IBM Education Assistance for z/OS V2R1

Item: Boundary Alignment
Element/Component: Binder

Material is current as of June 2013

Agenda

- Trademarks
- Presentation Objectives
- Overview
- Usage & Invocation
- Interactions & Dependencies
- Migration & Coexistence Considerations
- Presentation Summary
- Appendix

Trademarks

- See url http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.

Presentation Objectives

- What does boundary alignment do?
- How do I used boundary alignment?
- Are there boundary alignment compatibility issues?

Overview

- In the beginning load modules build from object modules were composed of sections that were always double-word aligned. The ability to page-align a section at bind time was also provided. Later the ability to have quad-word align sections in objects was added. When GOFF objects were introduced in the early 1990s they had architected the ability to define every alignment from byte to 2GB, but it was never fully exploited.
- Problem Statement / Need Addressed
  - Customers want to be able to align sections within a load module to other specified values
- Solution
  - The binder can now respect alignment information to allow customers more granularity in aligning the sections in their modules
- Benefit / Value
  - By aligning sections more granularly, customers may gain benefits of usability and possibly performance, without needlessly wasting space

Overview …

- This support comes in 3 parts
  - ALIGNT control statement (new)
    - For binding load modules and program objects
  - Alignment attributes coming from ESD records
    - From GOFF objects and program objects
  - Updates to APIs
    - ALIGNT regular binder API new VERSION=8
    - __iew_alignT2 new C/C++ API (__iew_alignT is unchanged)

Overview -- Background

- The generalization of a load module is a program object . Where load modules have just CSECTs, program objects have sections and classes, the cross-section of which is called an element.
- The default class is named B_TEXT, so you can think consider your CSECT *name* as being roughly the same as class B_TEXT in section *name*.
- The generalization of a pseudo-register is a part (the major difference is that a part can have initialization data)
- GOFF objects are required in order to define classes/sections aligned on other than double-word or quad-word boundary, and parts

Overview – Background …

- In order for the binder to align a section:
  - The class is given the most restrictive alignment (largest value) of all the sections in that class
    - The alignment attribute belongs to each class
  - The segment is given PAGE alignment if any classes are aligned at more than quad-word
    - For load modules the segment is the entire loaded module and there are no classes

Usage & Invocation – ALIGNT control statement

- ALIGNT is a new binder control statement, which is a generalization of the PAGE control statement

```
ALIGNT boundary,sectionname[(classname1 [,classname2]...)]
```

Usage & Invocation – ALIGNT control statement …

- Like PAGE and other binder control statements, ALIGNT caused a non-permanent change
  - You need to use it every time you (re-)bind your program
- ALIGNT is different than PAGE in these ways:
  - You can specify the alignment boundary
  - You can only specify a single section name on each statement
  - You can optionally specify one or more class names for that section
  - There is no interaction with the ALIGN2 binder option

Usage & Invocation – ALIGNT control statement …

- Some ALIGNT rules
  - The boundary can be 0-4096 and must be a power of 2; 0 means "reset", as if ALIGNT was never specified for that section (and classes)
  - ALIGNT for a section *name* with no classes affects *all* the classes, except merge classes (PAGE likewise excludes merge classes)
    - To use merge classes they must be explicitly specified
    - Only merge classes of parts are allowed, pseudo-registers are not
  - ALIGNT for the same section *name* but with classes is subtractive from the earlier specification with no classes (so multiple statements can be used)

Usage & Invocation – ALIGNT control statement …

- Like PAGE, because ALIGNT is not changing the ESD alignments
  - You only see the affect they have, so you see no changed alignment values in:
    - AMBLIST output for load modules
    - AMBLIST MODLIST output for program objects
  - You *do* see their alignment values in:
    - binder MODULE MAP
    - AMBLIST SEGMENT MAP TABLE for program objects

Usage & Invocation – ALIGNT control statement …

- Simple example:
  - Two object modules OBJ1 and OBJ2
  - Each happen to each be x'18' bytes long
  - INCLUDEd in that order
  - Bind into a load module (in a PDS):

Usage & Invocation – ALIGNT control statement …

- Simple example – no PAGE or ALIGNT:
  - Binder MODULE MAP

```
---------------
CLASS   B_TEXT            LENGTH =       30   ATTRIBUTES = CAT,   LOAD, RMODE= 24
                          OFFSET =        0 IN SEGMENT 001      ALIGN = DBLWORD
---------------
 SECTION     CLASS                                                 ------- SOURCE --------
  OFFSET     OFFSET   NAME                 TYPE      LENGTH   DDNAME    SEQ   MEMBER
                0     OBJ1                 CSECT         18   OBJS       01   **NULL**
               18     OBJ2                 CSECT         18   OBJS       01   **NULL**
```

  - AMBLIST MODLIST

```
RECORD# 1      TYPE 20 - CESD      ESDID 1                          ESD SIZE 32
                CESD#     SYMBOL     TYPE      ADDRESS      R/R/A     ID/LENGTH(DEC)    (HEX)
                   1      OBJ1       00(SD)     000000       00              24          18
                   2      OBJ2       00(SD)     000018       00              24          18
```

Usage & Invocation – ALIGNT control statement …

- Simple example – **PAGE OBJ2** control statement
  - Binder MODULE MAP

```
---------------
CLASS   B_TEXT            LENGTH =      1018  ATTRIBUTES = CAT,   LOAD, RMODE= 24
                          OFFSET =         0 IN SEGMENT 001     ALIGN = PAGE
---------------
 SECTION     CLASS                                                 ------- SOURCE --------
  OFFSET    OFFSET  NAME                 TYPE     LENGTH  DDNAME    SEQ   MEMBER
                 0  OBJ1                 CSECT        18  OBJS       01   **NULL**
              1000  OBJ2                 CSECT        18  OBJS       01   **NULL**
```

  - AMBLIST MODLIST

```
                    ***PAGE ALIGNMENT REQUIRED
...
RECORD# 1      TYPE 20 - CESD      ESDID 1                          ESD SIZE 32
                CESD#     SYMBOL     TYPE      ADDRESS      R/R/A     ID/LENGTH(DEC)    (HEX)
                   1     OBJ1       00(SD)      000000       00              24          18
                   2     OBJ2       00(SD)      001000       00              24          18
```

Usage & Invocation – ALIGNT control statement …

- Simple example – **ALIGNT 32,OBJ2** control statement

  – Binder MODULE MAP

```
---------------
CLASS   B_TEXT            LENGTH =       38  ATTRIBUTES = CAT,   LOAD, RMODE= 24
                          OFFSET =        0 IN SEGMENT 001     ALIGN = 32 BYTE
---------------
 SECTION     CLASS                                                  ------- SOURCE --------
  OFFSET     OFFSET   NAME                 TYPE     LENGTH  DDNAME    SEQ   MEMBER
                0     OBJ1                 CSECT        18  OBJS       01   **NULL**
               20     OBJ2                 CSECT        18  OBJS       01   **NULL**
```

  – AMBLIST MODLIST

```
                    ***PAGE ALIGNMENT REQUIRED
...
RECORD# 1      TYPE 20 - CESD      ESDID 1                          ESD SIZE 32
                CESD#     SYMBOL     TYPE      ADDRESS      R/R/A     ID/LENGTH(DEC)    (HEX)
                   1      OBJ1       00(SD)     000000       00              24          18
                   2      OBJ2       00(SD)     000020       00              24          18
```

Usage & Invocation – Alignment attributes coming from ESD records

- Without this support the binder changes class alignments on input from GOFF objects, to the next most restrictive alignment, supporting only:
  - Double-word
  - Quad-word
  - Page
- To tell binder to not change on input other alignment values, you must create a COMPAT(ZOSV2R1) or COMPAT(CURRENT) program object
  - Now all alignments up to 4096 (PAGE) can be input -- except
    - Alignments less than double-word are still changed to double-word
      - Otherwise assumptions made by assemblers & compilers might be invalid

Usage & Invocation – Alignment attributes coming from ESD records …

- Part alignments may now be any value up to 4096 (PAGE)
  - Previously values above quad-word could not be used
  - As always, parts must be aligned no more restrictively than the containing class

Usage & Invocation – Alignment attributes coming from ESD records …

- High Level Assembler has three ways to set alignments
  - The natural alignment of the type. For example this aligns a pseudo-register on a double-word boundary:

```
alignDW  DXD   0D
```

  - The SECTALGN option to align sections (and the default for all classes). Unless the GOFF is used, only double-word (default) and quad-word sections can be created:

```
*PROCESS  SECTALGN(32)
```

  - The CATTR statement ALIGN keyword to align a class; if a merge class then it also aligns all the parts within it:

```
MYPARTS  CATTR PART(partQW),ALIGN(4),RMODE(31)
```

Usage & Invocation – Alignment attributes coming from ESD records …

- Simple example
  - Identical to ALIGNT example except
    - High Level Assembler options used are for “octa-word” alignment
      - GOFF,SECTALGN(32)
    - Program is saved as a program object (PDS/E or UNIX filesystem)

Usage & Invocation – Alignment attributes coming from ESD records …

- Simple example – COMPAT(MIN)
  - Binder default: results unchanged from prior releases
  - Binder MODULE MAP

```
---------------
CLASS   B_TEXT              LENGTH =      1018  ATTRIBUTES = CAT,   LOAD, RMODE= 24
                            OFFSET =         0 IN SEGMENT 001     ALIGN = PAGE
---------------
 SECTION     CLASS                                                 ------- SOURCE --------
  OFFSET    OFFSET  NAME                  TYPE     LENGTH  DDNAME    SEQ   MEMBER
                 0  OBJ1                  CSECT        18  OBJS       01   **NULL**
              1000  OBJ2                  CSECT        18  OBJS       01   **NULL**
```

  - Binder calculated the class alignments based on the ESDs
    - There was no ALIGNT override
  - Without SECTALGN we would have seen ALIGN = DBLWORD

Usage & Invocation – Alignment attributes coming from ESD records …

- Simple example – COMPAT(MIN) …
  - Binder default: results unchanged from prior releases
  - AMBLIST MODLIST

```
                ***PAGE ALIGNMENT REQUIRED
. . .
           CONTROL SECTION:  OBJ1
B_TEXT(ED)
    CLASS:            B_TEXT      LENGTH:          18 (HEX)      CLASS OFFSET:          0 (HEX)
    NAME SPACE:            1      ALIGNMENT:          PAGE      BIND METHOD:        CATENATE
    TEXT                          LOAD                          FILL:                 UNSPEC
           CONTROL SECTION:  OBJ2
B_TEXT(ED)
    CLASS:            B_TEXT      LENGTH:          18 (HEX)      CLASS OFFSET:       1000 (HEX)
    NAME SPACE:            1      ALIGNMENT:          PAGE      BIND METHOD:        CATENATE
    TEXT                          LOAD                          FILL:                 UNSPEC
```

  - Binder permanently changed ESDs from ALIGN(5) to ALIGN(12)
    - From SECTALGN(32) to SECTALGN(4096)
  - Without SECTALGN default ALIGN(3) would be unchanged

Usage & Invocation – Alignment attributes coming from ESD records …

- Simple example – COMPAT(MIN) …
  - Binder default: results unchanged from prior releases
  - AMBLIST SEGMENT MAP TABLE

CLASS	SEGMENT	OFFSET	LENGTH	LOAD	TYPE	ALIGNMENT	RMODE
B_TEXT	1	0	1018	INITIAL	CAT	PAGE	24

  - Binder calculated the class alignment based on the ESDs
    - There was no ALIGNT override

Usage & Invocation – Alignment attributes coming from ESD records …

- Simple example – COMPAT(ZOSV2R1) / COMPAT(CURR)
  - Binder preserves input ESD alignments
  - Binder MODULE MAP

```
---------------
CLASS   B_TEXT             LENGTH =       38  ATTRIBUTES = CAT,   LOAD, RMODE= 24
                           OFFSET =        0 IN SEGMENT 001     ALIGN = 32 BYTE
---------------
 SECTION     CLASS                                                 ------- SOURCE --------
  OFFSET    OFFSET  NAME                TYPE     LENGTH  DDNAME    SEQ   MEMBER
                 0  OBJ1                CSECT        18  OBJS       01   **NULL**
                20  OBJ2                CSECT        18  OBJS       01   **NULL**
```

  - Binder calculated the class alignments based on the ESDs
    - There was no ALIGNT override

Usage & Invocation – Alignment attributes coming from ESD records …

- Simple example – COMPAT(ZOSV2R1) / COMPAT(CURR) …
  - Binder preserves input ESD alignments
  - AMBLIST MODLIST

```
                ***PAGE ALIGNMENT REQUIRED
. . .
          CONTROL SECTION:  OBJ1
B_TEXT(ED)
    CLASS:            B_TEXT      LENGTH:          18 (HEX)      CLASS OFFSET:          0 (HEX)
    NAME SPACE:            1      ALIGNMENT:        32 BYTE      BIND METHOD:        CATENATE
    TEXT                          LOAD                           FILL:                 UNSPEC
          CONTROL SECTION:  OBJ2
B_TEXT(ED)
    CLASS:            B_TEXT      LENGTH:          18 (HEX)      CLASS OFFSET:         20 (HEX)
    NAME SPACE:            1      ALIGNMENT:        32 BYTE      BIND METHOD:        CATENATE
    TEXT                          LOAD                           FILL:                 UNSPEC
```

  - Binder preserved input ESD as ALIGN(5)
    - Coming from SECTALGN(32)

Usage & Invocation – Alignment attributes coming from ESD records …

- Simple example – COMPAT(ZOSV2R1) / COMPAT(CURR) …
  - Binder preserves input ESD alignments
  - AMBLIST SEGMENT MAP TABLE

CLASS	SEGMENT	OFFSET	LENGTH	LOAD	TYPE	ALIGNMENT	RMODE
B_TEXT	1	0	38	INITIAL	CAT	32 BYTE	24

  - Binder calculated the class alignment based on the ESDs
    - There was no ALIGNT override

Usage & Invocation – APIs

- ALIGNT VERSION=8

```
FUNC=ALIGNT
    ,WORKMOD=workmod
    ,SECTION=section
    [,BDY=bdy]
    [,CLASSL=classl]
```

__iew_alignT2

```
#define  _IEW_TARGET_RELEASE IEW_ZOSV2R1_
#include <__iew_api.h>

int __iew_alignT2(
    _IEWAPIContext *__context,
    unsigned int __bdy,
    const char *__section,
    char * __class[]);
```

Interactions & Dependencies

- If a program is bound using COMPAT(ZOSV2R1) / COMPAT(CURR) to preserve alignment information and then saved to a load module (PDS)
  - The load module will be properly aligned however the preserved alignment information cannot be represented in a load module

```
IEW2536I 5257 ESD ALIGNMENT FOR SYMBOL symbol
CHANGED FROM  4096 TO  16.
```

  - Rebinding the load module with the now changed alignment may not produce the desired results

- There are no loader dependencies because this line item only supports more granular alignment within the module. The loader still only supports double-word and page alignment of programs (and segments).

Migration & Coexistence Considerations

- If a program objects is built COMPAT(ZOSV2R1) to preserve alignment information
  - This is a COMPAT sub-level which is compatible with releases back to z/OS V1R8. Programs bound with this option:
    - Can be loaded (executed) on any MVS system down z/OS V1R8
    - Cannot be inspected or reprocessed on any MVS system prior to z/OS V2R1
      - No rebind
      - No AMBLIST
      - No ZAP
      - etc.

Presentation Summary

- What boundary alignment does
- Ways to use boundary alignment
- Compatibility of programs taking advantage of boundary alignment

Appendix

- z/OS MVS Program Management: User's Guide and Reference – SA22-7643
  - Binder options and control statements
- z/OS MVS Program Management: Advanced Facilities – SA22-7644
  - Binder APIs
- z/OS V1R13.0 MVS Diagnosis: Tools and Service Aids – GA22-7589
  - AMBLIST
- z/OS V1R13.0 MVS System Messages, Vol 8 (IEF-IGD) – SA22-7638
  - IEW2001 - 2999