Documentation Updates for APAR PH52822

Preface

This document describes the IBM publication changes made to z/OS Language Environment by APAR PH52822.
The text marked in yellow are the newly added sections.

z/OS Language Environment Vendor Interfaces

Part 2. Language Environment vendor interfaces for AMODE 64 applications

Chapter 32. Member language information for AMODE 64 applications

Event handler calls

Event code 25 — static object constructor event

Purpose

Constructors and destructors are not resources, but are routines that are executed during the creation and destruction of application variables. Application variables can be static, automatic, or dynamic. Automatic variables are thread resources. The invocation of constructors and destructors for those variables is performed by each thread. Static variables and dynamic variables are enclave resources, so the constructors and destructors are executed once for the creation/destruction of each static and dynamic variable in the enclave.

Constructors and destructors for automatic and dynamic variables are driven by the languages libraries or compiled code, without specific support from Language Environment. Constructors and destructors for static variables are driven by language libraries from within the static constructor and static destructor events.

The static object constructor lets a member gain control to perform constructor initialization prior to the invocation of the main routine. CEECONST is a CWI, called by member languages from their the enclave initialization event logic, to register the member to gain control, by the member event handler, for two events:

1. Static constructor event (Event Code 25)

2. Static destructor event (Event Code 36)

By requiring member languages to register for these events, the overhead of the event calls is avoided for member languages that do not need the event.

Syntax

```
void CEECONST (fc)
 FEED_BACK *fc;
```

***fc* (output)**
A 12-byte feedback code that indicates the result of this service. This parameter must be specified. The following symbolic conditions can result from this service:

Condition		
CEE000	Severity	0
	Msg_No	N/A
	Message	The service completed successfully.
CEE38U	Severity	4

	Msg_No	3358
	Message	The service was invoked outside of the member enclave initialization. No action was taken.

The Static Constructor Event is designed to be used by languages with object oriented features to drive constructor functions (initialization methods) for all statically allocated objects. The event is driven during Language Environment enclave initialization, after the debugger initialization events but prior to the invocation of the main routine. The event is also driven by Language Environment whenever a new module has been loaded, immediately following the invocation of the DLL Initialization Event (22).

Call CEEEVnnn (25, *idinfo*, *loadinfo*)

```
INT4    *idinfo;
INT4    *loadinfo;
```

***idinfo* (input)**
A fullword that indicates to the member language additional information identifying the calling environment. A new executable unit (load module or program object) is introduced into the enclave by COBOL dynamic call, PL/1 or C fetch, CEEPIPI services, or DLL implicit or explicit load. The following bits are defined:

0 - 23

reserved

24 - 31

The value indicates the load_reason. The values are defined as follows:

0

The load was due to main Language Environment initialization.

1

The load was due to dynamic call, fetch, or ceepipi service. In this case, static constructors are run immediately.

2

The load was due to the explicit or implicit reference of a DLL. Static constructors will only be run if they are at the level represented by the initial DLL load (for example, all DLL initialization has been completed).

***loadinfo* (input)**
A fullword returned from the New Load Module (8) or DLL Initialization (22) event containing information about the module that was just loaded.

A return code is placed in R15 by the Event Handler. The following return codes (in decimal) are defined:

-4

The Event Handler does not want to process the event.

0

The Event Handler was successful.

16

The Event Handler encountered an unrecoverable error.

Usage notes

• This event is driven only if the member language has registered for static constructor/destructor events by calling the CEECONST CWI during the enclave initialization event.

• All services of Language Environment are available during this event.

• Application code may be driven during this event.

• In 64bit C/C++ module, LE get the function call's address of constructors and destructors from class C_@@SQINIT. The C_@@SQINIT class consists of records, with each record being used to invoke either a constructor or destructor or both.

the lowest bit of the function call's address in class C_@@SQINIT is used to indicates the address type to find constructors and destructors.

if the lowest bit is 1, then the address of function call is an absolute address content in the initial loaded class.

if the lowest bit is 0, then the address of function call is an offset within C_WSA64 class.

Event code 36 — static destructor event

Purpose

This event is designed to be used by languages with object oriented features to drive destructor functions (uninitialization methods) for all statically allocated objects. This event is driven during Language Environment enclave termination, after stack collapse, but prior to debugger termination events. This event occurs after the atterm event.

Syntax

```
void CEEEVnnn (event_code)
INT4 *event_code = 36;
```

***event_code* (input)**
Is a fullword integer with value 36 indicating that this is the static destructor event call.

Usage notes

• This event is driven only if the member language has registered for static constructor/destructor events by calling the CEECONST CWI during the enclave initialization event.

• All services of Language Environment are available during this event.

• Application code may be driven during this event.

• In 64bit C/C++ module, LE get the function call's address of constructors and destructors from class C_@@SQINIT. The C_@@SQINIT class consists of records, with each record being used to invoke either a constructor or destructor or both.

the lowest bit of the function call's address in class C_@@SQINIT is used to indicates the address type to find constructors and destructors.

if the lowest bit is 1, then the address of function call is an absolute address content in the initial loaded class.

if the lowest bit is 0, then the address of function call is an offset within C_WSA64 class.