OpenVMS Shareable Libraries: An Implementor's Guide

Thursday, October 24, 2024

2024 OpenVMS Bootcamp
Marriott Long Wharf
Boston, Massachusetts

Introduction

When and Why?

It is well known that shareable libraries make sense in heavily used applications. For example, the OpenVMS Run-Time library is implemented as a series of Shareable Libraries.

When and Why? (cont'd)

Not as well known are the benefits realized in program development and applications implementation. These benefits are completely user realizeable, and are separate from the traditional, well–known system–wide benefits of using shareable libraries.

Maintenance

No need to re-link entire program for a change in one routine.

Ability to quickly switch between new and old versions of routines.

Speed/Efficiency

- **INSTALLed shareable image**
- **Read-only pages shared by many processes**
- **Significant reduction in memory requirements**
- **Significant reduction in disk storage requirements**

“Leave No Stone Unturned”

Changes in object libraries require relinking to take effect

Relinking is a major task in a medium/large facility (tens or hundreds of programs)

Dynamic Code Generation

Permits execution time customization

Highly efficient

Simplifies code

Old tactic; but not well known

Why use Shareable Libraries?

- **efficiency/performance**
- **maintenance/change control**
- **eliminate regression**
- **leave “No Stone Unturned” (or program un-relinked!)**

Why use Shareable Libraries? (cont'd)

- **different programmers can work on different parts of the project at the same time without interfering with each other.**

What is an OpenVMS Shareable Library?

A Shareable Library is a section of code and/or data which is dynamically linked to your program at image activation.

Normal usage does not require any privileges not available to a Student user.

What kinds of code can be included in a Shareable Library?

Almost any code can be placed in a shareable library. The main requirement is that the code be referenced by one or more programs or developers.

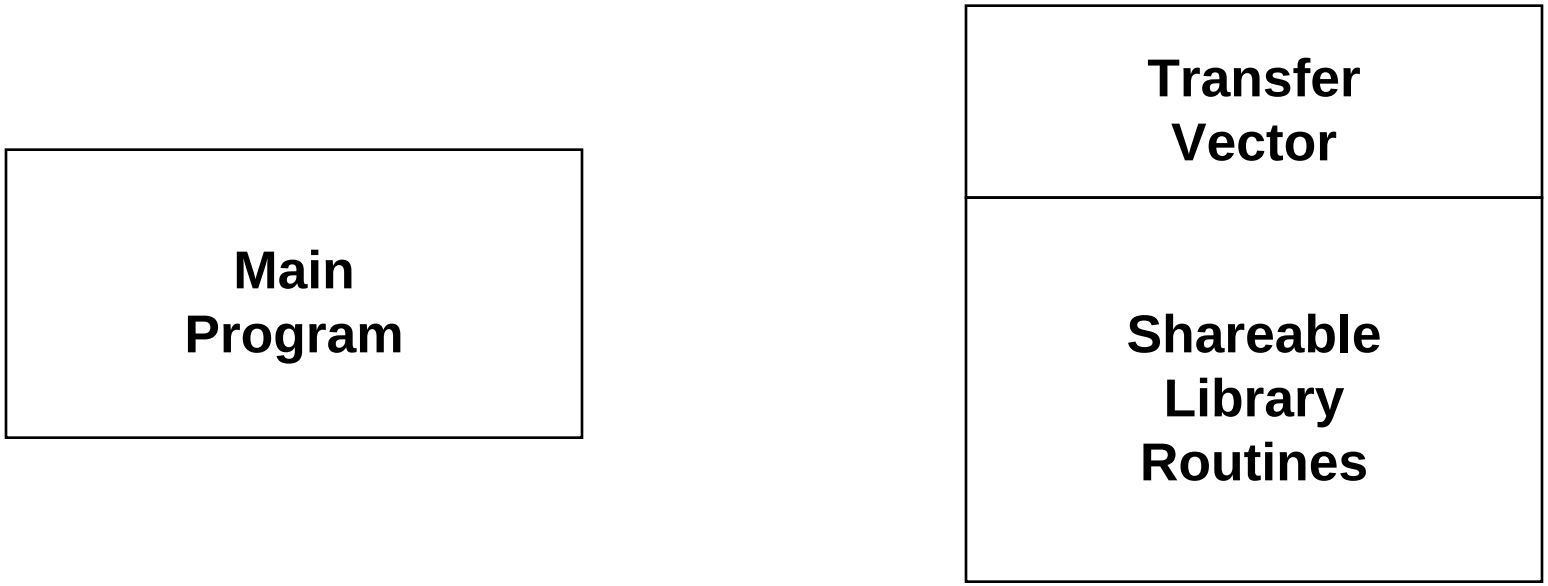
Can data be included in a Shareable Library?

Yes, data can be included in a shareable library.

However, to ensure safety, you should make sure that the data is

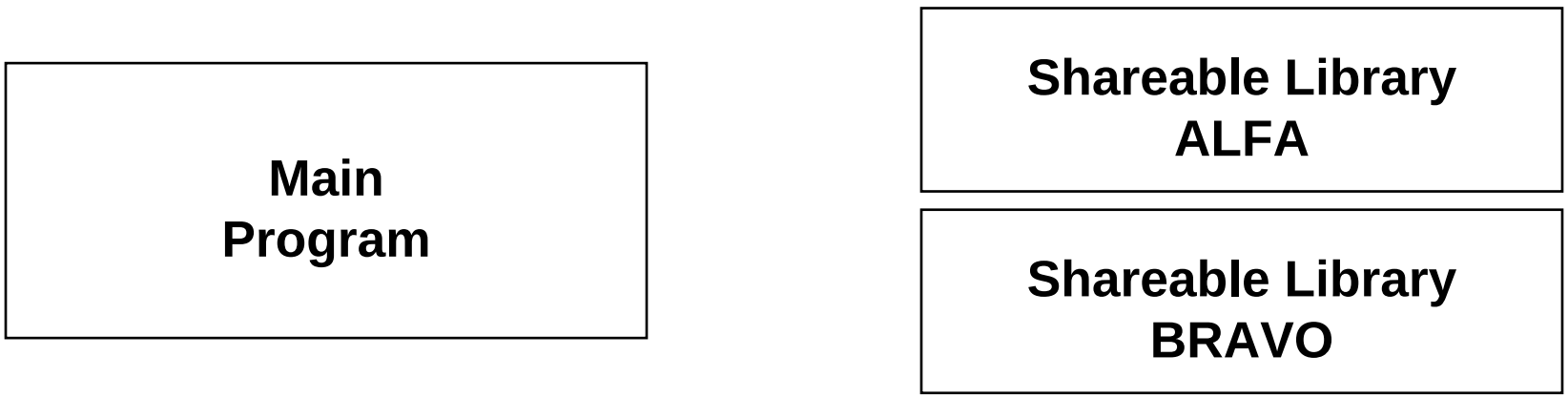
Read only (NOWRT)
OR
Copy on Reference.

What happens when I call a routine in a shareable library?



Can I use multiple shareable libraries at the same time?

YES!!!



How do I specify a shareable library at execution time?

Use logical names.

No privileges required!

```
$ TEKPLT_EXE:[GEZELTER]TEKPLT.EXE TEKPLT
$ RUN program
```

On OpenVMS–ALPHA/IA-64/x86-64

In the LINKER Options File:

```
SYMBOL_VECTOR=(name1=PROCEDURE,-
              name2=PROCEDURE,-
              SPARE,-
              SPARE,-
              SPARE,-
              SPARE)
```

What do I save by using Shareable Libraries?

- **link time (huge savings possible)**
- **disk space**
- **maintenance effort**
- **regression errors**
- **change management**
- **measured updates**

Guidelines:

- **provide ID entry points**
- **have main system produce optional revision listing of libraries used**
- **be careful of multiple versions**
- **be extremely careful of shareable, writeable data!!!! (JUST SAY NO!)**
- **enforce use of libraries**

Shareable Libraries – Concepts

All calls to entry points in shareable libraries are routed through transfer vectors.

Most data areas are allocated as non-shareable space or are located on the stack.

Normal use requires no privileges. Actual sharing of code/data requires the privileges to INSTALL the image.

Source Program Concerns

Avoid impure references; address constants, use MOVA-type instructions instead (a MACRO issue)

Watch out for:
COMMONs (FORTRAN); external variables (C); and similar structures

Compiler-related issues

Watch out for PSECT attributes!

Check LINK map!

In particular, the combination of SHR and WRT is generally a bad idea (when the image is installed, different processes will share Read/Write data).

LINKER-related issues

/SHARE switch on command

GSMATCH=LEQUAL,1,0 (in OPT file)

Fix PSECT attributes (if needed)

Be sure to check MAP file

Debugging Concerns

Try to debug before releasing shareable image to the world.

Local logical names override more global names, thus you can switch between production and test versions from minute to minute.

Cases from our Files:

We will present two case studies:

- **Development advantages**
- **Applications tool for dynamic code generation**

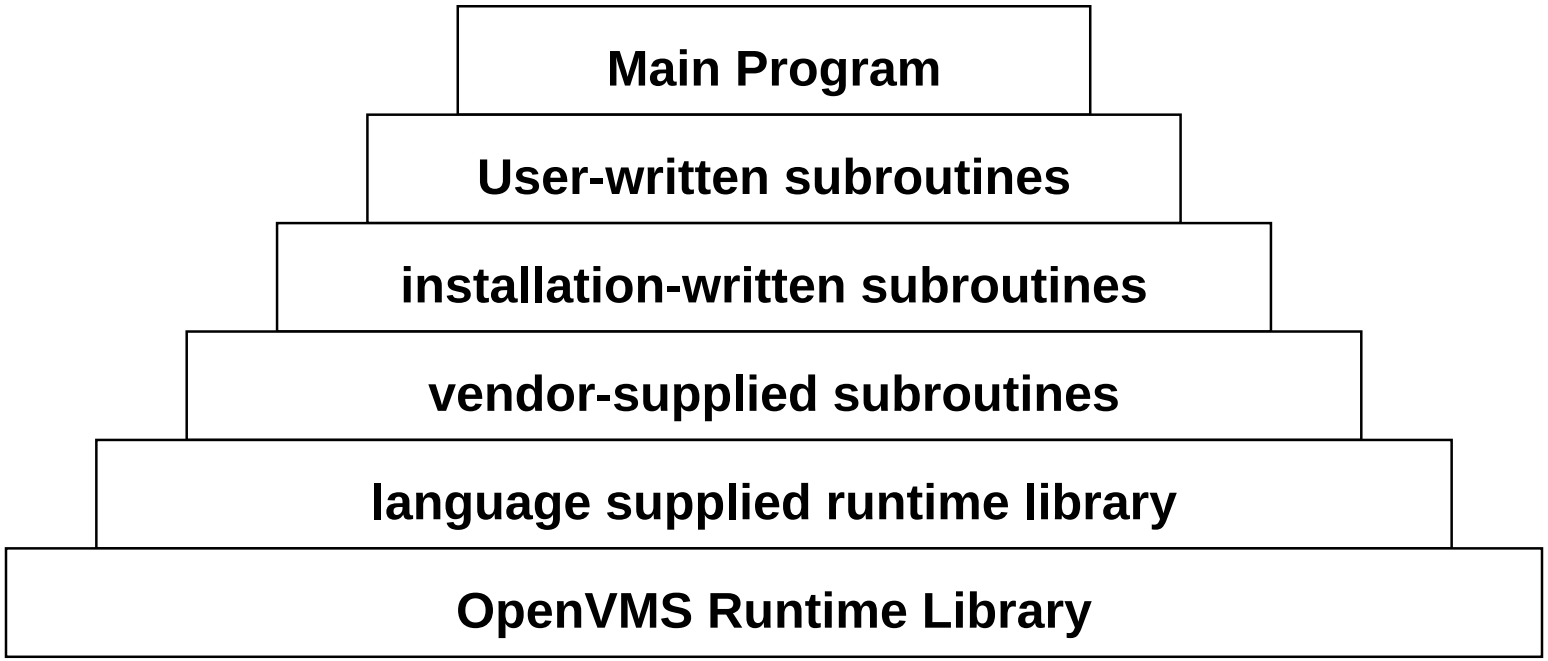
Case 1 – Development

Symptom:

Large Program – Slow Links

Linking this program reduced 99%; up to 20 minutes on a VAX–11/780

Problem: Programs are like pyramids – very large foundation

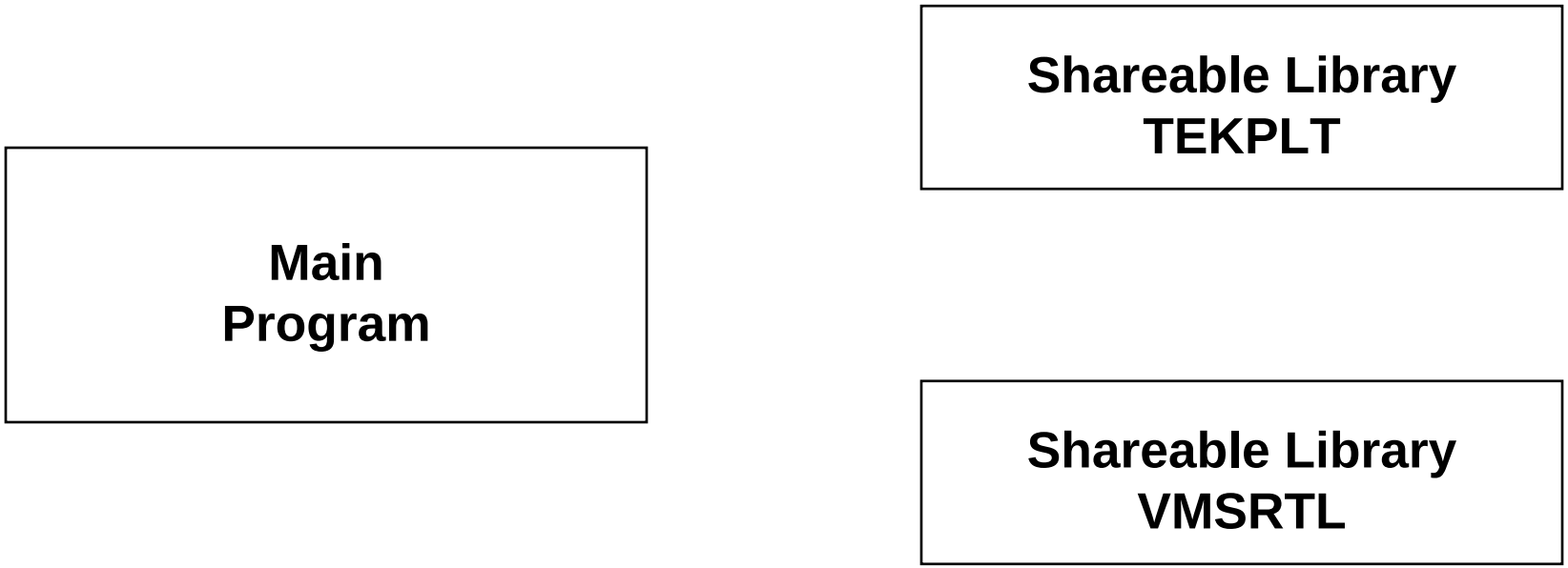


Solution:

Create one or more user shareable images containing most of the foundation elements.

Result:
Link time reduced 99%!

The support code, which is the bulk of the image, is in the shareable libraries!



Specify the shareable library at execution time

Use logical names.

No privileges required!

```
$ ASSIGN TEKPLT_EXE:[GEZELTER]TEKPLT.EXE TEKPLT
$ RUN program
```

Case 2 – Dynamic Linking a.k.a. Power Tools with Interchangeable Heads/Bits

Most programs are written to do a particular job.

How does one write a program to do many different jobs?

With Shareable Libraries, of course!

Case Subject: Mailing List System

Must generate:

- **Labels**
- **Envelopes**
- **Form letters**
- **Invitations**
- **Listings**
- **Attendee Lists**
- **...**

Problem: Complexity

Program complexity grows as an exponential (nm) of the number of different options AND the number of different values of the options**

Complexity

Research has shown that correctness of code is endangered by large numbers of nested IF statemets

Progamming by Components

Record Processing Initialize, Process, End
Accept Record
Name Builder
Utilities
Process Utilities

Programming By Chinese Menu

Pick:

1 from Column A

1 from Column B

3 from Column C

Conventional Programming

Column A: 5 possible choices

Column B: 7 possible choices

Column C: 30 possible choices

TOTAL: 1050 programs
(5 * 7 * 30)

Goal: Develop a large family of related programs with minimal effort

Maintain separation between different applications

Programming By Chinese Menu

5 Group A subroutine packages

7 Group B subroutine packages

30 Group C subroutine packages

1 Main Program

TOTAL: 43 programs / packages
(5 + 7 + 30 + 1)

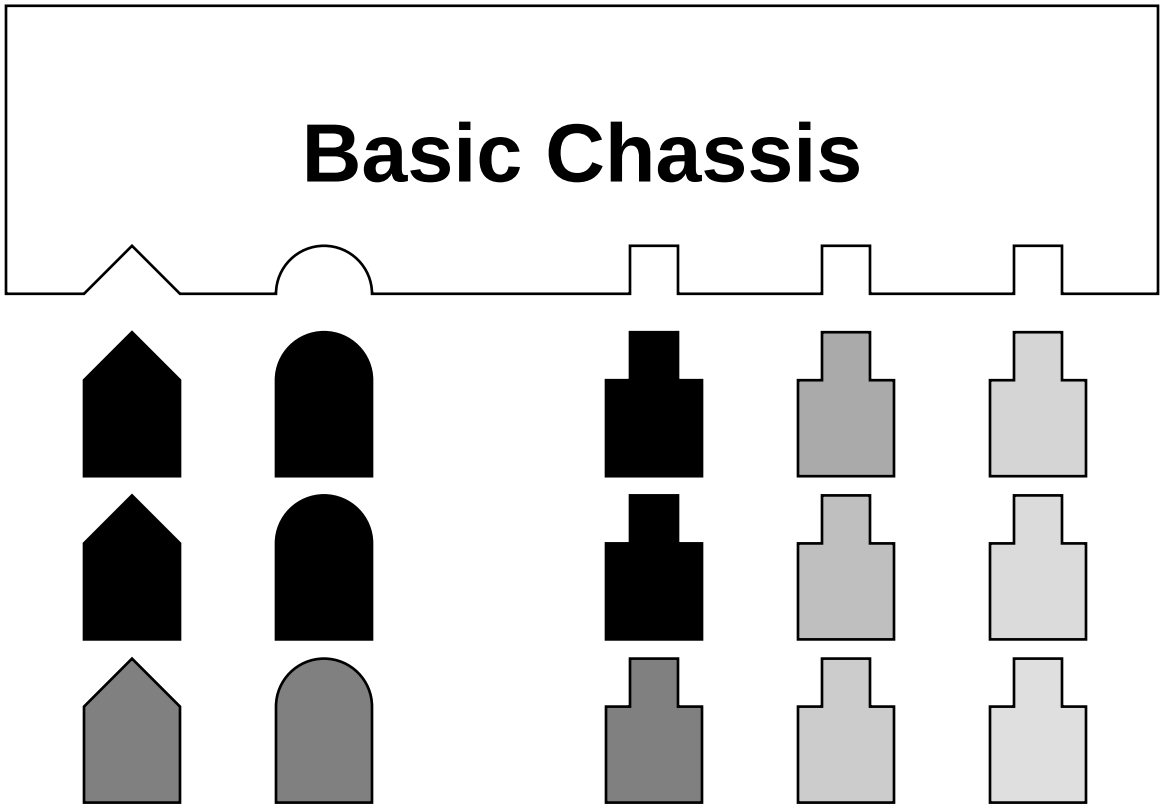
Conventional Programming vs. Chinese Menu – The Difference

Conventional:
1050 programs

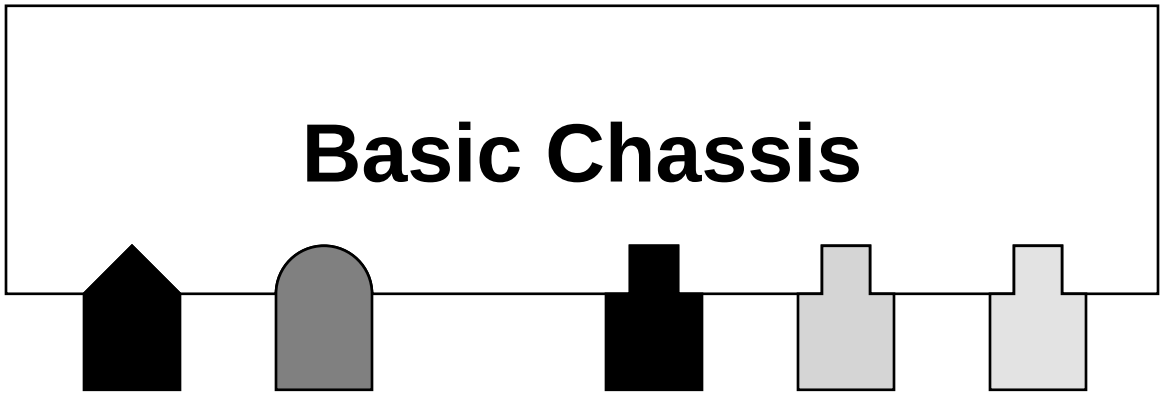
Chinese Menu:
43 modules/packages
3 interfaces

The Difference:
1007 programs!

Key Concept: Programming by Chassis



Programming by Chassis: Operation



Result:

Shareable libraries permit us to achieve the effect of multiple levels of nested IF statements without increasing program complexity.

Production Environment

The selection of components is driven by the menu system. There is little need for multiple levels of IF statements.

Complexity Solution

By hanging different applications components on the same chassis, we are able to achieve a wide variety of options WITH NO INCREASE APPLICATIONS COMPLEXITY

Invariably, new cases arise.

Another view:

This way of building applications is conceptually similar to genetics. You build applications (organisms) out of simple building blocks.

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/2024/index.html