#SHAREorg

SHARE

IEWBIND and IEWBFDAT – Learning to Use the Binder APIs Hands-on Lab Handout

Barry_Lichtenstein@us.ibm.com

February 2013
Session# 12929

SHARE in San Francisco 2013

STARTD: Start dialog

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

STARTD begins a dialog with the binder, establishing the processing environment and initializing the necessary control blocks. You specify the ddnames for the data sets to be accessed, how errors are to be handled, and the global binder options.

STARTD returns a dialog token that is included with later calls for the same dialog.

The syntax of the STARTD call is:

```
[symbol]    IEWBIND    FUNC=STARTD
                       [,VERSION=version]
                       [,RETCODE=retcode]
                       [,RSNCODE=rsncode]
                       ,DIALOG=dialog
                       [,FILES=filelist]
                       [,EXITS=exitlist]
                       [,OPTIONS=optionlist]
                       [,PARMS=parms]
                       [,ENVARS=envars]
```

FUNC=STARTD
Specifies that a dialog is opened and initialized.

VERSION=*1* | *2* | *3* | *4* | *5* | *6* | *7*
Specifies the version of the parameter list to be used. The default value is VERSION=1.
Notes:
1. If VERSION=1 is specified for the STARTD call, PARMS cannot be specified as a macro keyword. The parameter list ends with the OPTLIST parameter (with the high-order bit set). This exception is for Version 1 only.
2. ENVARS cannot be specified if VERSION is less then 6.

RETCODE=*retcode* — RX-type address or register (2-12)
Specifies the location of a fullword integer that is to receive the return code returned by the binder.

RSNCODE=*rsncode* — RX-type address or register (2-12)
Specifies the location of a 4-byte hexadecimal string that is to receive the reason code returned by the binder.

DIALOG=*dialog* — RX-type address or register (2-12)
Specifies the location of an 8-byte area where the binder places the dialog token. This token must not be modified.

FILES=*filelist* — RX-type address or register (2-12)
Specifies the address of a list containing one entry for each binder file for which a ddname or file name is provided. You code some or all of these file names in the list and provide a ddname or filename for each:

File name
Description
CALLIB
Automatic call library
MODLIB
Target program library
PRINT
Listing data set for messages produced by the LIST, MAP, and XREF options
TERM
Terminal data set for messages issued during binder processing
SIDEFILE
Data set to contain the side file of a DLL module.

The *ddnames* specified for PRINT and TERM can designate z/OS UNIX System Services files. CALLIB can designate a z/OS UNIX System Services directory and a z/OS UNIX System Services archive file. MODLIB can designate a z/OS UNIX System Services directory. SIDEFILE can designate a z/OS UNIX System Services directory or a z/OS UNIX System Services file.

Entries for all files on this list will accept a z/OS UNIX pathname in place of a ddname. Path names must begin with a slash (/) or a period and a slash (./).

EXITS=*exitlist* — RX-type address or register (2-12)
Specifies the address of a list of user exit names. You can specify the following user exits in this list:

MESSAGE exit
Specifies an entry into the calling program that receives control immediately prior to the binder issuing a message. See Message exit.
SAVE exit
Specifies an entry into the calling program that receives control if the binder is about to reject a primary name or an alias name or after an attempt to save a member or alias name. The save operation might have succeeded or failed. See Save exit.
Note:
This exit is not invoked if the target is a z/OS UNIX System Services file.
INTFVAL exit
The Interface Validation Exit (INTFVAL) allows your exit routine to examine descriptive data for both caller and called at each external reference. The exit can perform audits such as examining parameter passing conventions, the number of parameters, data types, and environments. It can accept the interface, rename the reference, or leave the interface unresolved. See Interface validation exit.

See User exits for additional information on writing user exit routines.

OPTIONS=*optionlist* — RX-type address or register (2-12)
Specifies the location of an options list that contains the address of binder options to be initialized during the STARTD call. Any option that can be set by SETO can be initialized by STARTD. See Setting options with the regular binder API for a list of allowable options.

However, the EXITS option listed in the table may not be specified as part of the OPTIONS parameter; use the EXITS or PARMS parameter on STARTD to specify user exits.

Note:

The negative option format (for example, NORENT) is not allowed. Use the corresponding keyword with a value. For example:

```
DC   CL8'REUS'
DC   AL4 (6) ,C'SERIAL'
```

PARMS=*parms* — RX-type address or register (2-12)

Specifies the location of a varying character string that contains a list of option specifications separated by commas. The STARTD FILES, EXITS and OPTIONS lists have precedence over the STARTD PARMS string in the STARTD call. See Setting options with the binder API for more information.

ENVARS=*envars* — RX-type address or register

Specifies the address of a list of 31-bit pointers. Each pointer in the list contains the address of a character string that is an environment variable, in the form of *name=value*, to be passed to the specified program. Each character string must end with zeros. Note that this is the same as passing the external variable, *environ*. See Environment variables for more information.

Passing lists to the binder

Any list passed to the binder must conform to a standard format, consisting of a fullword count of the number of entries followed by the entries. Each list entry consists of an 8-byte name, a fullword containing the length of the value string, and a 31-bit pointer to the value string. The list specification is provided in Table 28.

Table 28. Binder list structure

Field Name	Field Type	Offset	Length	Description
LIST_COUNT	Integer	0	4	Number of 16-byte entries in the list
LIST_ENTRY	Structure	4,20,...	16	Defines one list entry
ENTRY_NAME	Character	0	8	File, exit, or option name
ENTRY_LENGTH	Integer	8	4	Length of the value string
ENTRY_ADDRESS	Pointer	12	4	Address of the value string

Note:

ENTRY_NAME, ENTRY_LENGTH, and ENTRY_ADDRESS are repeated for each entry in the list up to the number specified in LIST_COUNT.

You code the data pointed to by ENTRY_ADDRESS according to the list type:

File list

Code one entry for each file name:

ENTRY_NAME:

`'CALLIB  '`, `'MODLIB  '`, and so on. See list of file names in FILES parameter description in STARTD: Start dialog.

ENTRY_LENGTH:

The byte length of the corresponding ddname.

ENTRY_ADDRESS:

The address of the string containing the ddname.

Each file name specified in the FILES parameter of the STARTDialog API must correspond to a currently defined ddname. Your data sets can be new or preallocated. Although you can use any valid ddname for a given FILE name, the following ddnames are recommended. Their allocation requirements are listed below:

FILE name
Recommended ddname

CALLIB
SYSLIB

MODLIB
SYSLMOD

PRINT
SYSPRINT

TERM
SYSTERM

SIDEFILE
SYSDEFSD

The SYSLIB Data Set (the CALLIB file)

This DD statement describes the automatic call library, which must reside on a direct access storage device. This is a required data set if you want to enable autocall processing while you bind your modules through the use of the BINDWorkmod API call (See the CALLIB parameter in the BINDW API).

The data set must be a library and the DD statement must not specify a member name. You can concatenate any combination of object module libraries and program libraries for the call library. If object module libraries are used, the call library can also contain any control statements other than INCLUDE, LIBRARY, and NAME. If this DD statement specifies a PATH parameter, it must specify a directory.

Table 29 shows the the SYSLIB data set attributes, which vary depending on the input data type.

Table 29. SYSLIB data set DCB parameters

LRECL	BLKSIZE	RECFM
80	80	F, FS, OBJ, XOBJ, control statements, and GOFF
80	32720 (maximum size)	FB, FBS OBJ, XOBJ, control statements, and GOFF

LRECL	BLKSIZE	RECFM
84+	32720 (maximum size)	V, VB, GOFF object modules
n/a	32720 (maximum size)	U, load modules
n/a	4096	U, program objects

The SYSLMOD data set (the MODLIB file)

It is the target library for your SAVEWorkmod API calls when ACCESS=BIND on your CREATEWorkmod API call. That is, SYSLMOD is the library that contains your bound modules. As such, it must be a partitioned data set, a PDSE, or a z/OS UNIX System Services file.

Although a member name can be specified on the SYSLMOD DD statement, it is used only if a name is not specified on the SAVEWorkmod SNAME parameter. (See SAVEW: Save workmod.) Therefore, a member name should not be specified if you expect to save more that one member in a binder dialog. For additional information on allocation requirements for SYSLMOD, see SYSLMOD DD statement in z/OS MVS Program Management: User's Guide and Reference.

The SYSPRINT data set (the print file)

The binder prints diagnostic messages to this data set. The binder uses a logical record length of 121 and a record format of FBA and allows the system to determine an appropriate block size.

Table 30 shows the data set requirements for SYSPRINT.

Table 30. SYSPRINT DCB parameters

LRECL	BLKSIZE	RECFM
121	121	FA
121	32670 (maximum size)	FBA
125		VA or VBA

The SYSTERM data set (the TERM file)

SYSTERM defines a data set for error and warning messages that supplements the SYSPRINT data set. It is always optional. SYSTERM output consists of messages that are written to both the SYSTERM and SYSPRINT data sets, and it is used mainly for diagnostic purposes.

Table 31 shows the data set requirements for SYSTERM.

Table 31. SYSTERM DCB parameters

LRECL	BLKSIZE	RECFM
80	32720 (maximum size)	FB

The SYSDEFSD data set (the SIDEFILE file)

When a module (call it module A) is enabled for dynamic linking through the DYNAM(DLL) binder option, a complementary file can be generated to go along with it. Module A becomes a DLL, and the complementary file becomes its *side file*. The side file is saved in the data set represented by the SYSDEFSD ddname. The side file contains the external symbols of DLL A, known as *exports*. These external symbols can be referenced by other DLLs and are known as *imports* to these modules. If module A does not export any symbols, no side file is generated for it. This applies to any DLL.

SYSDEFSD can be a sequential data set, a PDS, a PDSE, or a z/OS UNIX System Services file. If it is a sequential data set, the generated side files for multiple DLLs are appended one after another, provided that the DISP=MOD parameter is supplied in the SYSDEFSD ddname specification. If SYSDEFSD is a PDS or a PDSE, the side file is saved as a member with the same name as the DLL to which it belongs. Refer to the processing notes of the SAVEW API for additional information.

The SYSDEFSD DD statement is optional. However, when the ddname is absent, the binder issues a warning message if at bind time a program generates export records and the DYNAM(DLL) binder option has been specified.

Exit list

ENTRY_NAME:
'MESSAGE ', 'INTFVAL ', or 'SAVE '.
ENTRY_LENGTH:
12.
ENTRY_ADDRESS:
The location of a three-word area containing three addresses. See the message list addresses in Message exit, the save list addresses in Save exit and the interface validation list addresses in Interface validation exit for the contents of the three addresses.

Option list

ENTRY_NAME:
The option keyword (for example, 'LIST ', 'MAP ', 'CALLERID'). Option keywords cannot be truncated and negative options cannot be specified (NOLIST, NOPRINT, and so on). If INTENT=ACCESS, these keywords are not allowed: ALIGN2, CALL, CALLIB, EDIT, LET, MAP, OVLY, RES, TEST, XCAL, and XREF.
ENTRY_LENGTH:
The length of the option value as a character string; it can be zero.
ENTRY_ADDRESS:

The address of the character string encoded with the option's value. The address can be zero. The maximum length of the option value string is 256 bytes. Use commas and parentheses if a sublist is required.

Return and reason codes

The common binder API reason codes are shown in Table 3.

Return Code	Reason Code	Explanation
00	00000000	Normal completion.
04	83000112	The binder encountered an unsupported option in the OPTIONS file. The option is ignored.
04	83000204	The binder was unable to open the trace data set during initialization. Processing continues without trace.
08	83000108	An option value is missing or contains an invalid setting.
08	83000111	An OPTIONS option was encountered in the options file. The option is ignored.
08	83000200	The binder was unable to open the PRINT data set during initialization. Processing continues without PRINT.
08	83000201	One or more invalid options were passed on STARTD. Those options were not set, but processing continues.
12	83000203	The binder was unable to open the TERM data set during initialization. Processing stops.
08	83000205	The current time was not available from the operating system. Time and date information in printed listings and IDR records will be incorrect.
08	83000206	The binder was unable to open the SYSTERM data set because its DDNAME was not specified in the FILES parameter of STARTD. Processing continues without SYSTERM.
12	83000207	The binder was unable to open the SYSPRINT data set because its DDNAME was not specified in the FILES parameter of STARTD. Processing continues without SYSPRINT.

Environment variables

When the binder API is used by a program running in the z/OS UNIX subsystem, the environ parameter may be used to pass the C/C++ runtime variable of that name to the binder, in order to give the binder access to the array of environment variables. If a user sets binder environment variables (those documented below) in the UNIX shell, this is the only way the binder can get access to them. For further information about the C runtime 'environ' variable, refer to z/OS XL C/C++ Run-Time Library Reference.

Although it is not recommended, you may also construct your own 'envars' parameter using the same format as that of the C runtime variable. In this case, 'envars' must be the address of an array of pointers. Each pointer is the address of a null-terminated string representing an individual environment variable in the form 'keyword=*value*'. The last array entry must be a null pointer.

The following UNIX shell environment variables are recognized by the binder. They are specified in the form:

```
export  NAME=value
```

where NAME is the name of the environment variable.

IEWBIND_PRINT
the pathname or ddname to be used for SYSPRINT.
IEWBIND_TERM
the pathname or ddname to be used for SYSTERM.
IEWBIND_OPTIONS
the binder option string. These will be appended to options passed explicitly through the STARTD API call (and will take precedence). There are additional environment variables defined for the binder diagnostic data sets. See the "Binder Serviceability Aids" chapter inz/OS MVS Program Management: User's Guide and Reference for more information on passing them on STARTD.

Parameter list

If your program does not use the IEWBIND macro, place the address of the STARTD parameter list in general purpose register 1.

Table 32. STARTD parameter list

```
PARMLIST DS   0F
         DC   A(STARTD)          Function code
         DC   A(RETCODE)         Return code
         DC   A(RSNCODE)         Reason code
         DC   A(DIALOG)          Dialog token
         DC   A(FILELIST)        File list
         DC   A(EXITLIST)        Exit list
         DC   A(OPTLIST)         Option list
         DC   A(PARMSTR)         Parameters
         DC   A(ENVARS)          Environment Variables
STARTD   DC   H'01'              STARTD function code
         DC   H'version'         Interface version number
```

Note:
X'80000000' must be added to the last parameter. For version 1, that will be OPTLIST. For other versions it may be either PARMSTR or (beginning with version 6) ENVARS.

CREATEW: Create workmod

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

CREATEW initializes a workmod and initializes the module options to the defaults for the dialog. CREATEW also specifies the processing intent that determines the functions that can be performed on the workmod.

The syntax of the CREATEW call is:

```
[symbol]    IEWBIND    FUNC=CREATEW
                       [,VERSION=version]
                       [,RETCODE=retcode]
                       [,RSNCODE=rsncode]
                       ,WORKMOD=workmod
                       ,DIALOG=dialog
                       ,INTENT={BIND | ACCESS}
```

FUNC=CREATEW
Specifies that a workmod is created and initialized.

VERSION=*1 | 2 | 3 | 4 | 5 | 6 | 7*
Specifies the version of the parameter list to be used. The default value is VERSION=1.

RETCODE=*retcode* — RX-type address or register (2-12)
Specifies the location of a fullword integer that is to receive the return code returned by the binder.

RSNCODE=*rsncode* — RX-type address or register (2-12)
Specifies the location of a 4-byte hexadecimal string that is to receive the reason code returned by the binder.

WORKMOD=*workmod* — RX-type address or register (2-12)
Specifies the location of an 8-byte area that is to receive the workmod token for this request.

DIALOG=*dialog* — RX-type address or register (2-12)
Specifies the location of an 8-byte area that contains a dialog token for which the workmod is requested. The dialog token is obtained using the STARTD call and must not be modified.

INTENT={BIND | ACCESS}
Specifies the range of binder services that can be requested for this workmod. The values are as follows:

BIND
Specifies that the processing intent for this workmod is bind. The workmod will be bound and all binder functions can be requested.

ACCESS
Specifies that the processing intent for this workmod is access. The workmod will not be bound, and no services that alter the size or structure of the program module can be requested. See Processing intents for a list of services that are not allowable.

The value for INTENT can be abbreviated as **B** or **A**.

Processing notes

None.

Return and reason codes

The common binder API reason codes are shown in Table 3.

Return Code	Reason Code	Explanation
00	00000000	Normal completion. Workmod and workmod token created.

Parameter list

If your program does not use the IEWBIND macro, place the address of the CREATEW parameter list in general purpose register 1.

Table 9. CREATEW parameter list

```
PARMLIST DS   0F
         DC   A(CREATEW)            Function code
         DC   A(RETCODE)            Return code
         DC   A(RSNCODE)            Reason code
         DC   A(DIALOG)             Dialog token
         DC   A(WORKMOD)            Workmod token
         DC   A                     Processing intent and end-of-list indicator
              (INTENT+X'80000000')
CREATEW  DC   H'10'                 CREATEW function code
         DC   H'version'            Interface version number
INTENT   DC   CL1'A'                Processing intent
                                      'A' = Access
                                      'B' = Bind
```

__iew_openW() - Open workmod

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

The __iew_openW() creates a context(_IEWAPIContext) and initializes all of the parameters in the context and loads IEWBIND into memory. If successful, the context is created and returned to user for all subsequent API calls.

This is a 'C' function equivalent to binder API STARTD and CREATEW.

Format

```
#include <__iew_api.h>
_IEWAPIContext *__iew_openW(_IEWTargetRelease __release,
                                          _IEWIntent __intent,
                                          _IEWList *__file_list,
                                          _IEWList *__exit_list,
                                          const char *__parms,
                                          unsigned int *__return_code,
                                          unsigned int *__reason_code);
```

Parameters Descriptions

__release
target release can be one of the following:

- _IEW_ZOSV1R9
- _IEW_ZOSV1R10
- _IEW_ZOSV1R11
- _IEW_ZOSV1R12
- _IEW_ZOSV1R13

Note:
It is recommended that you define the feature test macro _IEW_TARGET_RELEASE prior to the #include <__iew_api.h>, and that _IEW_TARGET_RELEASE be used for __release. This definition ensures that structure mappings in <__iew_api.h> are consistent with the data returned by the other API access functions.

__intent
processing intent can be one of the following:

- _IEW_ACCESS
- _IEW_BIND

Note:
_IEW_ACCESS is more efficient but only allows the program to be inspected or copied. This can be changed later by __iew_resetW().

__file_list
file list is created by __iew_create_list().

__exit_list
exit list is created by __iew_create_list().

__parms
parameters - list of binder option.

__return_code
return code is passed back from STARTD or CREATEW.

__reason_code
reason code is passed back from STARTD or CREATEW

Returned Value

If successful, __iew_openW() returns API context.

If unsuccessful, __iew_openW() returns NULL.

Note:
The returned value is the same as the code returned by a subsequent __iew_get_return_code().

Utilities Functions

__iew_create_list()

__iew_fd_startName() - Starting a session with a DD name or path

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Begin inspecting a program object identified either by a path name or by a DDname and member name.

Format

```
#include <__iew_api.h>
int __iew_fd_startName(_IEWFDContext *__context,
                                        const char *__dd_or_path,
                                        const char *__member);
```

Parameters Descriptions

__context
FD context is created and returned by calling __iew_fd_open() and is used throughout the open session.
__dd_or_path
DD name or path name.
__member
member name.

Returned Value

If successful, __iew_fd_startName() returns 0.

If unsuccessful, __iew_fd_startName() returns nonzero.

Note:
The returned value is the same as the code returned by a subsequent __iew_fd_get_return_code ().

Utilities Functions

__iew_fd_get_reason_code()
__iew_fd_get_return_code()

SJ - Starting a session with a DD name or path

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

The calling application identifies the program object either directly by a UNIX path name or indirectly by the name of a DD statement which identifies either a UNIX path or one or more partitioned data sets. The application may also provide a member name.

Table 36. SJ parameter list

Parameter	Usage	Format	Content
1	in	structure	'SJ', X'0001'
2	in/out	binary word	mtoken Must contain zero when the service is called. The service will supply a value if the service is successful.
3	in	vstring	DD name or path If the string data begins with a slash or a period followed by a slash it is treated as a path, otherwise as a DD name.
4 (optional)	in	vstring	Member name or path extension If parameter 3 is a DD name defining a PDSE or concatenation this is a member name and is required. If parameter 3 is either a path or a DD name defining a path, this parameter is optional and is appended to the path name if present.

Sample assembler code

```
         CALL  (15),(SJIL,MTOKEN,UNIXDD),VL
*        Note: 4th optional parameter omitted here

SJIL     DC    C'SJ',X'0001'
MTOKEN   DC    F'0'          Replaced by the service
UNIXDD   DC    H'5','UPATH'  Name of a DD statement,
*                            which in turn has PATH.
```

Binder name list (version 7)

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Please note that the version 7 binder name list buffer has been slightly changed from the version 6 buffer.

Figure 49. Format for binder name list entries

	Field Name	Field Type	Off set	Leng	Description
	IEWBBNL				Binder Name List buffer, Version 7
	BNLH_BUFFER_ID	Char	0	8	Buffer identifier "IEWBBNL"
	BNLH_BUFFER_LENG	Binary	8	4	Length of the buffer, including the header
	BNLH_VERSION	Binary	12	1	Version identifier
	*** RESERVED ***	Binary	13	3	Reserved, must be zeros
	BNLH_ENTRY_LENG	Binary	16	4	Length of each entry in the list
	BNLH_ENTRY_COUNT	Binary	20	4	Number of entries in the buffer
	*** RESERVED ***	Binary	24	8	Reserved, initialize to zeros
	BNLH_ENTRY_ORIGIN		32		First namelist entry
1	BNL_ENTRY				Namelist Entry
2	BNL_CLS_LENGTH	Binary	0	4	Class segment length (NTYPE=CLASS only)
3	BNL_SECT_CU	Binary	0	4	Compile unit number (NTYPE=SECTION only)
	BNL_BIND_FLAGS	Bit	4	1	Bind Attributes(NTYPE=CLASS only)
	1...				Generated by Binder
	.1..				No data present
	..1.				Varying length records
	...1				Descriptive data (non-text)
	.... 1...				Class has initial data
	.... .1..				Fill character specified
	1				Class validation error
	BNL_LOAD_FLAGS	Bit	5	1	Loadability (NTYPE=CLASS only)
	1...				Read Only
	.xx.				Time of load
	.00.				Class is initially loaded
	.01.				A DEFER load class
	.10.				A NOLOAD class
	...1				Sharable
	.... 1...				Moveable (AdCon free)
	xx				Binding Method
	00				CAT(catenated text)
	01				MERGE(merged parts)
	BNL_NAME_CHARS	Binary	6	2	Length of name
	BNL_NAME_PTR	Pointer	8	4	Pointer to class or section name
	BNL_ELEM_COUNT	Binary	12	4	Number of elements in class or section
	BNL_SEGM_ID	Binary	16	2	Segment ID (NTYPE = CLASS only)
	BNL_ATTR	Bit	18	1	
	BNL_ALIGN	Bit		5	Alignment (NTYPE = CLASS only)
	00011				Doubleword
	00100				Quadword
	01100				Page (4K)
	BNL_RMODE	Bit		3	Residence mode (NTYPE = CLASS only)
	001				Rmode 24
	011				Rmode ANY
	100				Rmode 64
	BNL_NAMESPACE	Binary	19	1	Namespace (NTYPE = CLASS only)
	x'01' = catenate class				
	x'02' = pseudoregisters (merge class)				
	x'03' = parts (merge class)				
	BNL_SEGM_OFF	Binary	20	4	Class offset from start of segment (NTYPE=CLASS only)

Notes:

1. This entry is valid for output only.
2. BNL_SECT_CU is at the same offset as BNL_CLS_LENGTH.

GETN: Get names

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

GETN returns the names of each section or class in the workmod, a count of the total number of sections or classes, and the compile unit (CU) numbers for each section. The names returned also include names generated by the binder to represent private code sections, unnamed common, SEGTAB and ENTAB sections for overlay programs, and any other sections created by the binder. GETN can only be performed on a bound workmod.

The syntax of the GETN call is:

```
[symbol]    IEWBIND    FUNC=GETN
                       [,VERSION=version]
                       [,RETCODE=retcode]
                       [,RSNCODE=rsncode]
                       ,WORKMOD=workmod
                       [,AREA=buffer]
                       ,CURSOR=cursor
                       ,COUNT=count
                       ,TCOUNT=tcount
                       [,NTYPE={SECTION | CLASS}]
```

FUNC=GETN
Specifies that a count of the number of sections in a workmod and, optionally, the names of each section, be returned to a specified location.

VERSION=*1 | 2 | 3 | 4 | 5 | 6 | 7*
Specifies the version of the parameter list to be used. The default value is VERSION=1.
Note:
This version must match the version you specify with the IEWBUFF macro when you define the buffer passed on this call.

RETCODE=*retcode* — RX-type address or register (2-12)
Specifies the location of a fullword integer that is to receive the return code returned by the binder.

RSNCODE=*rsncode* — RX-type address or register (2-12)
Specifies the location of a 4-byte hexadecimal string that is to receive the reason code returned by the binder.

WORKMOD=*workmod* — RX-type address or register (2-12)
Specifies the location of an 8-byte area that contains the workmod token for this request.

AREA=*buffer* — RX-type address or register (2-12)
Specifies the location of a buffer to receive the names. This buffer must be in the format for section names (TYPE=NAME). See IEWBUFF - Binder API buffers interface assembler macro for generating and mapping data areas and Binder API buffer formats for information on buffer definition.

Section names will be moved until either the buffer is filled or all names have been moved. This keyword is optional. If it is not specified, only the number of section names in the workmod will be returned.

CURSOR=*cursor* — RX-type address or register (2-12)
Specifies the location of a fullword integer that contains the position relative to the start of the list of names where the binder should begin processing. Specifying a zero for this argument causes the binder to begin processing at the beginning of the list. Offsets are specified in records and are relative to the start of the list. The cursor value is modified before returning to the caller.

COUNT=*count* — RX-type address or register (2-12)
Specifies the location of a fullword integer in which the binder will indicate the number of names actually returned in the buffer.

TCOUNT=*tcount* — RX-type address or register (2-12)
Specifies the location of a fullword integer in which the binder will indicate the total name count. TCOUNT indicates the total number of sections or classes in the workmod, not just those returned in the buffer.

NTYPE={SECTION | CLASS}

Specifies the type of names to be returned and counted. **SECTION** causes the names of all sections in the workmod, including special sections, to be returned. In addition, the compile unit CU numbers are provided for buffer version 6 or higher. **CLASS** causes the names of all classes in the workmod containing data to be returned. The value for NTYPE can be abbreviated as **S** or **C**. **SECTION** is the default.

Processing notes

The CURSOR value identifies an index into the requested data beginning with 0 for the first name list entry. The name list buffer formats defined in Binder API buffer formats contain an entry length field in their headers. Multiplying the cursor value by the entry length provides a byte offset into the data. CURSOR is both an input and output parameter. On input to the service, the cursor specifies the first item to return. On exit from the service, it is updated to the index of the next sequential name list entry if not all entries have yet been retrieved.

Return and reason codes

The common binder API reason codes are shown in Table 3.

Return Code	Reason Code	Explanation
00	00000000	Normal completion.
04	83000800	An end-of-data condition was detected. Some data might have been returned in buffer. There is no message associated with this condition.
04	83000801	No section names exist. No data was returned.
08	83000750	The buffer is not large enough for one record. No data is returned.
04	83000810	Cursor is negative or beyond the end of the specified item. No data was returned.
12	83000102	Workmod is unbound. GETN request rejected.

Parameter list

If your program does not use the IEWBIND macro, place the address of the GETN parameter list in general purpose register 1.

Table 16. GETN parameter list

```
PARMLIST DS   0F
         DC   A(GETN)                 Function code
         DC   A(RETCODE)              Return code
         DC   A(RSNCODE)              Reason code
         DC   A(WORKMOD)              Workmod token
         DC   A(BUFFER)               Data buffer
         DC   A(CURSOR)               Starting position
         DC   A(COUNT)                Data count
         DC   A(TCOUNT)               Total count
         DC   A                       Name type to return and end-of-list indicator
              (NTYPE+X'800000000')
GETN     DC   H'60'                   GETN Function code
         DC   H'version'              Interface version number
NTYPE    DC   CL1'C'                  'C' = class; 'S' = section
```

__iew_getN() - Get names

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Returns a list of section or class names within the program together with information about the sections or classes.

Format

```
#include <__iew_api.h>
int __iew_getD(_IEWAPIContext *__context,
                          _IEWNameType __nametype,
                          unsigned int *__total_count,
                          _IEWNameListEntry ** __name_entry);
```

Parameters Descriptions

__context
API context is created and returned by calling __iew_openW() and is used throughout the open session.
__nametype
name type could be one of the following: _IEW_SECTION or _IEW_CLASS.
__total_count
total number of sections or classes in the workmod.
__name_entry
returned buffer - binder name list.

Returned Value

If successful, __iew_getN() returns > 0 (count, number of entries returned in the buffer).

If unsuccessful, __iew_getN() returns 0.

Utilities Functions

__iew_eod()
__iew_get_reason_code()
__iew_get_return_code()
__iew_get_cursor()
__iew_set_cursor()

__iew_fd_getN() - Get names

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Returns a list of section or class names within the program together with information about the sections or classes.

Format

```
#include <__iew_api.h>
int __iew_fd_getN(_IEWFDContext *__context,
                                _IEWNameType __nametype,
                                _IEWNameListEntry ** __name_entry);
```

Parameters Descriptions

__context
FD context is created and returned by calling __iew_fd_open() and is used throughout the open session.
__nametype
name type can be _IEW_SECTION or _IEW_CLASS
__name_entry
returned buffer - binder name list.

Returned Value

If successful, __iew_fd_getN() returns > 0 (count, number of entries returned in the buffer).

If unsuccessful, __iew_fd_getN() returns 0.

Utilities Functions

__iew_fd_eod()
__iew_fd_get_reason_code()
__iew_fd_get_return_code()
__iew_fd_get_cursor()
__iew_fd_set_cursor()

GN - Getting Names of sections or classes

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

This is often the first Get service requested, because it provides the names of sections or classes within the program object, thus providing information required for the GD or GE service calls. If this service is called with no buffer it returns, in the count field, the number of classes or sections in the program object.

Table 42. GN parameter list

Parameter	Usage	Format	Content
1	in	structure	'GN', X'0001'
2	in	binary word	mtoken
3	in	character	ntype: C for class or S for section May be upper or lower case.
4 (optional)	in/out	vstring	buffer Must be an NAME buffer formatted by IEWBUFF or as defined in Binder API buffer formats.
5	in/out	binary word	cursor - in items The item size depends on the buffer type and buffer version used.
6	out	binary word	count - in items

Sample assembler code

```
         CALL  (15),(GNIL,MTOKEN,TYPE,BUFF,CURS,CNT),VL

GNIL     DC    C'GN',X'0001'
MTOKEN   DS    F              As set at Start call
TYPE     DC    C'C'           To return class names/info
CURS     DC    F'0'           Start with first class
CNT      DS    F              Number of classes returned
BUFF IEWBUFF FUNC=MAPBUF,TYPE=NAME,VERSION=6,SIZE=50
*             i.e. a buffer big enough to hold 50 classes.
*             (Class names are never too long.)
```

ESD entry (version 5)

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Figure 44. Format for ESD entries

Field Name	Field Type	Off set	Leng	Description
IEWBESD				Binder ESD buffer, Version 5
ESDH_BUFFER_ID	Char	0	8	Buffer identifier "IEWBESD"
ESDH_BUFFER_LENG	Binary	8	4	Length of the buffer, including the header
ESDH_VERSION	Binary	12	1	Version identifier (Constant 2)
*** RESERVED ***	Binary	13	3	Reserved, must be zeros
ESDH_ENTRY_LENG	Binary	16	4	Length of each entry
ESDH_ENTRY_COUNT	Binary	20	4	Number of entries in the buffer
*** RESERVED ***	Binary	24	8	Reserved
ESDH_ENTRY_ORIGIN		32		First ESD entry
ESD_ENTRY				ESD entry
ESD_TYPE	Char	0	2	ESD Type
C' '				Null Entry
C'SD'				Control Section
C'LD'				Label Definition
C'ER'				External Reference
C'PR'				Part Reference
C'PD'				Part Definition
C'ED'				Element Definition
ESD_TYPE_QUAL	Char	2	2	ESD Type Qualifier
C' '				(no qualification)
C'SD'				Section Definition (SD)
C'CM'				Common (SD)
C'ST'				Segment Table (SD)
C'ET'				Entry Table (SD)
C'PC'				Unnamed Section (SD)
C'PR'				Part Reference (PR, PD)
C'PD'				Part Definition (PR, PD)
C'ER'				External Reference (ER)
C'WX'				Weak Reference (ER)
ESD_NAME_SPACE	Binary	4	1	Name Space for symbols
X'00'				Class and section names (SD, ED)
X'01'				Labels and references (LD, ER)
X'02'				Pseudoregisters (PR, PD)
X'03'				Parts(PR, PD) in merge classes
X'04'-x'07'				Reserved
ESD_SCOPE	Char	5	1	Scope of Name
' '				Not applicable
'S'				Section (Types SD/private,ST,ET)
'M'				Module (Types SD/CSECT,LD, ER/weak,CM,PR,DS,PD)
'L'				Library (Type ER/strong)
'X'				Symbol can be IMPORTED or EXPORTED.
ESD_NAME	Name	6	6	Symbol represented by ESD record
ESD_NAME_CHARS	Binary	6	2	length of name in bytes
ESD_NAME_PTR	Pointer	8	4	pointer to name string

Field Name	Field Type	Off set	Leng	Description
ESD_SYMBOL_ATTR	Binary	12	1	Symbol attributes
1...				ON = strong ref or def.
				OFF = weak ref or def.
.1..				ON = this symbol can be renamed
..1.				ON = Symbol is a descriptor
...1				ON = symbol is a C++ mangled name
.... 1...				ON = symbol uses XPLINK linkage conventions
.... .1..				Environment exists
.... ..11				** Reserved **
ESD_FILL_CHAR	Char	13	1	Value to fill with
1 ESD_RES_SECTION	Name	14	6	Name of containing section
ESD_RESIDENT_CHARS	Binary	14	2	length of name in bytes
ESD_RESIDENT_PTR	Pointer	16	4	pointer to name string
ESD_LENG	Binary	20	4	Length of defined element (ED, PD, PR)
ESD_ALIGN	Binary	24	1	Alignment specification from language processor. Indicates Alignment of section contribution within class segment (ED, PD, PR)
X'00'				Byte alignment (PD, PR)
X'01'				Halfword (PD, PR)
X'02'				Fullword (PD, PR)
X'03'				Doubleword (PD, PR, ED)
X'04'				Quadword (PR, PD, ED)
X'0C'				4K page (ED)
ESD_USABILITY	Binary	25	1	Reusability of Section (SD)
X'00'				Unspecified
X'01'				Nonreusable
X'02'				Reusable
X'03'				Reentrant
X'04'				Refreshable

Field Name	Field Type	Off set	Leng	Description
ESD_AMODE	Bit	26	1	Addressing Mode for Section or label (SD, LD)
X'00'				Unspecified
X'01'				AMODE 24
X'02'				AMODE 31
X'03'				AMODE ANY (24 or 31)
X'04'				AMODE MIN
X'05'				Unused, reserved

```
       X'06'                            AMODE 64
   ESD_RMODE           Bit     27   1   Residence Mode for class element
                                          (ED)
       X'01'                            RMODE 24
       X'03'                            RMODE ANY (24 or 31)
       X'04'                            RMODE 64   ESD_RECORD_FMT
                       Binary  28   2   Record format for class (ED)
       H'1'                             Byte stream
       H'>1'                            Fixed length records
   ESD_LOAD_FLAGS      Bit     30   1   Load Attributes (ED)
       1... ....                        Read-only
       .1.. ....                        Do not load with module
       ..1. ....                        Moveable
       ...1 ....                        Shareable
       .... 1...                        Deferred
       .... .111                        Reserved
   ESD_BIND_FLAGS      Bit     31   1   Bind Attributes
2      1... ....                        Binder generated (SD, ED, LD)
2      .1.. ....                        No class data available (ED)
2      ..1. ....                        Variable length records (ED)
       ...1 ....                        Descriptive data (not text) (ED)
       .... 1...                        ON = class contains part initializers (ED)
       .... .1..                        ON = fill character has been specified (ED)
       .... ..1.                        Class has padding
       .... ...1                        ** Reserved **
   ESD_BIND_CNTL       Bit     32   1   Bind control information
1      1... ....                        Removable class(ED)
1      .x.. ....                        ** Reserved **
1      ..xx ....                        Binding method (ED)
1      ..00 ....                          CAT (Catenated text)
1      ..01 ....                          MRG (Merged parts)
1      ..1x ....                          ** Reserved **
     ESD_ATTRIBUTES    Bit     33   1   General attributes
   4   1... ....                          Compiled as system LE
   4   .1.. ....                          Compiled as lightweight LE
       ..11 ....                          ** Reserved **
   4   .... xx..                          Error severity for dups (PD, LD)
       .... 00..                          - I-level
       .... 01..                          - W-level
       .... 10..                          - E-level
       .... 11..                          - S-level
       .... ..1.                          ** Reserved **
       .... ...1                          ** Reserved **

    Field              Field  Off Leng          Description
    Name               Type   set

   ESD_XATTR_CLASS     Name    34   6   Extended attributes class
                                        (LD, ER)

     ESD_XATTR_CLASS_CHARS
                       Binary  34   2     length of name in bytes
     ESD_XATTR_CLASS_PTR
                       Pointer 36   4     pointer to name string
   ESD_XATTR_OFFSET    Binary  40   4   Extended attributes element offset
                                        (LD, ER)
2  ESD_SEGMENT         Binary  44   2   Overlay segment number (SD)
2  ESD_REGION          Binary  46   2   Overlay region number (SD)
   ESD_SIGNATURE       Char    48   8   Interface signature
2  ESD_AUTOCALL        Binary  56   1   Autocall specification (ER)
       1... ....                          ** Reserved **
       .1.. ....                          Entry in LPA.  If ON, name is
                                          an alias.
       ..xx xxxx                          ** Reserved **
2  ESD_STATUS          Bit     57   1   Resolution status (ER)
       1... ....                          Symbol is resolved
       .1.. ....                          Processed by autocall
       ..1. ....                          INCLUDE attempted
       ...1 ....                          Member not found
       .... 1...                          Resolved outside module
       .... .1..                          NOCALL or NEVERCALL
       .... ..1.                          No strong references
       .... ...1                          Special call library
2  ESD_TGT_SECTION     Name    58   6   Target section (ER)
     ESD_TARGET_CHARS  Binary  58   2     length of name in bytes
     ESD_TARGET_PTR    Pointer 60   4     pointer to name string
   *** RESERVED ***    Char    64   2   Reserved
   ESD_RES_CLASS       Name    66   6   Name of containing class
                                        (LD, PD) or target class (ER)
     ESD_RES_CLASS_CHARS
                       Binary  66   2     length of name in bytes
     ESD_RES_CLASS_PTR
                       Pointer 68   4     pointer to name string
3  ESD_ELEM_OFFSET     Binary  72   4   Offset within class element
                                        (LD, ER)
2  ESD_CLASS_OFFSET    Binary  76   4   Offset within class segment
                                        (ED, LD, PD, ER)
   *** RESERVED ***    Char    80   2   Reserved
   ESD_ADA_CHARS       Binary  82   2   Environment name length (LD)
   ESD_ADA_PTR         Pointer 84   4   Pointer to name of environment (LD)
   *** RESERVED ***    Char    88   4   Reserved
   ESD_PRIORITY        Binary  92   4   Binding priority
```

Notes:

1. This entry is ignored on input to the binder.
2. Recalculated by the binder.
3. Calculated on the ED and ER records, required input to LD.
4. Valid for SDs only.

GETE: Get ESD data

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

GETE returns data from ESD items. GETE must be used on a bound workmod. Four optional parameters allow you to specify selection criteria for the ESD items to be returned: section name, ESD record type, offset in the section or module, and symbol name. Only ESD records that meet all of the selection criteria will be returned. Multiple selection criteria can be specified to retrieve exactly the records you need.

The syntax of the GETE call is:

```
[symbol]    IEWBIND    FUNC=GETE
                       [,VERSION=version]
                       [,RETCODE=retcode]
                       [,RSNCODE=rsncode]
                       ,WORKMOD=workmod
                       [,SECTION=section]
                       [,RECTYPE=rectype]
                       [,CLASS=class]
                       [,{OFFSET=offset | SYMBOL=symbol}]
                       ,AREA=buffer
                       ,CURSOR=cursor
                       ,COUNT=count
```

FUNC=GETE
Requests that data from ESD items in a workmod be returned to a specified location.

VERSION=*1 | 2 | 3 | 4 | 5 | 6 | 7*
Specifies the version of the parameter list to be used. The default value is VERSION=1.

Note:
If VERSION=1 is specified for the GETE call, CLASS cannot be specified as a macro keyword. The parameter list ends with the COUNT parameter (with the high-order bit set). This exception is for Version 1 only.

RETCODE=*retcode* — RX-type address or register (2-12)
Specifies the location of a fullword integer that is to receive the return code returned by the binder.

RSNCODE=*rsncode* — RX-type address or register (2-12)
Specifies the location of a 4-byte hexadecimal string that is to receive the reason code returned by the binder.

WORKMOD=*workmod* — RX-type address or register (2-12)
Specifies the location of an 8-byte area that contains the workmod token for this request.

SECTION=*section* — RX-type address or register (2-12)
Specifies the location of a 16-byte varying character string that contains the name of the section to be processed. This argument can be set to blanks to indicate blank common area. Sections will be retrieved in the same order that they were included in the workmod.

The default value is all sections. If this parameter is specified, only the indicated section is searched.

RECTYPE=*rectype* — RX-type address or register (2-12)
Specifies the location of a varying character string that contains a list of the ESD record types to be returned. If you do not specify this argument, all record types are returned.

Record types must be identified by one- or two-character codes, separated by commas and enclosed in parentheses. Embedded blanks are not allowed. Valid record types are:

SD
Section definition

ED
Element definition

LD
Label definition

PD
Part definition

PR
Part reference

ER
External reference

CM
Common

ST
Segment table

ET
Entry table

DS
Dummy section definition

CM
Common section definition

ET
ENTAB

ST
SEGTAB

PC
Private code section definition

WX
Weak external reference

In addition, you can use a generic code to reference more than one ESD type:

S
Section definition records (SD, CM, ST, ET, PC, and DS)

U
Unresolved external references (ER, ESD_STATUS=unresolved)

CLASS=*class* — RX-type address or register (2-12)
Specifies the location of a 16-byte varying character string containing the name of the text class referenced by the ESD record to be selected. If class has not been specified, ESD records are returned without regard to class.

OFFSET=*offset* — RX-type address or register (2-12)
Specifies the location of a fullword integer that contains the offset within the specified section. If a section name has not been specified, a module offset is assumed. If you specify OFFSET you cannot specify SYMBOL but must specify CLASS.

SYMBOL=x*symbol* — RX-type address or register (2-12)
Specifies the location of a varying character string that contains a symbol to be used as a selection criterion. If you specify SYMBOL you cannot specify OFFSET.

If neither **OFFSET** nor **SYMBOL** is provided, processing begins at the start of the item.

AREA=*buffer* — RX-type address or register (2-12)

Specifies the location of a buffer to receive the data. This buffer must be allocated and initialized in ESD format. See IEWBUFF - Binder API buffers interface assembler macro for generating and mapping data areas for information on buffer handling.

CURSOR=*cursor* — RX-type address or register (2-12)
Specifies the location of a fullword integer that indicates the position within the section or module where the binder should begin processing. Specifying a zero for this argument causes the binder to begin processing at the first ESD entry. Offsets are specified in records and are relative to the start of the selected ESD item(s).

COUNT=*count* — RX-type address or register (2-12)
Specifies the location of a fullword integer in which the binder will store the number of entries it has returned.

Processing notes

The binder returns ESD records that meet the selection criteria specified on the call:

- If SECTION is specified, only that section of the ESD will be searched. All sections is the default.
- If RECTYPE is specified, only ESD records of the types appearing in the supplied list are returned.
- If OFFSET is specified and rectype="S", the ESD record for the control section (or common area) containing the specified offset, is returned for buffer version 1. The SD record mapping in other buffer versions does not contain an offset and no records will be returned. If OFFSET is specified and rectype="LD", then all LD ESD records for the symbols defined at or before that location (within the containing section) will be returned.
- If SYMBOL is specified, all ESD records of the type(s) specified with that symbol name are returned. If CLASS is specified, only ESD records that define locations in that class are returned. Some records, such as SD and ER, are not associated with any class and are never returned if class is specified.

Note:
Processing of the ESD records returned by a GETE call should not make assumptions about the order of the returned ESD records. For example, such processing should not assume that LD type ESD records are returned in the order of their offsets in the section.

The CURSOR value identifies an index into the requested ESD data beginning with 0 for the first ESD. The ESD buffer formats defined in Binder API buffer formats contain an entry length field in their headers. Multiplying the cursor value by the entry length provides a byte offset into the data. CURSOR is an input and output parameter. On input to the service, the cursor specifies the first record to return. On exit from the service, it is updated to the index of the next sequential ESD if not all data has yet been retrieved.

The binder will typically return multiple entries in a single call, depending on the size of the buffer. Data is reformatted, if necessary, to conform to the version identified in the caller's buffer. The COUNT parameter is set to the number of records actually returned in the buffer.

The ESD buffer formats defined in Binder API buffer formats contain a record length field in their headers giving the length of each ESD record. This provides a way for the caller to index through the returned records or to access a specific record in the returned data buffer.

In some cases where OFFSET is specified and the parameter list is version 6 or less, return code 0 and reason code 0 will be returned on an end-of-data condition. The version 7 API call will always return return code 4 and reason code 83000800 on an end-of-data condition, while the COUNT may be non-zero indicating that data was returned.

Return and reason codes

The common binder API reason codes are shown in Table 3.

Return Code	Reason Code	Explanation
00	00000000	Normal completion.
04	83000705	The specified symbol could not be located in the workmod. No data is returned in the buffer.
04	83000800	An end-of-data condition was detected. Some data might have been returned in buffer. There is no message associated with this condition.
04	83000801	The requested item was not found in the workmod, or was empty, or no records met the specified criteria. No data returned.
04	83000812	The specified offset was negative or beyond the end of the designated item or module. No data is returned in the buffer.
12	83000101	OFFSET and SYMBOL have both been specified. Request rejected.
12	83000102	Workmod is unbound. GETE request rejected.

Parameter list

If your program does not use the IEWBIND macro, place the address of the GETE parameter list in general purpose register 1.

Table 15. GETE parameter list

```
PARMLIST DS   0F
         DC   A(GETE)              Function code
         DC   A(RETCODE)           Return code
         DC   A(RSNCODE)           Reason code
         DC   A(WORKMOD)           Workmod token
         DC   A(SECTION)           Section name
         DC   A(RECTYPE)           ESD record type(s)
         DC   A(OFFSET)            Offset in module or section. If not a selection
                                   criterion, set to -1.
         DC   A(SYMBOL)            Symbol name
         DC   A(BUFFER)            Data buffer
         DC   A(CURSOR)            Starting position
         DC   A(COUNT)             Data count
         DC   A(CLASS)             Text class
GETE     DC   H'62'                GETE function code
         DC   H'version'           Interface version number
RECTYPE  DC   H'7',CL7'(SD,CM)'    Sample varying string
```

Note:
X'80000000' must be added to either the COUNT parameter (for Version 1) or the CLASS parameter (for Version 2 or higher).

__iew_getE() - Get ESD data

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Returns external symbol dictionary information selected by various criteria.

Format

```
#include <__iew_api.h>
int __iew_getE(_IEWAPIContext *__context,
                          const char *__sect, const char *__class,
                          const char *__sym, const char *__rec_type
                          int *__offset,
                          _IEWESDEntry ** __esd_entry);
```

Parameters Descriptions

__context
API context is created and returned by calling __iew_openW() and is used throughout the open session.
__sect
section name.
__class
class name. See class under Understanding binder programming concepts for details.
__sym
symbol name. See “External symbol dictionary" in Chapter 2 of Program Management User's Guide and Reference for details.
__rec_type
ESD record type.
__offset
offset in module or section
__esd_entry
returned buffer - ESD entry

Returned Value

If successful, __iew_getE() returns > 0 (count, number of entries returned in the buffer).

If unsuccessful, __iew_getE() returns 0.

Utilities Functions

__iew_api_name_to_str()
__iew_eod()
__iew_get_reason_code()
__iew_get_return_code()
__iew_get_cursor()
__iew_set_cursor()

__iew_fd_getE() - Get ESD data

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Returns external symbol dictionary information selected by class and/or section to which it refers.

Format

```
#include <__iew_api.h>
int __iew_fd_getE(_IEWFDContext *__context,
                                const char *__sect, const char *__class,
                                _IEWESDEntry ** __esd_entry);
```

Parameters Descriptions

__context
FD context is created and returned by calling __iew_fd_open() and is used throughout the open session.
__sect
section name.
__class
class name.
__esd_entry
returned buffer - ESD entry.

Returned Value

If successful, __iew_fd_getE() returns > 0 (count, number of entries returned in the buffer).

If unsuccessful, __iew_fd_getE() returns 0.

Utilities Functions

__iew_api_name_to_str()
__iew_fd_eod()
__iew_fd_get_reason_code()
__iew_fd_get_return_code()
__iew_fd_get_cursor()
__iew_fd_set_cursor()

GE - Getting External Symbol Dictionary data

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

This service is a specialized variant of GD, used to retrieve only one class, B_ESD. This can be a confusing class, though, because ESD records are owned by elements in a second class and may point to elements in a third class. For example, an ESD may describe an adcon in class C_CODE that refers to an address in class B_TEXT. Using the 'GD' service you would only be able to indicate that you were interested in class B_ESD, and would have to retrieve all ESDs to locate the specific ones you were interested in. The 'GE' service allows the caller to screen ESDs returned, limiting the output to those owned by a specified class. The calling application can also ask for ESDs in a specific section.

Table 41. GE parameter list

Parameter	Usage	Format	Content
1	in	structure	'GE', X'0001'
2	in	binary word	mtoken
3 (optional)	in	vstring	class
4 (optional)	in	vstring	section
5	in/out	structure	buffer Must be formatted by IEWBUFF or as defined in Binder API buffer formats, and appropriate to the class requested.
6	in/out	binary word	cursor - in items The item size depends on the buffer type and buffer version used.
7	out	binary word	count - in items

Sample assembler code

```
         CALL  (15),(GEIL,MTOKEN,CLASS,,BUFF,CURS,CNT),VL

GEIL     DC    C'GE',X'0001'
MTOKEN   DS    F              As set at Start call
CLASS    DC    H'6',C'C_CODE' Limit ESDs returned
CURS     DC    F'0'           Start with first ESD
CNT      DS    F              Number of ESDs returned
BUFF IEWBUFF FUNC=MAPBUF,TYPE=ESD,VERSION=6,SIZE=50
*             i.e. a buffer big enough to hold 50 ESDs,
*             assuming the names are not too long.
```

Text data buffer (version 1)

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Figure 32. Format for TXT entries

Field Name	Field Type	Offset	Leng	Description
IEWBTXT				Binder Text buffer, Version 1
TXTH_BUFFER_ID	Char	0	8	Buffer identifier "IEWBTXT"
TXTH_BUFFER_LENG	Binary	8	4	Length of the buffer, including the header
TXTH_VERSION	Binary	12	1	Version identifier
*** RESERVED ***	Binary	13	3	Reserved, must be zeros
TXTH_ENTRY_LENG	Binary	16	4	Length of entries (always 1)
TXTH_ENTRY_COUNT	Binary	20	4	Number of entries (bytes) in the buffer
*** RESERVED ***	Binary	24	8	Reserved, initialize to zeros
TXT_ARRAY	Undef.	32	var	Program Text (length varies from 1 to 2**31-1 bytes, depending on value in TXTH_ENTRY_COUNT)

GETD: Get data

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

GETD returns data from items in a workmod. The values of the **CLASS** and **SECTION** parameters determine which item is returned. If **SECTION** is omitted, all sections are returned as a single unit. This service can only be performed on a bound workmod.

The syntax of the GETD call is:

```
[symbol]    IEWBIND    FUNC=GETD
                       [,VERSION=version]
                       [,RETCODE=retcode]
                       [,RSNCODE=rsncode]
                       ,WORKMOD=workmod
                       ,CLASS=class
                       [,SECTION=section]
                       ,AREA=buffer
                       ,CURSOR=cursor
                       ,COUNT=count
                       [,RELOC=reloc]
```

FUNC=GETD
Requests that data from items in a workmod be returned to a specified location.

VERSION=*1 | 2 | 3 | 4 | 5 | 6 | 7*
Specifies the version of the parameter list to be used. The default value is VERSION=1.

RETCODE=*retcode* — RX-type address or register (2-12)
Specifies the location of a fullword integer that is to receive the return code returned by the binder.

RSNCODE=*rsncode* — RX-type address or register (2-12)
Specifies the location of a 4-byte hexadecimal string that is to receive the reason code returned by the binder.

WORKMOD=*workmod* — RX-type address or register (2-12)
Specifies the location of an 8-byte area that contains the workmod token for this request.

CLASS=*class* — RX-type address or register (2-12)
Specifies the location of a 16-byte varying character string containing a class name. The class name might have been defined by the binder, a compiler, or an end user. See Understanding binder programming concepts for binder class names. B_PMAR is also accepted as a class name, although it is not an actual class in a binder workmod.

SECTION=*section* — RX-type address or register (2-12)
Specifies the location of a varying character string that contains the name of the section to be processed. If omitted, this defaults to a concatenation of all sections in the specified class. If the processing intent is bind, the sections are ordered by virtual address. If the processing intent is access, they are returned in the same order that they were included in the workmod.

AREA=*buffer* — RX-type address or register (2-12)
Specifies the location of a buffer to receive the data. The binder returns data until either this buffer is filled or the specified items have been completely moved. See IEWBUFF - Binder API buffers interface assembler macro for generating and mapping data areas for information on buffer handling.

CURSOR=*cursor* — RX-type address or register (2-12)
Specifies the location of a fullword integer that contains the position within the item(s) where the binder should begin processing. Specifying a zero for the argument causes the binder to begin processing at the start of the item. The cursor value is specified in bytes for items in the TEXT class, in records for all other classes. The value is relative to the start of the item. The cursor value is modified before returning to the caller.

COUNT=*count* — RX-type address or register (2-12)
Specifies the location of a fullword that is to receive the number of bytes of TEXT or the number of entries returned by the binder.

RELOC=*reloc*
Specifies a base address to be used for relocation. You can only use this parameter with **VERSION**=6 or higher. You will need to know the load segment for the data you are requesting. You can map text classes into load segments using GETN. *reloc* is a single 8–byte address. The relocation address will relocate the adcons in the returned text buffer as though the program segments had been loaded at the designated address. If you do not use the **RELOC** parameter, it should set to zero.

Processing notes

The CURSOR value identifies an index into the requested data beginning with 0 for the first data item. Each of the buffer formats defined in Binder API buffer formatscontains an entry length field in

its header. Multiplying the cursor value by the entry length provides a byte offset into the data. Note that CUI, LIB, PMAR, and text data is always treated as having entry length 1. The CURSOR value is both an input and output parameter. On input to the service, the cursor specifies the first item to return. On exit from the service, it is updated to an appropriate value for continued sequential retrieval if not all data has yet been retrieved. For text data, this may or may not be the next byte after the last one returned, because pad bytes between sections and uninitialized data areas within sections may have been skipped. Any data skipped should be treated by the calling application as containing the fill character (normally X'00').

On the next GETD request, the binder begins processing where the last request left off.

If you interrupt a series of successive GETD calls, you should reset the value of the cursor before continuing. Otherwise, the cursor value might be invalid and the results of a GETD request are unpredictable.

If a section name is not passed on a GETD API invocation for a text class and the target is an overlay module, the cursor is interpreted as an offset into the module and laid out sequentially in segment order, using the alignment as specified in the object modules.

Return and reason codes

The common binder API reason codes are shown in Table 3.

Return Code	Reason Code	Explanation
00	00000000	Normal completion.
04	83000800	Normal completion. Some data might have been returned in the buffer, and an end-of-data condition was encountered. There is no message associated with this condition.
04	83000801	The requested item did not exist or is empty. No data has been returned.
08	83000750	The buffer is not large enough for one record. No data is returned.
08	83000813	The buffer version is not compatible with the module content. No data is returned.
08	83002349	Not all adcons were successfully relocated. This condition could occur because relocation addresses for all the segments were not passed, or because the adcon length was insufficient to contain the address.
12	83002379	Binder encountered a bad cursor for class B_PARTINIT and processing has been stopped.
12	83000102	Workmod was in an unbound state. GETD request could not be processed.
12	83002375	The class was not a text class.

Parameter list

If your program does not use the IEWBIND macro, place the address of the GETD parameter list in general purpose register 1.

Table 14. GETD parameter list

```
PARMLIST DS     0F
         DC     A(GETD)              Function code
         DC     A(RETCODE)           Return code
         DC     A(RSNCODE)           Reason code
         DC     A(WORKMOD)           Workmod token
         DC     A(CLASS)             Class name
         DC     A(SECTION)           Section name
         DC     A(BUFFER)            Data buffer
         DC     A(CURSOR)            Starting position
         DC     A                    Data count and end-of-list indicator
                (COUNT+X'80000000')
         DC     A(RELOC)             Relocation address
GETD     DC     H'61'                GETD function code
         DC     H'version'           Interface version number
```

__iew_getD() - Get data

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Returns data associated with a specified class (and optionally section) in the program.

Format

```
#include <__iew_api.h>
int __iew_getD(_IEWAPIContext *__context,
                          const char *__class, const char * __sect,
                          long long * __reloc,
                          void ** __data_entry);
```

Parameters Descriptions

__context
API context is created and returned by calling __iew_openW() and is used throughout the open session.
__class
class name.
__sect
section name.
__reloc
relocation address.
__data_entry
returned buffer - based on class name.

Returned Value

If successful, __iew_getD() returns a number greater than zero representing the number of data items or bytes returned in the buffer.

If unsuccessful, __iew_getD() returns 0.

Utilities Functions

__iew_api_name_to_str()
__iew_eod()
__iew_get_reason_code()
__iew_get_return_code()
__iew_get_cursor()
__iew_set_cursor()

__iew_fd_getD() - Get data

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Return data associated with a specified class (and optionally section) in the program.

Format

```
#include <__iew_api.h>
int __iew_fd_getD(_IEWFDContext *__context,
                                const char *__class, const char * __sect,
                                long long *__reloc,
                                void ** __data_entry);
```

Parameters Descriptions

__context
FD context is created and returned by calling __iew_fd_open() and is used throughout the open session.
__class
class name.
__sect
section name.
__reloc
relocation address.
__data_entry
returned buffer - based on class.

Returned Value

If successful, __iew_fd_getD() returns > 0 (count, could be number of bytes of TEXT (if class=TEXT) or number of entries returned in the buffer).

If unsuccessful, __iew_fd_getD() returns 0.

Utilities Functions

__iew_api_name_to_str()
__iew_fd_eod()
__iew_fd_get_reason_code()
__iew_fd_get_return_code()
__iew_fd_get_cursor()
__iew_fd_set_cursor()

GD - Getting Data from any class

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

This service is often used to access program text, but can also be used to retrieve data from other compiler-defined or binder-defined classes. The data can optionally be limited to that associated with a particular section.

One special feature for program text is that the addresses within the text can be relocated in the same way that the loader would do, relative to a specified starting address.

Note that many programs are built with multiple text classes. The GD service is not able to combine data from different classes in a single call, and the cursor used for positioning is always relative to the beginning of the specified class (or section contribution within the class if the optional section parameter is provided). Typically the calling application views program text as continuous across classes within a loadable segment. The application can adjust for this by using the class starting offset within the segment as returned by the GN service in the BNL_SEGM_OFF field.

Table 40. GD parameter list

Parameter	Usage	Format	Content
1	in	structure	'GD', X'0001'
2	in	binary word	mtoken
3	in	vstring	class
4 (optional)	in	vstring	section
5	in/out	structure	buffer Must be formatted by IEWBUFF or as defined in Binder API buffer formats, and appropriate to the class requested.
6	in/out	binary word	cursor - in items The item size depends on the buffer type and buffer version used. For text data, this might not be returned as the next location after the last text byte returned, because pad bytes between sections and uninitialized data areas within sections may have been skipped. Any data skipped should be treated by the caller as containing the fill character (normally X'00').
7	out	binary word	count - in items
8 (optional)	in	64-bit address	relocation value An assumed address for the start of the class, to be used for address constant relocation.

Sample assembler code

```
         CALL  (15),(GDIL,MTOKEN,CLASS,,BUFF,CURS,CNT),VL

GDIL     DC    C'GD',X'0001'
MTOKEN   DS    F             As set at Start call
CLASS    DC    H'6',C'B_TEXT' One particular text class
CURS     DC    F'0'          Start at beginning of text
CNT      DS    F             Number of bytes returned
*                           (since text item size is 1)
BUFF  IEWBUFF  FUNC=MAPBUF,TYPE=TEXT,BYTES=8192
*        Note default to V1, but text buffer hasn't changed
```

CUI entry (version 6)

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Figure 46. Format for CUI entries

Field Name	Field Type	Off set	Leng	Description
IEWBCUI				Binder CUI buffer, Version 6
CUIH_BUFFER_ID	Char	0	8	Buffer identifier "IEWBCUI"
CUIH_BUFFER_LENG	Binary	8	4	Length of the buffer, including the header
CUIH_VERSION	Binary	12	1	Version identifier
*** RESERVED ***	Binary	13	3	Reserved, must be zeros
CUIH_ENTRY_LENG	Binary	16	4	Length of entries
CUIH_ENTRY_COUNT	Binary	20	4	Number of entries (bytes) in the buffer
*** RESERVED ***	Binary	24	8	Reserved, initialize to zeros
CUIH_ENTRY_ORIGIN		32		First compile unit entry
CUI_ENTRY				Compile Unit Entry
CUI_CU	Binary	0	4	Compile unit number
CUI_SOURCE_CU	Binary	4	4	Source of compile unit number
*** RESERVED ***	Binary	8	2	Reserved, must be zero
CUI_MEMBER_LEN	Binary	10	2	Length of member
CUI_MEMBER_PTR	Pointer	12	4	Pointer to the member
*** RESERVED ***	Binary	16	2	Reserved, must be zero
CUI_PATH_LEN	Binary	18	2	Length of path
CUI_PATH_PTR	Pointer	20	4	Pointer to the path
*** RESERVED ***	Binary	24	2	Reserved, must be zero
CUI_DSNAME_LEN	Binary	26	2	Length of dsname
CUI_DSNAME_PTR	Pointer	28	4	Pointer to the dsname
CUI_DDNAME	Char	32	8	Ddname
*** RESERVED ***	Binary	40	2	Reserved, must be zero
CUI_CONCAT	Binary	42	1	Concat
CUI_TYPE	Binary	43	1	Source type
X'00'				Load module
X'01'				Generated by PUTD API version 1
X'02'				Generated by PUTD API version 2 or higher
X'10'				PO1 (PM1) format program object
X'11'				Object module (traditional format)
X'12'				Object module (XOBJ format)
X'13'				Object module (GOFF format)
X'14'				Unknown
X'15'				Workmod
X'1E'				Generated by the binder
X'20'				PO2 (PM2) format program object
X'30'				PO3 (PM3) format program object
X'41'				PO4 (PM4) format program object, z/OS 1.3 compatible
X'42'				z/OS 1.5 compatible
X'43'				z/OS 1.7 compatible
X'51'				PO5 (PM5) format program object, z/OS 1.8 compatible
*** RESERVED ***	Binary	44	4	Reserved, must be zero
*** RESERVED ***	Binary	48	2	Reserved, must be zero
CUI_C_MEMBER_LEN	Binary	50	2	Length of member (original)
CUI_C_MEMBER_PTR	Pointer	52	4	Pointer to the member (original)
*** RESERVED ***	Binary	56	2	Reserved, must be zero
CUI_C_PATH_LEN	Binary	58	2	Length of path (original)
CUI_C_PATH_PTR	Pointer	60	4	Pointer to the path (original)
*** RESERVED ***	Binary	64	2	Reserved, must be zero
CUI_C_DSNAME_LEN	Binary	66	2	Length of dsname (original)
CUI_C_DSNAME_PTR	Pointer	68	4	Pointer to the dsname
*** RESERVED ***	Char	72	3	Reserved, must be zero
CUI_C_TYPE	Binary	75	1	Source type within the object file
CUI_C_SEQ	Binary	76	4	CU sequence number within the object file

Notes:

1. The header record contains information about the target workmod as a whole. The format is the same as that of the other records. CUI_CU, CUI_SOURCE_CU and CUI_C_SEQ will always be zero in the header record.
2. Fields that have the comment original refer to the object module used to build the program object the first time. These field contents are not changed if the program object is rebound. However, this information is available only if the program object is compatible with the z/OS 1.5 format or higher version. Thus, complete information is returned for nonheader records only if the CUI_TYPE value is 42 or greater. CUI_TYPE for program objects in formats compatible with releases earlier than z/OS 1.5. Load modules have CUI_TYPE set to the format of the target module.
3. Entries for compile units representing sections created by the binder, including section 1, contain no information other than the compile unit number (CUI_CU) and the type (CUI_TYPE).

GETC: Get compile unit list

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

GETC returns data which is mapped to a new CUI buffer format (Version 6). The **COMPILEUNITLIST** parameter determines which data is returned.

The syntax of the GETC call is:

[*symbol*]	**IEWBIND**	**FUNC=GETC** **,VERSION**=*version* [**,RETCODE**=*retcode*] [**,RSNCODE**=*rsncode*] **,WORKMOD**=*workmod* [**,COMPILEUNITLIST**=*compileunitlist*]] **,AREA**=*buffer* **,CURSOR**=*cursor* **,COUNT**=*count*

FUNC=GETC
Requests that data from items in a workmod be returned to a specified location.

VERSION=*6*
Specifies the version of the parameter list to be used (6 or higher).

RETCODE=*retcode* — RX-type address or register (2-12)
Specifies the location of a fullword integer that is to receive the return code returned by the binder.

RSNCODE=*rsncode* — RX-type address or register (2-12)
Specifies the location of a 4-byte field that is to receive the reason code returned by the binder. Reason codes are documented as a sequence of 8 hexadecimal digits.

WORKMOD=*workmod* — RX-type address or register (2-12)
Specifies the location of an 8-byte area that contains the workmod token for this request.

COMPILEUNITLIST=*compileunitlist*
Determines which data is returned. If **COMPILEUNITLIST** is specified, one record for each compile unit in a list of compile units will be returned. If **COMPILEUNITLIST** is omitted, one record of each of all compile units will be returned. The header record, the first compile unit record, is built when the cursor is zero.

The compile unit list is a structure:

```
Count    DC    F'5'   List with 5 entries
List     DS    5F     Returned by GETN
```

Note:
compileunitlist must be composed of values returned in BFNL_6_SECT_CU resulting from a GETN TYPE=SECTION,VERSION=6 API call.

When **INTENT=ACCESS** is specified in the CREATEW API call, information about the input module (the target module of the GETC call) is placed in the header record. This information includes the program object version and the source of the input module (data set name or path name, ddname, and member name).

AREA=*buffer* — RX-type address or register (2-12)
Specifies the location of a CUI buffer to receive the data. The binder returns data until either this buffer is filled or the specified items have been completely moved. See IEWBUFF - Binder API buffers interface assembler macro for generating and mapping data areas for information on buffer handling.

CURSOR=*cursor* — RX-type address or register (2-12)
Specifies the location of a fullword integer that contains the position within the items where the binder begins processing. Specifying a zero causes the binder to return the header record, the first compile unit record. The information is provided on the DASD location of the program object. The cursor value is modified before returning to the caller.

When no compile unit list is provided, the cursor is an index into an ordered list of all CUI entries that can be returned. If the application does not modify the cursor during the retrieval process, multiple calls return all CUI records in the order by CU number because the buffer is full. When a compile unit list is provided, the cursor is an index into that application-provided list. CUI records are returned in the order specified in the CU list. If the application still does

not modify the cursor during the retrieval process, multiple calls continue with subsequent entries in the list because the buffer is full. End of data is signalled when the end of the application-provided list is reached.

COUNT=*count* — RX-type address or register (2-12)
Specifies the location of a fullword that receives the number of CUI records returned by the binder.

Processing notes

The CURSOR value identifies an offset into the requested data beginning with 0. It is both an input and output parameter. On input to the service, the cursor specifies the first byte to return. On exit from the service, it is updated to the next byte for continued sequential retrieval if not all data has yet been retrieved.

For load modules and program object formats at a compatability level prior to z/OS V1R5 , a compile unit is the same as a section. For z/OS V1R5 compatible modules, a compile unit corresponds to a single object module. Each compile unit in a workmod is assigned a unique number; however, this assigned number may change when a module is rebound. Furthermore, the compile unit number will be zero for all binder generated sections (IEWBLIT or section 1, for example).

Return and reason codes

The common binder API reason codes are shown in Table 3.

Return Code	Reason Code	Explanation
00	00000000	Normal completion. There might be additional data that did not fit in the buffer.
04	83000800	End of data. Some data might have been returned in the buffer, but no more is available.
04	83000801	No section names exist. No data was returned.
04	83000810	Cursor is negative or beyond the end of the specified item. No data was returned.
08	83002342	Some of the passed compile unit numbers do not exist in workmod. Data for the valid compile units is returned.
12	83000102	Workmod was in an unbound state.

Parameter list

If your program does not use the IEWBIND macro, place the address of the GETC parameter list in general purpose register 1.

Table 13. GETC parameter list

```
PARMLIST DS     0F
         DC     A(GETC)              Function code
         DC     A(RETCODE)           Return code
         DC     A(RSNCODE)           Reason code
         DC     A(WORKMOD)           Workmod token
         DC     A(CULIST)            Compile unit list
         DC     A(BUFFER)            Data buffer
         DC     A(CURSOR)            Starting position
         DC     A                    Data count
                (COUNT+X'80000000')
GETC     DC     H'64'                GETC function code
         DC     H'6'                 Interface version number
```

__iew_getC() - Get compile unit list

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Obtains source information about the program and the compile units from which it was constructed.

Format

```
#include <__iew_api.h>
int __iew_getC(_IEWAPIContext *__context,
                          int __culist[],
                          _IEWCUIEntry ** __cui_entry);
```

Parameters Descriptions

__context
API context is created and returned by calling __iew_openW() and is used throughout the open session.

__culist
compile unit list - array of compile units where the first entry is used to specify the total number of compile unit entries. If the first entry is non zero, then one record for each compile unit in a list of compile units will be returned. If the first entry is zero, then one record of each of all compile units will be returned.

__cui_entry
returned buffer - CUI entry, one record for each compile unit in a list of compile units is returned.

Returned Value

If successful, __iew_getC() returns > 0 (count, number of entries returned in the buffer).

If unsuccessful, __iew_getC() returns 0.

Utilities Functions

__iew_eod()
__iew_get_reason_code()
__iew_get_return_code()
__iew_get_cursor()
__iew_set_cursor()

__iew_fd_getC() - Get compile unit list

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Obtains source information about the program and the compile units form which it was constructed. The program object source "header record" returned by fast data in the CUI buffer for a program object in a PDSE will never identify the data set containing the object.

Format

```
#include <__iew_api.h>
int __iew_fd_getC(_IEWFDContext *__context,
                                int __culist[],
                                _IEWCUIEntry ** __cui_entry);
```

Parameters Descriptions

__context
FD context is created and returned by calling __iew_fd_open() and is used throughout the open session.

__culist
compile unit list - array of compile units where the first entry is used to specify the total number of compile unit entries. If the first entry is non zero, then one record for each compile unit in a list of compile units will be returned. If the first entry is zero, then one record of each of all compile units will be returned.

__cui_entry
returned buffer - CUI entry, one record for each compile unit in a list of compile units will be returned.

Returned Value

If successful, __iew_fd_getC() returns > 0 (count, number of entries returned in the buffer).

If unsuccessful, __iew_fd_getC() returns 0.

Utilities Functions

__iew_fd_eod()
__iew_fd_get_reason_code()
__iew_fd_get_return_code()
__iew_fd_get_cursor()
__iew_fd_set_cursor()

GC - Getting Compile unit information

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Compile unit information is primarily data passed to the binder by compilers, identifying the source of each program making up the program object being inspected. As an important special case, though, the first compile unit entry returned when the cursor is specified as zero provides information on the DASD location of the program object itself. The program object source "header record" returned by fast data in the CUI buffer for a program object in a PDSE will never identify the data set containing the object.

When no culist is provided, the cursor is an index into an ordered list of all CUI entries that can be returned. If the application does not modify the cursor during the retrieval process, multiple calls return all CUI records in the order by CU number because the buffer is full. When the culist is provided, the cursor is an index into that application-provided list. CUI records are returned in the order specified in the culist. If the application still does not modify the cursor during the retrieval process, multiple calls continue with subsequent entries in the list because the buffer is full. End of data is signalled when the end of the application-provided list is reached.

Table 39. GC parameter list

Parameter	Usage	Format	Content
1	in	structure	'GC', X'0001'
2	in	binary word	mtoken
3 (optional)	in	binary words	culist An array of numbers. The first word is the number of additional words that follow it. Each additional word is a compile unit number returned in BNL_SECT_CU by a 'GN' call. If no compile unit numbers are passed, the first (and only) word must be zero.
4	in/out	structure	buffer Must be a CUI buffer formatted by IEWBUFF or as defined in Binder API buffer formats.
5	in/out	binary word	cursor Cursor is an index within the culist or into an ordered list of all CUI entries.
6	out	binary word	count The number of CUI records returned by the binder.

Sample assembler code

```
         CALL  (15),(GCIL,MTOKEN,NULL,BUFF,CURS,CNT),VL

GCIL     DC    C'GC',X'0001'
MTOKEN   DS    F             As set at Start call
NULL     DC    F'0'          Omitted to get all CU's
CURS     DC    F'0'          Start with PO information
CNT      DS    F             Number of records returned
BUFF  IEWBUFF  FUNC=MAPBUFF,TYPE=CUI,VERSION=6,SIZE=2000
```

SETO: Set option

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

SETO specifies options for processing and module attributes. Each option is set at either the dialog or workmod level by providing a token in the call. The options that can be specified are listed in Setting options with the regular binder API.

The syntax of the SETO call is:

[*symbol*]	**IEWBIND**	**FUNC=SETO** [**,VERSION**=*version*] [**,RETCODE**=*retcode*] [**,RSNCODE**=*rsncode*] [**,WORKMOD**=*workmod*] [**,DIALOG**=*dialog*] **,OPTION**=*option* **,OPTVAL**=*optval* [**,PARMS**=*parms*]

FUNC=SETO
Specifies that you are requesting specific processing options or module attributes for a dialog or workmod.

VERSION=*1 | 2 | 3 | 4 | 5 | 6 | 7*
Specifies the version of the parameter list to be used. The default value is VERSION=1.
Note:
If VERSION=1 is specified for the SETO call, PARMS cannot be specified as a macro keyword. The parameter list ends with the OPTVAL parameter (with the high-order bit set). This exception is for version 1 only.

RETCODE=*retcode* — RX-type address or register (2-12)
Specifies the location of a fullword integer that is to receive the return code returned by the binder.

RSNCODE=*rsncode* — RX-type address or register (2-12)
Specifies the location of a 4-byte hexadecimal string that is to receive the reason code returned by the binder.

WORKMOD=*workmod* — RX-type address or register (2-12)
Specifies the location of an 8-byte area that contains the workmod token for this request. WORKMOD and DIALOG are mutually exclusive. To set the options at the workmod level, provide the WORKMOD token.

DIALOG=*dialog* — RX-type address or register (2-12)
Specifies the location of an 8-byte area that contains the appropriate dialog token. WORKMOD and DIALOG are mutually exclusive. To set the options at the dialog level, provide the DIALOG token.

OPTION=*option* — RX-type address or register (2-12)
Specifies the location of an 8-byte varying character string that contains an option keyword. Except for CALLIB, all keywords can be truncated to three characters. See Setting options with the regular binder API for a complete list of keywords.

OPTVAL=*optval* — RX-type address or register (2-12)
Specifies the location of a varying character string that contains a value or a list of values for the specified option.

PARMS=*parms* — RX-type address or register (2-12)
Specifies the location of a varying character string that contains a list of option specifications separated by commas.

Processing notes

Option values are coded as *value* or *(value1,value2)*. A list of values is enclosed in parentheses. A value containing special characters is enclosed in single quotation marks. An imbedded single quotation mark is coded as two consecutive single quotation marks. Special characters include all EBCDIC characters other than upper and lower case alphabetics, numerics, national characters (@ # $), and the underscore. YES and NO values can be abbreviated Y and N, respectively.

Options specified for a workmod override any corresponding options specified for that dialog. Options specified at the dialog level override the corresponding system defaults, and apply to all workmods within the dialog unless overridden. If INTENT=ACCESS, these keywords are not allowed: ALIGN2, CALL, CALLIB, EDIT, LET, MAP, OVLY, RES, TEST, XCAL, and XREF.

The options list specified in the PARMS= parameter is a character string identical to the PARM= value defined in the "Binder options reference" chapter of z/OS MVS Program Management: User's Guide and Reference, with the following restrictions:

- The list is not enclosed with apostrophes or parentheses
- Environmental options cannot be specified on SETO. See the list of environmental options in Setting options with the binder API
- The EXITS and OPTIONS options are also not allowed in this list.

The OPTION and OPTVAL operands are used together to specify a single option and its value.

- None of the environmental options can be specified. See Setting options with the binder API.
- The following invocation options may not be specified on the OPTION/OPTVAL operands of SETO because they are really mapped to something different: EXITS, OPTIONS, REFR, RENT, and the YES, NO, or default values for REUS.
- The negative option format (for example, NORENT) is not allowed. The corresponding option with a value must be used (for example, OPTION=REUS,OPTVAL=SERIAL).
- An option specified using the OPTION and OPTVAL operands overrides any value for that same option specified within the PARMS operand.

You can specify a z/OS UNIX System Services file as the CALLIB parameter value on a SETO call.

Return and reason codes

The common binder API reason codes are shown in Table 3.

Return Code	Reason Code	Explanation
00	00000000	Normal completion.
08	83000109	One or more options designated as environmental have been specified on SETO. Option ignored.
12	83000100	Neither dialog token nor workmod token were specified. Request rejected.
12	83000106	The option specified is invalid for a workmod specified with INTENT=ACCESS. Request rejected.
12	83000107	Invalid option keyword specified. Request rejected.
12	83000108	The option value is invalid for the specified keyword. Request rejected.
12	83000113	An option you specified is valid only for the STARTD function. The request is rejected.

Parameter list

If your program does not use the IEWBIND macro, place the address of the SETO parameter list in general purpose register 1.

Table 27. SETO parameter list

```
PARMLIST DS 0F
         DC A(SETO)              Function code
         DC A(RETCODE)           Return code
         DC A(RSNCODE)           Reason code
         DC A(DIALOG)            Dialog token
         DC A(WORKMOD)           Workmod token
         DC A(OPTION)            Option keyword
         DC A(OPTVAL)            Option value
         DC A(PARMS)             Options list
SETO     DC H'20'                SETO function code
         DC H'version'           Interface version number
```

Note:

The PARMS parameter is an addition for Version 2 and X'80000000' must be added to either the OPTION parameter (for Version 1) or the PARMS parameter.

__iew_setO() - Set option

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Set binder options to be used for subsequent processing. See Setting options with the regular binder API for details. Environment options cannot be specified here.

Format

```
#include <__iew_api.h>
int __iew_setO(_IEWAPIContext *__context,
                          const char *__parms);
```

Parameters Descriptions

__context
API context is created and returned by calling __iew_openW() and is used throughout the open session.

__parms
list of binder options.

Returned Value

If successful, __iew_setO() returns 0.

If unsuccessful, __iew_setO() returns nonzero.

Note:
The returned value is the same as the code returned by a subsequent __iew_get_return_code().

Utilities Functions

__iew_get_reason_code()
__iew_get_return_code()

Binder options reference

z/OS MVS Program Management: User's Guide and Reference
SA22-7643-11

Guideline: This topic refers to binder processing. These concepts apply equally to linkage editor and batch loader processing, unless noted otherwise in Processing and attribute options reference. The linkage editor and batch loader cannot process program objects.

This section describes the processing and attribute options that can be requested. Binder options are specified in a number of ways. These are broadly classified as interfaces that pass option strings and interfaces that have tailored option capabilities.

The following interfaces pass option strings:

- The PARM field of the JCL EXEC statement
- The first parameter passed to
  - IEWBLINK
  - IEWBLOAD
  - IEWBLODI or IEWBLDGO

  when using CALL, LINK, ATTACH, or XCTL from another program
- An options file identified by the OPTIONS option
- An options file specified by the DD name IEWPARMS
- The SETOPT control statement
- Installation option defaults
- The PARMS parameter of the IEWBIND FUNC=STARTD or FUNC=SETO call.

The following interfaces have tailored option capabilities:

- Arguments passed to the TSO LINK or LOADGO commands
- Arguments passed to the z/OS UNIX System Services c++, c89, cc, or ld commands
- The OPTIONS parameter of the IEWBIND FUNC=STARTD call
- The OPTION and OPTVAL parameters of the IEWBIND FUNC=SETO call.

Note:
IEWBIND is fully documented in z/OS MVS Program Management: Advanced Facilities

Many options have the possible values YES and NO. These options usually have an associated option that begins with *N* or *NO*. For example, you can specify MAP to produce a module map, and NOMAP to suppress production of a module map. You can also specify the MAP option as **MAP**=*YES* or **MAP(***YES***)** and **MAP**=*NO* or **MAP(***NO***)**. Table 7 shows the associated negative option if the option's values are *YES* and *NO*.

The options you specify, through any means, when invoking the binder, always override similar data from included modules. For example, if you specify **PARM**=*RENT*, the resultant module is marked "reentrant" regardless of the reusability of any included modules.

If more than one output module is produced by a single binder instance, the options specified will apply to all output modules, unless overridden by a SETOPT control statement, or IEWBIND FUNC=SETO call.

Options precedence rules (low to high)

1. Installation options from IEWBODEF
2. Primary invocation options, from one of the following:
   1. The PARM field of the JCL EXEC statement
   2. The first parameter passed to IEWBLINK, IEWBLOAD, etc.
   3. The PARMS parameter of IEWBIND FUNC=STARTD
3. ***The IEWPARMS DD statement – introduced in z/OS V1R11 !***
4. The OPTIONS parameter of IEWBIND FUNC=STARTD
5. IEWBIND_OPTIONS environment variables via the ENVARS parameter of IEWBIND FUNC=STARTD
6. Dynamic option changes from either:
   1. Options set from attributes by an INCLUDE -ATTR control statement or
   2. The SETOPT control statement, or
   3. The PARMS parameter, followed by the OPTION/OPTVAL parameter, of IEWBIND FUNC=SETO

__iew_api_name_to_str() - Convert API name into string

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

Data buffers returned by the binder and fast data APIs often contain pointers to names. Those are character data, but not stored as C null-delimited strings. Instead they have a separate length field in the buffer structure. This function returns a C string equivalent to the name returned by the API.

The function also provides special handling for binder-generated names, which are returned as binary numbers. The function will convert those to special displayable strings which will be recognized and automatically reconverted by the functions in this suite if they are passed back to the API later.

Format

```
#include <__iew_api.h>
void __iew_api_name_to_str(const char *__name,
                                                short __len,
                                                char *__str);
```

Parameters Descriptions

__name
input: varying string characters.
__len
input: varying string size.
__str
output: string.
Note:
You need to allocate storage for string. The length should be the greater of 10 and input len+1.

Returned Value

None.

__iew_create_list() - Create list

z/OS MVS Program Management: Advanced Facilities
SA22-7644-14

The binder's lists consist of a fullword count of the number entries followed by the entries. Each list entry contains an 8-byte name, a fullword containing the length of the value string, and a 31-bit pointer to the value string. The __iew_create_list() function creates a list of keywords and values in “binder list format". It is used to support file lists and exit lists for the __iew_openW() call. There are two types of lists:

1. A list of DD names with keywords being any of the standard binder DD names as strings, and values being string replacement names to use for them.
2. A list of exit routines with keywords being any of the strings “MESSAGE", “INTFVAL", or “SAVE", and values being an array of three pointers, to:
    - exit routine entry point
    - application data passes to the exit
    - message exit severity (unused, but must be provided as zero, for the other two exits)

Format

```
#include <__iew_api.h>
_IEWList *__iew_create_list(int __size,
                                                char *__keys[],
                                                void *__values[]);
```

Parameters Descriptions

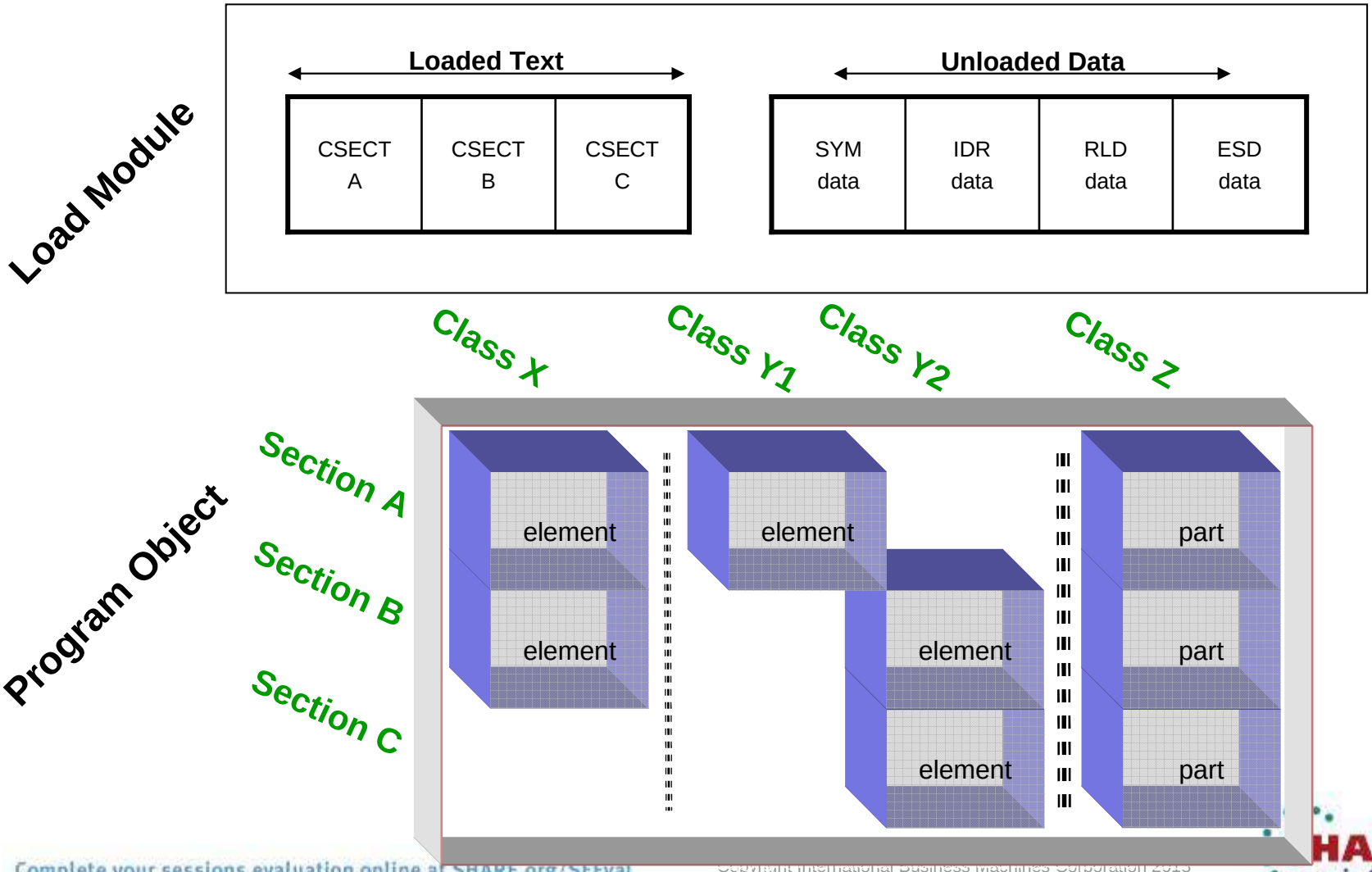
__size
input: size of the list.
__keys
input: list of keywords.
__values
input: list of values.

Returned Value

If successful, __iew_create_list() returns API list.

If unsuccessful, __iew_create_list() returns null.

load module vs. program object



What else comes with the binder? Binder APIs …