Wednesday, October 22, 2025, 11:00 AM, Amphitheater

OpenVMS DEBUG: A Powerful Tool for Sophisticated Debugging

2025 OpenVMS Bootcamp
Portsmouth Sheraton
Portsmouth, New Hampshire USA

Agenda –

- OpenVMS DEBUG is powerful, but underused
- “Work smarter, not harder”
- Think process – combine DEBUG with internal fault logic

References

- OpenVMS DEBUG Manual

Introduction

Basic DEBUG usage

- RSX-11 Octal Debugging Technique (“ODT”)
- Load; Store; Breakpoints; non-symbolic
- Better than PRINT statements

OpenVMS DEBUG

- Major improvements over ODT-type use
- Symbolic
- Orders of magnitude more powerful
- Power can be leveraged for more sophisticated use

Advanced Features

- Initialization files
- Session Logging
- Conditional breakpoint actions
- Nestable debugging scripts
- Standalone calls

Sample Program

```
#include <types.h>
#include <stdio.h>

long usestring(char *string)
{
     return 2;
     }

int main(int argc, char **argv[])
{
     long value1[100];
     long value2[100];
     int i, j, k;
     char buffer[200];

     for (i=0; i<100; i++)
         {
          value1[i] = i;
          value2[i] = i * i;
          sprintf(&buffer[0], "Value 1[%d]: %ld; Value2[%d]: %ld\n",
              i, value1[i], i, value2[i]);
          usestring(&buffer[0]);
          }

     }
```

Compile and Link Sample Program for Debugging

- Remember to compile with /DEBUG for symbols
- Optimization makes debugging harder, user /NOOPTIMIZE

```
$ cc/list/debug/nooptimize test
$ link/debug test
```

Initialization File DBG$INIT

- DBG$INIT is processed when DEBUG starts
- User or Supervisor logical name
- Saves typing, reduces errors
- Length not limited

```
$ create init.dbg
! Author: Robert Gezelter     10-October-2025
!
set break/exception
go
^Z
$ assign/user init.dbg dbg$init
$ run test

         OpenVMS Alpha Debug64 Version V8.4-002

%DEBUG-I-INITIAL, Language: C, Module: TEST
%DEBUG-I-NOTATMAIN, Type GO to reach MAIN program
break at routine TEST\main
  1900:         for (i=0; i<100; i++)
DBG> go
%DEBUG-I-EXITSTATUS, is '%SYSTEM-S-NORMAL, normal successful completio
DBG> ^Z
```

Session Logging

- Let DEBUG keep your notes
- Every command (including comments ["!"];)
  every output
- Faster than hand, more accurate, less distraction
- Basis for later script

Session Logging (continued)

```
$ run test

         OpenVMS Alpha Debug64 Version V8.4-002

%DEBUG-I-INITIAL, Language: C, Module: TEST
%DEBUG-I-NOTATMAIN, Type GO to reach MAIN program
DBG> set log session.log
DBG> set output log
DBG> show output
noverify, terminal, noscreen_log, logging to session.log
DBG> set break %line 1906
DBG> go
break at routine TEST\main
  1900:         for (i=0; i<100; i++)
DBG> go
break at TEST\main\%LINE 1906
  1906:                 usestring(&buffer[0]);
DBG> ^Z
$
$
$ type session.log
show output
!noverify, terminal, noscreen_log, logging to session.log
set break %line 1906
go
!break at routine TEST\main
!  1900:         for (i=0; i<100; i++)
go
!break at TEST\main\%LINE 1906
!  1906:                 usestring(&buffer[0]);
EXIT
```

Conditional Break Actions

- Many uses, e.g., ignore irrelevant breakpoint trips
- Take actions based on certain values
- Highly leverageable

Conditional Break Actions (continued)

```
$ run test

         OpenVMS Alpha Debug64 Version V8.4-002

%DEBUG-I-INITIAL, Language: C, Module: TEST
%DEBUG-I-NOTATMAIN, Type GO to reach MAIN program
DBG> set break %line 1906 do (if i == 50 then (examine value1[i],
value2[[i]; go) else (go))
DBG> go
break at routine TEST\main
  1900:         for (i=0; i<100; i++)
DBG> go
break at TEST\main\%LINE 1906
  1906:                 usestring(&buffer[0]);
break at TEST\main\%LINE 1906
  1906:                 usestring(&buffer[0]);
...
break at TEST\main\%LINE 1906
  1906:                 usestring(&buffer[0]);
TEST\main\value1[50]:   50
TEST\main\value2[50]:   2500
break at TEST\main\%LINE 1906
  1906:                 usestring(&buffer[0]);
break at TEST\main\%LINE 1906
  1906:                 usestring(&buffer[0]);
,,,
break at TEST\main\%LINE 1906
  1906:                 usestring(&buffer[0]);
%DEBUG-I-EXITSTATUS, is '%SYSTEM-S-NORMAL, normal successful completion
DBG> ^Z
```

Nestable Scripts

- Callable at any point
- Can be arbitrarily nested

```
$ run test

         OpenVMS Alpha Debug64 Version V8.4-002

%DEBUG-I-INITIAL, Language: C, Module: TEST
%DEBUG-I-NOTATMAIN, Type GO to reach MAIN program
DBG> @ifpoint.dbg
DBG> show break
breakpoint at routine TEST\main [temporary]
breakpoint at TEST\main\%LINE 1906
   do (if i == 50 then (examine value1, value2;go) else (go))
DBG> ^Z
```

Standalone Calls

- DEBUG allows direct invocation of image subroutines
- Parameters are allowed
- More complex processing than DEBUG's record display
- Semantically-aware decoding, diagnosis

```
$ run test
         OpenVMS Alpha Debug64 Version V8.4-002
%DEBUG-I-INITIAL, Language: C, Module: TEST
%DEBUG-I-NOTATMAIN, Type GO to reach MAIN program
DBG> call usestring
value returned is 2
DBG>^Z
```

Combinations

- DEBUG facilities can be combined: DBG$INIT can invoke module-specific scripts to establish strategic breakpoints, that when triggered invoke subroutines to display useful information
- Frees developer from the keyboard drudge
- Repeatable actions

Common Actions

- Tired of repeatedly typing same commands - use script
- Doing the same setup for every session? DBG$INIT
- Breakpoint conditional actions allow complex reactions: less typing, more thinking

Summary

- Debugging large, complex programs is different than debugging small programs
- OpenVMS Debug has features to make debugging easier
- Resolve/prevent conflict
- Strategic use of DEBUG features complements internal fault detection and response

Questions?

Robert Gezelter Software Consultant
35 – 20 167th Street, Suite 215
Flushing, New York 11358 – 1731
United States of America

+1 (718) 463 1079
gezelter@rlgsc.com
http://www.rlgsc.com

Session Notes & Materials:
http://www.rlgsc.com/openvms-bootcamp/2025/index.html