Thursday, October 29, 2026, 2:00 PM, Lear Room

OpenVMS x86-64 Reentrant and Reusable Code

2026 OpenVMS Bootcamp
Portsmouth Sheraton
Portsmouth, New Hampshire USA

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Agenda –

- Reusable v Reentrant
- Where unsafety lines
- Reentrant code
- OpenVMS support
- Reusable and reentrancy are complementary, not mutually exclusive

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

References

- OpenVMS System Management Manual
- OpenVMS C User's Manual
- OpenVMS LIB$ Reference Manual
- OpenVMS STR$ Reference Manual
- OpenVMS System Services Reference Manuals
- OpenVMS Programming Concepts Manual

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Introduction

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reusable v Reentrant

- Reusable is an OpenVMS feature, e.g., INSTALL
- Reentrant refers to “simultaneous” use within an OpenVMS process
- Both require caution wrt static and global storage
- Generally, storage shared between processes is complex and to be avoided

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reusable Code – Overview

- Reusable code means that a library is external to an executable image
- INSTALLing reusable code means that OpenVMS will keep one copy of the reusable code in memory
- Global and static variables can be shared, per-process private, or some combination

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reusable Code – Overview

- A routine can be both reusable AND reentrant
- INSTALLed images use copy-on-write for writeable, non-shared regions
- Reentrant code must use either stack frames or dynamically allocated memory for writeable storage
- Beware memory leaks; beware out of scope variables
- If you have routines that invoke QIO and similar

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reusable Code – Shared/Private Data

- Ensure global/static data is process-private
- Compiler pragmas, e.g., C #pragma extern_model
- LINKER psect commands
- Failures can be subtle and hard to reproduce

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reusable Code – Compiling

- Verify that data is correctly shared
- Per process is NOSHR, WRT
- Shared among processes is SHR, WRT
- Tedious and challenging to troubleshoot

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reusable Code – Linking

- Verify and possibly correct PSECT attributes
- Inspect LINK map to verify data attributes
- Keep symbol table, map files

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reusable Code – Non-Privileged User

- LINK with specified library
- At run time, ensure that logical name is defined
- RUN program as usual
- Code/Data segments will not be shared
- Multiple shareable libraries are fully supported, see my 1993 presentation on Shareable Library Case Study

```
$ ASSIGN/USER_MODE DISK$USERS:[GEZELTER]XYZLIB.EXE XYZLIB
```


© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reusable Code – Privileged User

- Users holding CMKRNL can run INSTALL
- INSTALL the shareable image
- Image now available globally
- All processes are using the same copy
- Beware shared data regions!

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reentrant Code – Overview

- Code executed simultaneously within a process
- Can be direct (recursion) or indirect
- One or more threads, e.g., DECthreads, AST
- Requires precautions to avoid problems

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reentrant Code – OpenVMS Compiler Options

- /RENTRANT switches have limited meaning, e.g., set flags used by RTL; do not affect code generation
- Compiler Built-ins, e.g., C/C++ builtins.h

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reentrant Code – Forbidden Choices

- Unsafe writeable static and/or global data
- Deferred references to dynamic data, e.g., QIO IOSB allocated within the stack frame
- Problems can surface at any time, often long after release to production

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reentrant Code – Is shared state necessary?

- Is shared state really needed?
- Is state tied to the process, thread, or call
- Timing relationships are often difficult to visualize

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reentrant Code – Interlocked Instructions

- Understand your data!
- Some is global within a process, e.g., random number seeds
- Some structures simply persist beyond local scope, e.g., I/O request structures and buffers
- Structures that are used by different threads MUST use atomic update sequences, see builtins, e.g., COMPARE & SWAP in concert with memory barriers
- Timing problems are devilishly difficult to find

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reentrant Code – Interlocked Instructions (continued)

- Memory barriers not needed as much on x86-64; but are more necessary on Alpha/IA64; could be needed in future
- Timing problems are devilishly difficult to find
- Be careful of cache line alignment issues, e.g., may differ on different implementations of same instruction set

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reentrant Code – No shared state

- Where possible – Avoid shared state
- Temporary structures local in stack frame
- see “Introduction to AST Programming” from earlier OpenVMS Bootcamps

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Reentrancy – Summary

- Reentrant routines with no persistent state – straightforward
- Persistent state or persistent structures – complexity quickly increases

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

Summary

- Reusable and Reentrant code is fine, provided caution and safe practices are followed
- Reusability and reentrancy are not mutually exclusive
- Unwitting shared static data is the hazard
- Check LINKER maps carefully
- Compiler options control RTL usage, not your code
- Single user/thread mode can be deceptive, be alert for unusual behavior

© 2026, Robert Gezelter Software Consultant, All Rights Reserved

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/2026/index.html

© 2026, Robert Gezelter Software Consultant, All Rights Reserved