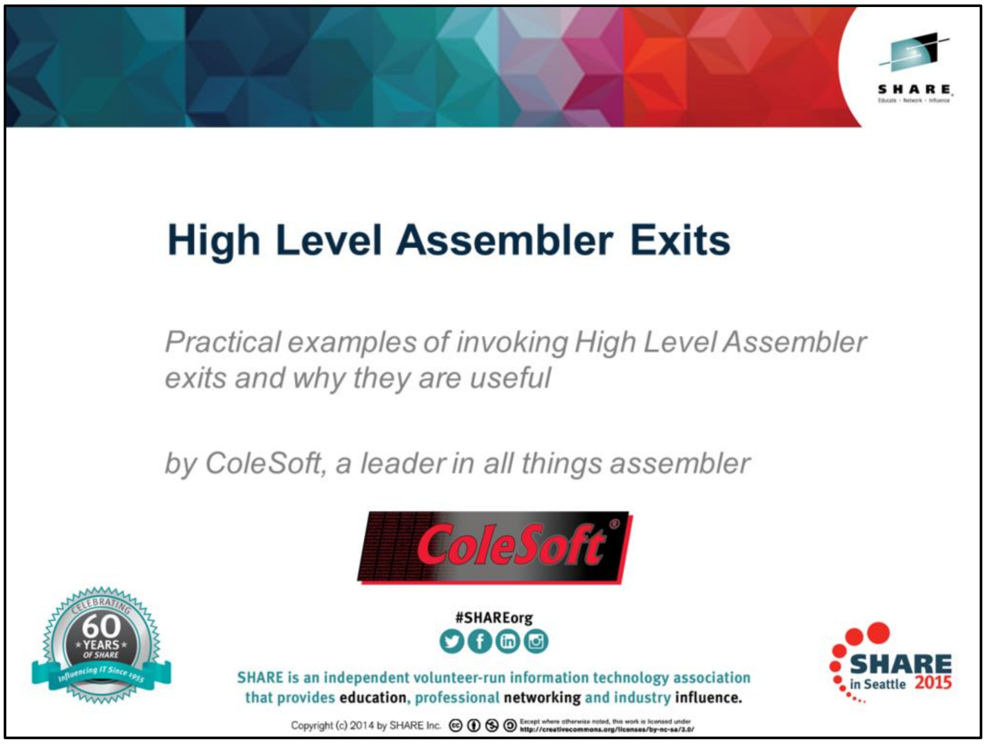


High Level Assembler Exits

Practical examples of invoking High Level Assembler exits and why they are useful

by ColeSoft, a leader in all things assembler

#SHAREorg

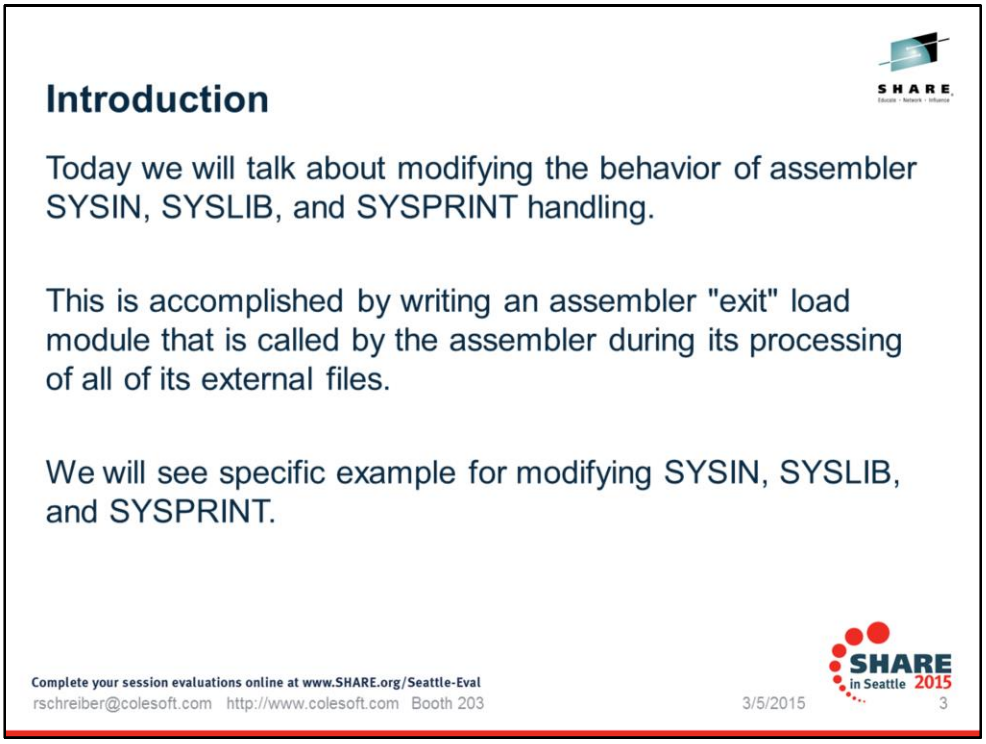
SHARE is an independent volunteer-run information technology association that provides education, professional networking and industry influence.

SHARE
Educate • Network • Influence
CELEBRATING 60 YEARS OF SHARE
Influencing IT Since 1955
SHARE in Seattle 2015

Thank you for your interest in this topic. I hope to provide you with a 50 minute informative and interesting presentation and at the end of the presentation we will have a Q&A period.

If you have more in-depth questions after the presentation please feel free to contact me at the e-E-Mail address below.

Lets begin…

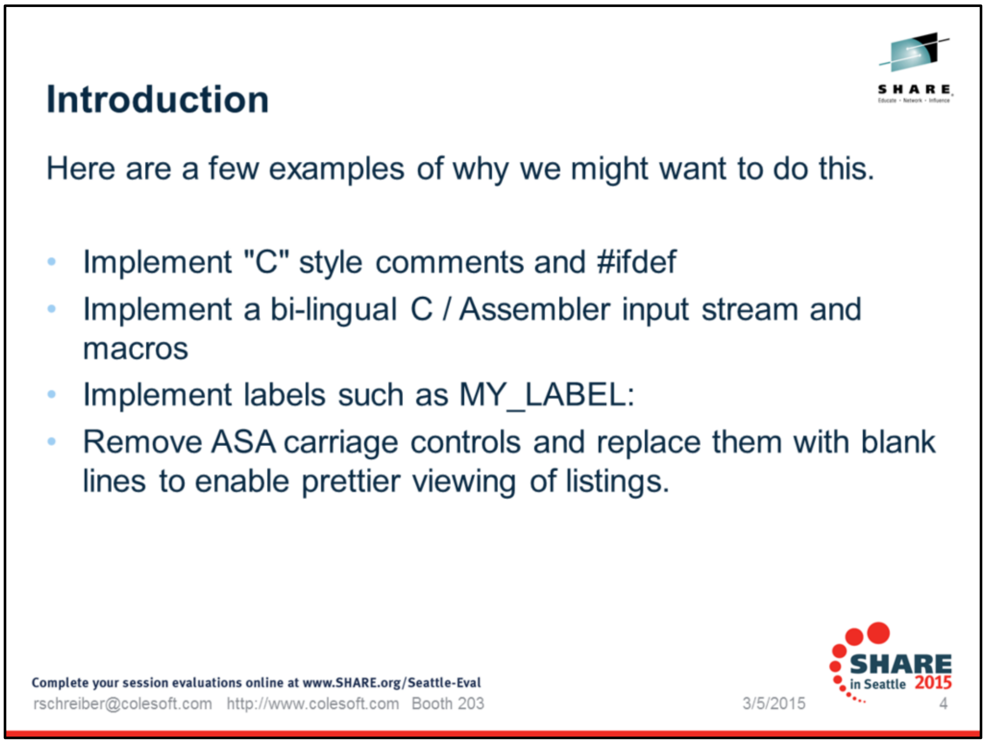


Introduction

Today we will talk about modifying the behavior of assembler SYSIN, SYSLIB, and SYSPRINT handling.

This is accomplished by writing an assembler "exit" load module that is called by the assembler during its processing of all of its external files.

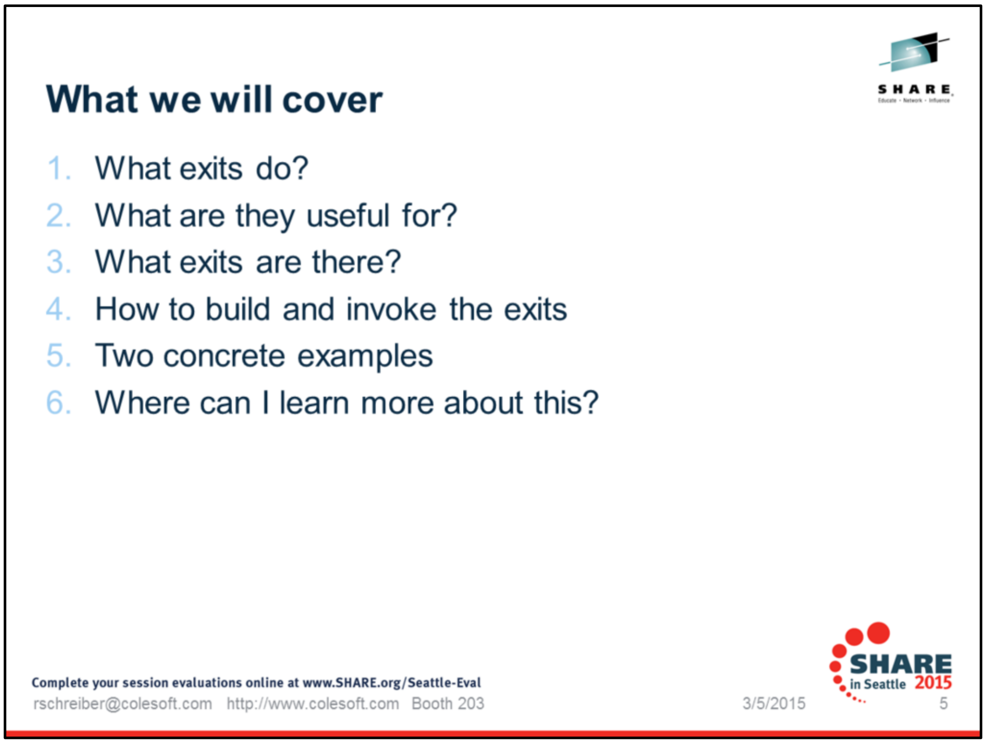
We will see specific example for modifying SYSIN, SYSLIB, and SYSPRINT.



Introduction

Here are a few examples of why we might want to do this.

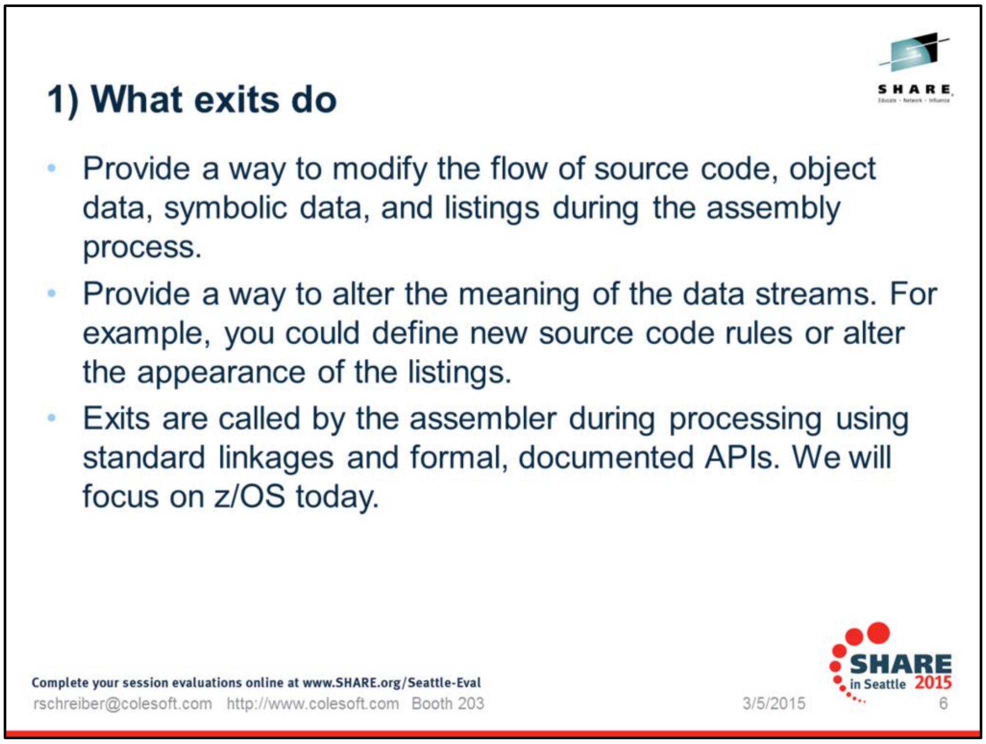
- Implement "C" style comments and #ifdef
- Implement a bi-lingual C / Assembler input stream and macros
- Implement labels such as MY_LABEL:
- Remove ASA carriage controls and replace them with blank lines to enable prettier viewing of listings.



What we will cover

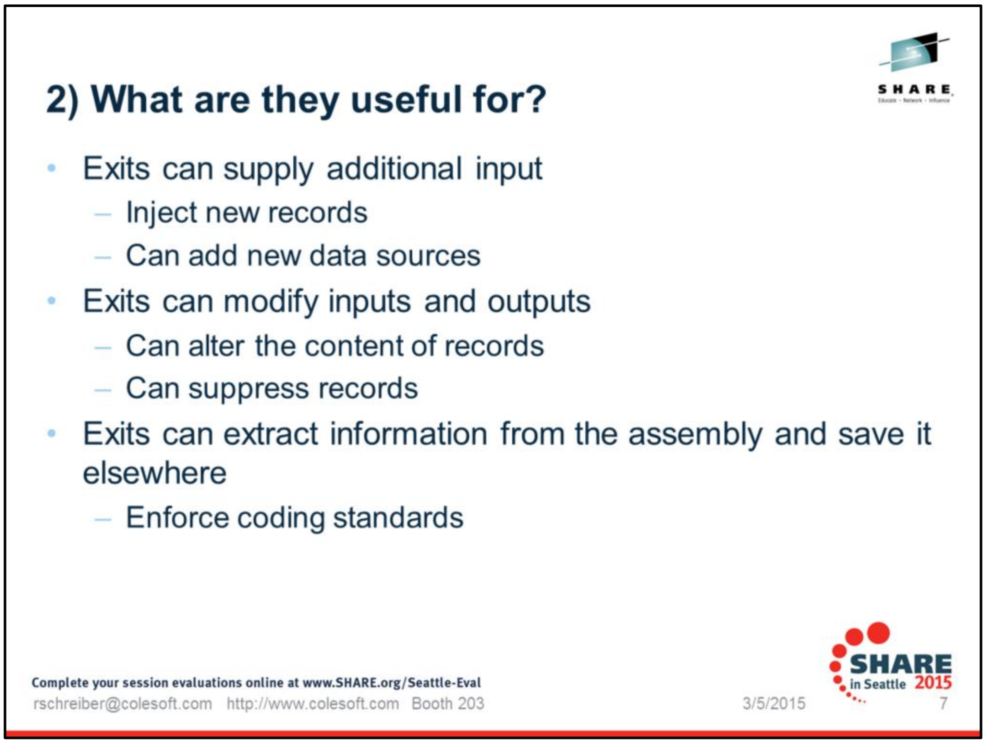
1. What exits do?
2. What are they useful for?
3. What exits are there?
4. How to build and invoke the exits
5. Two concrete examples
6. Where can I learn more about this?

[click to advance to each topic]



1) What exits do

- Provide a way to modify the flow of source code, object data, symbolic data, and listings during the assembly process.
- Provide a way to alter the meaning of the data streams. For example, you could define new source code rules or alter the appearance of the listings.
- Exits are called by the assembler during processing using standard linkages and formal, documented APIs. We will focus on z/OS today.



2) What are they useful for?

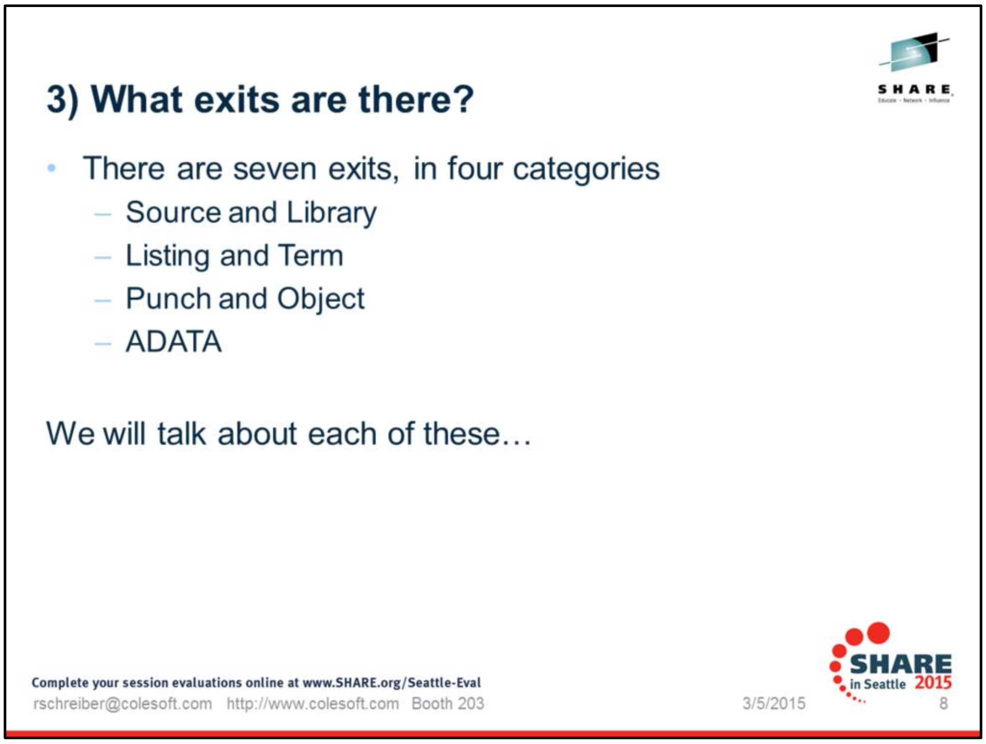
- Exits can supply additional input
  - Inject new records
  - Can add new data sources
- Exits can modify inputs and outputs
  - Can alter the content of records
  - Can suppress records
- Exits can extract information from the assembly and save it elsewhere
  - Enforce coding standards

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 7

Data sources may be from more than just SYSIN and SYSLIB.

Given appropriate calls, just about any data source can be accessed.



3) What exits are there?

- There are seven exits, in four categories
  - Source and Library
  - Listing and Term
  - Punch and Object
  - ADATA

We will talk about each of these…

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 8

There are seven exits.

The Source and Library exits provide the ability to control SYSIN and SYSLIB input.

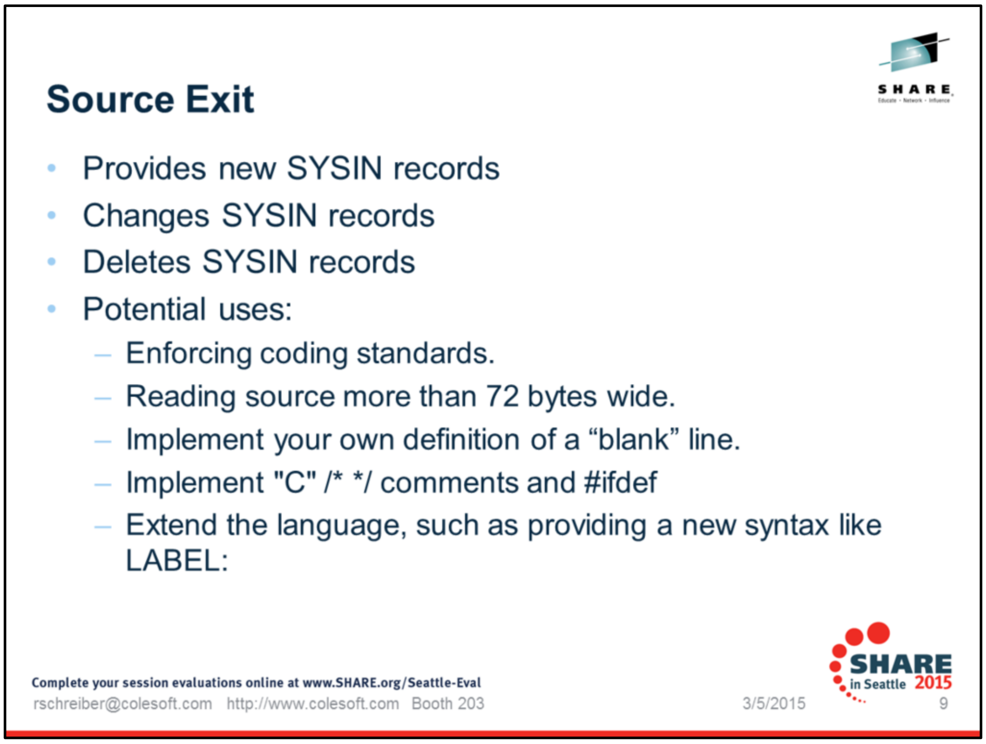
Listing and Term provide control over SYSPRINT and SYSTERM output.

Punch and Object provide control over SYSPUNCH and SYSLIN object deck creation.

ADATA provides control over SYSADATA creation.

In general, they can add, modify, delete, and extract information in all the assembler's data streams.

They can also provide alternate ways of processing the data, for example, by using different DCBs or DDNAMEs.



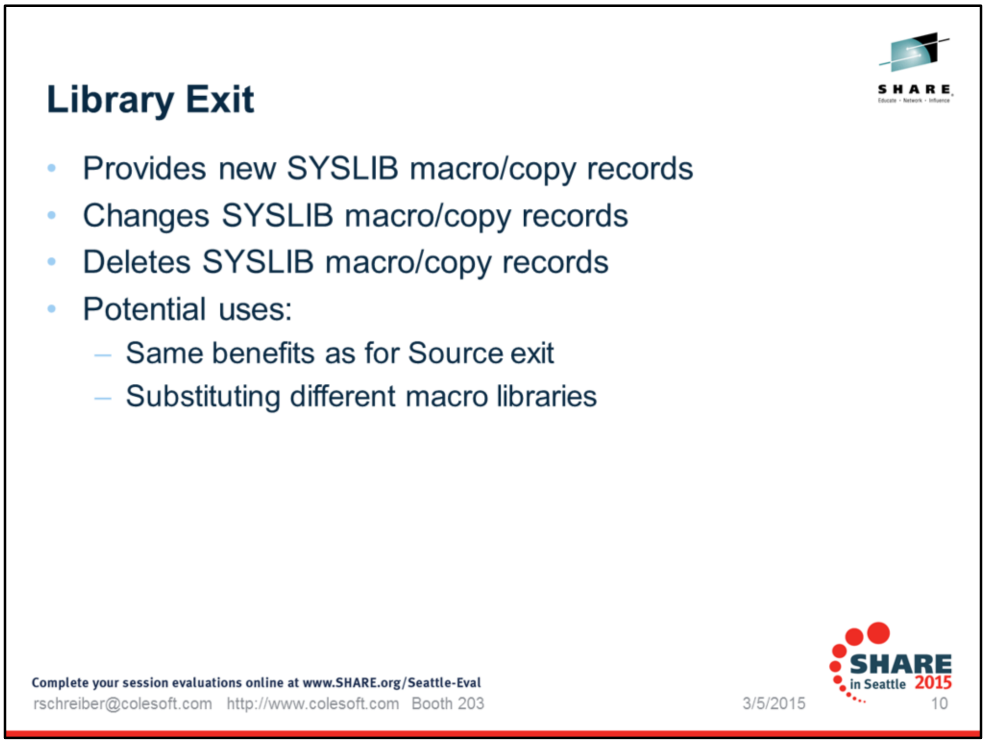
Source Exit

- Provides new SYSIN records
- Changes SYSIN records
- Deletes SYSIN records
- Potential uses:
  - Enforcing coding standards.
  - Reading source more than 72 bytes wide.
  - Implement your own definition of a “blank” line.
  - Implement "C" /* */ comments and #ifdef
  - Extend the language, such as providing a new syntax like LABEL:

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 9

Checking for programmer ID or control information out near column 60.
Absorb records from a VB dataset and convert them to RECFM=FB
Change a blank line to be blank in only columns 1-60, say
Implement bi-lingual C/ASM. Support multi-line /* */
Implement #ifdef, #ifndef, #else, #endif
Provide LABEL: SYSIN syntax

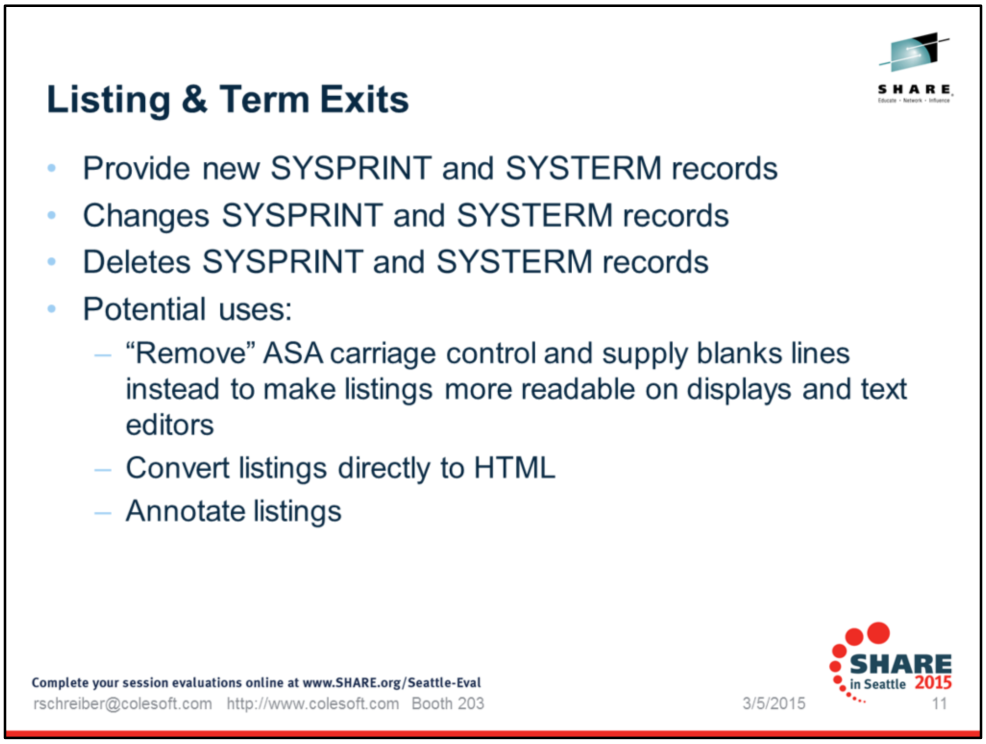


Library Exit

- Provides new SYSLIB macro/copy records
- Changes SYSLIB macro/copy records
- Deletes SYSLIB macro/copy records
- Potential uses:
  - Same benefits as for Source exit
  - Substituting different macro libraries

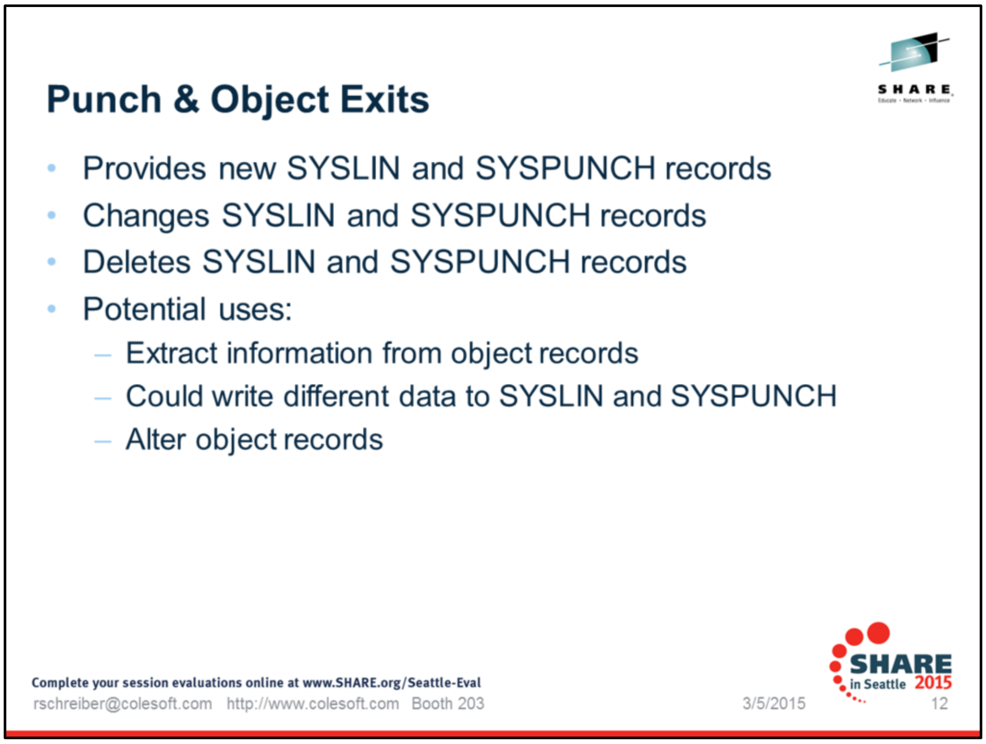
Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 10

Note that the Library exit is pretty much the same as the Source exit, but it gives the same controls over MACRO and COPY code inclusion.



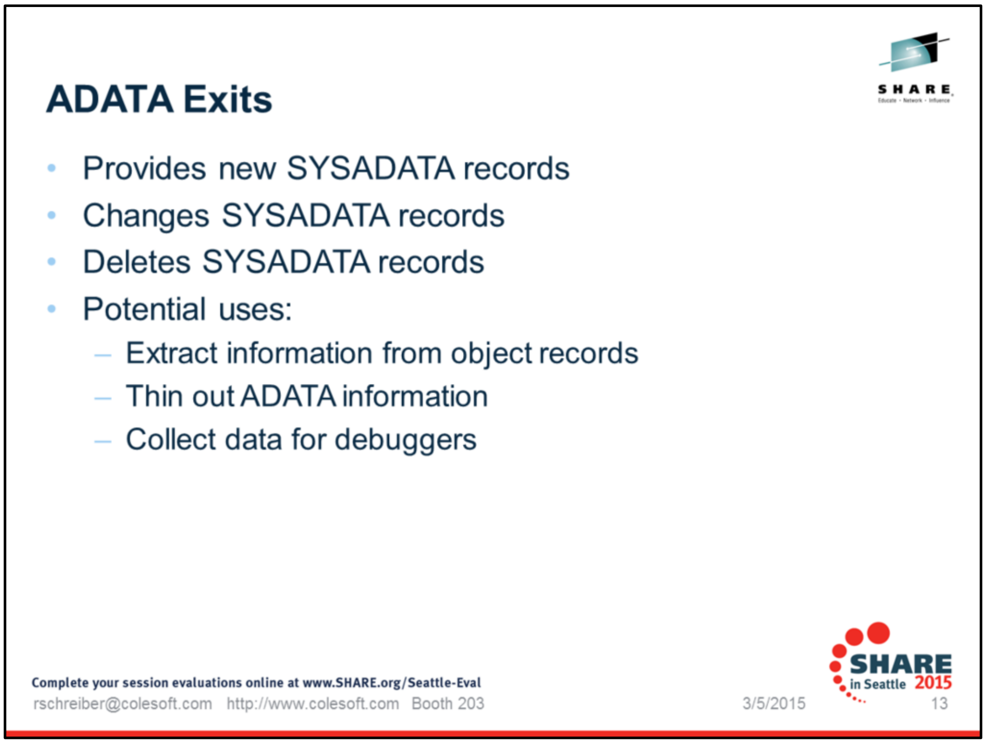
Listing & Term Exits

- Provide new SYSPRINT and SYSTERM records
- Changes SYSPRINT and SYSTERM records
- Deletes SYSPRINT and SYSTERM records
- Potential uses:
  - "Remove" ASA carriage control and supply blanks lines instead to make listings more readable on displays and text editors
  - Convert listings directly to HTML
  - Annotate listings



Punch & Object Exits

- Provides new SYSLIN and SYSPUNCH records
- Changes SYSLIN and SYSPUNCH records
- Deletes SYSLIN and SYSPUNCH records
- Potential uses:
  - Extract information from object records
  - Could write different data to SYSLIN and SYSPUNCH
  - Alter object records

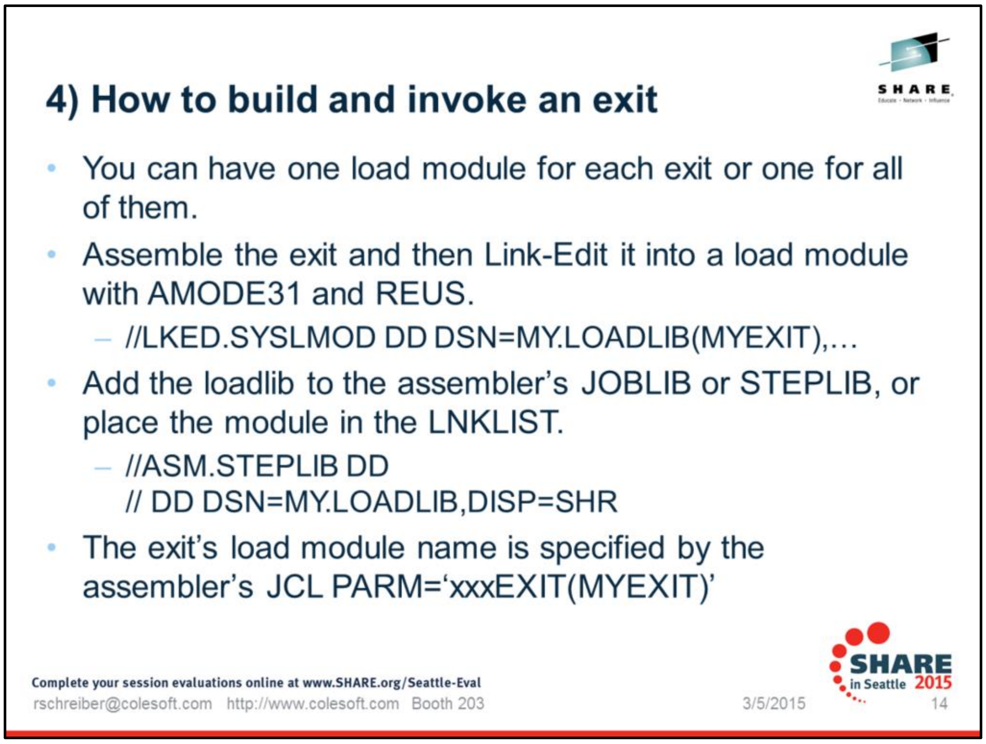


ADATA Exits

- Provides new SYSADATA records
- Changes SYSADATA records
- Deletes SYSADATA records
- Potential uses:
  - Extract information from object records
  - Thin out ADATA information
  - Collect data for debuggers

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 13



4) How to build and invoke an exit

- You can have one load module for each exit or one for all of them.
- Assemble the exit and then Link-Edit it into a load module with AMODE31 and REUS.
  - //LKED.SYSLMOD DD DSN=MY.LOADLIB(MYEXIT),…
- Add the loadlib to the assembler's JOBLIB or STEPLIB, or place the module in the LNKLIST.
  - //ASM.STEPLIB DD
    // DD DSN=MY.LOADLIB,DISP=SHR
- The exit's load module name is specified by the assembler's JCL PARM='xxxEXIT(MYEXIT)'

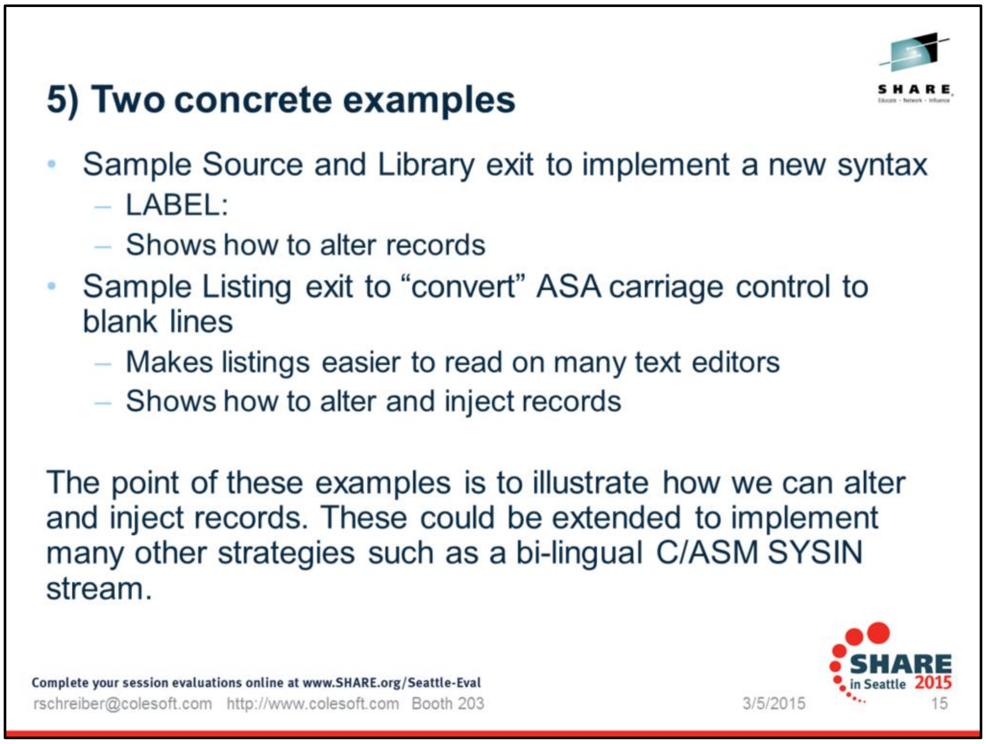
Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 14

To me, having one load module makes sharing of common code easier to write and maintain.

You can also write it as a re-entrant module if you wish.

The example that I give here is reusable but not reentrant.

Having the REUS (or RENT) option is important. When it is present then the assembler will LOAD the module only once and just branch to it on each call. If it is not REUS then it will be LOADed on every call and the performance will be pretty bad.



5) Two concrete examples

- Sample Source and Library exit to implement a new syntax
  - LABEL:
  - Shows how to alter records
- Sample Listing exit to "convert" ASA carriage control to blank lines
  - Makes listings easier to read on many text editors
  - Shows how to alter and inject records

The point of these examples is to illustrate how we can alter and inject records. These could be extended to implement many other strategies such as a bi-lingual C/ASM SYSIN stream.

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 15

Next we are going to talk about two simple exits.

The LABEL: exit illustrates how we can alter a SYSIN/SYSLIB record.

The SYSPRINT exit shows how we can alter and inject new records.

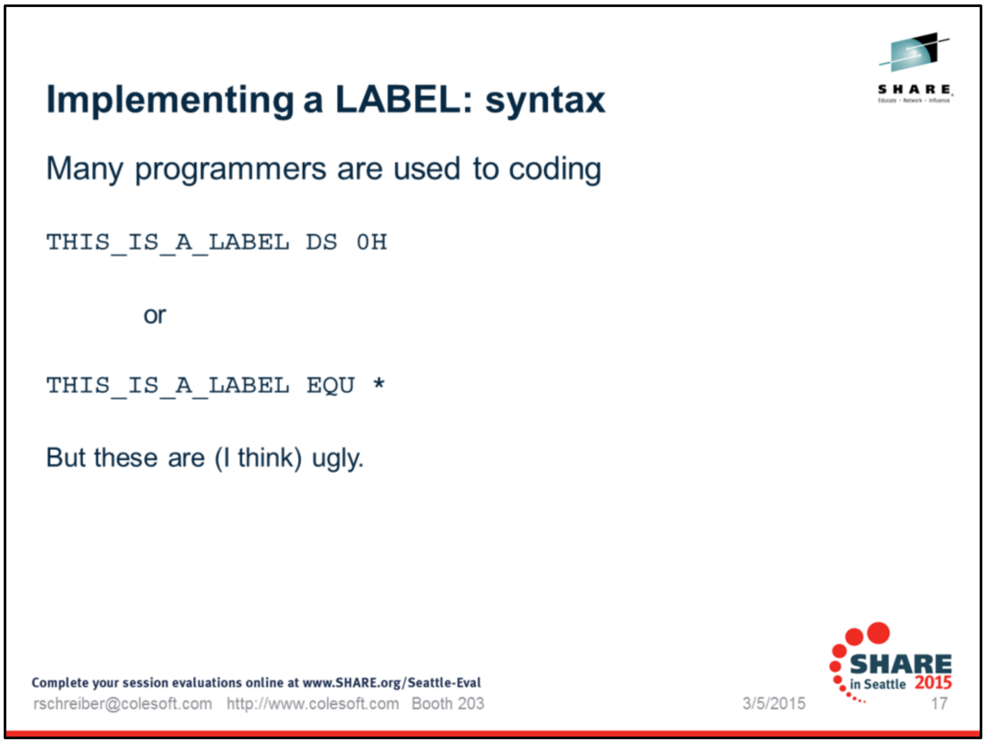
5.1 Implementing a LABEL: syntax

The following slides will illustrate an implementation of our sample exit that converts

```
THIS_IS_A_LABEL:
```

to

```
THIS_IS_A_LABEL       DS   0H
```



Implementing a LABEL: syntax

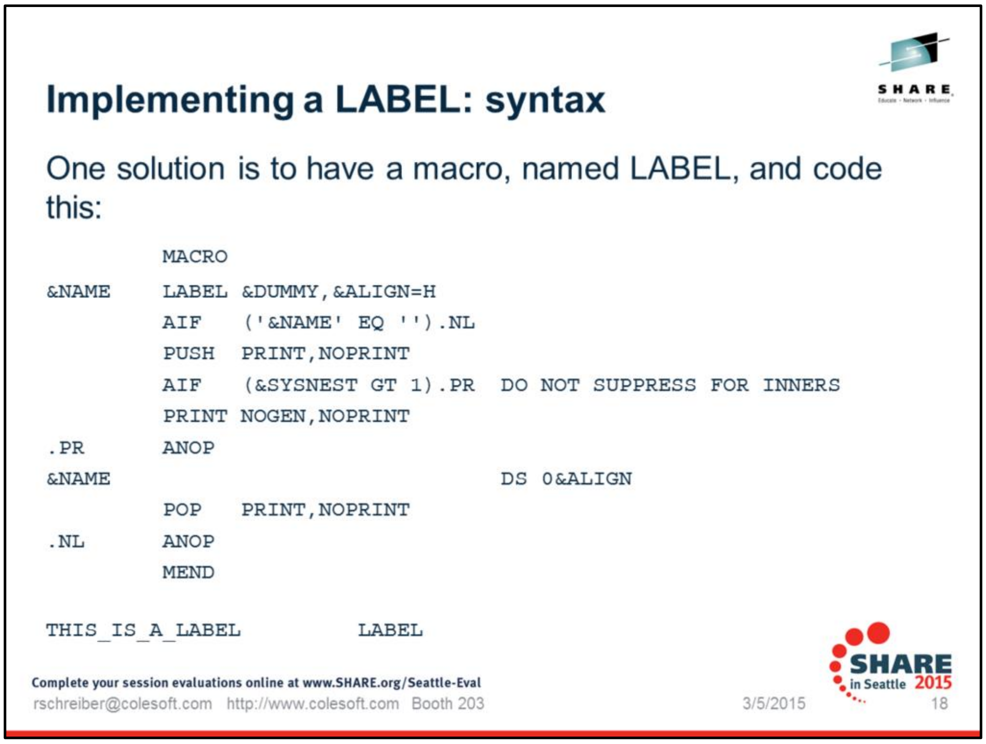
Many programmers are used to coding

```
THIS_IS_A_LABEL DS 0H
```

or

```
THIS_IS_A_LABEL EQU *
```

But these are (I think) ugly.



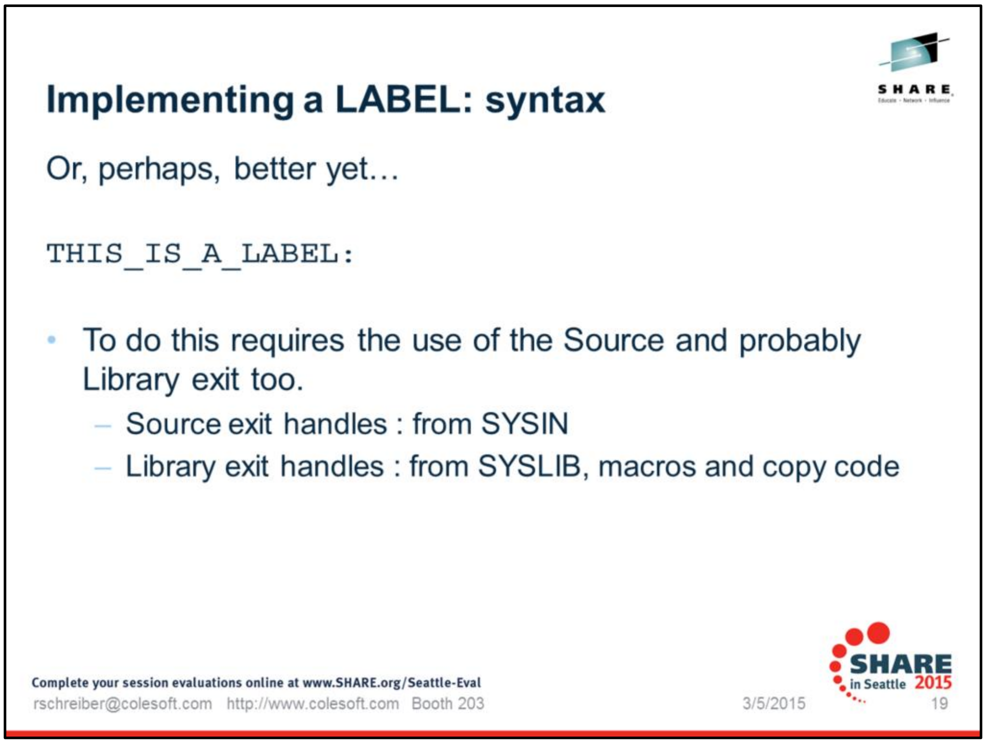
Implementing a LABEL: syntax

One solution is to have a macro, named LABEL, and code this:

```
         MACRO
&NAME    LABEL &DUMMY,&ALIGN=H
         AIF   ('&NAME' EQ '').NL
         PUSH  PRINT,NOPRINT
         AIF   (&SYSNEST GT 1).PR  DO NOT SUPPRESS FOR INNERS
         PRINT NOGEN,NOPRINT
.PR      ANOP
&NAME                              DS 0&ALIGN
         POP   PRINT,NOPRINT
.NL      ANOP
         MEND

THIS_IS_A_LABEL         LABEL
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 18

One fairly elegant way to do "labels" is with a LABEL macro. The opcode, LABEL, is placed out in column 36 to make is seems to be a comment.
This has the advantage of not needing an exit to implement it.


Implementing a LABEL: syntax

Or, perhaps, better yet…

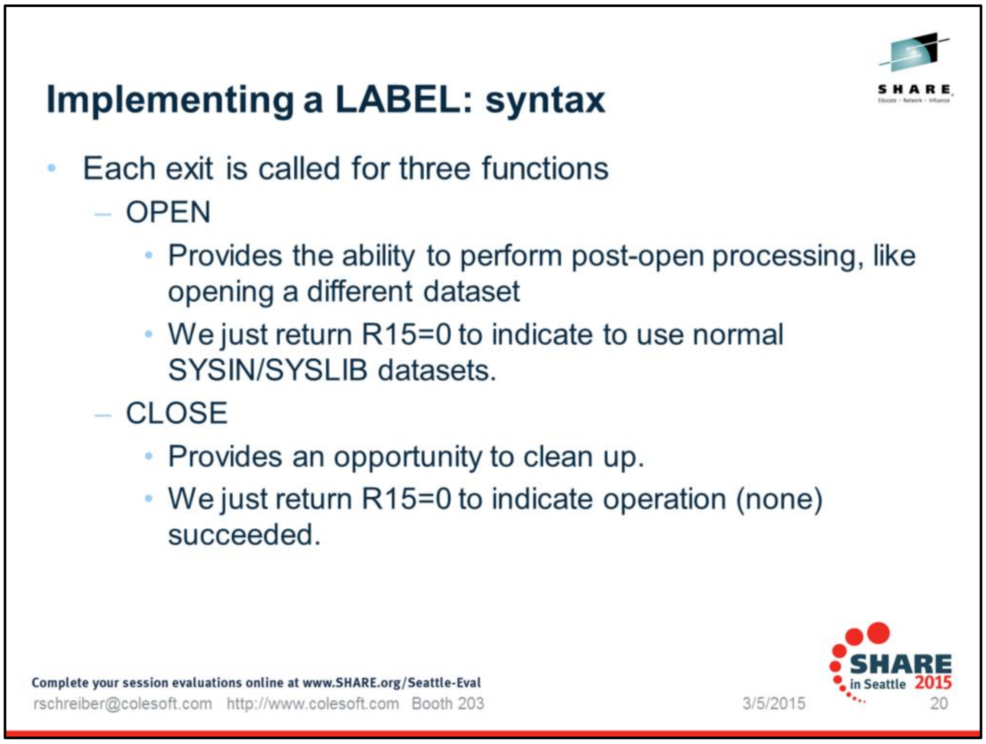
```
THIS_IS_A_LABEL:
```

- To do this requires the use of the Source and probably Library exit too.
  - Source exit handles : from SYSIN
  - Library exit handles : from SYSLIB, macros and copy code

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 19

One alternative is have the exit convert some thing like this into something that the assembler likes.

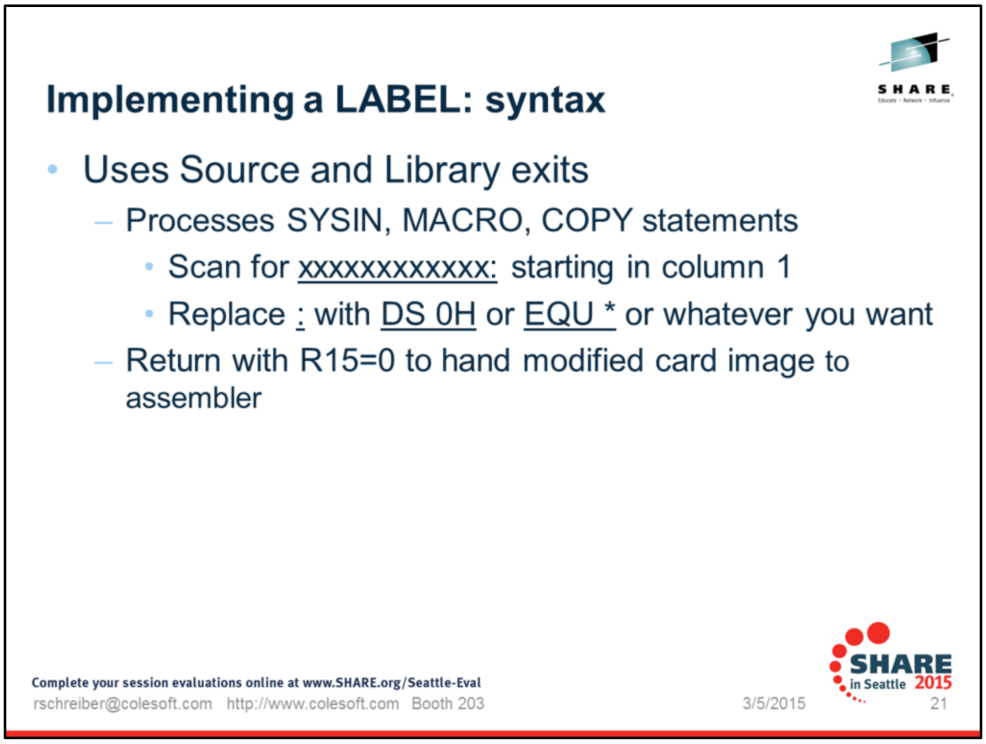
I tend to ignore listings for the most part and want the SYSIN and SYSLIB records to be pretty.



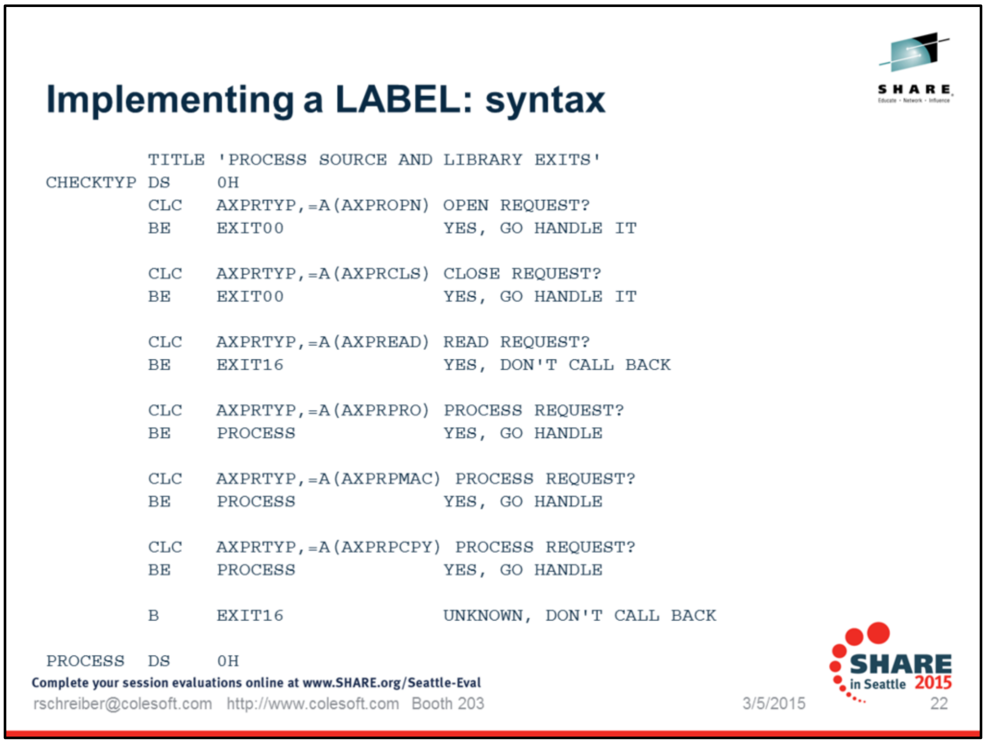
Implementing a LABEL: syntax

- Each exit is called for three functions
  - OPEN
    - Provides the ability to perform post-open processing, like opening a different dataset
    - We just return R15=0 to indicate to use normal SYSIN/SYSLIB datasets.
  - CLOSE
    - Provides an opportunity to clean up.
    - We just return R15=0 to indicate operation (none) succeeded.

The READ operations is not called when the assembler wants to read data. It is called…


Implementing a LABEL: syntax

- Uses Source and Library exits
  - Processes SYSIN, MACRO, COPY statements
    - Scan for <u>xxxxxxxxxxxx:</u> starting in column 1
    - Replace <u>:</u> with <u>DS 0H</u> or <u>EQU *</u> or whatever you want
  - Return with R15=0 to hand modified card image to assembler



Implementing a LABEL: syntax

```
         TITLE 'PROCESS SOURCE AND LIBRARY EXITS'
CHECKTYP DS    0H
         CLC   AXPRTYP,=A(AXPROPN) OPEN REQUEST?
         BE    EXIT00              YES, GO HANDLE IT

         CLC   AXPRTYP,=A(AXPRCLS) CLOSE REQUEST?
         BE    EXIT00              YES, GO HANDLE IT

         CLC   AXPRTYP,=A(AXPREAD) READ REQUEST?
         BE    EXIT16              YES, DON'T CALL BACK

         CLC   AXPRTYP,=A(AXPRPRO) PROCESS REQUEST?
         BE    PROCESS             YES, GO HANDLE

         CLC   AXPRTYP,=A(AXPRPMAC) PROCESS REQUEST?
         BE    PROCESS             YES, GO HANDLE

         CLC   AXPRTYP,=A(AXPRPCPY) PROCESS REQUEST?
         BE    PROCESS             YES, GO HANDLE

         B     EXIT16              UNKNOWN, DON'T CALL BACK

PROCESS  DS    0H
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 SHARE in Seattle 2015 22

This is the beginning of the code for the Source and Library sample exits.

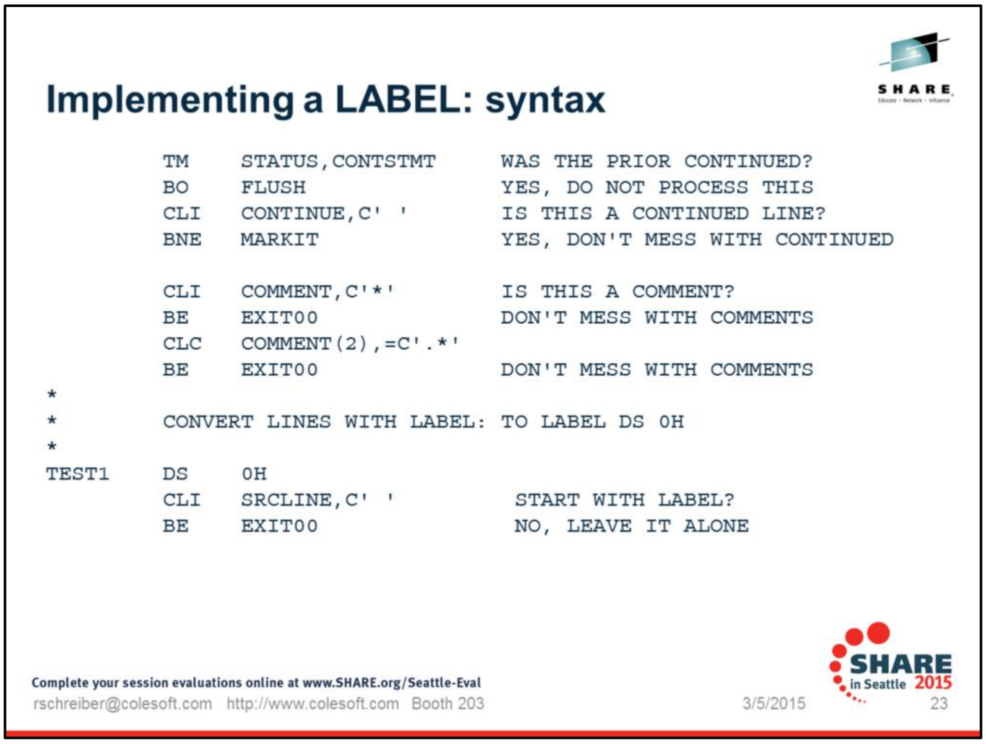
Both exits start executing at label CHECKTYP.

The EXIT00 routine will return with RC=0 and REASON=0 indicating that we are done "not" modifying the SYSIN and SYSLIB dataset specifications.

The EXIT04 routine (discussed later) will return with RC=0 and REASON=4.

The EXIT16 routine will return with RC=16 and REASON=0.

The three calls to PROCESS are the meat of this and will cause further actions.



Implementing a LABEL: syntax

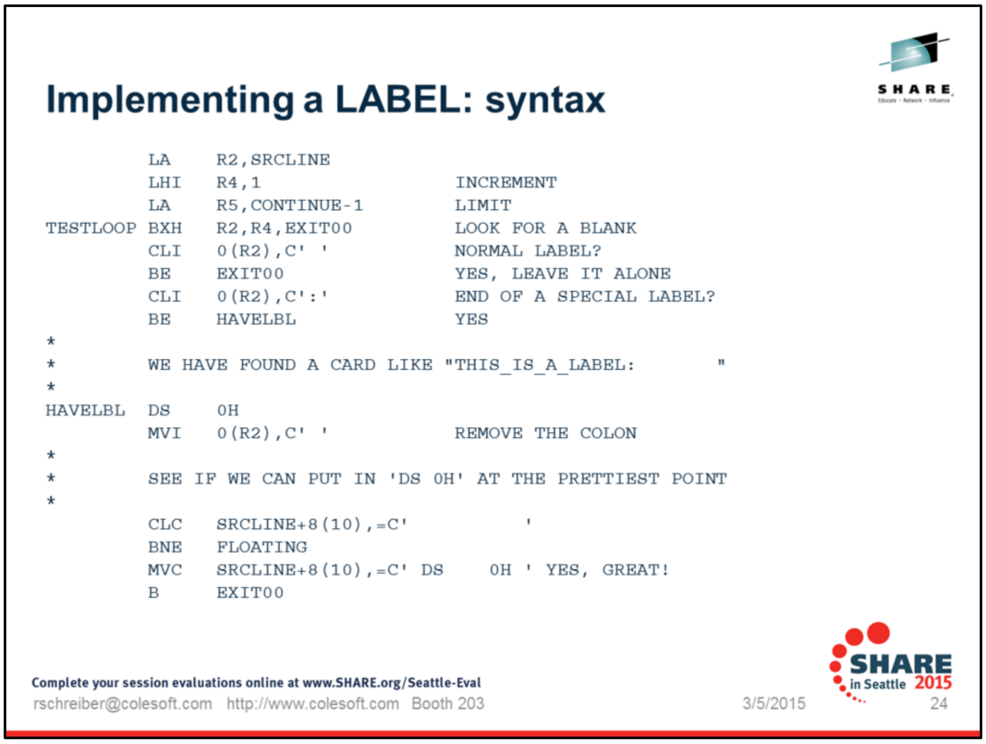
```
         TM    STATUS,CONTSTMT     WAS THE PRIOR CONTINUED?
         BO    FLUSH               YES, DO NOT PROCESS THIS
         CLI   CONTINUE,C' '       IS THIS A CONTINUED LINE?
         BNE   MARKIT              YES, DON'T MESS WITH CONTINUED

         CLI   COMMENT,C'*'        IS THIS A COMMENT?
         BE    EXIT00              DON'T MESS WITH COMMENTS
         CLC   COMMENT(2),=C'.*'
         BE    EXIT00              DON'T MESS WITH COMMENTS
*
*        CONVERT LINES WITH LABEL: TO LABEL DS 0H
*
TEST1    DS    0H
         CLI   SRCLINE,C' '         START WITH LABEL?
         BE    EXIT00               NO, LEAVE IT ALONE
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 23

Here we avoid process continued statements and comment cards.

At test1 we begin analyzing the statement to see if it has a label.



Implementing a LABEL: syntax

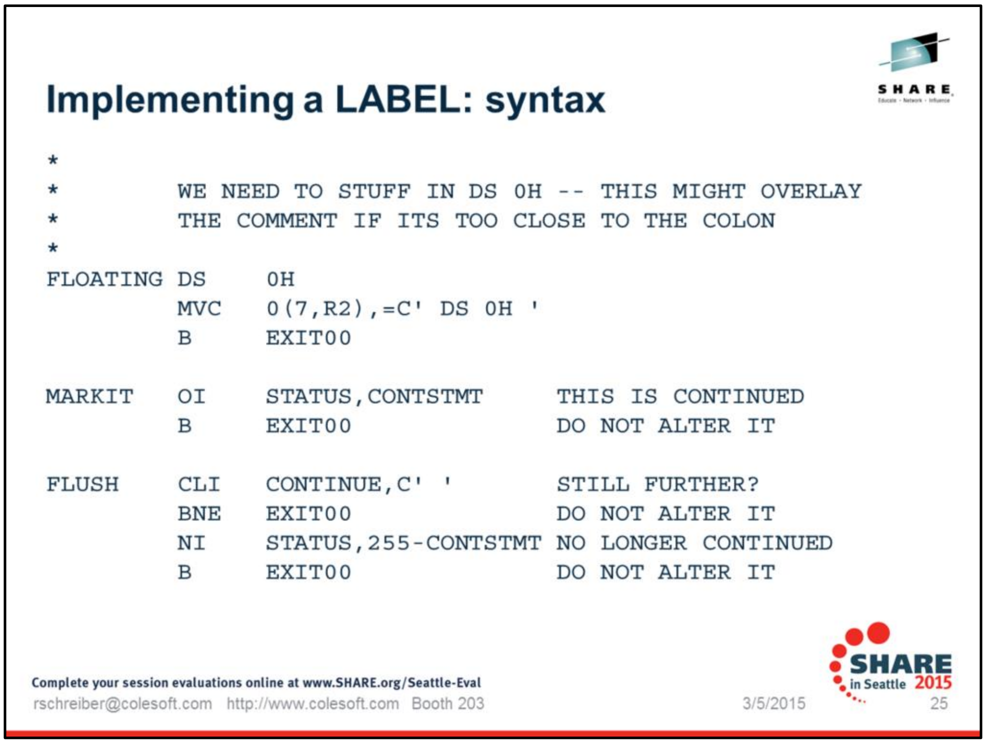
```
         LA    R2,SRCLINE
         LHI   R4,1                 INCREMENT
         LA    R5,CONTINUE-1        LIMIT
TESTLOOP BXH   R2,R4,EXIT00         LOOK FOR A BLANK
         CLI   0(R2),C' '           NORMAL LABEL?
         BE    EXIT00               YES, LEAVE IT ALONE
         CLI   0(R2),C':'           END OF A SPECIAL LABEL?
         BE    HAVELBL              YES
*
*        WE HAVE FOUND A CARD LIKE "THIS_IS_A_LABEL:       "
*
HAVELBL  DS    0H
         MVI   0(R2),C' '           REMOVE THE COLON
*
*        SEE IF WE CAN PUT IN 'DS 0H' AT THE PRETTIEST POINT
*
         CLC   SRCLINE+8(10),=C'          '
         BNE   FLOATING
         MVC   SRCLINE+8(10),=C' DS    0H ' YES, GREAT!
         B     EXIT00
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 24

Here we have dertermined that the statement does indeed have a label and forward scan it to see if it has a colon.

If the label was short then we try to place the DS 0H in column 10 and then exit.



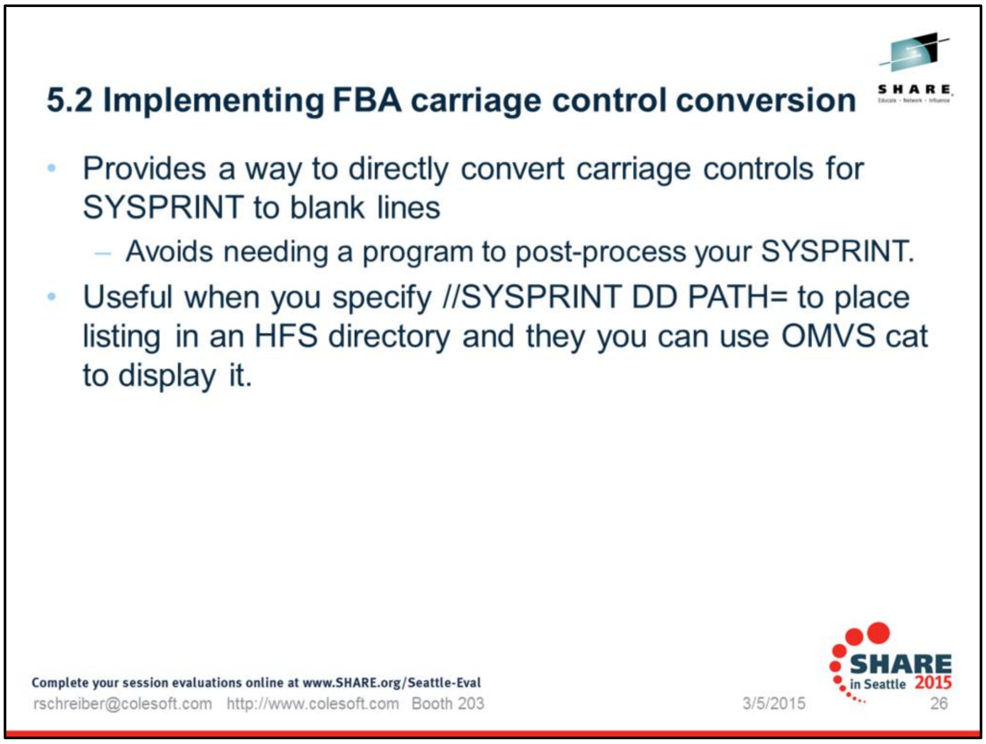
Implementing a LABEL: syntax

```
*
*        WE NEED TO STUFF IN DS 0H -- THIS MIGHT OVERLAY
*        THE COMMENT IF ITS TOO CLOSE TO THE COLON
*
FLOATING DS     0H
         MVC    0(7,R2),=C' DS 0H '
         B      EXIT00

MARKIT   OI     STATUS,CONTSTMT      THIS IS CONTINUED
         B      EXIT00               DO NOT ALTER IT

FLUSH    CLI    CONTINUE,C' '        STILL FURTHER?
         BNE    EXIT00               DO NOT ALTER IT
         NI     STATUS,255-CONTSTMT NO LONGER CONTINUED
         B      EXIT00               DO NOT ALTER IT
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 25

If that did not work then we jam the DS 0H over where the colon was, and exit.



5.2 Implementing FBA carriage control conversion

- Provides a way to directly convert carriage controls for SYSPRINT to blank lines
  - Avoids needing a program to post-process your SYSPRINT.
- Useful when you specify //SYSPRINT DD PATH= to place listing in an HFS directory and they you can use OMVS cat to display it.

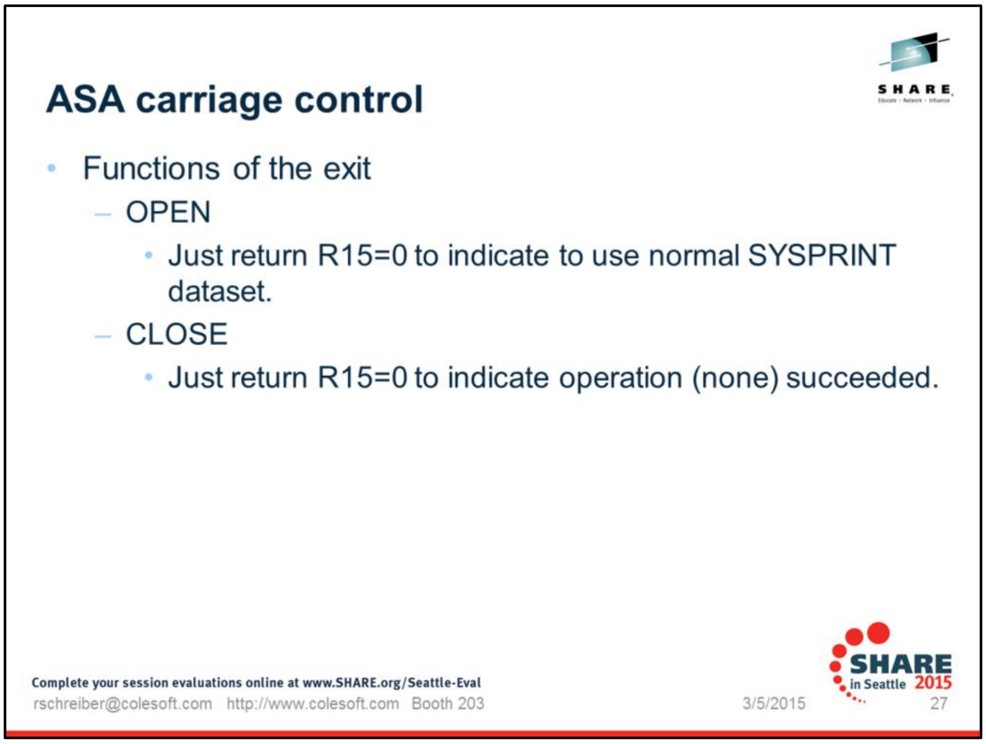
Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 26

This Listing exit provides the conversion of ASA carriage controls to blank lines.

In my shop I place the SYSPRINT data into an HFS file that is available to me via the z/OS SAMBA server. Then I use my Windows system to map a drive to that directory and then use my favorite ASCII editor to browse the listing.

You could also do this with Linux.

Here is how it works…



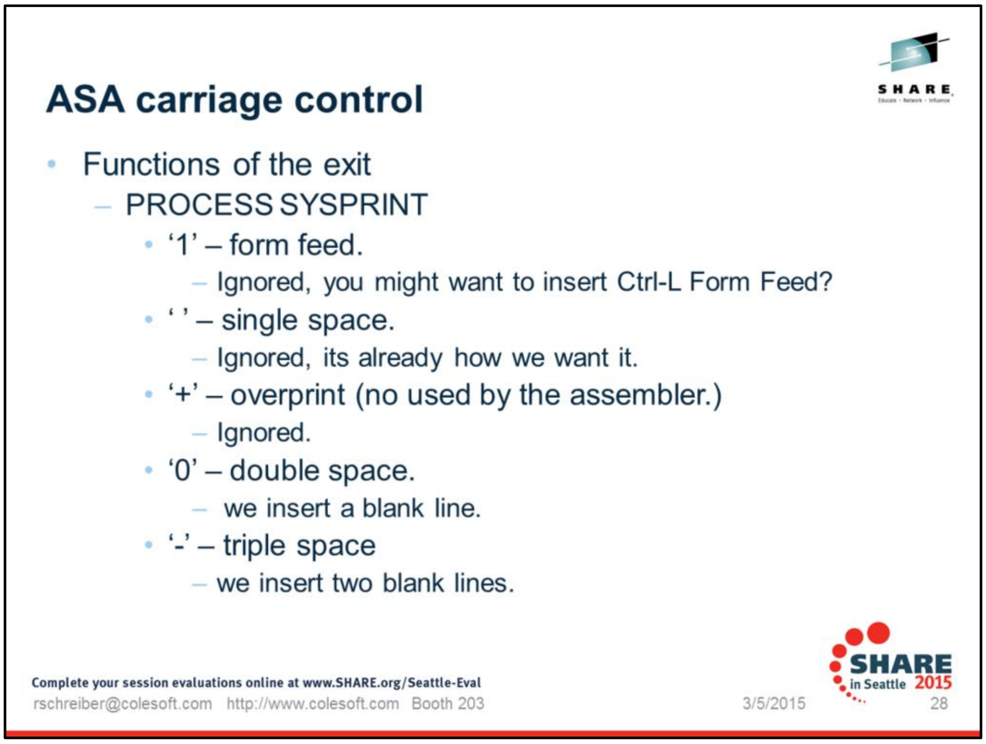
ASA carriage control

- Functions of the exit
  - OPEN
    - Just return R15=0 to indicate to use normal SYSPRINT dataset.
  - CLOSE
    - Just return R15=0 to indicate operation (none) succeeded.

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 27

Here we basically want to ignore Open and Close since we don’t need to remap them to a different DCB or DDNAME.

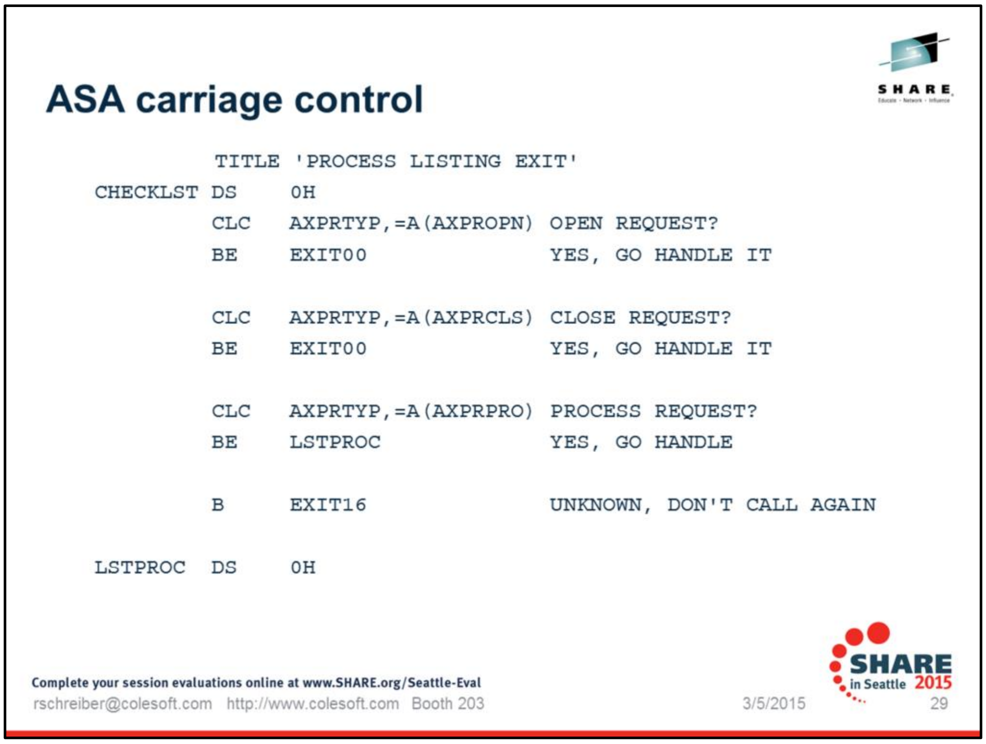

ASA carriage control

- Functions of the exit
  - PROCESS SYSPRINT
    - ‘1’ – form feed.
      - Ignored, you might want to insert Ctrl-L Form Feed?
    - ‘ ’ – single space.
      - Ignored, its already how we want it.
    - ‘+’ – overprint (no used by the assembler.)
      - Ignored.
    - ‘0’ – double space.
      - we insert a blank line.
    - ‘-’ – triple space
      - we insert two blank lines.

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 28

For 0 and – we will be inserting one or two blank lines in front of the current line.

This will require injecting new records into SYSPRINT.

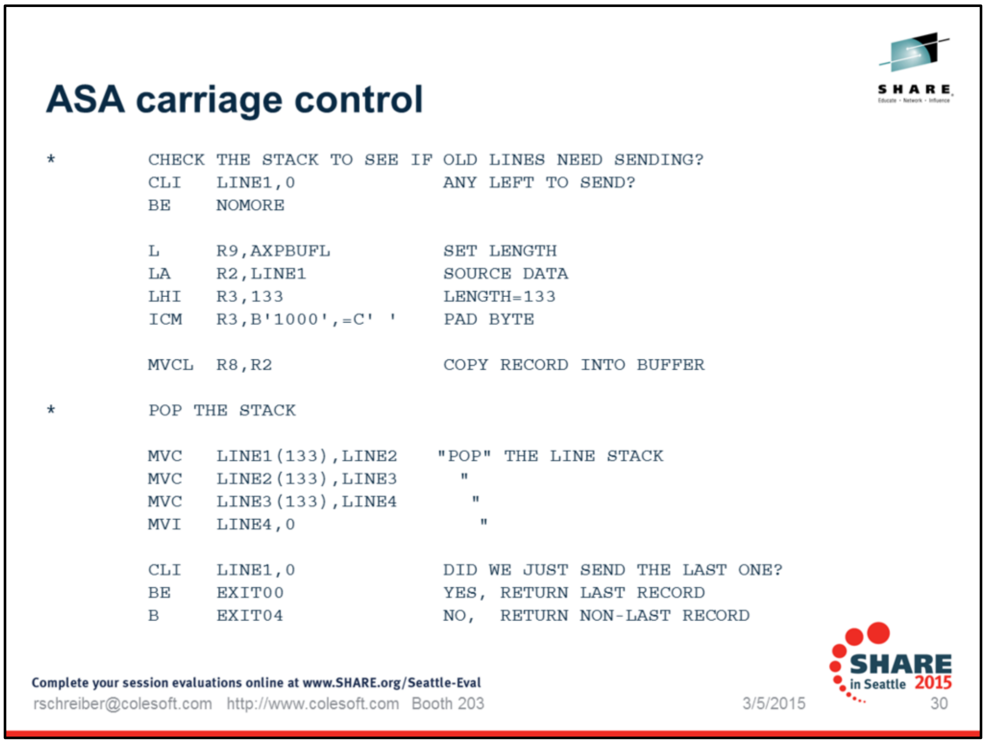


ASA carriage control

```
          TITLE 'PROCESS LISTING EXIT'
CHECKLST DS     0H
         CLC    AXPRTYP,=A(AXPROPN) OPEN REQUEST?
         BE     EXIT00              YES, GO HANDLE IT

         CLC    AXPRTYP,=A(AXPRCLS) CLOSE REQUEST?
         BE     EXIT00              YES, GO HANDLE IT

         CLC    AXPRTYP,=A(AXPRPRO) PROCESS REQUEST?
         BE     LSTPROC             YES, GO HANDLE

         B      EXIT16              UNKNOWN, DON'T CALL AGAIN

LSTPROC  DS     0H
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 29

Similar to the Source and Library exits we ignore Open and Close and honor Process.



ASA carriage control

```
*         CHECK THE STACK TO SEE IF OLD LINES NEED SENDING?
          CLI   LINE1,0            ANY LEFT TO SEND?
          BE    NOMORE

          L     R9,AXPBUFL         SET LENGTH
          LA    R2,LINE1           SOURCE DATA
          LHI   R3,133             LENGTH=133
          ICM   R3,B'1000',=C' '   PAD BYTE

          MVCL  R8,R2              COPY RECORD INTO BUFFER

*         POP THE STACK

          MVC   LINE1(133),LINE2  "POP" THE LINE STACK
          MVC   LINE2(133),LINE3    "
          MVC   LINE3(133),LINE4     "
          MVI   LINE4,0               "

          CLI   LINE1,0            DID WE JUST SEND THE LAST ONE?
          BE    EXIT00             YES, RETURN LAST RECORD
          B     EXIT04             NO,  RETURN NON-LAST RECORD
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 SHARE in Seattle 2015 30

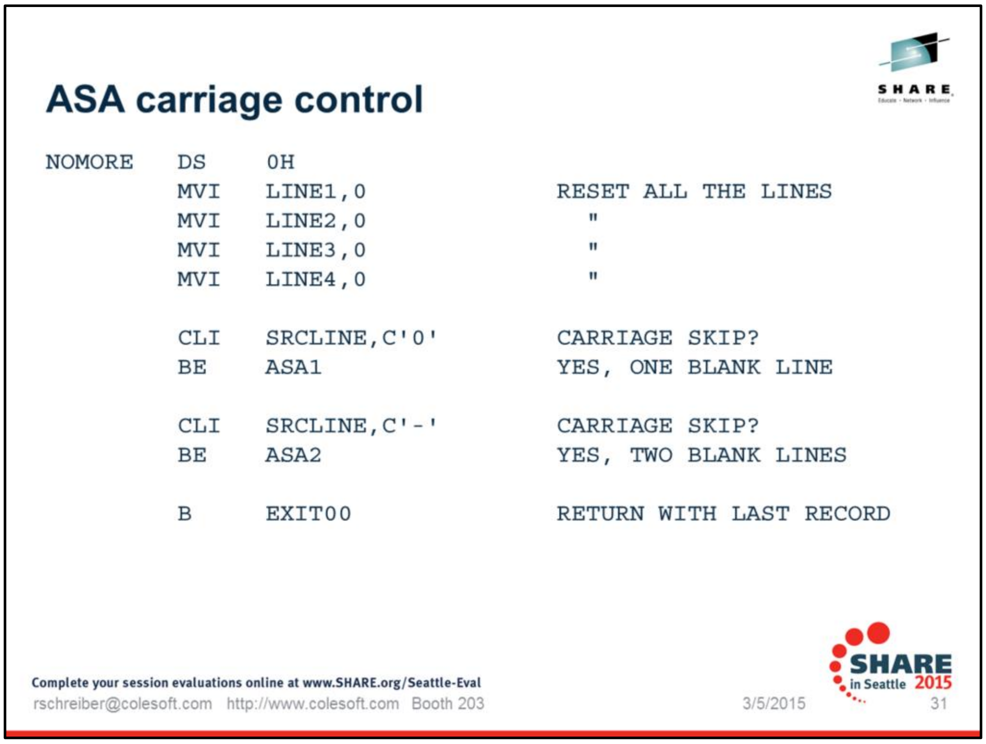
The code operates by maintaining a stack of blank and output lines.

If the stack is empty then we reload it with the current record. (NOMORE)

LINE1 gets the output line and its also copied to LINES2-4.

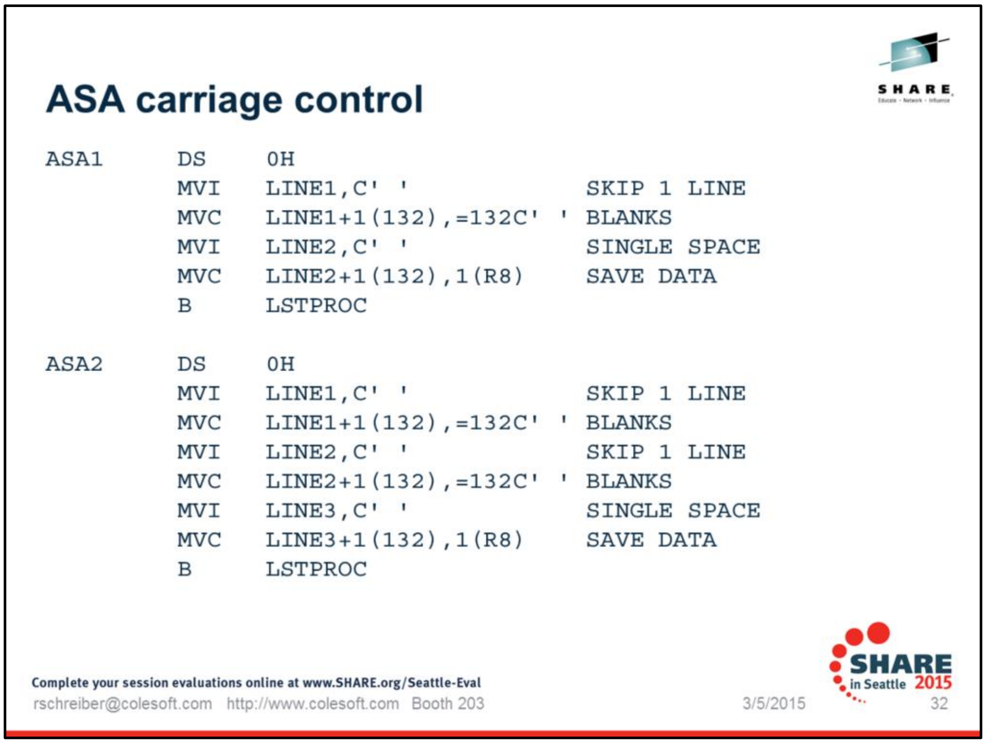
If LINE1 has a x’00’ then we have emptied the stack and tell the assembler to move onto its next output record.

If LINE1 is non-zero then we tell the



ASA carriage control

```
NOMORE   DS    0H
         MVI   LINE1,0                RESET ALL THE LINES
         MVI   LINE2,0                  "
         MVI   LINE3,0                  "
         MVI   LINE4,0                  "

         CLI   SRCLINE,C'0'           CARRIAGE SKIP?
         BE    ASA1                   YES, ONE BLANK LINE

         CLI   SRCLINE,C'-'           CARRIAGE SKIP?
         BE    ASA2                   YES, TWO BLANK LINES

         B     EXIT00                 RETURN WITH LAST RECORD
```



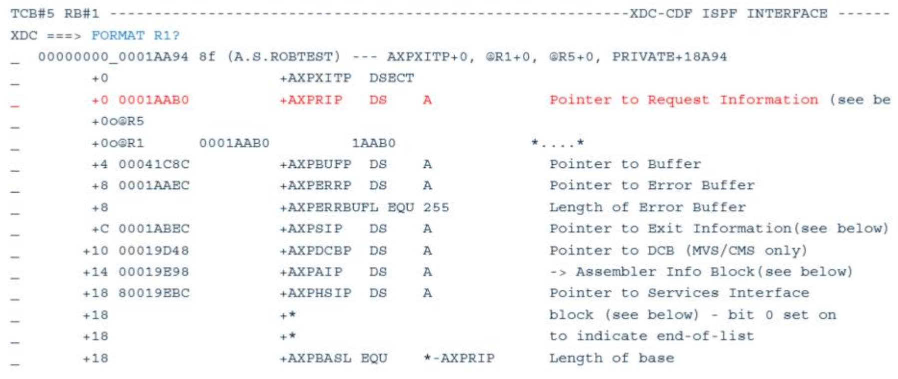
ASA carriage control

```
ASA1     DS    0H
         MVI   LINE1,C' '            SKIP 1 LINE
         MVC   LINE1+1(132),=132C' ' BLANKS
         MVI   LINE2,C' '            SINGLE SPACE
         MVC   LINE2+1(132),1(R8)    SAVE DATA
         B     LSTPROC

ASA2     DS    0H
         MVI   LINE1,C' '            SKIP 1 LINE
         MVC   LINE1+1(132),=132C' ' BLANKS
         MVI   LINE2,C' '            SKIP 1 LINE
         MVC   LINE2+1(132),=132C' ' BLANKS
         MVI   LINE3,C' '            SINGLE SPACE
         MVC   LINE3+1(132),1(R8)    SAVE DATA
         B     LSTPROC
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 32

Here we handle adding one line and the data to the stack (of two)

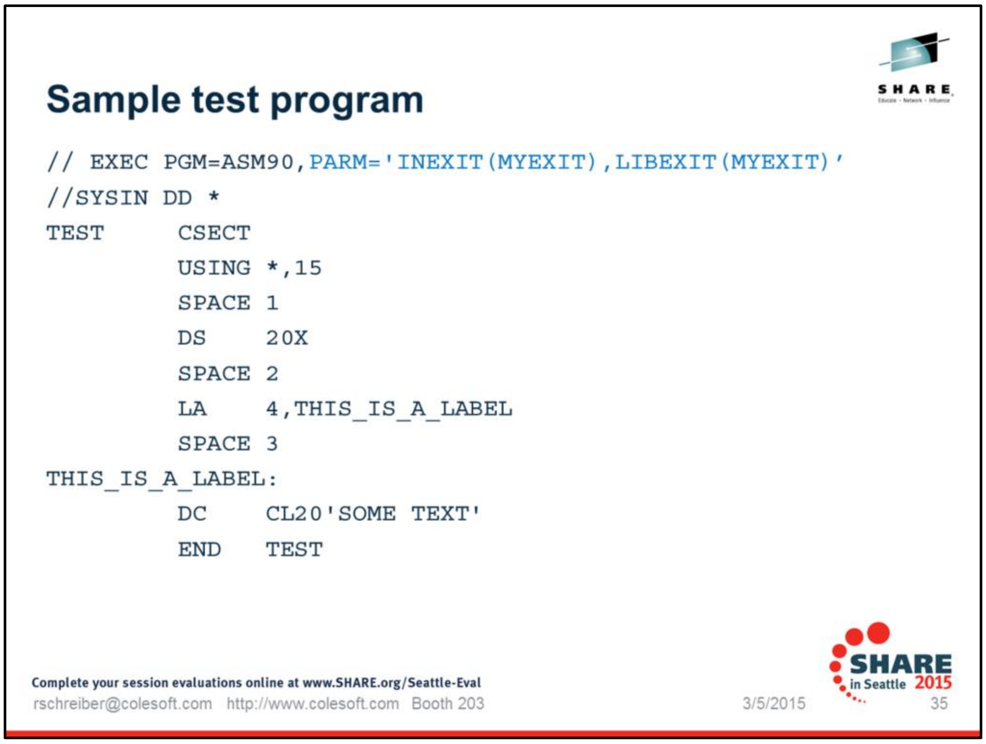
AXPXITP upon entry – via R1



```
TCB#5 RB#1 ----------------------------------------------------------XDC-CDF ISPF INTERFACE ------
XDC ===> FORMAT R1?
_  00000000_0001AA94 8f (A.S.ROBTEST) --- AXPXITP+0, @R1+0, @R5+0, PRIVATE+18A94
_        +0                    +AXPXITP  DSECT
_        +0 0001AAB0           +AXPRIP   DS    A             Pointer to Request Information (see be
_        +0o@R5
_        +0o@R1       0001AAB0          1AAB0               *....*
_        +4 00041C8C           +AXPBUFP  DS    A             Pointer to Buffer
_        +8 0001AAEC           +AXPERRP  DS    A             Pointer to Error Buffer
_        +8                    +AXPERRBUFL EQU 255           Length of Error Buffer
_        +C 0001ABEC           +AXPSIP   DS    A             Pointer to Exit Information(see below)
_       +10 00019D48           +AXPDCBP  DS    A             Pointer to DCB (MVS/CMS only)
_       +14 00019E98           +AXPAIP   DS    A             -> Assembler Info Block(see below)
_       +18 80019EBC           +AXPHSIP  DS    A             Pointer to Services Interface
_       +18                    +*                            block (see below) - bit 0 set on
_       +18                    +*                            to indicate end-of-list
_       +18                    +AXPBASL EQU    *-AXPRIP      Length of base
```

AXPRIL control block for SYSPRINT call

```
TCB#5 RB#1 ---------------------------------------------------------XDC-CDF ISPF INTERFACE ----------
XDC ===> FORMAT .AXPRIL
_  00000000_0001AAB0 8f (A.S.ROBTEST) --- AXPRIL+0, @R10+0, @R3+0, @R1+1C, @R11+1C, @R5+1C,
_           AXPXITP+1C, PRIVATE+18AB0
_       +0                    +AXPRIL   DSECT                Request Information List
_       +0 00000003           +AXPLVER  DS    F              EXIT list version number
_       +4                    +AXPVER3  EQU   3              Exit Parameter List Version 3
_       +4 00000003           +AXPTYPE  DS    F              EXIT Type (see values below)
_       +4                    +AXPTSRC  EQU   1              SOURCE (SYSIN)
_       +4                    +AXPTLIB  EQU   2              LIBRARY (SYSLIB)
_       +4                    +AXPTLST  EQU   3              LISTING (SYSPRINT)
_       +4                    +AXPTPUN  EQU   4              PUNCH (SYSPUNCH)
_       +4                    +AXPTOBJ  EQU   5              OBJECT (SYSLIN)
_       +4                    +AXPTAD   EQU   6              ADATA (SYSADATA)
_       +4                    +AXPTTRM  EQU   7              TERM (SYSTERM)
_       +8 00000005           +AXPRTYP  DS    F              Request Type (see values below)
_       +8                    +AXPRPRO  EQU   5              PROCESS - exit receives control to inspect
_       +8                    +*                                       and/or modify record provided by
_       +8                    +*                                       the assembler (Not LIBRARY exit)
```



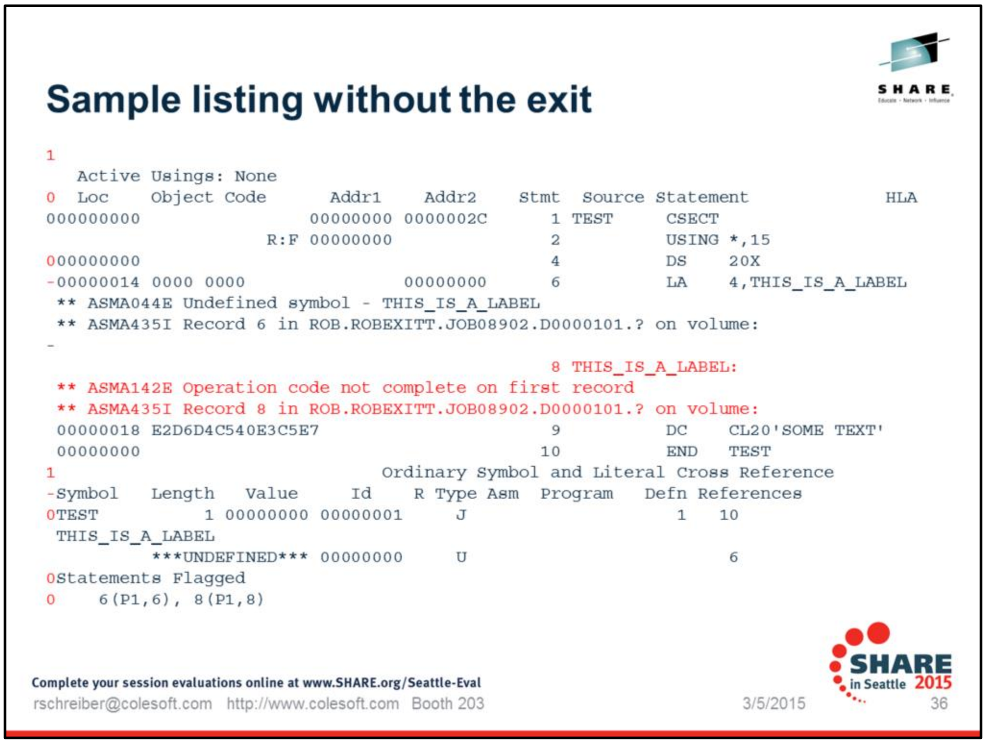
Sample test program

```
// EXEC PGM=ASM90,PARM='INEXIT(MYEXIT),LIBEXIT(MYEXIT)'
//SYSIN DD *
TEST     CSECT
         USING *,15
         SPACE 1
         DS    20X
         SPACE 2
         LA    4,THIS_IS_A_LABEL
         SPACE 3
THIS_IS_A_LABEL:
         DC    CL20'SOME TEXT'
         END   TEST
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 35

This is a tiny program that illustrates the features we have implemented.

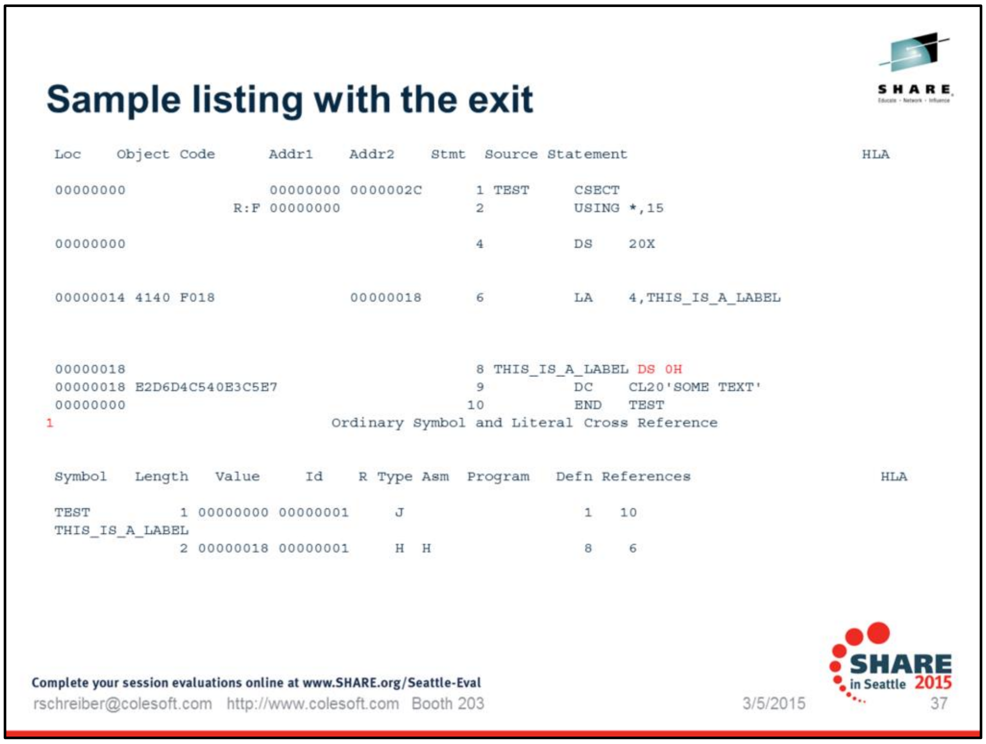


Sample listing without the exit

```
1
   Active Usings: None
0  Loc     Object Code       Addr1     Addr2     Stmt  Source Statement              HLA
000000000                 00000000 0000002C      1 TEST      CSECT
                     R:F 00000000                2          USING *,15
000000000                                        4          DS     20X
-00000014 0000 0000                00000000      6          LA     4,THIS_IS_A_LABEL
 ** ASMA044E Undefined symbol - THIS_IS_A_LABEL
 ** ASMA435I Record 6 in ROB.ROBEXITT.JOB08902.D0000101.? on volume:
-
                                                 8 THIS_IS_A_LABEL:
 ** ASMA142E Operation code not complete on first record
 ** ASMA435I Record 8 in ROB.ROBEXITT.JOB08902.D0000101.? on volume:
 00000018 E2D6D4C540E3C5E7                       9          DC     CL20'SOME TEXT'
 00000000                                       10          END    TEST
1                                Ordinary Symbol and Literal Cross Reference
-Symbol   Length   Value     Id     R Type Asm  Program    Defn References
0TEST          1 00000000 00000001     J                     1    10
 THIS_IS_A_LABEL
         ***UNDEFINED*** 00000000     U                          6
0Statements Flagged
0    6(P1,6), 8(P1,8)
```

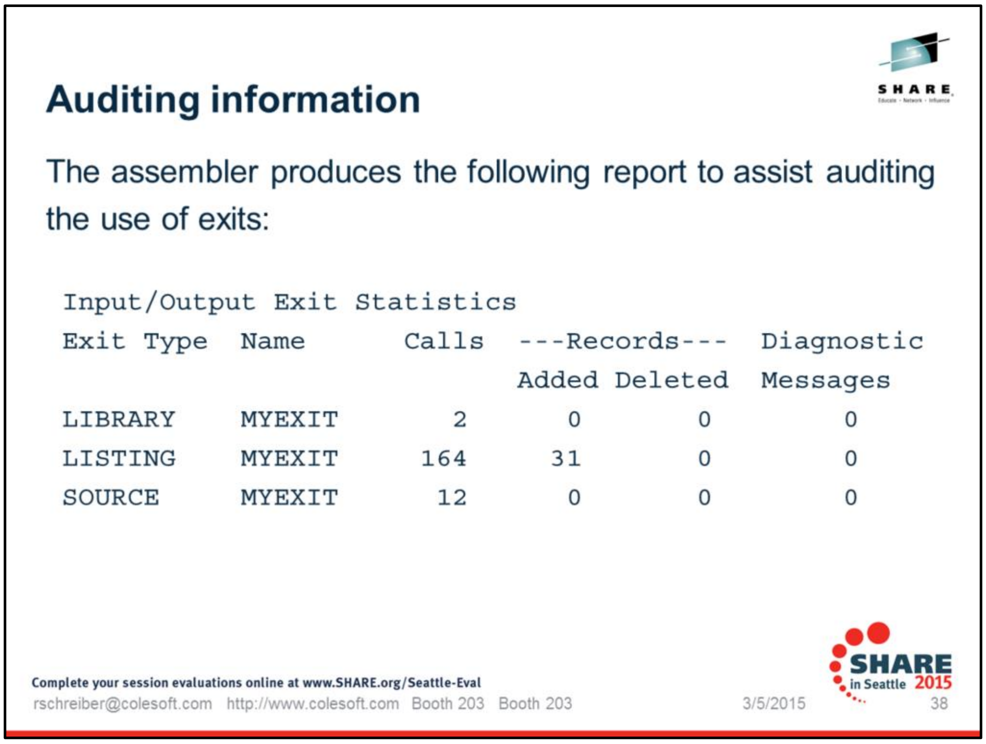
Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 36



Sample listing with the exit

```
 Loc     Object Code       Addr1     Addr2    Stmt  Source Statement                                 HLA

 00000000                  00000000 0000002C      1 TEST     CSECT
                       R:F 00000000               2          USING *,15

 00000000                                         4          DS    20X


 00000014 4140 F018                 00000018      6          LA    4,THIS_IS_A_LABEL



 00000018                                         8 THIS_IS_A_LABEL DS 0H
 00000018 E2D6D4C540E3C5E7                        9          DC    CL20'SOME TEXT'
 00000000                                        10          END   TEST
1                              Ordinary Symbol and Literal Cross Reference


 Symbol   Length   Value     Id     R Type Asm  Program   Defn References                         HLA

 TEST          1 00000000 00000001     J                    1   10
 THIS_IS_A_LABEL
               2 00000018 00000001     H  H                 8    6
```



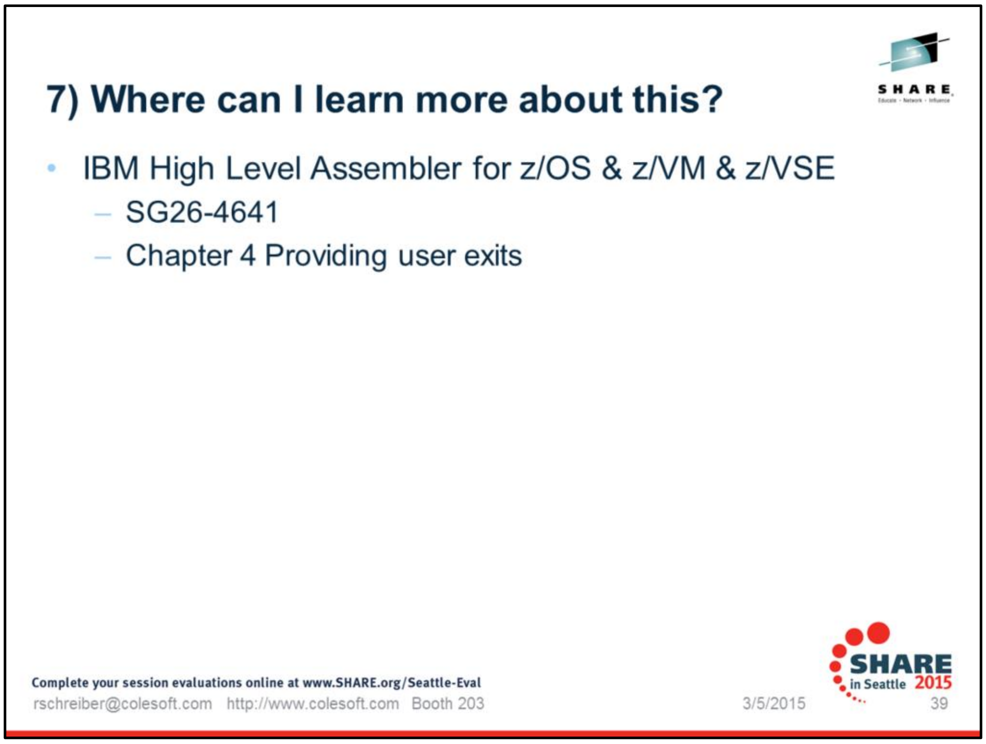
Auditing information

The assembler produces the following report to assist auditing the use of exits:

```
Input/Output Exit Statistics
Exit Type  Name       Calls  ---Records---  Diagnostic
                             Added Deleted  Messages
LIBRARY    MYEXIT        2       0       0         0
LISTING    MYEXIT      164      31       0         0
SOURCE     MYEXIT       12       0       0         0
```

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 Booth 203 3/5/2015 38

It does not produce a count of the number of records that were modified.

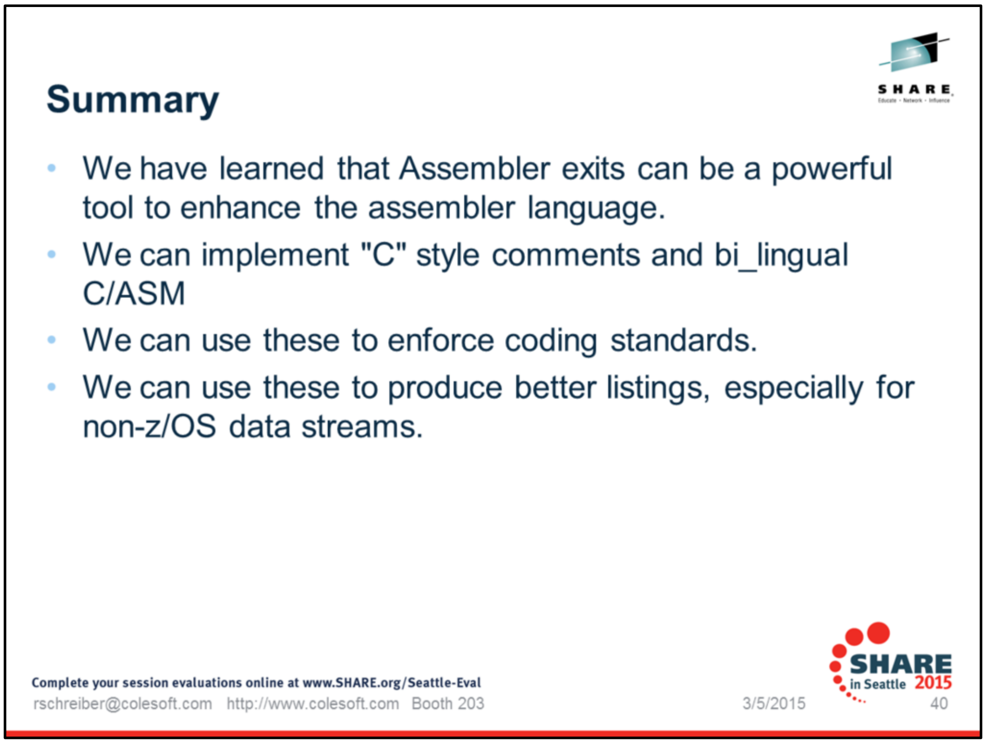


7) Where can I learn more about this?

- IBM High Level Assembler for z/OS & z/VM & z/VSE
  - SG26-4641
  - Chapter 4 Providing user exits

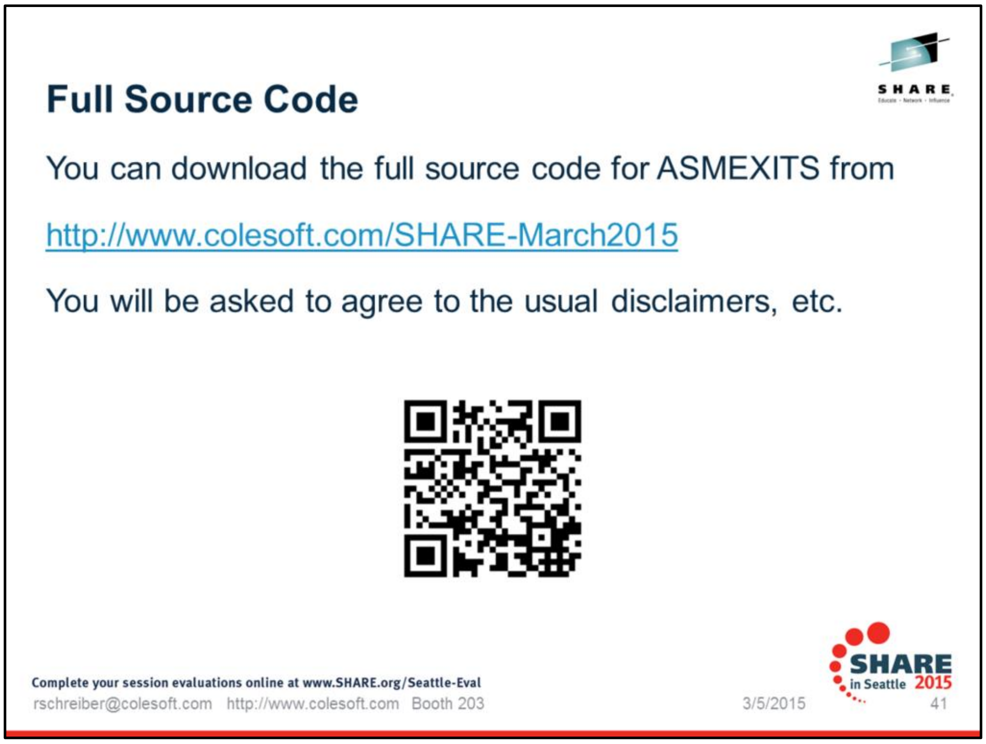
Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 39

The example exits that we present here do not include all the source code to build it. You can visit our web site to download the full source code.



Summary

- We have learned that Assembler exits can be a powerful tool to enhance the assembler language.
- We can implement "C" style comments and bi_lingual C/ASM
- We can use these to enforce coding standards.
- We can use these to produce better listings, especially for non-z/OS data streams.



Full Source Code

You can download the full source code for ASMEXITS from

http://www.colesoft.com/SHARE-March2015

You will be asked to agree to the usual disclaimers, etc.

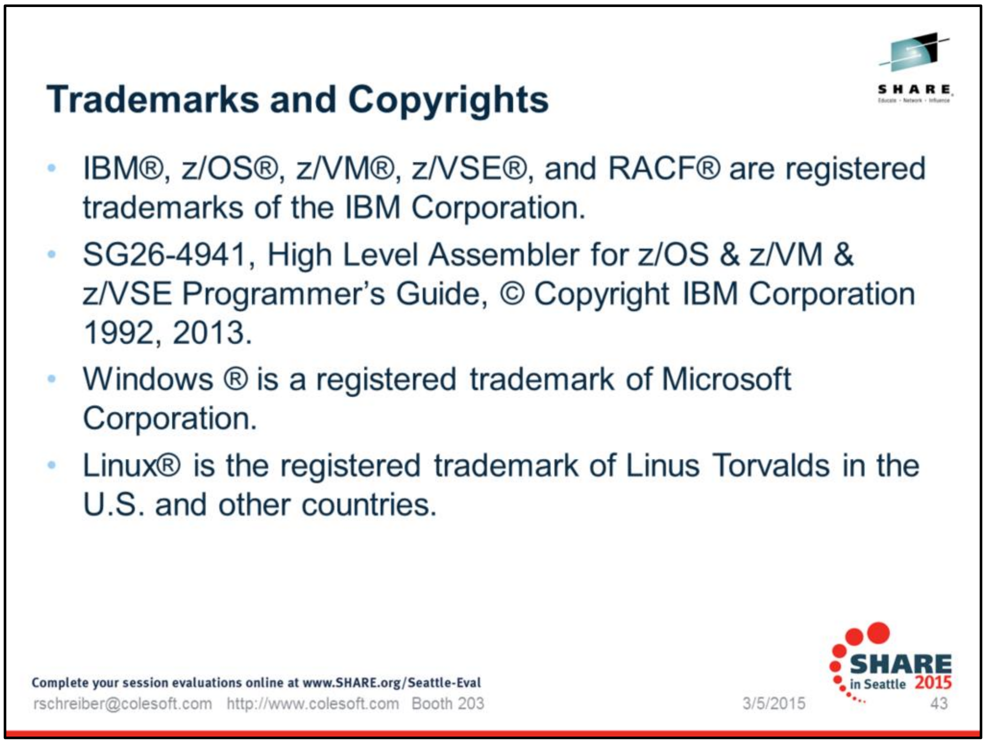
Complete your session evaluations online at www.SHARE.org/Seattle-Eval
rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 41

SHARE in Seattle 2015

Questions?

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

rschreiber@colesoft.com http://www.colesoft.com Booth 203 3/5/2015 42



Trademarks and Copyrights