time Coordinated as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582.
- Query Century (CEEQCEN) queries the century within which 2-digit year values are assumed to lie.
- Return Default Date and Time Strings for Country or Region (CEEFMDT) returns the default date and time picture strings for the country or region specified in the country/region_code parameter.
- Return Default Date String for Country or Region (CEEFMDA) returns the default date picture string for the country or region specified in the country/region_code parameter.
- Return Default Time String for Country or Region (CEEFMTM) returns the default time picture string for the country or region specified in the country/region_code parameter.
- Set Century (CEESCEN) sets the century within which 2-digit year values are assumed to lie.

Date and Time Notation and Limits

Calendars used by the date and time ILE CEE APIs have specific notation and limits associated with them. Following is a description of the notation and the limits.

Notation

Calendars based on eras use unique strings to identify these eras. An era is defined by a major event in time from which date calculations take place. These identification strings are called picture strings. The following picture strings are supported:

Japanese era

Identified by one 6-character string

```
<JJJJ>
```

Republic of China (ROC) era

Identified by one 6-character string

```
<CCCC>
```

or by one 10-character string

```
<CCCCCCCC>
```

The era picture string begins with the less than character (<) and ends with the greater than character (>). The characters within these characters are either uppercase Js or uppercase Cs.

Note: An era is defined by a major event in time from which date calculations take place.

Limits

Date variables associated with certain calendars have certain limitations. These limits are:

starting Lilian date

The beginning of the Lilian date range is set to 15 October 1582. This is defined as Lilian day 1. Lilian day zero is defined as 14 October 1582, and is used for calculation purposes in several of the APIs. Lilian second 1 is defined as 00:00:01 on 14 October 1582, and is used for calculation purposes in several of the APIs. Only Lilian dates greater than or equal to 1 are valid as input or produced as output in the date and time APIs. This is the date of adoption of the Gregorian calendar. Lilian dates preceding this date are undefined.

end Lilian date

The end of the Lilian date range is set to 31 December 9999. Lilian dates following this date are undefined.

limit of current era

The maximum future date that can be expressed in the era system must be within the first 999 years of the current era. Future dates beyond year 999 of the current era are not defined.

number of eras

The eras supported by ILE are shown within this section. No other past eras are supported by ILE at the present time. Future eras will be added as required.

Top | ILE CEE APIs | APIs by category

Calculate Day of Week from Lilian Date (CEEDYWK) API

Required Parameter Group:

1	input_Lilian_date	Input	INT4
2	output_day_no	Output	INT4

Omissible Parameter:

3	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Calculate Day of Week from Lilian Date (CEEDYWK) API returns the day of the week as a number between 1 and 7.

Required Parameter Group

input_Lilian_date (input)

A 32-bit binary integer representing the Lilian date, which is the number of days since 14 October 1582. For example, 16 May 1988 is day number 148 138. The valid range is 1 to 3 074 324 (31 December 9999).

output_day_no (output)

A 32-bit binary integer representing the day of week of the *input_Lilian_date*. For the day of week, 1 indicates Sunday, 2 indicates Monday, ..., 7 indicates Saturday. If *input_Lilian_date* is not valid, *output_day_no* is set to 0 and CEEDYWK ends with a nonzero feedback code.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2512	The value for the given Lilian date is not valid
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The number returned by CEEDYWK is useful for end-of-week calculations.

API Introduced: V2R3

Convert Date to Lilian Format (CEEDAYS) API

Required Parameter Group:

1	input_char_date	Input	VSTRING
2	picture_string	Input	VSTRING
3	output_Lilian_date	Output	INT4

Omissible Parameter:

4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Convert Date to Lilian Format (CEEDAYS) API converts a string representing a date into a number representing the number of days since 14 October 1582. This API makes it easier to do calculations such as the number of days between two dates.

Required Parameter Group

input_char_date (input by descriptor)

A character string representing a date or timestamp in the format shown by *picture_string*. Field width is 5 to 255 characters. *Input-char-date* can contain leading or trailing blanks. Parsing for a date begins with the first non-blank character unless the picture string contains leading blanks, in which case CEEDAYS skips exactly that many positions before parsing begins. After a valid date is parsed, remaining characters are ignored. Valid dates are in the range 15 October 1582 to 31 December 9999.

picture_string (input by descriptor)

A character string indicating the format of the date value in *input_char_date*, for example MM/DD/YY. Each character in *picture_string* represents a character in *input_char_date*. If delimiters such as the slash (/) appear in the picture string, then leading zeros can be omitted. For example:

```
CALL CEEDAYS('6/2/88'  , 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('06/02/88', 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('060288'  , 'MMDDYY'  , lildate, fc);
CALL CEEDAYS('88154'   , 'YYDDD'   , lildate, fc);
```

would all assign the same value to variable lildate. If any time characters are included, for example HH:MI:SS YY/MM/DD, they count as place holders but are otherwise ignored.

Picture Characters Used in Picture Strings contains a list of valid picture characters, and Examples of Picture Strings Recognized by ILE Date and Time APIs has examples of valid picture strings.

If *picture_string* is null or blank, CEEDAYS obtains *picture_string* based on the current job value for the country or region ID (CNTRYID). For example, if the current value for CNTRYID is US (United States), the date format is MM/DD/YY. If the current job value for CNTRYID is FR (France), the date format is DD.MM.YYYY.

This default mechanism makes it easy for translators to specify the preferred date format, and also easy for application programs and library procedures to automatically use this format.

output_Lilian_date (output)

A 32-bit binary integer representing the Lilian date, which is the number of days since 14 October 1582. For example, 16 May 1988 is day number 148 138. If *input_char_date* does not contain a valid date, *output_Lilian_date* is set to 0 and CEEDAYS ends with a nonzero feedback code.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0501	The operational descriptor data type is not valid
Severity: 30	
CEE0502	Missing operational descriptor
Severity: 30	
CEE2507	Insufficient data provided
Severity: 30	
CEE2508	The value for day is not valid
Severity: 30	
CEE2509	The value for era is not valid
Severity: 30	
CEE2513	The value for Lilian date is not valid
Severity: 30	
CEE2517	The value for month is not valid
Severity: 30	
CEE2518	The picture string specification is not valid
Severity: 30	
CEE2521	The value for year is not valid
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The inverse of CEEDAYS is CEEDATE. The CEEDATE API converts a date in the Lilian format to character format.
- Date calculations can be performed easily on the *output_Lilian_date*, because it is an integer. Leap year and end-of-year anomalies are avoided.
- See Set Century (CEESCEN) API and Query Century (CEEQCEN) API for information on 2-digit years.
- If picture_string includes a Japanese era symbol <JJJJ>, the YY position in *input_char_date* is assumed to contain the year within Japanese era. Examples of Picture Strings Recognized by ILE Date and Time APIs has an example. Japanese Eras Used by ILE Date and Time APIs When <JJJJ> Specified contains a list of the Japanese eras recognized by CEEDAYS.
- If *picture_string* includes a Republic of China (ROC) era symbol <CCCC> or <CCCCCCCC>, the YY position in *input_char_date* is assumed to contain the year within ROC era. Examples of Picture Strings Recognized by ILE Date and Time APIs has an example. Republic of China Eras Used by ILE Date and Time APIs When <CCCC> or <CCCCCCCC> Specified contains a list of the ROC eras recognized by CEEDAYS. The coding of YYY or ZYY without including one of the era symbols is a picture string specification error.

Picture Characters Used in Picture Strings

Picture Characters	Explanation	Valid Values	Notes
Y YY YYY ZYY YYYY	1-digit year 2-digit year 3-digit year 3-digit year within era 4-digit year	0-9 00-99 000-999 1-999 1582-9999	Y valid for output only. YY implies the years xx00-xx99. (The years are dependent on the century start value and the system date.) YYY or ZYY used with <JJJJ>, <CCCC>, and <CCCCCCCC>.
<JJJJ>	Japanese era name in DBCS characters	*Heisei* (X'0E458D45BA0F') *Showa* (X'0E45B3457A0F') *Taisho* (X'0E455B45770F') *Meiji* (X'0E45A645840F')	Affects YY field: if <JJJJ> specified, YY means the year within Japanese era. For example, 1988 = Showa 63. See example in Examples of Picture Strings Recognized by ILE Date and Time APIs.

`<CCCC>`
`<CCCCCCCC>`
	Republic of China (ROC) era name in DBCS characters	*`MinKow`*
		`(X'0E4D8256CE0F')`
		`ChuHwaMinKow`
		`(X'0E4C845ADD4D8256CE0F')`
			Affects YY field: if <CCCC> specified, YY means the year within ROC era. For example, 1988 = Minkow 77. See example in Examples of Picture Strings Recognized by ILE Date and Time APIs.

`MM`
`ZM`
	`2-digit month`
	`1- or 2-digit month`
		`01-12`
		`1-12`
			For output, leading zero suppressed. For input, ZM treated as MM.

`RRRR` `RRRZ`	Roman numeral month	I-XII (Left-justified)	For input, source string is folded to uppercase. For output, uppercase only. I=Jan, II=Feb, ..., XII=Dec.

`MMM`
`Mmm`
`MMMMMMMMMM`
`Mmmmmmmmmm`
`MMMMMMMMMZ`
`Mmmmmmmmmz`
	`3-char month, uppercase`
	`3-char month, mixed case`
	`20-char month, uppercase`
	`20-char month, mixed case`
	`trailing blanks suppressed`
	`trailing blanks suppressed`
		`JAN-DEC`
		`Jan-Dec`
		`JANUARY   -DECEMBER`
		`January   -December`
		`JANUARY-DECEMBER`
		`January-December`
			For input, source string always folded to uppercase. For output, M generates uppercase and m generates lowercase. Output is padded with blanks ( ) (unless Z specified) or truncated to match the number of Ms.

`DD`
`ZD`
`DDD`
	`2-digit day of month`
	`1- or 2-digit day of month`
	`day of year (Julian day)`
		`01-31`
		`1-31`
		`001-366`
			For output, leading zero suppressed. For input, ZD and DD are equivalent. That is, each accepts the format of the other.

HH ZH	2-digit hour 1- or 2-digit hour	00-23 0-23	For output, leading zero suppressed. For input, ZH and HH are equivalent. That is, each accepts the format of the other. If AP is specified, valid values are 01-12.
MI	minute	00-59	
SS	second	00-59	
9 99 999	tenths of a second hundredths of a second thousandths of a second	0-9 00-99 000-999	No rounding.
AP ap A.P. a.p.	AM/PM indicator	AM or PM am or pm A.M. or P.M.	AP affects HH/ZH field. For input, source string always folded to uppercase. For output, AP generates uppercase and ap generates lowercase.
W WWW Www WWWWWWWWWW Wwwwwwwwww WWWWWWWWWZ Wwwwwwwwwz	1-char day-of-week 3-char day, uppercase 3-char day, mixed case 10-char day, uppercase 10-char day, mixed case trailing blanks suppressed trailing blanks suppressed	S, M, T, W, T, F, S SUN-SAT Sun-Sat SUNDAY -SATURDAY Sunday -Saturday SUNDAY-SATURDAY Sunday-Saturday	For input, Ws are ignored. For output, W generates uppercase and w generates lowercase. Output padded with blanks (unless Z specified) or truncated to match the number of Ws.
All others, not including numbers 0 through 9 and any characters used in the month or day-of-week names.	delimiters	X'01'-X'FF'(X'00' reserved for ILE use.)	For input, treated as delimiters between the month, day, year, hour, minute, second, and fraction of a second. For output, copied exactly as is to the target string.

Examples of Picture Strings Recognized by ILE Date and Time APIs

Picture String	Example	Notes
YYMMDD YYYYMMDD YYYY-MM-DD <JJJJ> YY.MM.DD <CCCC> YY.MM.DD	880516 19880516 1988-05-16 *Showa* 63.05.16 *MinKow* 77.05.16	1988-5-16 would also be valid input. *Showa* is a Japanese era name. *Showa* 63 = 1988. *MinKow* is an ROC era name. *MinKow* 77 = 1988.
MMDDYY MM/DD/YY ZM/ZD/YY MM/DD/YY MM/DD/Y	050688 05/06/88 5/6/88 05/06/1988 05/06/8	1-digit year format (Y) valid for output only
DD.MM.YY DD-RRRR-YY DD MMM YY DD Mmmmmmmmmm YY ZD Mmmmmmmmmz YY Mmmmmmmmmz ZD, YYYY ZDMMMMMMMMZYY	09.06.88 09- VI-88 09 JUN 88 09 June 88 9 June 88 June 9, 1988 9JUNE88	Z suppresses zeros and blanks
YY.DDD YYDDD YYYY/DDD	88.137 88137 1988/137	Julian date
YYMMDDHHMISS YYYYMMDDHHMISS YYYY-MM-DD HH:MI:SS.999 WWW, ZM/ZD/YY HH:MI AP Wwwwwwwwwz, DD Mmm YYYY, ZH:MI AP	880516204229 19880516204229 1988-05-16 20:42:29.046 MON, 5/16/88 08:42 PM Monday, 16 May 1988, 8:42 PM	Timestamp -- valid only for CEESECS and CEEDATM. If used with CEEDATE, time positions are left blank. If used with CEEDAYS, HH, MI, SS, and 999 fields are ignored.

Japanese Eras Used by ILE Date and Time APIs When <JJJJ> Specified

First date of Japanese Era	Era Name	Era Name in IBM Japanese DBCS Code	Valid Year (YY, ZYY) Values
1868-09-08	Meiji	X'0E45A645840F'	01-45
1912-07-30	Taisho	X'0E455B45770F'	01-15

1926-12-25	Showa	X'0E45B3457A0F'	01-64
1989-01-08	Heisei	X'0E458D45BA0F'	01-999 (01 = 1989)

Republic of China Eras Used by ILE Date and Time APIs When <CCCC> or <CCCCCCCC> Specified

First date of ROC Era	Era Name	Era Name in Traditional Chinese DBCS Code	Valid Year (YY, ZYY) Values
1912-01-01	*MinKow*	X'0E4D8256CE0F'	01-999 (77 = 1988)
	ChuHwaMinKow	X'0E4C845ADD4D8256CE0F'	

Example

- Convert a date to the Lilian format to calculate the date 60 days hence:

```
CALL CEEDAYS ('880516','YYMMDD', ndays, fc);
ndays = ndays + 60;
CALL CEEDATE (ndays, 'YYMMDD', newdate, fc);
```

API Introduced: V2R3

Convert Integers to Seconds (CEEISEC) API

Required Parameter Group:

1	input_year	Input	INT4
2	input_month	Input	INT4
3	input_day	Input	INT4
4	input_hours	Input	INT4
5	input_minutes	Input	INT4
6	input_seconds	Input	INT4
7	input_milliseconds	Input	INT4
8	output_seconds	Output	FLOAT8

Omissible Parameter:

9	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe:

The Convert Integers to Seconds (CEEISEC) API converts separate binary integers representing year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 14 October 1582. Use CEEISEC instead of CEESECS when the input is in numeric format rather than character format.

Required Parameter Group

input_year (input)

A 32-bit binary integer representing year. The range is 1582 through 9999.

input_month (input)

A 32-bit binary integer representing month. The range is 1 through 12.

input_day (input)

A 32-bit binary integer representing day. The range is 1 through 31.

input_hours (input)

A 32-bit binary integer representing hours. The range is 0 through 23.

input_minutes (input)

A 32-bit binary integer representing minutes. The range is 0 through 59.

input_seconds (input)

A 32-bit binary integer representing seconds. The range is 0 through 59.

input_milliseconds (input)

A 32-bit binary integer representing milliseconds. The range is 0 through 999.

output_seconds (output)

A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The range is 86,400 to 265 621 679 999.999 (23:59:59.999 31 December 9999).

If any input values are not valid, *output_seconds* is set to zero.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2510	The value for hour is not valid
Severity: 30	
CEE2511	The value for day is not valid
Severity: 30	
CEE2513	The value for Lilian date is not valid
Severity: 30	
CEE2514	The value for Lilian year is not valid
Severity: 30	
CEE2515	The value for millisecond is not valid
Severity: 30	
CEE2516	The minute value is not valid
Severity: 30	
CEE2517	The value for month is not valid
Severity: 30	
CEE2519	The value for second is not valid
Severity: 30	

CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The inverse of CEEISEC is CEESECI. The CEESECI API converts number of seconds to integer year, month, day, and so forth.
- To convert *output_seconds* to a Lilian day number, divide *output_seconds* by 86 400 (number of seconds in a day).
- CEEISEC can be used to do date arithmetic that cannot otherwise be done with Lilian dates or number of seconds. For example, to add exactly 6 months to a date rather than add 180 days, use CEEISEC.

API Introduced: V2R3

Top | ILE CEE APIs | APIs by category

Convert Lilian Date to Character Format (CEEDATE) API

Required Parameter Group:

1	input_Lilian_date	Input	INT4
2	picture_string	Input	VSTRING
3	output_char_date	Output	VSTRING

Omissible Parameter:

4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Convert Lilian Date to Character Format (CEEDATE) API formats a number representing a Lilian date. The output is a character string such as 1988/07/26.

Required Parameter Group

input_Lilian_date (input)

A 32-bit binary integer representing the Lilian date, which is the number of days since 14 October 1582. For example, 16 May 1988 is day number 148 138. Valid range is 1 to 3 074 324 (31 December 9999).

picture_string (input by descriptor)

A character string representing the desired format of *output_char_date*, for example MM/DD/YY. Each character in *picture_string* represents a character in *output_char_date*. If delimiters such as the slash (/) appear in the picture string, they are copied as is to *output_char_date*.

Picture Characters Used in Picture Strings has a list of valid picture characters, and Examples of Picture Strings Recognized by ILE Date and Time APIs contains examples of valid picture strings.

If *picture_string* is null or blank, CEEDATE obtains *picture_string* based on the current job value for the country or region ID (CNTRYID). For example, if the current job value for CNTRYID is US (United States), the date format is MM/DD/YYYY. If the current job value for CNTRYID is FR (France), the date format is DD.MM.YYYY.

This default mechanism makes it easy for translators to specify the preferred date format, and also easy for application programs and library procedures to automatically use this format.

output_char_date (output by descriptor)

A character string that is the result of converting *input_Lilian_date* to the format specified by *picture_string*. If necessary, output will be truncated to the length of *output_char_date*. Sample Output of the CEEDATE API contains sample output dates. If *input_Lilian_date* is not valid, *output_char_date* is set to all blanks and CEEDATE ends with a nonzero *feedback-code*.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0501	The operational descriptor data type is not valid
Severity: 30	
CEE0502	Missing operational descriptor
Severity: 30	
CEE2512	The value for the given Lilian date is not valid
Severity: 30	
CEE2518	The picture string specification is not valid
Severity: 30	
CEE2522	Lilian date outside of era
Severity: 30	
CEE2526	Date truncated
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The inverse of CEEDATE is CEEDAYS. The CEEDAYS API converts character dates to the Lilian format.
- If *picture_string* includes a Japanese era symbol <JJJJ>, the YY position in *output_char_date* is replaced by "year within Japanese era". Examples of Picture Strings Recognized by ILE Date and Time APIs has an example. Japanese Eras Used by ILE Date and Time APIs When <JJJJ> Specified contains a list of Japanese eras supported by CEEDATE.
- If *picture_string* includes a Republic of China (ROC) Era symbol <CCCC> or <CCCCCCCC>, the YY position in *output_char_date* is replaced by the year within ROC era. Examples of Picture Strings Recognized by ILE Date and Time APIs has an example. Republic of China Eras Used by ILE Date and Time APIs When <CCCC> or <CCCCCCCC> Specified contains a list of ROC eras supported by CEEDATE.

Sample Output of the CEEDATE API

input_Lilian_date	picture_string	output_char_date

148138	`YY`
	`YYMM`
	`YY-MM`
	`YYMMDD`
	`YYYYMMDD`
	`YYYY-MM-DD`
	`YYYY-ZM-ZD`
	`<JJJJ> YY.MM.DD`
	`<CCCC> YY.MM.DD`
		`88`
		`8805`
		`88-05`
		`880516`
		`19880516`
		`1988-05-16`
		`1988-5-16`
		Showa `63.05.16 (in a DBCS string)`
		MinKow `77.05.16 (in a DBCS string)`

148139	`MM`
	`MMDD`
	`MM/DD`
	`MMDDYY`
	`MM/DD/YYYY`
	`ZM/DD/YYYY`
		`05`
		`0517`
		`05/17`
		`051788`
		`05/17/1988`
		`5/17/1988`

148140	`DD`
	`DDMM`
	`DDMMYY`
	`DD.MM.YY`
	`DD.MM.YYYY`
	`DD Mmm YYYY`
		`18`
		`1805`
		`180588`
		`18.05.88`
		`18.05.1988`
		`18 May 1988`

148141	`DDD`
	`YYDDD`
	`YY.DDD`
	`YYYY.DDD`
		`140`
		`88140`
		`88.140`
		`1988.140`

148142	`YY/MM/DD HH:MI:SS.99`
	`YYYY/ZM/ZD ZH:MI AP`
		`88/05/20 00:00:00.00`
		`88/5/20 0:00 AM`

148143	`WWW., MMM DD, YYYY`
	`Www., Mmm DD, YYYY`
	`Wwwwwwwwwww, Mmmmmmmmmmm DD, YYYY`
	`Wwwwwwwwwwz, Mmmmmmmmmmz DD, YYYY`
		`SAT., MAY 21, 1988`
		`Sat., May 21, 1988`
		`Saturday   , May         21,   1988`
		`Saturday, May 21, 1988`

Example

- Convert a date from the Lilian format to IBM USA standard format MM/DD/YYYY:

  ```
  CALL CEEDATE (lildate, 'MM/DD/YYYY', usadate, fc);
  ```

API Introduced: V2R3

Convert Seconds to Character Timestamp (CEEDATM) API

Required Parameter Group:

1	input_seconds	Input	FLOAT8
2	picture_string	Input	VSTRING
3	output_timestamp	Output	VSTRING

Omissible Parameter:

4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Convert Seconds to Character Timestamp (CEEDATM) API formats a number representing the number of seconds since 00:00:00 14 October 1582. The output is a character string such as 1988/07/26 20:37:00.

Required Parameter Group

input_seconds (input)

A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range is 86 400 to 265 621 679 999.999 (23:59:59.999 31 December 9999).

picture_string (input by descriptor)

A character string representing the desired format of *output_timestamp*, for example MM/DD/YY HH:MM AP. Each character in *picture_string* represents a character in *output_timestamp*. If delimiters such as the slash (/) appear in the picture string, they are copied as is to *output_char_date*.

Picture Characters Used in Picture Strings contains a list of valid picture characters, and Examples of Picture Strings Recognized by ILE Date and Time APIs has examples of valid picture strings.

If *picture_string* is null or blank, CEEDATM obtains *picture_string* based on the current job value for the country or region ID (CNTRYID). For example, if the current job value for CNTRYID is US (United States), the date-time format is MM/DD/YYYY HH:MI AP. If the current job value for CNTRYID is FR (France), the date-time format is YYYY-MM-DD HH.MI.

This default mechanism makes it easy for translators to specify the preferred timestamp format, and also easy for application programs and library procedures to automatically use this format.

output_timestamp (output by descriptor)

A character string that is the result of converting *input_seconds* to the format specified by *picture_string*. If necessary, output is truncated to the length of *output_timestamp*. Sample Output of the CEEDATM API shows sample output. If input_seconds is not valid, *output_timestamp* is set to all blanks and CEEDATM ends with a nonzero *feedback-code*.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0501	The operational descriptor data type is not valid
Severity: 30	
CEE0502	Missing operational descriptor
Severity: 30	
CEE2505	The value for seconds is out of range
Severity: 30	
CEE2506	The value for seconds is outside of era
Severity: 30	
CEE2518	The picture string specification is not valid
Severity: 30	
CEE2527	Timestamp truncated
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The inverse of CEEDATM is CEESECS. The CEESECS converts timestamp values to number-of-seconds values.
- If the input value is a Lilian date instead of seconds, multiply the Lilian date by 86 400 (number of seconds in a day), and pass the new value to CEEDATM.
- If *picture_string* includes the Japanese era symbol <JJJJ>, the YY position in *output_timestamp* is replaced by the year within Japanese era. Examples of Picture Strings Recognized by ILE Date and Time APIs has an example. Japanese Eras Used by ILE Date and Time APIs When <JJJJ> Specified contains a list of Japanese eras supported by CEEDATM.
- If *picture_string* includes the Republic of China (ROC) era symbol <CCCC> or <CCCCCCCC>, the YY position in *output_timestamp* is replaced by the year within ROC era. See Examples of Picture Strings Recognized by ILE Date and Time APIs for an example. See Republic of China Eras Used by ILE Date

and Time APIs When <CCCC> or <CCCCCCCC> Specified for a list of ROC eras supported by CEEDATM.

Sample Output of the CEEDATM API

input_seconds	picture_string	output_timestamp
12 799 191 601.000	`YYMMDD`	`880516`
	`HH:MI:SS`	`19:00:01`
	`YY-MM-DD`	`88-05-16`
	`YYMMDDHHMISS`	`880516190001`
	`YY-MM-DD HH:MI:SS`	`88-05-16 19:00:01`
	`YYYY-MM-DD HH:MI:SS AP`	`1988-05-16 07:00:01 PM`
12 799 191 661.986	`DD Mmm YY`	`16 May 88`
	`DD MMM YY HH:MM`	`16 MAY 88 19:01`
	`WWW, MMM DD, YYYY ZH:MI AP`	`MON, MAY 16, 1988 7:01 PM`
	`Wwwwwwwwwz, ZM/ZD/YY HH:MI:SS.99`	`Monday, 5/16/88 19:01:01.98`
12 799 191 662.009	`YYYY`	`1988`
	`YY`	`88`
	`Y`	`8`
	`MM`	`05`
	`ZM`	`5`
	`RRRR`	`V`
	`MMM`	`MAY`
	`Mmm`	`May`
	`Mmmmmmmmmm`	`May`
	`Mmmmmmmmmz`	`May`
	`DD`	`16`
	`ZD`	`16`
	`DDD`	`137`
	`HH`	`19`
	`ZH`	`19`
	`MI`	`01`
	`SS`	`02`
	`99`	`00`
	`999`	`009`
	`AP`	`PM`
	`WWW`	`MON`
	`Www`	`Mon`
	`Wwwwwwwwww`	`Monday`
	`Wwwwwwwwwz`	`Monday`

Example

- Convert number of seconds to YYYY/MM/DD HH.MM.SS format:

```
CALL CEEDATM (secs, 'YYYY/MM/DD HH.MI.SS', timestmp,
              fc);
```

API Introduced: V2R3

Convert Seconds to Integers (CEESECI) API

Required Parameter Group:

1	input_seconds	Input	FLOAT8
2	output_year	Output	INT4
3	output_month	Output	INT4
4	output_day	Output	INT4
5	output_hours	Output	INT4
6	output_minutes	Output	INT4
7	output_seconds	Output	INT4
8	output_milliseconds	Output	INT4

Omissible Parameter:

9	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Convert Seconds to Integers (CEESECI) API converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond. Use CEESECI instead of CEEDATM when the output is needed in numeric format rather than in character format.

Required Parameter Group

input_seconds (input)

A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86 401 (24*60*60 + 01). The valid range is 86 400 to 265 621 679 999.999 (23:59:59.999 31 December 9999).

If *input_seconds* is not valid, all output parameters are set to zero.

output_year (output)

A 32-bit binary integer representing year. The range is 1582 through 9999.

output_month (output)

A 32-bit binary integer representing month. The range is 1 through 12.

output_day (output)

A 32-bit binary integer representing day. The range is 1 through 31.

output_hours (output)

A 32-bit binary integer representing hours. The range is 0 through 23.

output_minutes (output)

A 32-bit binary integer representing minutes. The range is 0 through 59.

output_seconds (output)

A 32-bit binary integer representing seconds. The range is 0 through 59.

output_milliseconds (output)

A 32-bit binary integer representing milliseconds. The range is 0 through 999.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2505	The value for seconds is out of range
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The inverse of CEESECI is CEEISEC. The CEEISEC API converts integer year, month, day, hour, minute, second, and millisecond to number of seconds.
- If the input value is a Lilian date instead of seconds, multiply the Lilian date by 86 400 (number of seconds in a day), and pass the new value to CEESECI.
- CEESECI can be used to do date arithmetic that cannot otherwise be done with Lilian dates or number of seconds.

API Introduced: V2R3

Convert Timestamp to Number of Seconds (CEESECS) API

Required Parameter Group:

1	input_timestamp	Input	VSTRING
2	picture_string	Input	VSTRING
3	output_seconds	Output	FLOAT8

Omissible Parameter:

4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Convert Timestamp to Number of Seconds (CEESECS) API converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582. This API makes it easier to do time calculations, such as the elapsed time between two timestamps.

Required Parameter Group

input_timestamp (input by descriptor)

A character string representing a date or timestamp in the format shown by **picture_string**. The field width is 5 through 255 characters. **Input-timestamp** may contain leading or trailing blanks. After a valid date or timestamp is parsed, remaining characters are ignored. Valid dates are in the range 15 October 1582 to 31 December 9999. A full date must be specified. Valid times are in the range 00:00:00.000 to 23:59:59.999.

If any part or all of the time value is omitted, zeros are substituted for the remaining values. For example,

```
1988-05-17-19:02 is equivalent to 1988-05-17-19:02:00
1988-05-17       is equivalent to 1988-05-17-00:00:00
```

picture_string (input by descriptor)

A character string indicating the format of the date or timestamp value in **input_timestamp**, for example

```
YY/MM/DD
HH.MI.SS.
```

Each character in **picture_string** represents a character in **input_timestamp**. If delimiters such as the slash (/) appear in the picture string, leading zeros may be omitted. For example, these calls assign the same value to the variable *secs*.

```
CALL CEESECS('88/06/03 15.35.03',
             'YY/MM/DD HH.MI.SS', secs, fc);

CALL CEESECS('88/6/3 15.35.03'  ,
             'YY/MM/DD HH.MI.SS', secs, fc);

CALL CEESECS('88/6/3 3.35.03 PM',
             'YY/MM/DD HH.MI.SS AP', secs, fc);

CALL CEESECS('88.155 3.35.03 pm',
             'YY.DDD HH.MI.SS AP', secs, fc);
```

See Picture Characters Used in Picture Strings for a list of valid picture characters, and Examples of Picture Strings Recognized by ILE Date and Time APIs for examples of valid picture strings.

If **picture_string** is null or blank, CEESECS obtains **picture_string** based on the current job value for the country or region ID (CNTRYID). For example, if the current job value for CNTRYID is US (United States), the date format is MM/DD/YYYY. If the current job value for CNTRYID is FR (France), the date format is DD.MM.YYYY.

This default mechanism makes it easy not only for translators to specify the preferred date format, but also for application programs and library routines to automatically use this format.

output_seconds (output)

A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second 86 401 (24*60*60 + 01). 19:00:01.12 on 16 May 1988 is second 12 799 191 601.12. The largest value that can be represented is 23:59:59.999 on 31 December 9999, which is second 265 621 679 999.999.

Note: A 64-bit double floating-point value can accurately represent approximately 16 significant decimal digits without loss of precision. Therefore, accuracy is available to the nearest millisecond (15 decimal digits).

If **input_timestamp** does not contain a valid date or timestamp, **output_seconds** is set to 0 and CEESECS ends with a nonzero feedback code.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0501	The operational descriptor data type is not valid

Severity: 30	
CEE0502	Missing operational descriptor
Severity: 00	
CEE2508	The value for day is not valid
Severity: 30	
CEE2509	The value for era is not valid
Severity: 30	
CEE2510	The value for hour is not valid
Severity: 30	
CEE2513	The value for Lilian date is not valid
Severity: 30	
CEE2515	The value for millisecond is not valid
Severity: 30	
CEE2516	The minute value is not valid
Severity: 30	
CEE2517	The value for month is not valid
Severity: 30	
CEE2518	The picture string specification is not valid
Severity: 30	
CEE2519	The value for second is not valid
Severity: 30	
CEE2521	The value for year is not valid
Severity: 30	
CEE2525	Timestamp picture mismatch
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The inverse of CEESECS is CEEDATM. The CEEDATM API converts **output_seconds** to character format.
- Elapsed time calculations can be performed easily on the **output_seconds**, because it represents elapsed time. Leap year and end-of-year anomalies are avoided.
- See Set Century (CEESCEN) API and Query Century (CEEQCEN) API for information on 2-digit years.
- If **picture_string** includes a Japanese era symbol <*JJJJ*>, the *YY* position in **input_timestamp** is assumed to contain the year within Japanese era. See Examples of Picture Strings Recognized by

ILE Date and Time APIs for an example. See Japanese Eras Used by ILE Date and Time APIs When <JJJJ> Specified for a list of the Japanese eras recognized by CEESECS.

- If **picture_string** includes an ROC era symbol <*CCCC*> or <*CCCCCCCC*> , the *YY* position in **input_timestamp** is assumed to contain the year within ROC era. See Examples of Picture Strings Recognized by ILE Date and Time APIs for an example. See Republic of China Eras Used by ILE Date and Time APIs When <CCCC> or <CCCCCCCC> Specified for a list of the ROC eras recognized by CEESECS.

Example

- Calculate the difference between two timestamps, in hours:

```
CALL CEESECS ('19880516190001','YYYYMMDDHHMISS',
              secs1, fc);
CALL CEESECS ('1988-05-17-03.00.01',
              'YYYY-MM-DD-HH.MI.SS', secs2, fc);
              diff = (secs2 - secs1) / 3600;
              /* Assume floating-point division */
```

- Convert a timestamp to number of seconds, to calculate the date and time 36 hours ago:

```
now = '1988/07/26 19:55:00';
CALL CEESECS (now, 'YYYY/MM/DD HH:MI:SS', secs, fc);
secs = secs - 36*60*60;
CALL CEEDATM (secs, 'YYYY/MM/DD HH:MI:SS', before,
              fc);
```

API Introduced: V2R3

Get Current Greenwich Mean Time (CEEGMT) API

The Get Current Greenwich Mean Time (CEEGMT) API is an alias of CEEUTC. See Get Universal Time Coordinated (CEEUTC) API for details.

API Introduced: V2R3

Get Current Local Time (CEELOCT) API

Required Parameter Group:

1	output_Lilian	Output	INT4
2	output_seconds	Output	FLOAT8
3	output_Gregorian	Output	CHAR23

Omissible Parameter:

4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe:

The Get Current Local Time (CEELOCT) API returns the current local time in three formats: Lilian date (the number of days since 14 October 1582), Lilian timestamp (the number of seconds since 00:00:00 14 October 1582), and Gregorian character string (in the form YYYYMMDDHHMISS999'). These values are compatible with the other ILE date and time APIs and with existing language intrinsic functions. CEELOCT performs the same service, faster, than calling CEEUTC, CEEUTCO, and CEEDATM in succession.

Required Parameter Group

output_Lilian (output)

A 32-bit binary integer representing the current *local* date in the Lilian format. That is, day 1 is 15 October 1582, day 148 887 is 4 June 1990. If local time is not available from the system, *output_Lilian* is set to 0 and CEELOCT ends with a nonzero feedback code.

output_seconds (output)

A 64-bit double floating point number representing the current *local* date and time as the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86 401 (24*60*60 + 01). 19:00:01.078 on 4 June 1990 is second number 12 863 905 201.078. If local time is not available from the system, *output_seconds* is set to 0 and CEELOCT ends with a nonzero feedback code.

output_Gregorian (output)

A 17-byte character string in the form YYYYMMDDHHMISS999 representing local year, month, day, hour, minute, second, and millisecond.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2502	Local time not available
Severity: 30	

Usage Notes

- Use CEEUTC to determine Universal Time Coordinated (UTC).
- Use CEEUTCO to obtain the offset from UTC to local time.
- The character value returned by CEELOCT is designed to match that produced by existing language intrinsic functions. The numeric values returned can be used to simplify date calculations.
- If the format of *output_Gregorian* is inappropriate, CEEDATM can be used to convert *output_seconds* to any required format.

Example

- Extract current local date and time in the form YYYYMMDDHHMISS999:

```
CALL CEELOCT (days, secs, localdatetime, fc);
```

API Introduced: V2R3

Get Offset from Universal Time Coordinated to Local Time (CEEUTCO) API

Required Parameter Group:

1	offset_hours	Output	INT4
2	offset_minutes	Output	INT4
3	offset_seconds	Output	FLOAT8

Omissible Parameter:

4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Get Offset from Universal Time Coordinated to Local Time (CEEUTCO) API provides three values representing the current offset from Universal Time Coordinated (UTC) to local system time. *Offset_seconds* can be used with CEEUTC to calculate local date and time. *Offset_hours* and *offset_minutes* express the offset from UTC in terms of hours and minutes.

Required Parameter Group

offset_hours (output)

A 32-bit binary integer representing the offset from UTC to local time, in hours; for Pacific Standard Time *offset_hours* is -8. The range for *offset_hours* is -12 to -13, where +13 is Daylight Savings Time in the +12 time zone. If local time offset is not available, *offset_hours* is set to 0 and CEEUTCO ends with a nonzero feedback code.

offset_minutes (output)

A 32-bit binary integer representing the number of additional minutes that local time is ahead of, or behind, UTC. The range for *offset_minutes* is 0 to 59. If the local time offset is not available, *offset_minutes* is set to 0 and CEEUTCO ends with a nonzero feedback code.

offset_seconds (output)

A 64-bit double floating point (output) number representing the offset from UTC to local time, in seconds. For example, Pacific Standard Time is eight hours behind UTC. If the system is in the Pacific time zone during standard time, CEEUTCO returns -28 800 (-8 * 60 * 60). The range for *offset_seconds* is -43 200 to +46 800. If the local time offset is not available from the system, *offset_seconds* is set to 0 and CEEUTCO ends with a nonzero feedback code.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2503	UTC offset not available
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- The values returned by CEEUTCO and CEEUTC can be used together to calculate the local date and time.
- The CEEDATM API can be used to convert number of seconds to character timestamp.

Example

- Extract current UTC and convert to local date and time in YYYY-MM-DD HH.MM.SS format:

```
CALL CEEUTC (days, secs, fc);
CALL CEEUTCO (hrs, mins, secoffset, fc);
secs = secs + secoffset;
CALL CEEDATM (secs, 'YYYY-MM-DD HH.MI.SS', timestmp,
              fc);
```

API Introduced: V2R3

Get Universal Time Coordinated (CEEUTC) API

Required Parameter Group:

1	output_UTC_Lilian	Output	INT4
2	output_UTC_seconds	Output	FLOAT8

Omissible Parameter:

3	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Get Universal Time Coordinated (CEEUTC) API returns the current Universal Time Coordinated as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. These values are compatible with the other ILE date and time APIs.

Required Parameter Group

output_UTC_Lilian (output)

A 32-bit binary integer representing the current date in Greenwich, England, in the Lilian format. That is, the number of days since 14 October 1582, also known as Universal Time Coordinated (UTC). For example, 16 May 1988 is day number 148 138. If UTC is not available from the system, *output_UTC_Lilian* is set to 0 and CEEUTC ends with a nonzero feedback code.

output_UTC_seconds (output)

A 64-bit double floating-point number representing the current date and time in Greenwich, England as the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86 401 (24*60*60 + 01). 19:00:01.078 on 16 May 1988 is second number 12 799 191 601.078. If UTC is not available from the system, *output_UTC_seconds* is set to 0 and CEEUTC ends with a nonzero feedback code.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2502	Local time not available
Severity: 30	
CEE2531	UTC not available
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- Use CEELOCT to obtain local time.
- Use CEEUTCO to obtain the offset from UTC to local time.
- The values returned by CEEUTC are handy for **wall-clock** calculations.
- CEEDATE will convert *output_UTC_Lilian* to character date, and CEEDATM will convert *output_UTC_seconds* to character timestamp.

Example

- Extract current UTC and convert to local date and time in YYMMDDHHMMSS format:

```
CALL CEEUTC (days, secs, fc);
CALL CEEUTCO (hrs, mins, secoffset, fc);
secs = secs + secoffset;
CALL CEEDATM (secs, 'YYMMDDHHMISS', localdatetime,
              fc);
```

API Introduced: V2R3

Query Century (CEEQCEN) API

Required Parameter:			
1	century_start	Output	INT4
Omissible Parameter:			
2	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Query Century (CEEQCEN) API queries the century within which 2-digit year values are assumed to lie. Use it in conjunction with CEESCEN when it is necessary to save and restore the current setting. If the ILE default is in effect, all 2-digit years are assumed to lie within a 100-year window. This window starts 80 years prior to the system date and CEEQCEN returns a value of 80.

Required Parameter

century_start (output)

An integer between 0 and 100. A value of 80 (the ILE default) means all 2-digit years lie within the 100-year window starting 80 years before the system date. For example, in 1990, all 2-digit years are assumed to represent dates between 1910 and 2009, inclusive.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

API Introduced: V2R3

Top | ILE CEE APIs | APIs by category

Return Default Date and Time Strings for Country or Region (CEEFMDT) API

Omissible Parameter:			
1	country/region_code	Input	CHAR2
Required Parameter:			
2	datetime_str	Output	VSTRING
Omissible Parameter:			
3	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Return Default Date and Time String for Country or Region (CEEFMDT) API returns the default date and time picture strings for the country or region specified in the country/region_code parameter.

Omissible Parameter

country/region_code (input)

The 2-character string that represents the country or region code. See Country/Region Codes for values. If this value is blank, the default country or region code is used.

Required Parameter

datetime_str (output by descriptor)

The default date and time picture string for the country or region code is placed into this character string variable.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE3470	The resulting string is truncated
Severity: 20	
CEE3471	The country/region_code identifier is not valid for CEEFMDT
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Country/Region Code Identifiers

The following table lists the country or region code identifiers.

Country/Region Codes

Country/Region	**ID**
Albania	AL
Algeria	DZ
Argentina	AR
Australia	AU
Austria	AT
Bahrain	BH
Belgium	BE
Bolivia	BO
Brazil	BR
Bulgaria	BG
Canada	CA
Chile	CL

Colombia	CO
Costa Rica	CR
Countries of the former Yugoslavia	YU
Czech Republic	CZ
Denmark	DK
Dominican Republic	DO
Ecuador	EC
Egypt	EG
El Salvador	SV
Finland	FI
France	FR
Germany	DE
Greece	GR
Guatemala	GT
Honduras	HN
Hungary	HU
Iceland	IS
India	IN
Iran	IR
Iraq	IQ
Ireland	IE
Israel	IL
Italy	IT
Japan	JP
Jordan	JO
Korea, Democratic People's Republic	KP
Korea, Republic of	KR
Kuwait	KW
Lebanon	LB
Libya	LY
Mexico	MX
Morocco	MA
Netherlands	NL
New Zealand	NZ

Norway	NO
Oman	OM
Pakistan	PK
Panama	PA
Paraguay	PY
People's Republic of China	CN
Peru	PE
Poland	PL
Portugal	PT
Qatar	QA
Romania	RO
Russia	RU
Saudi Arabia	SA
Slovakia	SK
Slovenia	SI
South Africa	ZA
Spain	ES
Sudan	SD
Sweden	SE
Switzerland	CH
Syria	SY
Taiwan	TW
Thailand	TH
Tunisia	TN
Turkey	TR
United Arab Emirates	AE
United Kingdom	GB
United States	US
Uruguay	UY
Venezuela	VE
Yemen	YE

API Introduced: V2R3

Top | ILE CEE APIs | APIs by category

Return Default Date String for Country or Region (CEEFMDA) API

Omissible Parameter:			
1	country/region_code	Input	CHAR2
Required Parameter:			
2	date_pic_str	Output	CHAR
Omissible Parameter:			
3	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Return Default Date String for Country or Region (CEEFMDA) API returns the default date picture string for the country or region specified in the country/region_code parameter.

Omissible Parameter

country/region_code (input)

The 2-character string that represents the country or region code. See Country/Region Codes for specific codes. If this value is blank, the default country or region code is used.

Required Parameter

date_pic_str (output by descriptor)

The default date picture string for the country or region code is placed into this character string variable.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE3466	The date picture string is truncated
Severity: 20	
CEE3467	The country/region_code identifier is not valid for CEEFMDA
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

API Introduced: V2R3

Return Default Time String for Country or Region (CEEFMTM) API

Omissible Parameter:			
1	country/region_code	Input	CHAR2
Required Parameter:			
2	time_pic_str	Output	VSTRING
Omissible Parameter:			
3	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Return Default Time String for Country or Region (CEEFMTM) API returns the default time picture string for the country or region specified in the country/region_code parameter.

Omissible Parameter

country/region_code (input)

The 2-character string that represents the country or region code. See Country/Region Codes for values. If this value is blank, the default country or region code will be used.

Required Parameter

time_pic_str (output)

The default time picture string for the country or region code is placed into this character string variable.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE3468	The time picture string is truncated
Severity: 20	
CEE3469	The country/region_code identifier is not valid for CEEFMTM
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

API Introduced: V2R3

Set Century (CEESCEN) API

Required Parameter:

1	century_start	Input	INT4

Omissible Parameter:

2	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Set Century (CEESCEN) API sets the century within which 2-digit year values are assumed to lie. Use it in conjunction with CEEDAYS or CEESECS when processing date values that contain 2-digit years (in the YYMMDD format), and when the ILE default century interval does not meet the requirements of a particular application.

Required Parameter

century_start (input)

An integer between 0 and 100. A value of 80 (the ILE default) means all 2-digit years lie within the 100-year window starting 80 years before the system date. For example, in 1990, all 2-digit years are assumed to represent dates between 1910 and 2009, inclusive.

Omissible Parameter

fc (output/optional)

A 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2502	Local time not available
Severity: 30	
CEE2533	The century_start value is not between 0 - 100

Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

API Introduced: V2R3

Math APIs

The ILE math bindable APIs can be called either through the intrinsic functions of an ILE high-level language, or from an ILE HLL as a call to the API.

For additional information on using the math APIs, see:

- Data Types and Limits
- Calling Math Bindable APIs
- Math Bindable APIs are Procedures
- ILE Math Bindable API Descriptions
- Message Descriptions

Messages that can be generated by the math bindable APIs are summarized in Message Descriptions.

The following math bindable APIs are presented in the Math API Descriptions table.

- Absolute Function (CEESxABS)
- Arccosine (CEESxACS)
- Arcsine (CEESxASN)
- Arctangent (CEESxATN)
- Arctangent2 (CEESxAT2)
- Conjugate of Complex (CEESxCJG)
- Cosine (CEESxCOS)
- Cotangent (CEESxCTN)
- Error Function and Its Complement (CEESxERx)
- Exponential Base e (CEESxEXP)
- Exponentiation (CEESxXPx)
- Factorial (CEE4SxFAC)
- Floating Complex Divide (CEESxDVD)
- Floating Complex Multiply (CEESxMLT)
- Gamma Function (CEESxGMA)
- Hyperbolic Arctangent (CEESxATH)
- Hyperbolic Cosine (CEESxCSH)
- Hyperbolic Sine (CEESxSNH)
- Hyperbolic Tangent (CEESxTNH)
- Imaginary Part of Complex (CEESxIMG)
- Log Gamma Function (CEESxLGM)
- Logarithm Base 10 (CEESxLG1)
- Logarithm Base 2 (CEESxLG2)
- Logarithm Base e (CEESxLOG)
- Modular Arithmetic (CEESxMOD)
- Nearest 64-Bit Integer (CEESxNJN)

- Nearest Integer (CEESxNIN)
- Nearest Whole Number (CEESxNWN)
- Positive Difference (CEESxDIM)
- Sine (CEESxSIN)
- Square Root (CEESxSQT)
- Tangent (CEESxTAN)
- Transfer of Sign (CEESxSGN)
- Truncation (CEESxINT)

The following math APIs can be accessed individually:

- Basic Random Number Generation (CEERAN0)
- Convert 64-Bit Integer to Character String (CEE4JNTS)
- Convert Character String to 64-Bit Integer (CEE4JSTN)

Top | ILE CEE APIs | APIs by category

Data Types and Limits

Following is a description of the data types used in the math bindable APIs. The special values and limits of the data types are also listed.

The semantics and requirements of a high-level language may affect the values of the data types.

Integer Data Types

The following table shows the range of values that math bindable APIs can represent and use with the integer data types.

Precise Limit Values of Integer Data Types

Data Type	Length (bytes)	Range of values
INT2	2	-32 768 through 32 767
INT4	4	-2 147 483 648 through 2 147 483 647
INT8 (See note)	8	-9223372036854775808 through 9223372036854775807
Note: The data type is not currently available. For ILE C, storage for this variable could be defined as char(8).		

The operating system represents integers internally in two's complement notation, and the leftmost bit is the sign of the number.

Real Data Types

The following table shows the range of values that math bindable APIs can represent and use with the real data types.

Approximate Limit Values of Real Data Types

Data Type	Length (bytes)	Approximate Absolute Nonzero Minimum	Approximate Absolute Maximum	Approximate Precision (Decimal Digits)
FLOAT4	4	1.175494×10^{-38}	3.402823×10^{38}	7
FLOAT8	8	$2.225074 \times 10^{-308}$	1.797693×10^{308}	15

The operating system represents real data type values in IEEE floating-point format.

The following table lists the special values for floating-point operations. The values adhere to the IEEE standard for binary floating-point arithmetic.

Floating-point Special Values

Data Type	Description	Values

FLOAT4	+INF	0x7F800000
FLOAT4	HUGE	0x7F7FFFFF
FLOAT4	+0	0x00000000
FLOAT4	-0	0x80000000
FLOAT4	-INF	0xFF800000
FLOAT4	masked NaN	0x7FC00000
FLOAT4	unmasked NaN	0x7F800010
FLOAT8	+INF	0x7FF00000 0x00000000
FLOAT8	HUGE	0x7FEFFFFF 0xFFFFFFFF
FLOAT8	+0	0x00000000 0x00000000
FLOAT8	-0	0x80000000 0x00000000
FLOAT8	-INF	0xFFF00000 0x00000000
FLOAT8	masked NaN	0x7FF80000 0x00000000
FLOAT8	unmasked NaN	0x7FF00000 0x00000001

Complex Data Types

The following table shows the range of values that the math bindable APIs can represent and can use with complex data types.

Approximate Limit Values of Complex Data Types

Data Type	Length (bytes)	Approximate Values
COMPLEX8	8	Both real and imaginary parts are _FLOAT4. See Approximate Limit Values of Real Data Types
COMPLEX16	16	Both real and imaginary parts are FLOAT8. See Approximate Limit Values of Real Data Types

Calling Math Bindable APIs

ILE math bindable APIs can be called through the intrinsic functions of an ILE language or through a call statement specific to the ILE language. In either of the calling conventions, you need to specify one or two input parameters for the math bindable API being called. The number of input parameters is indicated in Table 5 under the No. Inputs column.

For example, if the number of input parameters for CEESxnnn is 1, the API is based on the following format:

Required Parameter:

1	parm1	Input	Data type
2	result	Output	Data type

Omissible Parameter:

3	fc	Output	FEEDBACK

If the number of input parameters for CEESxnnn is 2, the API is based on the following format:

Required Parameter Group:

1	parm1	Input	Data type
2	parm2	Input	Data type
3	result	Output	Data type

Omissible Parameter:

4	fc	Output	FEEDBACK

CEES*xnnn*

The name of the math bindable API being called. The character *x* identifies the type of parameters being passed to the API, and may be one of the following:

I	32-bit binary integer. The data type is INT4.
J	64-bit binary integer. The data type is INT8.
S	32-bit single floating-point number. The data type is FLOAT4.
D	A 64-bit double floating-point number. The data type is FLOAT8.
T	A 32-bit single floating-complex number (both real and imaginary parts are 32 bits long). The data type is COMPLEX8.

E A 64-bit double floating-complex number (both real and imaginary parts are 64 bits long). The data type is COMPLEX16.

The character *nnn* identifies the API being called. contains information about each of the ILE math bindable APIs.

parm1

The first input parameter to the API. The declared type of this parameter must match the type specified by the *x* in the called API.

parm2

The second input parameter to the API. The declared type of this parameter must match the type specified by the *x* in the called API.

fc

This omissible parameter represents a 12-byte feedback code.

result

The output parameter containing the result of the computations performed by the API.

Math Bindable APIs Are Procedures

The math bindable APIs are procedures, and therefore parameters are passed by reference. The math library handles all exceptions by returning a feedback code, if one was specified on the call. If the pointer to the feedback area is null (0), the feedback area is created and signaled to the ILE condition manager and a program message is generated by the condition manager.

The APIs that perform computations will do so in the rounding mode of "round to nearest" to achieve better accuracy. The APIs save the current rounding mode (before the computations start), and restore the previous one before the APIs exit. This is a general rule for the math bindable APIs.

ILE Math Bindable API Descriptions

The following information is provided for each math bindable API.

Definition

This column states the nature of the computation performed by the API.

API Name

This column gives the API names of the procedure.

Parameter Type

This column describes the acceptable parameter types that are input to and output by the specified API, and is indicated by the following characters:

I	32-bit binary integer.
J	64-bit binary integer.
S	32-bit single floating-point number.
D	A 64-bit double floating-point number.
T	A 32-bit single floating-complex number (both real and imaginary parts are 32 bits long).
E	A 64-bit double floating-complex number (both real and imaginary parts are 64 bits long).

Note: The characters correspond to the fifth character of the called API.

Equation

This column gives a math equation that represents the computation.

The following notation is used in the equations of the APIs:

$\|x\|$	denotes the absolute value of x.
sign(x)	is +1 if x >= 0 or -1 if x < 0.
f	denotes a function result.
z	denotes a complex argument, where `z = x + iy.`
z1 and z2	denote two complex arguments, where `z1 = x1 + iy1 and z2 = x2 +iy2.`

No. Inputs

This column states how many input parameters must be passed to the routine. See Calling Math Bindable APIs for a description of the format of APIs with 1 or 2 input parameters.

Input Range

This column gives the valid range for input parameters. If a parameter is not within the range, an error message is issued. (See the table below for a description of the messages that can be generated by the math bindable APIs.)

For output ranges, see Data Types and Limits for information concerning integer data types, real data types, and complex data types.

Math API Descriptions

Definition	API Name	Para-meter Type	Equation	No. Inputs	Input Range[1]

Absolute Function	CEESxABS	I
		J
		S
		D
		T
		E
			`f = |x|`
			or
			`f = |z|`
				1	Any integer number (for parameter type I and J).
					
					Any real number (for parameter types S and D).
					
					Any complex number (for parameter types T and E).

Arccosine	CEESxACS	S D	$f = \cos^{-1}(x)$	1	$\|x\| <= 1$
Arcsine	CEESxASN	S D	$f = \sin^{-1}(x)$	1	$\|x\| <= 1$
Arctangent	CEESxATN	S D T E	$f = \tan^{-1}(x)$ or $f = \tan^{-1}(z) = \tan^{-1}(x + iy)$	1	Any real argument (for parameter types S and D). Any complex argument (for parameter types T and E).

Arctangent2	CEESxAT2	S D	f = atan2(y/x) For x > 0, atan2(y, x) = $\tan^{-1}$(y/x). For x < 0 and y > 0, atan2(y, x) = pi + $\tan^{-1}$(y/x). For x < 0 and y < 0, atan2(y, x) = -pi + $\tan^{-1}$(x/y).	2	y is not equal to 0.0 and x is not equal to 0.0; y is not equal to +INF or -INF and x is not equal to +INF or -INF
Conjugate of Complex	CEESxCJG	T E	f = x - iy for argument z = x + iy.	1	Any complex number.
Cosine	CEESxCOS	S D T E	f = cos(x) or f = cos(z) = cos (x + iy)	1	Any real argument, in radians, such that |x| <= 0x432921FB54442D18 ~= pi * 2^{50} ~= 3537118876014220.0. Any complex number such that |y| <= 88.7228 for FLOAT4 and |y| <= 709.7827 for FLOAT8, and x is any real argument.
Cotangent	CEESxCTN	S D	f = cot(x)	1	Any real argument, in radians, such that |x| <= 0x432921FB54442D18 ~= pi * 2^{50} ~= 3537118876014220.0.
Error Function and its Complement	CEESxERF CEESxERC	S D	$f = erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t2dt}$	1	Any real argument

Exponential Base e	CEESxEXP	S D T E	$f = e^x$ or $f = e^z = e^{x+iy}$	1	-87.3365 <= x <= 88.7228 (for parameter type S). -708.3964 <= x <= 709.7827 (for parameter type D). -87.3365 <= x <= 88.7228 and y is any real number (for parameter type T). -708.3964 <= x <= 709.7827 and y is any real number where |y| <= 0x432921FB54442D18 ~= pi * 2^{50} ~= 3537118876014220.0. (for parameter type E).
Exponentiation	CEESxXPy	I J S D T E	$f = x^y$ (See note 2)	2	Any integer arguments (for CEESIXPI, CEESSXPI, CEESDXPI, CEESTXPI, CEESEXPI, CEESJXPJ, CEESSXPJ, CEESDXPJ, CEESTXPJ, CEESEXPJ). Any real argument subject to the condition that if x is negative, y must be an integer (for CEESSXPS, CEESDXPD). For complex data types, see the argument for real data types above (for CEESTXPT, CEESEXPE).
Factorial	CEE4SxFAC	I J	n! = 1*2*3 ... (n-1)*n.	1	For parameter type I, any integer <=12. For parameter type J, any integer <=20.
Floating Complex Divide	CEESxDVD	T E	$f = z_1 / z_2 = (x_1 + iy_1) / (x_2 + iy_2)$	2	Any complex number.
Floating Complex Multiply	CEESxMLT	T E	$f = z_1 * z_2 = (x_1 + iy_1) * (x_2 + iy_2)$	2	Any complex number.
Gamma Function	CEESxGMA	S D	$f = \Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$	1	0 < x <= 35.04 (for parameter type S). 0 < x <= 171.6243 (for parameter type D).

Hyperbolic Arctangent	CEESxATH	S D T E	$f = \tanh^{-1}(x)$ or $f = \tanh^{-1}(z) = \tanh^{-1}(x + iy)$	1	Any real argument such that $\|x\| <= 1$ (for parameter types S and D). Any complex number such that x is not equal to 1.0 and y is not equal to 0 (for parameter types T and E).
Hyperbolic Cosine	CEESxCSH	S D T E	f = cosh(x) or f = cosh(z) = cosh(x + iy)	1	Any real argument such that $\|x\| <= 89.4159$ (for FLOAT4) and $\|x\| <= 709.7827$ (for FLOAT8). Any complex number such that $\|x\| <= 89.4159$ (for FLOAT4) $\|x\| <= 709.7827$ (for FLOAT8).
Hyperbolic Sine	CEESxSNH	S D T E	f = sinh(x) or f = sinh(z) = sinh(x + iy)	1	Any real argument such that $\|x\| <= 89.4159$ (for FLOAT4) and $\|x\| <= 709.7827$ (for FLOAT8). Any complex number such that $\|x\| <= 89.4159$ (for FLOAT4) and $\|x\| <= 709.7827$ (for FLOAT8).
Hyperbolic Tangent	CEESxTNH	S D T E	f = tanh(x) or f = tanh(z) = tanh(x + iy)	1	Any real argument (for parameter types S and D). Any complex number such that $\|x\| < 43.66825$ (for parameter type T), and $\|x\| < 354.1982$ (for parameter type E).
Imaginary Part of Complex	CEESxIMG	T E	f = y, where z = x + iy.	1	Any complex number.
Log Gamma Function	CEESxLGM	S D	f = ln(Gamma (x))	1	$0 < x <= 4.0850*10^{36}$ (for parameter type S), and $0.0 < x <= 2^{1014}$ (for parameter type D).

Logarithm Base 10	CEESxLG1	S D	f = $\log_{10}(x)$	1	$x > 0.0$
Logarithm Base 2	CEESxLG2	S D	f = $\log_{2}(x)$	1	$x > 0.0$
Logarithm Base e	CEESxLOG	S D T E	f = $\log_e(x)$ or f = $\ln(z) = \log_e(\|z\|) + i\text{atan2}(y, x)$	1	x > 0.0 (for parameter types S and D). z is not equal to 0+*i*0 (for parameter types T and E).
Modular Arithmetic	CEESxMOD	I J S D	f = remainder of x/y (See note 3)	2	Any two integer numbers such that y is not equal to 0 (for parameter types I and J). Any two real numbers such that y is not equal to 0.0 (for parameter types S and D).
Nearest 64-Bit Integer	CEESxNJN	S D	If x >= 0.0 f = sign(x)*n where n is the largest 64-bit integer <= |x + 0.5| or If x < 0.0 f = sign(x)*n where n is the largest 64-bit integer <= |x - 0.5|	1	Any real number.
Nearest Integer	CEESxNIN	S D	If x >= 0.0 f = sign(x)*n where n is the largest integer <= |x + 0.5| or If x < 0.0 f = sign(x)*n where n is the largest integer <= |x - 0.5|	1	Any real number.

Nearest Whole Number	CEESxNWN	S D	If x >= 0.0 f = sign(x)*n where n is the largest integer <= |x + 0.5| or If x < 0.0 f = sign(x)*n where n is the largest integer <= |x - 0.5|	1	Any real number.
Positive Difference	CEESxDIM	I J S D	If x > y, f = x - y or If x <= y, f = 0	2	Any integer argument (for parameter types I and J). Any real number (for parameter types S and D).
Sine	CEESxSIN	S D T E	f = sin(x) or f = sin(z) = sin(x + iy)	1	Any real argument, in radians, such that |x| <= 0x432921FB54442D18 ~= pi*2^{50} ~= 3537118876014220.0 (for parameter types S and D). Any complex number such that |y| <= 88.7228 (for parameter type T), and |y| <= 709.7827, and |x| <= 0x432921FB54442D18 ~= pi*2^{50} ~= 3537118876014220.0 (for parameter type E).
Square Root	CEESxSQT	S D S D	$f=\sqrt{x}$ or $f=\sqrt{z}=\sqrt{x+iy}$	1	Any real argument such that x >= 0.0 (for parameter types S and D). Any complex number such that |z| + |x| <= 1.797693*10^{308} (for parameter types T and E).

Tangent	CEESxTAN	S
		D
		T
		E
			`f = tan(x)`
			`or`
			`f = tan(z) = tan(x + iy)`
				1	`Any real argument, in radians, such that`
					`|x| <= 0x432921FB54442D18 ~= pi*`2^{50}
					`~= 3537118876014220.0`
					`(for parameter types S and D).`
					`Any complex number such that`
					`|y| < 43.66825 (for parameter`
					`type T), and |y| < 354.1982`
					`(for parameter type E).`

Transfer of Sign	CEESxSGN	I
		J
		S
		D
			f = sign(y)|x|	2	`Any integer argument`
					`(for parameter types I and J).`
					`Any real number`
					`(for parameter types S and D).`

Truncation	CEESxINT	S
		D
			`f = sign(x)*n,`
			`where n is the largest integer`
			`<= |x|`
				1	Any real number.

Notes:

1

For output ranges, see Data Types and Limits, including information on Integer Data Types, Real Data Types, and Complex Data Types.

2

Where x and y have the following combinations, respectively:

```
integer, integer
real FLOAT4, integer
real FLOAT8, integer
single complex, integer
double complex, integer
real FLOAT4, integer
real FLOAT8, integer
single complex, integer
double complex, integer
64-bit integer, 64-bit integer
real FLOAT4, 64-bit integer
real FLOAT8, 64-bit integer
single complex, 64-bit integer
double complex, 64-bit integer
real FLOAT4, real FLOAT4
real FLOAT8, real FLOAT8
```

```
single complex, single complex
double complex, double complex
```

3

The absolute value of the result is always less than the absolute value of y.

Message Descriptions for Math Bindable APIs

A summary of the messages generated by the math bindable APIs is contained in the table below.

Msg_No	Msg_Id	Explanation
2002	CEE2002	The argument is too close to multiple (pi/2)'s for tangent and multiple pi's for cotangent. This causes an inaccurate result.
2003	CEE2003	In (x**y) both x and y are integers, x = 0 and y < 0 for procedure &1. The result is undefined. The fixed-point zero-divide exception occurs.
2004	CEE2004	In (x**y) x is real and y is integer, x = 0.0 and y < 0 for procedure &1. The result is undefined. The floating-point zero-divide exception occurs.
2005	CEE2005	The value of the argument for &1 is outside the range &2. It causes a floating-point overflow exception.
2006	CEE2006	In (x**y) both x and y are real, x = 0.0 and y < 0.0 for procedure &1. The result is undefined. The floating-point zero-divide exception occurs.
2008	CEE2008	For an exponentiation operation (Z**P) where the complex base Z equals 0, the real part of the complex exponent P is less than or equal to 0. The floating-point zero-divide exception occurs in procedure &1.
2009	CEE2009	The value of the real part of a complex argument is greater than &2 for procedure &1. It causes a floating-point overflow exception.
2010	CEE2010	The value of the argument is less than 0.0 for procedure &1. It is not valid for the square root function. The result is undefined.
2011	CEE2011	The argument for procedure &1 is greater than &2. It causes a floating-point overflow exception.
2012	CEE2012	The argument for procedure &1 is negative. It is not valid for the logarithmic function.
2013	CEE2013	The absolute value of the imaginary part of the complex argument for &1 is greater than &2. This causes floating-point overflow.
2014	CEE2014	Both arguments to the arctangent2 function are either 0 or infinity for procedure &1. They are not valid for arctangent2 function.
2015	CEE2015	The value of the real part of a complex argument for procedure &1 is less than &2. It causes a floating-point underflow exception.
2016	CEE2016	The absolute value of the argument for procedure &1 is greater than &2. The argument is out of range and the result is undefined.
2017	CEE2017	The absolute value of the argument for procedure &1 is greater than or equal to &2. This causes a floating-point overflow exception.
2018	CEE2018	The real and imaginary parts of the argument for procedure &1 are zero or infinity. They are not valid for complex logarithmic functions.

2019	CEE2019	The absolute value of the real part of the complex argument for procedure &1 is greater than &2. This causes a floating-point overflow exception.
2020	CEE2020	In (x**y) x < 0.0 and y is not an integer for procedure &1. It causes a floating-point zero-divide exception. The result is undefined.
2022	CEE2022	The complex argument is not valid for procedure &1. It may be one of the following: • The real part of the argument is 0. • Real**2 + imaginary**2 = 1.0.
2023	CEE2023	The calculated result overflows the result field in procedure &1.
2024	CEE2024	Floating-point overflow exception occurred in procedure &1.
2025	CEE2025	Floating-point underflow exception occurred in procedure &1.
2026	CEE2026	The denominator is 0. The operand for the modular function is not valid. It causes a floating-point zero-divide exception.
2027	CEE2027	Floating-point zero divide exception occurred in procedure &1.
2101	CEE2101	The argument is an unmasked NaN for procedure &1. If the argument is a complex number, either its real part or imaginary part is an unmasked NaN. It causes a floating-point incorrect operand exception.
2102	CEE2102	The argument for procedure &1 is less than &2. It causes a floating-point underflow exception.
2103	CEE2103	Floating-point operand exception occurred in &1.
2117	CEE2117	The values of the real part and imaginary part cannot be 1.0 and 0.0 respectively at the same time for procedure &1. The result is undefined.
2118	CEE2118	The sum of the absolute value of the complex number, and the absolute value of its real part, is greater than the maximum FLOAT8 (1.797693*10**308) for procedure &1.

Basic Random Number Generation (CEERAN0) API

Required Parameter Group:		
1 seed	I/O	INT4
2 random_no	Output	FLOAT8
Omissible Parameter:		
3 fc	Output	FEEDBACK
Service Program Name: QILE		
Default Public Authority: *USE		

The Basic Random Number Generation (CEERAN0) API generates a sequence of uniform pseudorandom numbers between 0 and 1 using the multiplicative congruential method with a user-specified seed.

Required Parameter Group

seed (I/O)

A 32-bit binary integer representing an initial value used to generate random numbers. It must be a variable; that is, it cannot be an input-only parameter. The valid range is 0 to 2 147 483 646.

If the value of the seed parameter is 0, the seed is generated from the current Greenwich Mean Time.

On return, CEERAN0 changes the value of *seed* so that it may be used as the new seed in the next call.

random_no (output)

A 64-bit double floating-point pseudorandom number with a value between 0 and 1, exclusive. If *seed* is not valid, *random_no* is set to -1 and CEERAN0 ends with a nonzero feedback code.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2523	UTC not available to generate random seed from system time
Severity: 10	
CEE2524	Seed value for &1 is not valid
Severity: 30	

API Introduced: V2R3

Convert 64-Bit Integer to Character String (CEE4JNTS) API

Required Parameter Group:

1	integer	Input	INT8
2	sign	Input	INT4
3	length	I/O	INT4
4	output_string	Output	VSTRING

Omissible Parameter:

5	fc	Output	FEEDBACK

Default Public Authority: *USE

The Convert 64-Bit Integer to Character String (CEE4JNTS) API converts a 64-bit binary integer to its character string equivalent.

Required Parameter Group

integer (input)

A 64-bit integer that needs to be formatted.

sign (input)

A 32-bit binary integer of one of the following values.

0	Parameter 1 is an unsigned 64-bit integer
1	Parameter 1 is a signed 64-bit integer

length (I/O)

A 32-bit binary integer representing the length of the output string provided.

On return, this parameter will contain the number of bytes used to represent the string.

output_string (output)

A character string that is the result of formatting the 64-bit integer. The string is left-justified. The condition CEE3470 will be signaled to the condition manager if output is truncated.

The output_string will not contain the plus character (+) if the integer provided is unsigned.

The output_string length should contain enough space to store the resulting string. The maximum storage needed is 20 bytes based on a maximum value of 18446744073709551615 and a minimum value of -9223372036854775808.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE3470	The resulting string is truncated
Severity: 30	

API Introduced: V2R3

Top | ILE CEE APIs | APIs by category

Convert Character String to 64-Bit Integer (CEE4JSTN) API

Required Parameter Group:

1	input_string	Input	VSTRING
2	length	Input	INT4
3	output_integer	output	INT8

Omissible Parameter:

5	fc	Output	FEEDBACK

Default Public Authority: *USE

The Convert Character String to 64-Bit Integer (CEE4JSTN) API converts a character string representation of an integer to its 64-bit integer equivalent.

Required Parameter Group

input_string (input)

A character string representing the integer that needs to be formatted. The valid range for signed integer is $(-2^{63}) < x < (2^{63} - 1)$. The range for unsigned is $(0 < x < 2^{64} - 1)$.

The following rules apply to input_string.

The decimal separator is retrieved from the job attributes. The input string is scanned left to right until the decimal point or end of string is reached. Any character to the right of the decimal point will be ignored.

Multiple - and + are allowed. The first sign, starting from right to left will indicate the sign of the input.

Imbedded spaces, "-", "+", "." and "," are valid characters.

For examples of this process, see API Examples.

length (input)

A 32-bit binary integer representing the number of characters in the input string provided including the sign.

output_integer (output)

A 64-bit storage in which the result is stored.

Omissible Parameter

fc (output)

A 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE2537	Intput string contains characters that are not valid
Severity: 30	
CEE2539	Intput integer is too large or too small
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Examples

In the following examples, "." is the decimal separator.

Input	**Output**
" +241-"	-241
" +1 001.23-"	-1001
" 537,072.34+ "	537072

API Introduced: V4R3

Message Services APIs

Bindable APIs are provided for message services. The APIs can be used to dispatch, format, obtain, retrieve, and store messages.

The message services APIs are:

- Dispatch a Message (CEEMOUT) dispatches a message string.
- Get a Message (CEEMGET) retrieves a message and stores it in a buffer.
- Get, Format, and Dispatch a Message (CEEMSG) obtains, formats, and dispatches a message that corresponds to an input condition token.

Dispatch a Message (CEEMOUT) API

Required Parameter Group:			
1	message_string	Input	VSTRING
2	destination_code	Input	INT4
Omissible Parameter:			
3	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe:			

The Dispatch a Message (CEEMOUT) API is used to dispatch a message string.

Required Parameter Group

message_string (input by descriptor)

A valid ILE string, passed by reference with a descriptor, to be dispatched as a message. It is not necessary that this string first be retrieved by the CEEMGET API. Insert data cannot be inserted with this call.

destination_code (input)

A 4-byte binary integer of one of the following values:

1	Dispatch the message for output to the standard output (console or file). The message is also logged in the *EXT message queue of the job.
2	Log the message to the system message file only; the message is not displayed.

Note: Messages sent through this API are subject to the normal server rules for message filtering.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0451	Unsupported destination code &2 passed to &1
Severity: 30	
CEE0501	The operational descriptor data type is not valid
Severity: 30	
CEE0502	Missing operational descriptor
Severity: 30	
CEE3101	&1 cannot be called in the default activation group
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- A CPF9898 message of type *INFO is used to log the message being dispatched. It is sent to the message queue of the caller of CEEMOUT. Message CPF9898 appends a period to the displayed text.

API Introduced: V2R3

Get a Message (CEEMGET) API

Required Parameter Group:

1	cond_token	Input	FEEDBACK
2	message_area	Output	VSTRING
3	msg_ptr	I/O	INT4

Omissible Parameter:

4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe:

The Get a Message (CEEMGET) API retrieves a message and stores it in a buffer for manipulation or output by the caller.

The API retrieves a message and places it in the storage location referenced by the message_area parameter.

The msg_ptr parameter has a value of zero on the initial call to the CEEMGET API. If the message is too large to be contained in *message_area*, *msg_ptr* is returned containing an index into the message. The index is used in subsequent calls to CEEMGET to retrieve the remaining portion of the message. When the entire message has been retrieved, *msg_ptr* is returned containing a value of zero.

Required Parameter Group

cond_token (input)

A 12-byte condition token. See for a description of the condition token.

message_area (output by descriptor)

A valid ILE string variable, passed by reference with a descriptor. The CEEMGET API places the retrieved message into this string variable.

msg_ptr (input/output)

A 4-byte integer with a value of 0 on the initial call to CEEMGET to retrieve a message. If the message is too large to be contained in the *message_area*, *msg_ptr* will be returned containing an index into the message. The index is used in subsequent calls to CEEMGET to retrieve the remaining portion of the message. When the entire message has been retrieved, *msg_ptr* is returned with a value of 0.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0102	The condition token passed to &1 is not valid
Severity: 30	
CEE0454	&1 cannot find message &3 in message file &2
Severity: 30	
CEE0455	The message returned is truncated
Severity: 10	
CEE0458	&1 cannot find message file &2
Severity: 30	
CEE0501	The operational descriptor data type is not valid
Severity: 30	
CEE0502	Missing operational descriptor
Severity: 30	
CEE3103	Cannot allocate storage in &1
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- If *msg_ptr* is greater than the length of the message being retrieved, then *msg_ptr* is set to 0, and a zero-length string is copied into *message_area*.
- Insert data cannot be inserted with this call.

API Introduced: V2R3

Get, Format, and Dispatch a Message (CEEMSG) API

Required Parameter Group:			
1	cond_token	Input	FEEDBACK
2	destination_code	Input	INT4
Omissible Parameter:			
3	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Get, Format, and Dispatch a Message (CEEMSG) API is used to obtain, format, and dispatch a message corresponding to an input condition token.

Required Parameter Group

cond_token (input)

A 12-byte condition token. See for a description of the condition token.

destination_code (input)

A 4-byte binary integer of one of the following values:

1	Dispatch the message for output to the standard output. The message is also logged in the system message file.
2	Log the message to the system message file only; the message is not displayed.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0102	The condition token passed to &1 is not valid
Severity: 30	
CEE0451	Unsupported destination code &2 passed to &1
Severity: 30	
CEE0458	&1 cannot find message file &2
Severity: 30	
CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- A CPF9898 message of type *INFO is used to log the message being dispatched. It is sent to the message queue of the caller of CEEMSG. Message CPF9898 appends a period to the displayed text.

API Introduced: V2R3

Program or Procedure Call APIs

ILE CEE APIs are provided to retrieve operational descriptor information and to test for omitted arguments.

The program or procedure call APIs are:

- Get String Information (CEEGSI) retrieves string information about a parameter used in the call to this API.
- Retrieve Operational Descriptor Information (CEEDOD) retrieves operational descriptor information about a parameter used in the call to this API.
- Test for Omitted Argument (CEETSTA) is used to test for the presence or absence of an omissible argument.

Get String Information (CEEGSI) API

Required Parameter Group:

1	posn	Input	INT4
2	datatype	Output	INT4
3	currlen	Output	INT4
4	maxlen	Output	INT4

Omissible Parameter:

5	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe:

The Get String Information (CEEGSI) API retrieves string information about a parameter used in the call to this API. String information describes the elements of a parameter, such as the type and the length of the string.

Required Parameter Group

posn (input)

The ordinal position in the parameter list of the formal parameter whose operational descriptor is required. A value of 1 denotes the leftmost parameter.

datatype (output)

The binary value of the data type field. The possible values and their data types are:

1	(typeEsc) An element descriptor type (descElmt) that is not one of the following data types.
2	(typeChar) A string of SBCS characters with values in the range X'00' through X'FF'.
3	(typeCharZ) A string of SBCS characters with values in the range X'01' through X'FF' that ends with a null byte (X'00').
4	(typeCharV2) A string of SBCS characters prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.
5	(typeCharV4) A string of SBCS characters prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

6 (typeBit) A string of bits with values of 0 or 1.

7 (typeBitV2) A string of bits prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of bits.

8 (typeBitV4) A string of bits prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of bits.

9 (typeGChar) A string of DBCS characters with values in the range X'0000' through X'FFFF'.

10 (typeGCharZ) A string of DBCS characters with values in the range X'0001' through X'FFFF' that end with a null byte (X'0000').

11 (typeGCharV2) A string of DBCS characters prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

12 (typeGCharV4) A string of DBCS characters prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

currlen (output)

A binary number containing the current number of elements in the string, as follows:

- For strings of typeEsc, this is the length from the descriptor, and is equal to the maximum length returned in *maxlen*.
- For strings of typeCharV2 and typeCharV4, this is the length from the argument (binary prefix) itself.
- For strings of typeCharZ, the API determines the current length (in number of characters, either SBCS or DBCS) of the string by scanning for the null character. If the length in the descriptor is nonzero (the compiler of the caller knew the extent), the search is confined to that length. Otherwise, the scan continues until a null character is found.
- If the data type was undefined, the contents of this field are undefined.

maxlen (output)

A binary number containing the maximum number of string elements, as follows:

- For strings of typeEsc, this is the length from the descriptor, and is equal to the current length returned in *currlen*.
- For strings of typeCharV2 and typeCharV4, this is the length from the descriptor (which does not include the length of the prefix).
- For strings of typeCharZ, the maximum length is the number of the characters excluding the null character. It is the maximum length from the descriptor minus 1 (to account for the SBCS or DBCS null character). If the length in the descriptor is zero, the maximum length is set equal to the current length.
- If the data type was undefined, the contents of this field are undefined.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0501	The operational descriptor data type is not valid
Severity: 30	
CEE0502	Missing operational descriptor
Severity: 30	
CEE0505	A NULL-terminated string is not found
Severity: 10	

Usage Notes

- It is an error to use CEEGSI to test an argument that is not a reference argument or that is preceded in the argument list by other arguments that are not reference arguments.

  Note: This error may not be diagnosed.
- The results are undefined if this API is used in a function that does not have operational descriptors specified.
- CEEGSI is implemented as a builtin and therefore cannot have its address taken or be called through a procedure pointer.

API Introduced: V2R3

Retrieve Operational Descriptor Information (CEEDOD) API

Required Parameter Group:

1	posn	Input	INT4
2	desctype	Output	INT4
3	datatype	Output	INT4
4	descinf1	Output	INT4
5	descinf2	Output	INT4
6	datalen	Output	INT4

Omissible Parameter:

7	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Retrieve Operational Descriptor Information (CEEDOD) API retrieves operational descriptor information about a parameter used in the call to this API. The operational descriptor communicates additional information about a parameter, such as size and shape.

Required Parameter Group

posn (input)

The ordinal position in the parameter list of the formal parameter whose operational descriptor is required. A value of 1 denotes the leftmost parameter.

desctype (output)

The binary value of the descriptor type field. The possible values and their descriptor types are:

1	(descEsc) An escape descriptor.
2	(descElmt) An element descriptor. Elements are objects such as numbers and strings, that can be aggregated into arrays and structures.
3	(descArray) An array descriptor.
4	(descStruct) A structure descriptor.

datatype (output)

The binary value of the data type field. The possible values and their data types are:

1	(typeEsc) An element descriptor type (descElmt) that is not one of the following data types.
2	(typeChar) A string of SBCS characters with values in the range X'00' through X'FF'.
3	(typeCharZ) A string of SBCS characters with values in the range X'01' through X'FF' that ends with a null byte (X'00').
4	(typeCharV2) A string of SBCS characters prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.
5	(typeCharV4) A string of SBCS characters prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.
6	(typeBit) A string of bits with values of 0 or 1.
7	(typeBitV2) A string of bits prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of bits.
8	(typeBitV4) A string of bits prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of bits.
9	(typeGChar) A string of DBCS characters with values in the range X'0000' through X'FFFF'.
10	(typeGCharZ) A string of DBCS characters with values in the range X'0001' through X'FFFF' that end with a null byte (X'0000').
11	(typeGCharV2) A string of DBCS characters prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.
12	(typeGCharV4) A string of DBCS characters prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

descinf1 (output)

The binary value of the first descriptor information field. If the descriptor omits this field, *descinf1* is set to 0.

descinf2 (output)

The binary value of the second descriptor information field (used for bit alignment, for example). If the descriptor omits this field, *descinf2* is set to 0.

datalen (output)

The 4-byte binary value of the descriptor length field.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0501	The operational descriptor data type is not valid
Severity: 30	
CEE0502	Missing operational descriptor
Severity: 30	

Usage Notes

- CEEDOD is implemented as a builtin and therefore cannot have its address taken or be called through a procedure pointer.
- It is an error to use CEEDOD to test an argument that is not a reference argument. It is also an error to test an argument that is preceded in the argument list by other arguments that are not reference arguments.
- The results are undefined if you use the CEEDOD API in a function that does not specify operational descriptors.

API introduced: V2R3

Test for Omitted Argument (CEETSTA) API

Required Parameter Group:

1	presence_flag	Output	INT4
2	arg_num	Input	INT4

Omissible Parameter:

3	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Test for Omitted Argument (CEETSTA) API is used to test for the presence or absence of an omissible argument.

Required Parameter Group

presence_flag (output)

The presence or absence of the argument tested. A value of 1 indicates that the argument is present; a value of 0 indicates that the argument was omitted.

arg_num (input)

The argument number to be tested (in the argument list of the procedure that called the procedure where the CEETSTA call is coded), counting the first argument as 1.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	

CEE0503	The argument number is not valid **Note:** ILE does not check the condition where the argument number is greater than the number of arguments in the list.
Severity: 30	
CEE3005	The argument being tested is not an address
Severity: 30	

Usage Notes

- It is an error to use CEETSTA to test an argument that is not a reference argument or that is preceded in the argument list by other arguments that are not reference arguments.

  Note: This error may not be diagnosed.

- CEETSTA is implemented as a builtin and therefore cannot have its address taken or be called through a procedure pointer.

API Introduced: V2R3

Storage Management APIs

ILE CEE APIs are provided for all heap operations. Applications can be written using either the storage management APIs, language-intrinsic functions, or both.

The storage management APIs fall into the following categories:

- Basic heap operations
- Extended heap operations
- Heap allocation strategies

See Allocation Strategy Type (CEE4ALC) for information on the attributes that are used to define the characteristics of the storage allocated for heaps.

Allocation Strategy Type (CEE4ALC)

The CEE4ALC allocation strategy type contains attributes that are used to define the characteristics of the storage allocated for heaps. ILE C defines the attributes as shown in CEE4ALC Definition for ILE C. ILE COBOL defines the attributes as shown in CEE4ALC Definition for ILE COBOL, and ILE RPG defines the attributes as shown in CEE4ALC Definition for ILE RPG.

User-Defined Allocation Strategy

You can define an allocation strategy by altering the attributes of the CEE4ALC allocation strategy type. You do this with the Define Heap Allocation Strategy (CEE4DAS) API. Then, when you use the Create Heap (CEECRHP) API, you can specify the allocation strategy that you defined for the heap attributes you require.

If you use the CEECRHP API, but did not define an allocation strategy, a default allocation strategy provides the heap attributes. See The Default Heap for a description of the default allocation strategy.

Note: The creation size and extension size values of CEE4ALC may be overridden. You do this by specifying values for the initial_size parameter and the increment parameter on the CEECRHP bindable API.

For ILE C the CEE4ALC allocation strategy type is defined as follows:

CEE4ALC Definition for ILE C

```
struct _CEE4ALC {
         _INT4 max_sngl_alloc;  /* maximum size of a single allocation */
         _INT4 min_ddy;         /* minimum boundary alignment of any
allocation */
         _INT4 crt_size;        /* initial creation size of the heap */
         _INT4 ext_size;        /* the extension size of the heap */
         _INT2 reserved1;       /* must be binary 0 */
         _BITS alloc_strat:1;   /* a choice for allocation strategy */
         _BITS no_mark:1;       /* a group deallocation choice */
         _BITS blk_xfer:1;      /* a choice for block transfer of a heap */
         _BITS PAG:1;           /* a choice for heap creation in a PAG */
         _BITS alloc_init:1;    /* a choice for allocation initialization
*/
         _UCHAR init_value;     /* initialization value */
         _BITS reserved2:3;     /* must be binary 0 */
         _BITS reserved3:32;    /* must be binary 0 */
        };
```

For ILE COBOL the CEE4ALC allocation strategy type is defined as follows:

CEE4ALC Definition for ILE COBOL

```
 01  CEE4ALC-my-own.
   05 max-sngl-alloc pic 9(9) BINARY.
   05 min-bdy        pic 9(9) BINARY.
   05 crt-size       pic 9(9) BINARY.
   05 ext-size       pic 9(9) BINARY.
   05 reserved1      pic 9(4) BINARY VALUE 0.
   05 my-bits        pic x VALUE X"F8".
* Using initial value of X"F8" ,   alloc_strat, no_mark,
* blk_xfer, PAG, and alloc_init are set to TRUE
   05 init-value     pic x.
   05 my-reserved2   pic x(5).
   05 my-reserved3   pic x(5).
```

For ILE RPG the CEE4ALC allocation strategy type is defined as follows:

CEE4ALC Definition for ILE RPG

```
D CEE4ALC         DS
D   MaxSglAloc              10I 0
D   MinBdy                  10I 0
D   CrtSiz                  10I 0
D   ExtSiz                  10I 0
D   res1                     5I 0
D   AllocBits                1A
D   InitValue                1A
D   res2                     5A
  * Use BITON to set on the appropriate bit.
  * For example    BITON     AllocInit     AllocBits
D AllocStrat      C                  '0'
D NoMark          C                  '1'
D BlkXfer         C                  '2'
D PAG             C                  '3'
D AllocInit       C                  '4'
```

max_sngl_alloc

The maximum allocation size, in bytes, of any single allocation from the heap. This attribute is useful for controlling the use of the heap. The minimum value for this attribute is 4 bytes, and the maximum value is 16MB minus 64KB. If 0 is specified, the default value of 16MB minus 64KB is used.

min_bdy

The minimum boundary alignment, in bytes, associated with any allocation from the heap. The minimum value for a boundary alignment is 4 bytes, and maximum value is 512 bytes. To allow valid pointers to be stored in a storage allocation with the heap, a minimum boundary alignment of 16 bytes is required. If zero is specified, a default value of 16 bytes is used for the boundary alignment. The minimum boundary alignment is rounded up to a power of 2.

crt_size

The creation size, in bytes, of the heap. The minimum value for the size of the heap is 512 bytes and the maximum value is 16MB minus 1KB. If 0 is specified, the system computes a default value. The value is rounded up to a 512-byte boundary.

Note: If system resources are constrained, the system may override the value specified.

ext_size

The extension size of the heap in bytes. The minimum value for extension size is 512 bytes and the maximum value is 16MB minus 1KB. If 0 is specified the system computes a default value. The value is rounded up to a 512-byte boundary.

Note: If system resources are constrained, the system may override the value specified.

reserved1

Must be binary 0.

alloc_strat

Allows a choice between:

0 Normal allocation strategy.

1 Create a process space on each allocation.

Note: This option should be used only in unusual situations, such as in debugging application problems caused by references past the end of an allocation.

no_mark

Allows a choice between:

0 Allow the use of the CEEMKHP and CEERLHP APIs.

1 Do not allow the use of the CEEMKHP and CEERLHP APIs.

blk_xfer

Used to increase the performance of a heap based on prior knowledge of how the heap is used. This attribute is used only when the heap is not a member of a process access group (PAG). The values are:

0 Transfer the minimum storage transfer size (that is, 1 storage unit).

1 Transfer the machine default storage transfer size (that is, 8 storage units).

Note: On the reduced instruction set computer (RISC) hardware, this parameter has no effect. Regardless of whether 0 or 1 is specified for blk_xfer, the machine default storage transfer size will always be transferred.

PAG

A heap can be created as a process access group (PAG) member. The values are:

0 Do not create the heap in the PAG.

1 Create the heap in the PAG.

Note: It is possible for the PAG to overflow, at which point any requested PAG heap creations or extensions will not reside in the PAG. Therefore, the system may ignore the request to create the heap in the PAG.

alloc_init

Allows the user to specify if all storage allocations from the heap being created will be initialized to the initialization value. The values are:

0 Do not initialize the heap with Init_Value.

1 Initialize the heap with Init_Value.

init_value

The value used to initialize the storage allocations. This value is not used if *alloc_value* is 0.

reserved2

Must be binary 0.

reserved3

Must be binary 0.

The Default Heap

From the programmer's viewpoint, a default heap is always available in the activation group. In fact, the first request to allocate storage results in the creation of the default heap from which the storage allocation takes place.

The attributes of the default heap are defined by the system through a default allocation strategy. You cannot change this default allocation strategy. Following is an example of the default allocation strategy:

```
Max_Sngl_Alloc = 16MB - 64KB
Min_Bdy        = 16
Crt_Size       = 4KB
Ext_Size       = 4KB
Alloc_Strat    = 0
No_Mark        = 1
Blk_Xfer       = 0
PAG            = 0
Alloc_Init     = 0
Init_Value     = 0x00  /* This value is not used */
                       /* if Alloc_Init is 0     */
```

In addition, the default heap in an activation group has the following special characteristics:

- It may not be discarded by CEEDSHP (discard a heap); it is guaranteed to be present for the life of the activation group.
- It is referred to with a heap identifier of 0.

Languages that do not have an intrinsic multiple-heap storage model (such as ILE C) use the default heap. This heap cannot be discarded and is immune to the Mark Heap (CEEMKHP) and Release Heap (CEERLHP) APIs. Storage allocated within the default heap can be freed only by explicit free operations or when the owning activation group ends.

This implementation ensures that the storage is not inadvertently released in mixed-language applications. Release heap and discard heap operations are considered insecure for the following reasons:

- Large applications that re-use existing code with potentially different storage models.
- The programmer is not completely familiar with the internals of each procedure.

If release heap operations were valid for the default heap, then procedures that correctly use different storage management capabilities separately might fail when used in combination.

Basic Heap Operations APIs

The basic heap operations APIs are:

- Allocation Strategy Type (CEE4ALC) specifies the allocation strategy defined for the heap attributes.
- Free Storage (CEEFRST) frees one previously allocated heap storage.
- Get Heap Storage (CEEGTST) allocates storage within a heap.
- Reallocate Storage (CEECZST) changes the size of previously allocated storage.

Free Storage (CEEFRST) API

Required Parameter:			
1	address	Input	POINTER
Omissible Parameter:			
2	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Free Storage (CEEFRST) API frees previously allocated heap storage.

Required Parameter

address (input)

The address returned by a previous CEEGTST call or a language-intrinsic function. The storage at this address is deallocated.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0802	The storage headers are damaged
Severity: 40	
CEE0810	The starting address for reallocation is not valid
Severity: 30	

Usage Notes

- The heap identifier is inferred from the address of the storage to be freed. The storage is deallocated from the heap in which it was allocated. The deallocate operation may be issued from an activation group other than the activation group that owns the heap.
- The pointer to the address passed as the argument is no longer valid after this call. The storage freed by this operation can be reallocated on a subsequent CEEGTST call. If the pointer is not reassigned, any further use of it will have unpredictable results.

API introduced: V2R3

Get Heap Storage (CEEGTST) API

Required Parameter Group:

1	heap_id	Input	INT4
2	size	Input	INT4
3	address	Output	POINTER

Omissible Parameter:

4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe:

The Get Heap Storage (CEEGTST) API allocates storage within a heap.

Required Parameter Group

heap_id (input)

A heap identifier of the heap in which the storage is allocated.

size (input)

The number of bytes of storage to be allocated.

address (output)

The address of the first byte of allocated storage.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0802	The storage headers are damaged
Severity: 40	
CEE0803	The heap identifier does not match any existing heap
Severity: 30	
CEE0808	Requested storage size is not valid
Severity: 30	
CEE0813	Insufficient storage available to satisfy the request
Severity: 30	

Usage Notes

- The size value is rounded up to a multiple of the minimum boundary specified when the heap space is created. The minimum boundary for the activation group default heap is 16 bytes.
- The address of the first byte of the allocation begins on a boundary at least as great as the minimum boundary specified when the heap is created. The minimum boundary for the activation group default heap is 16 bytes.
- A *heap_id* of 0 specifies the default heap in the activation group.
- Individual allocations within a heap may not be contiguous.
- Each allocation associated with a heap provides a sequence of contiguously addressable bytes. The bytes do not cross a 64KB boundary unless the allocation size is greater than 64KB.

API Introduced: V2R3

Reallocate Storage (CEECZST) API

Required Parameter Group:

1	address	I/O	POINTER
2	new_size	Input	INT4

Omissible Parameter:

3 fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Reallocate Storage (CEECZST) API changes the size of a previously allocated storage block, preserving the contents.

Required Parameter Group

address (I/O)

On input, this parameter contains an address returned by a previous CEEGTST call or a language intrinsic function. On output, the address of the first byte of the newly allocated storage is returned in this parameter.

In effect, reallocation of a storage block is accomplished by allocating a new storage block, of size *new_size*, and copying the contents of the old block to the new block.

new_size (input)

The number of bytes of storage to be allocated for the new storage block. This value is rounded up to a multiple of the minimum boundary specified when the heap was created. The minimum boundary for the activation group default heap is 16 bytes.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

A message severity of 10 or less represents success. If the severity is greater than 10:

- No storage is allocated.
- The previous allocation remains intact.
- The value in *address* remains unchanged.

CEE0000	The API completed successfully
Severity: 00	
CEE0802	The storage headers are damaged
Severity: 40	
CEE0808	Requested storage size is not valid
Severity: 30	
CEE0810	The starting address for reallocation is not valid
Severity: 30	
CEE0813	Insufficient storage available to satisfy the request
Severity: 30	

Usage Notes

- The heap identifier is inferred from the address. The new storage block is allocated from the same heap that contained the old block. The reallocate operation may be issued from an activation group other than the one that owns the heap.
- The contents of the old storage block are preserved in the following way:
    - If *new_size* is greater than the old size, the entire contents of the old storage block are copied to the new block.
    - If *new_size* is less than the old size, the contents of the old block are truncated to the size of the new block.
    - If *new_size* is equal to the old size, the contents of the old storage block are copied to the new block.

  Note: Because the new storage may be allocated at a different location than the existing allocation, any pointers that referred to the old storage are no longer valid. Continued use of such pointers will give unpredictable, or incorrect results.
- Storage that is reallocated maintains the same mark and release status as the old storage block. If the old storage block was marked, the new storage block carries the same mark and is released by a release operation that specifies the mark.

API introduced: V2R3

Extended Heap Operations APIs

The extended heap operations APIs are:

- Create Heap (CEECRHP) creates a new heap.
- Discard Heap (CEEDSHP) discards an existing heap.
- Mark Heap (CEEMKHP) returns a token that can be used to identify heap storage to be freed by the CEERLHP API.
- Release Heap (CEERLHP) frees all storage allocated in the heap since the mark was specified.

Create Heap (CEECRHP) API

Required Parameter:			
1	heap_id	Output	INT4
Omissible Parameter Group:			
2	initial_size	Input	INT4
3	increment	Input	INT4
4	alloc_strat_id	Input	INT4
5	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Create Heap (CEECRHP) API creates a new heap.

Required Parameter

heap_id (output)

The heap identifier of the created heap.

Omissible Parameter Group

initial_size (input)

The initial amount of storage, in bytes, allocated for the new heap. If this parameter is 0 or omitted, and there is no corresponding value in the user-defined allocation strategy, the system computes a default value. The minimum value that can be specified is 512 bytes. If values between 1 and 512 bytes are specified, the system rounds the value up to 512 bytes automatically. The maximum value that can be specified is 16MB minus 1KB. The value specified is rounded up to a 512-byte boundary.

Note: If system resources are constrained, the system may override the value specified.

increment (input)

The number of bytes by which the heap is extended when it is necessary to enlarge the heap to satisfy an allocation request. If this parameter is 0 or omitted, and there is no corresponding value in the user-defined allocation strategy, the system computes a default value. The minimum value that can be specified is 512 bytes. If values between 1 and 512 bytes are specified, the system rounds the value up to 512 bytes automatically. The maximum value that can be specified is 16MB minus 1KB. The value specified is rounded up to a 512-byte boundary.

Note: If system resources are constrained, the system may override the value specified.

alloc_strat_ID (input)

The allocation strategy used. ILE allows allocation strategy values of 0, 1, and 40 through 44 from the possible range of values 0 through 99, where:

0	Specifies use of an allocation strategy that is the optimal default for the system.
1	Same as 0.
40-44	User-defined allocation strategies. See Define Heap Allocation Strategy (CEE4DAS) API for information on defining allocation strategies.

For allocation strategy values outside of the values specified above:

2-39	The values are reserved for other systems. If they are specified, message CEE0814 is issued.
45-49	The values are reserved for other systems. If they are specified, message CEE0815 is issued.
50-99	The values are not supported by ILE. If they are specified, message CEE0806 is issued.

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

Severities of 10 or less represent success. For higher severities, no heap is created, and all other output values are undefined.

CEE0000	The API completed successfully
Severity: 00	
CEE0804	The initial_size parameter is not valid
Severity: 30	
CEE0805	The increment parameter is not valid
Severity: 30	
CEE0806	Allocation strategy identifier value is not valid
Severity: 30	
CEE0809	The maximum number of heaps supported by ILE has been reached
Severity: 30	
CEE0813	Insufficient storage available to satisfy the request
Severity: 30	
CEE0814	Allocation strategy identifier is not supported by ILE
Severity: 30	

CEE0815	Allocation strategy identifier is not supported by ILE
Severity: 30	
CEE3006	At least one field in the allocation strategy is not valid
Severity: 30	

Usage Notes

- If a user-defined allocation strategy value is specified for which the CEE4DAS API was not called, the default user-created heap strategy is used. See User-Defined Allocation Strategy.
- The *initial_size* and *increment* values, if specified, will override any default or user-created heap strategy values.
- If one of the fields in the allocation strategy CEE4DAS is not valid, the heap is not created and an error condition is raised. This API checks the allocation strategy.

API introduced: V2R3

Discard Heap (CEEDSHP) API

Required Parameter:			
1	heap_id	Input	INT4
Omissible Parameter:			
2	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Discard Heap (CEEDSHP) API deletes an existing heap.

Required Parameter

heap_id (input)
The heap identifier of the heap to be discarded.

Omissible Parameter

fc (output)
A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0803	The heap identifier does not match any existing heap
Severity: 30	
CEE0812	The basic initial heap cannot be marked and discarded
Severity: 30	
CEE0850	The heap cannot be marked
Severity: 30	

Usage Notes

- A *heap_id* of 0 is not valid. This is the heap identifier of the default heap; it cannot be discarded.
- After this call, there may still be pointers to storage that had been allocated from this heap. Their use can cause unpredictable or erroneous results.

API introduced: V2R3

Mark Heap (CEEMKHP) API

Required Parameter Group:			
1	heap_id	Input	INT4
2	mark	Output	POINTER
Omissible Parameter:			
3	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe:			

The Mark Heap (CEEMKHP) API returns a token, which can be used with the Release Heap (CEERLHP) API to free a heap storage group. After a CEEMKHP call, all storage subsequently obtained by the CEEGTST API can be freed by a single CEERLHP call. A heap may have multiple marks.

Required Parameter Group

heap_id (input)

A heap identifier token specifying the heap to be marked.

mark (output)

The mark token for use in a subsequent CEERLHP call.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0803	The heap identifier does not match any existing heap
Severity: 30	

CEE0812	The basic initial heap cannot be marked and discarded
Severity: 30	

Usage Notes

- The default heap in the activation group cannot be marked.
- Multiple marks are maintained for each heap.

  When a release operation is performed, the specified mark and all subsequent marks are removed from the list of active marks.
- A mark should only be used in release operations. It should not be used as a pointer.

API Introduced: V2R3

Release Heap (CEERLHP) API

Required Parameter:			
1	mark	Input	POINTER
Omissible Parameter:			
2	fc	Output	FEEDBACK
Service Program Name: QILE			
Default Public Authority: *USE			
Threadsafe: Yes			

The Release Heap (CEERLHP) API frees all storage allocated since the specified mark in the heap.

Required Parameter

mark (input)

A mark returned by a previous CEEMKHP call. All storage allocated in the marked heap since the corresponding mark operation is released. Marks obtained after the specified mark are also discarded.

Omissible Parameter

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0807	The mark does not match any existing mark
Severity: 30	

Usage Notes

- No heap identifier is required for this call. The heap identifier of the heap to be released is inferred from the mark address. The release operation may be issued from an activation group other than the activation group that owns the heap.
- Multiple marks can be maintained for each heap. A release operation deallocates all storage allocated since the specified mark operation.

  Mark and release operations treat the heap in a fashion similar to a stack. There must be a mark operation outstanding at the location used in a release operation. A release operation can reset the heap to any mark; a release operation can clear one or several mark operations.
- Pointers obtained by CEEGTST for storage that is freed by the release operation are no longer valid after this call. Continued use of these pointers will give unpredictable or incorrect results.

API Introduced: V2R3

Heap allocation strategies APIs

The heap allocation strategies API is:

- Define Heap Allocation Strategy (CEE4DAS) defines an allocation strategy that determines the attributes for a heap created with the CEECRHP API.

Define Heap Allocation Strategy (CEE4DAS) API

Required Parameter Group:

1	alloc_strat_id	Input	INT4
2	alloc_strat_in	Input	CEE4ALC

Omissible Parameter Group:

3	alloc_strat_out	Output	CEE4ALC
4	fc	Output	FEEDBACK

Service Program Name: QILE

Default Public Authority: *USE

Threadsafe: Yes

The Define Heap Allocation Strategy (CEE4DAS) API defines a system-specific allocation strategy and associates the defined strategy with a specified allocation strategy identifier. When creating a heap using the CEECRHP API, the allocation strategy identifier can be used within the activation group in which it was defined.

Required Parameter Group

alloc_strat_ID (input)

The allocation strategy identifier being defined. The valid values are 40 through 44.

alloc_strat_in (input)

A structure of allocation strategy type CEE4ALC that defines the new allocation strategy to be assigned to the specified allocation strategy identifier. See Allocation Strategy Type (CEE4ALC) for a description.

Omissible Parameter Group

alloc_strat_out (output)

An optional structure of type CEE4ALC that will be set to the value of the previous allocation strategy that was assigned to the specified allocation strategy identifier. If no previous allocation strategy was assigned, the default allocation strategy is returned. See User-Defined Allocation Strategy for more information.

fc (output)

A 12-byte feedback code.

Feedback Codes and Conditions

CEE0000	The API completed successfully
Severity: 00	
CEE0816	The allocation strategy identifier is out of range
Severity: 30	

Usage Notes

The CEE4DAS API does not perform a validity check on the contents of *alloc_strat_in*. When the defined allocation strategy is used, checking is enforced by the CEECRHP API.

API introduced: V2R3
