z/TPF SOLD Driver

User's Guide

This page intentionally left blank.

ZTEST SOLD

Shared Object Linkage Driver (SOLD) is a driver written specifically to test intra and inter module calls between C Shared Objects (CSO), BAL shared objects (BSO) and ELF executables (C program with main()). While a C program with main() can call other SOs, main() cannot be called by other programs. The CSOs come in two variations. One is where a CSO contains a single entry point. These are not meant to export any functions other than the function by the name of the module. The second type is where a CSO is a library. These shared objects export all functions defined within them that are not defined as static. The CSO with a single entry point is linked (ld) the same way as other shared objects. But when the callers of these programs are linked (ld), single entry CSOs are not provided as input libraries. These calls are resolved at fetch time. Export of variables is not supported from any modules.

All modules can contain object code of one or more source files. The CSOs contain objects of C source files or objects of BAL written C functions. Each object can contain more than one function. BAL written C functions are assembled to generate a GOFF object and translated to an ELF object. In z/TPF, linkage or parameter passing is same for C and C++. For linkage testing, the only difference is in the way function names are identified. In C++, the function is identified by the name and the parameters passed to it (to allow for overloading). The C functions are identified by their names alone. One C++ test case exists for the purpose of function trace verification.

Each BAL source file can only contain one BAL program. Each BSO can contain objects of many source files and hence can contain many BAL programs. Each source file is assembled to a GOFF object, converted to an ELF object and linked (ld) as a BSO.

The SOLD test cases cover calls between all program types. It also covers calls to functions via function pointers and C function calls from BAL via the CALLC macro. In addition, there is an option to cause a dump after a test case completes for dump verification and function trace purposes.

The driver can run on any subsystem in either 1052 or NORM state, and it runs in multiple I-streams. An error message is displayed if one attempts to run SOLD while the system is cycling.

SOLD can run one test case at a time (with the user supplying the specific test case number), or it can run in continuous mode. For continuous mode, the sequence is in no particular order (i.e., it is not in numerical order by test case number).

Appropriate error messages are displayed to the terminal if an invalid test case number is supplied or passed, there is a return to a segment that did an ENTDC or ENTNC, and the like.

Requirements and restrictions

None.

Format

```
>>---ZTEST--+-----+-- --SOLD-- --+-CASE-xx--+----------------+--+----><
            +- -i-+              |          | .- -NODUMP-.    |  |
            '- -*-'              |          +-+----------+----+  |
                                 |          | '- -DUMP---'    |  |
                                 |          | .- -NOSHOWVER-. |  |
                                 |          '-+-------------+-'  |
                                 |            '- -SHOWVER---'    |
                                 +-START-------------------------+
                                 +-STOP--------------------------+
                                 +-STATUS------------------------+
                                 +-HELP--------------------------+
                                 '-?-----------------------------'
```

i

indicates the specific I-stream in which the driver will be run. If *i* is not specified, the test case(s) will be executed on the I-stream on which the command is entered.

*****

specifies the driver will be invoked on all currently defined and available I-streams.

CASE-*xx*

indicates that one specific test case is to be executed. The *xx* variable is required with the CASE parameter and specifies the test case number to be executed.

NODUMP

specifies that the driver will not dump after completion. This is the default setting.

DUMP

causes the driver to dump after completion.

NOSHOWVER

specifies that the driver will not display the program version and program variation number when entering a segment. This is the default setting.

SHOWVER

displays the program version and program variation number when entering a segment.

START

specifies that the SOLD driver is to be run in continuous mode.

STOP

specifies that this driver halt continuous mode.

STATUS

displays the status of the running driver. The status may be requested for a specific I-stream or for all defined I-streams.

HELP | ?

displays the correct syntax of the command.

Source code information

The SOLD driver consists of the following program segments:

Header Files

Header File	Description
c_sold.h	This header file was specifically created to hold all SOLD prototypes and various definitions.
cpp_sold.hpp	This header file was created to hold all C++ SOLD definitions.

BSOs

Module	Makefile	Segment	Description
QSB1	qsb1.mak	qsb1.asm	Standard 64-bit BAL segment with QSB1 entry point. The tracegroup for QSB1 is TRG_2.
		qsb2.asm	64-bit BAL segment with 2 transfer vectors (QSB2 and QSB3).
		qsb4.asm	64-bit BAL segment that is greater than 4K in size and has multiple base registers. QSB4 entry point.
QSB5	qsb5.mak	qsb5.asm	64-bit BAL segment with 3 transfer vectors (QSB5, QSBZ, and QSB6). QSBZ is a dummy transfer vector not meant to be used. The tracegroup for QSB5 is TRG_4.
QSB8	qsb8.mak	qsb8.asm	Standard 64-bit BAL segment with QSB8 entry point.
		qsb9.asm	64-bit BAL segment with no base register and QSB9 entry point.
		qsbf.asm	31-bit BAL segment with no base register and QSBF entry point.
QSBA	qsba.mak	qsba.asm	31-bit BAL segment with QSBA entry point.
		qsbb.asm	31-bit BAL segment that is greater than 4K in size and with multiple base registers. QSBB is the entry point.
QSBC	qsbc.mak	qsbc.asm	31-bit BAL segment with 2 transfer vectors (QSBC and QSBD).
		qsb7.asm	Standard 64-bit BAL segment with QSB7 entry point.
QSBE	qsbe.mak	qsbe.asm	Standard 31-bit BAL segment with QSBE entry point.
QSWP	qswp.mak	qswp.asm	31-bit BAL segment with a program base register and no transfer vectors.

CSOs

SOLD Driver Infrastructure			
Module	**Makefile**	**Segment**	**Description**
QSCA	qsca.mak	qsca.c	Affinity of QSCA has to be set to PROGRAM in order to start/stop continuous mode on a specific I-stream. Points to the driver, which parses the arguments and then invokes the appropriate function. The qsca.c prolog contains a more detailed description of the test cases.
QSCB	qscb.mak	qscb.c	Affinity of QSCB has to be set to PROGRAM in order to start/stop continuous mode on a specific I-stream. Operates the driver in continuous mode.
QSCC	qscc.mak	qscc.c	Contains the functions that execute the individual test cases.
QSCD	qscd.mak	qscd.c	Contains utility functions that are used to print common messages to the screen.
		qscd98.c	Contains functions to test long function names that have greater than 2000 characters.
		qscd99.c	Contains functions to test long function names including boundary conditions.

ELF Executables (main()) Code			
Module	**Makefile**	**Segment**	**Description**
QME1	qme1.mak	qme1.c	ELF executable with accessory functions.
QME2	qme2.mak	qme2.c	Standard ELF executable.
QME3	qme3.mak	qme3.c	ELF executable with an accessory function.
QME4	qme4.mak	qme4.cpp	C++ ELF executable with various accessory functions for function trace verification.

CSO Libraries			
Module	**Makefile**	**Segment**	**Description**
QSC1	qsc1.mak	qsc1a.c	Contains a simple C function.
		qsc1b.c	Contains a simple C function.
		qsc1trace.c	Contains C functions for function trace verification.
QSC2	qsc2.mak	qsc2a.c	Contains a simple C function.
		qsc2b.c	Contains a simple C function.
QSC3	qsc3.mak	qsc3a.c	Contains a simple C function.
		qsc3b.c	Contains a simple C function.

CSOs with Single Entry Point			
Module	**Makefile**	**Segment**	**Description**
QCE1	qce1.mak	qce1.c	CSO with a static variable, accessory function and QCE1 entry point. The tracegroup for QCE1 is TRG_3.
QCE2	qce2.mak	qce2.c	Standard CSO with an accessory function and QCE2 entry point. The tracegroup for QCE2 is TRG_2.
		qce210.asm	Tests PRLGC 31/64 bit support.
QCE3	qce3.mak	qce3.c	Standard CSO with QCE3 entry point. The tracegroup for QCE3 is TRG_1.

Note: We may need more QSBx segments as you can not have more than one BAL program in each file (BEGIN/FINIS can only be called once for each file).

Linkage information

This section contains the linkage information for the SOLD driver.

Notes:

- Please see the qsca.c segment for a more detailed description of the test cases and for the exact paths between the modules for each test case.
- As far as possible, all test case calls should have some parameters, preferably 6 or more. From caller to called, there should be a way to verify if the parameters are received correctly. One way is to pass parameters that add up to zero. Have some signed and some unsigned. Same applies for calls to BAL programs. Pass some values in registers (TPF_regs - with values for some of registers 0-7) and have a way to verify. Pass signed as well as unsigned.
- Keep one CSO without any writable static variables. In other CSOs, define and use static variables in alternate source files.

C to C: (test case range 1-25)

1) C entry → C entry → C function (intra) → C function (inter) → return all the way back.
2) C entry → C function (inter) and pass a function pointer (intra) → call passed function → return all the way back.
3) C entry → C function (inter) and pass a function pointer (inter from a 3rd module) → call passed function → return all the way back.
4) C entry → C function (intra) and pass a function pointer (inter from a 3rd module) → call passed function → return all the way back.
5) C (main) → C function (intra) and pass a function pointer (intra) → call passed function → return all the way back.
6) C (main) → C entry → C function (intra) → C function (inter) → return all the way back.
7) C (main) → C function (intra) → C entry → C function (inter) → return all the way back.
8) C (main) → C function (inter) → C entry → return all the way back.
9) C (main) → C function (inter) and pass a function pointer (intra) → call passed function → return all the way back.

10) C (main) → C function (inter) and pass a function pointer (inter from a 3rd module) → call passed function → return all the way back.
11) Test user-defined errno values.
12) Test ENTRC with new options.
13) Test long function names, including boundary conditions.
14) Test long function names using names greater than 2000 characters.
15) Test BSO calls QQQQ.
16) Test CSO calls QQQQ.
17) Test BSO calls data program QXAX.
18) Test CSO calls data program QXAX.
19) Test BSO calls GTAL.
20) Test CSO calls GTAL.
21) Test CSO calls nonexistent function.
22) Run test case 15-21.

23 to 25 unused.

Note: Above test cases can be repeated for C++ but may not be of great value. In z/TPF, there is very little difference between C and C++ unless C++ features are used. C team will add any test cases that test C++ features separately. C team will also cover test cases that call CTAL style functions (C functions written in BAL).

BAL to BAL: (test case range 26-50)
Note: Programs in BSO can be either 32-bit or 64-bit. There will be one function (BAL program) that matches the BSO name. The file names do not have to match the program name. So the entry function can be in BSO name source file. Other functions in the same BSO can be in file names with BSO prefix. Make some of the BAL programs greater than 4K and with multiple base registers. Some BAL programs will be base-less.

26) BAL (entry) → BAL (intra) → BAL (inter, base-less) → BAL TV inter → return all the way back.
27) BAL (entry) → BAL (intra >4K >1 base) → BAL (inter) → BAL TV inter → return all the way back
28) BAL64 (entry) → BAL64 (intra) → BAL64 inter TV → BAL64 (inter, base-less) → return all the way back.
29) BAL64 (entry) → BAL64 (intra, >4K >1 base) → BAL64 inter TV → BAL64 (inter) → return all the way back.
30) BAL (entry) → BAL64 (entry) → BAL64 intra TV → BAL64 (inter TV) → return all the way back.
31) BAL64 (entry) → BAL TV → BAL64 (intra) → BAL (inter) → BAL64 (intra) → return all the way back
32) BAL (entry) → ENTDC to → BAL TV inter → ENTNC to → BAL TV intra → EXIT
33) BAL64 (entry) → ENTNC to → BAL64 TV intra → ENTNC to → BAL64 TV inter → EXIT

34) BAL (entry) BSS → CROSC to → BAL TV WP → return
Note: Case 34 used also to validate save/restore of PBI across CROSC ENTxC

35-50) Unused

<u>Mixed: (C, BAL, C(main)): (test case range 51-75)</u>

51) C entry → BAL64 (entry) → C entry → BAL (entry) → return all the way

52) C entry → BAL → C entry → BAL64 → return all the way

53) BAL (entry) → C entry → BAL64 TV → C entry → BAL TV → return all the way

54) BAL64 → C entry → BAL64 TV → C entry → BAL TV → return all the way

55) C (main) → BAL64 → C entry → BAL → return all the way

56) C (main) → BAL → C entry → BAL64 → return all the way

57) C (main) → BAL64 TV → C entry → BAL → return all the way

58) BAL64 → via CALLC CALL a C function → Called function → BAL → return all the way

59) BAL64 → via CALLC CALL a C function → Called function → BAL64 → return all the way

60) BAL → via CALLC call a C function → called function → C entry → return all the way

61) C(main) → C entry →|C entry → BSO → returns all the way
→|C entry → BSO64 → returns all the way

62) ECB Trace test case: C entry → BAL64 → C entry → BAL64 → return all the way
Note: This test is not included when the driver is run in continuous mode.

63) ECB Trace test case: C entry → BAL64 → C entry → BAL64 → C entry → return all the way
Note: This test is not included when the driver is run in continuous mode.

64-75) Unused

<u>Function Trace Tests: (C(main), C++(main)): (test case range 100-125)</u>

100) C++ entry → various C++ functions (intra) → return all the way

101) C entry → C functions to test char (inter) → return all the way

102) C entry → C functions to test unsigned char (inter) → return all the way

103) C entry → C functions to test short (inter) → return all the way

104) C entry → C functions to test unsigned short (inter) → return all the way

105) C entry → C functions to test int (inter) → return all the way

106) C entry → C functions to test unsigned int (inter) → return all the way

107) C entry → C functions to test long (inter) → return all the way

108) C entry → C functions to test unsigned long (inter) → return all the way

109) C entry → C functions to test long long (inter) → return all the way

110) C entry → C functions to test unsigned long long (inter) → return all the way

111) C entry → C functions to test float (inter) → return all the way

112) C entry → C functions to test double (inter) → return all the way

113) C entry → C functions to test long double (inter) → return all the way

114) C entry → C functions to test variable parameters (inter) → return all the way

115) C entry → C functions to test void parameters (inter) → return all the way

116) C entry → C functions to test tpf_trace_info() (inter) → return all the way

117) C entry → C functions to test tpf_trace_info() (inter) → return all the way

118) C entry → C functions to test tpf_trace_info() (inter) → return all the way

119) C entry → C functions to test tpf_trace_info() (inter) → return all the way

120) C entry → C functions to test tpf_trace_info() (inter) → return all the way

121) C entry → C functions to test typedef (inter) → return all the way

122) C entry → C functions to test structures in registers (inter) → return all the way

123) C entry → C functions to test structures in registers (inter) → return all the way

124 - 125) Unused

Additional information

None.

Examples

The following examples will display the help menu:

```
ZTEST SOLD HELP
ZTEST SOLD ?
```

The following example will run test case 7 on I-stream 1:

```
ZTEST SOLD CASE-7
```

The following example is the same as above, but dumps after running the test:

```
ZTEST SOLD CASE-7 DUMP
```

The following example shows the program version when entering a segment (excludes test cases 15-22):

```
ZTEST SOLD CASE-7 SHOWVER
```

The following example will run test case 12 on I-stream 2:

```
ZTEST 2 SOLD CASE-12
```

The following example will run test case 99 on all I-streams:

```
ZTEST * SOLD CASE-99
```

The following example will execute the driver in continuous mode on I-stream 3:

```
ZTEST 3 SOLD START
```

The following example stop the driver in continuous mode on all defined I-streams:

```
ZTEST * SOLD STOP
```

The following example will display the status of executed test cases on I-stream 2:

```
ZTEST 2 SOLD STATUS
```

The following example will display the status of executed test cases on all I-streams:

```
ZTEST SOLD STATUS
```

Messages

Below is a list of the SOLD driver messages:

SOLD0001I SOLD IS STARTING IN CONTINUOUS MODE ON ISTREAM *x*

Explanation: The operator entered ZTEST SOLD START and the SOLD driver is starting in continuous mode. Enter ZTEST SOLD STOP to stop continuous mode.

SOLD0002I SOLD STATUS

Explanation: SOLD status message.

SOLD0003I TEST CASE *d* HAS STARTED

Where

d test case number entered by ZTEST SOLD CASE-*xx*

Explanation: Test case *d* was requested through the ZTEST SOLD CASE-*xx* command. Execution of the test case has started.

SOLD0004I TEST CASE *d* HAS COMPLETED

Where

d test case number entered by ZTEST SOLD CASE-*xx*

Explanation: Test case *d* was requested through the ZTEST SOLD CASE-*xx* command. Execution of the test case has completed.

SOLD0005I STOPPING ALL INSTANCES OF SOLD ON ISTREAM *x*

Explanation: The operator entered ZTEST SOLD STOP and the SOLD is in the process of stopping all instances of the SOLD driver on I-stream *x*.

SOLD0006I SOLD STOPPED ON ISTREAM *x*

Explanation: All instances of the SOLD driver on I-stream *x* are stopped.

SOLD0007I SOLD IS IN THE PROCESS OF STOPPING

Explanation: The SOLD driver is in the process of being stopped.

SOLD0010I SOLD SYNTAX:

```
  ZTEST i SOLD CASE-xx ((NO)DUMP|(NO)SHOWVER)
     Execute one test case.
  ZTEST i SOLD START
     Start continuous operation.
  ZTEST i SOLD STATUS
     Display status of running driver.
  ZTEST i SOLD STOP
     Stop SOLD in continuous mode

PARAMETERS:
  i : I-STREAM NUMBER
  xx: TEST CASE NUMBER, 1-125
  DUMP: CAUSE THE DRIVER TO DUMP AFTER COMPLETION
  SHOWVER: DISPLAY THE PROGRAM VERSION AND VARIATION NUMBER
```

Explanation: Help message for SOLD driver invoked through ZTEST SOLD HELP or ZTEST SOLD ?.

SOLD0086W SYSTEM BELOW LODIC BATCH SHUTDOWN LEVELS - SOLD IN DEFER LOOP

Explanation: SOLD is running in continuous mode and is unable to start new test cases because the system is currently running below the BATCH shutdown levels. To keep the system from running out of resources, SOLD has put itself into a defer loop. Stop the SOLD driver or increase system resources.

SOLD0087E UNABLE TO START - SOLD IS STOPPING

Explanation: SOLD START was entered, but the driver was in the process of stopping. A new instance of the driver is not started.

SOLD0088E ERROR GETTING SYSTEM HEAP FOR CONTROL BLOCK

Explanation: SOLD START was entered, but was unable to obtain system heap for the SOLD control structure. START processing is aborted.

SOLD0089E *xxxx*: BASE REGISTER NOT RESTORED CORRECTLY

Where
xxxx: 4-character name of segment issuing the error message

Explanation: Segment *xxxx* did not correctly restore a base register after making a function call.

SOLD0090E *xxxx*: STATIC VARIABLE HAS WRONG VALUE FOR TEST CASE *d*

Where
xxxx: 4-character name of segment issuing the error message
d: test case number

Explanation: Segment *xxxx* did not find the correct value of a static variable for test case *d*.

SOLD0091E *xxxx*: BAD RETURN VALUE FROM *yyyy* FOR TEST CASE *d*

Where
xxxx: 4-character name of segment issuing the error message
yyyy: 4-character name of segment that is being returned from
d: test case number

Explanation: Segment *xxxx* did not receive the correct return values from segment *yyyy* for test case *d*.

SOLD0092E *xxxx*: PARAMETER PASSING ERROR IN TEST CASE *d*

Where
xxxx: 4-character name of segment issuing the error message
d: test case number

Explanation: Segment *xxxx* received invalid input parameters for test case *d*.

SOLD0093E *xxxx*: INVALID TEST CASE NUMBER *d*

Where
xxxx: 4-character name of segment issuing the error message
d: test case number

Explanation: Segment *xxxx* received was called for test case number *d*, but should not get called for this particular test case.

SOLD0094E INVALID TEST CASE NUMBER *d*
SEE QSCA.C FOR THE COMPLETE LIST OF VALID
SOLD DRIVER TEST CASE NUMBERS

Where

d: test case number

Explanation: An invalid test case number was entered on the ZTEST SOLD CASE-*xx* command. See segment qsca.c for a list of valid test case numbers.

SOLD0095E SOLD IS NOT RUNNING

Explanation: The STOP or STATUS command was entered but the SOLD driver is not currently running.

SOLD0096E SOLD CONTROL BLOCK NOT FOUND

Explanation: The driver expected to find the SOLD control block, but a NULL pointer was returned by tpf_fsysc. The driver exits.

SOLD0097E SOLD IS ALREADY STOPPING ON ISTREAM *x*

Explanation: Multiple STOP requests have been defined while the original STOP request is still being handled. A single STOP request stops all instances of the SOLD driver on that specific I-stream.

SOLD0098E SYSTEM IS CYCLING - SOLD CANNOT BE RUN

Explanation: The system is cycling to another state. Wait until system cycle is completed before issuing the ZTEST SOLD command.

**SOLD0099E You entered ZTEST SOLD with invalid syntax.
Enter ZTEST SOLD HELP to display the correct syntax.**

Explanation: Invalid syntax was entered for the ZTEST SOLD command.

SOLD0100E QSWP: CROSC SS ERROR FOR TEST CASE ..

Explanation: UATBC returned an error when QSWP attempted to retrieve the SS name.

SOLD0101I QSWP: CURRENT SS ....

Explanation: Informational display to track the subsystem hop.

SOLD0093I QCE2: CASE 61 PSW Check OK..

Explanation: Test of PSW for 31-bit mode is verified.

SOLD0093I QCE2: CASE 61 PSW Check FAILED.

Explanation: Test of PSW for 31-bid mode has failed.

SOLD0093I QCE2: CASE 61 TESTING PRLGC 31-BIT mode.

Explanation: Informational display to indicate start of test case.

SOLD0093I QCE2: CASE 61 MALLOC FAILED.

Explanation: Request for malloc space failed.

SOLD0093I QCE2: CASE 61 STACK FAILED.

Explanation: Stack pointer save/restore over CALLC failed.

SOLD0093I QCE2: CASE 61 STACK OK.

Explanation: Stack pointer save/restore over CALLC validated.

SOLD0093I QCE2: CASE 61 TESTING PRLGC 64-BIT mode.

Explanation: Informational display to indicate start of test case.

References

For more information about reading syntax diagrams, also referred to as railroad diagrams, see *Accessibility information* in the TPF Product Information Center.