Chapter One

COURSE INTRODUCTION

Introduction

Developing and maintaining application code in the PFPC environment requires that a code designer have a background in a number of areas including The OpenVMS operating system, the Fortran programming language, and a number of SuRPAS applications and utilities. Over the next two weeks we will learn about the application environment in which we live and how to be most productive with the tools available to us.

Course Objectives

To become familiar with and knowledgeable in the following:

- The OpenVMS Digital Command Language (DCL)
- OpenVMS and PFPC Editors (EVE & AMS)
- The OpenVMS FORTRAN language and its compiler
- OpenVMS utilities to link, debug, and troubleshoot program code
- The SuRPAS utilities
- The SuRPAS WorkBench
- SuRPAS I/O commands (FALIOS)
- SuRPAS Workflow Process
- SuRPAS Software Environment
- The various SuRPAS Release Processes
- SuRPAS Report Creation and Screen Generation

Course Information

Schedule

Classes begin at 9:00 am each day and conclude at 5:00 pm. We break for lunch at about noon for one hour. There is a 15 minute break during the morning and afternoon sessions. If you need to miss a session please let the instructor know in advance, when possible.

Student Accounts on the Alpha

Each student is assigned a student account on the Alpha for used during the course. If you have already been assigned an employee account please do not use it during this course. The student account assigned to you may have privileges not available to your employee account and these privileges may be required to accomplish some of the course assignments. In addition, the instructor will be placing required course files in your student account. At the conclusion of this course you will be given the option of moving the files in your student account to a directory in your employee account.

Course Assignments

During this three-week course, class assignments will be provided to complement the course examples found in the student manual. These assignments can be found in chapters twenty six and twenty seven. Students are encouraged to take advantage of the time allotted during class to complete these assignments. The assignments are intended to amplify and solidify the information provided by the instructor.

Course Flow

Monday Morning, Day #1 (9:00 am – 12:00 noon)

Welcome: Introductions and Logistical Information
Chapter One - Course Introduction
Course and Manual Overview
Introduction to OpenVMS
Chapter Two - Logging In To The OpenVMS Operating System

Monday Afternoon, Day #1 (1:00 pm – 5:00 pm)

Chapter Three - Naming & Command Conventions, VMS HELP
Chapter Four - Naming and Storing Files in VMS

Tuesday Morning, Day #2 (9:00 am – 12:00 noon)

Chapter Five - The EVE/AMS Editor

Tuesday Afternoon, Day #2 (1:00 pm – 5:00 pm)

Chapter Six - Making Copies of Files

Wednesday Morning, Day #3 (9:00 am – 12:00 noon)

Chapter Seven - SuRPAS & SuRPAS WorkBench - Part 1
Chapter Eight - Customizing your OpenVMS Working Environment

Wednesday Afternoon, Day #3 (1:00 pm – 5:00 pm)

Chapter Nine - The SuRPAS File Environment
Software and Databases
The LNM Utility (Logical Name Translation)
Chapter Ten - OpenVMS File Security

Thursday Morning, Day #4 (9:00 am - 12:00 noon)

Chapter Eleven - Handling Input and Output in a Command Procedure
Chapter Twelve - Manipulating Data In Command Procedures

Thursday Afternoon, Day #4 (1:00 pm - 5:00 pm)

Chapter Thirteen - Executing Code and Commands

Friday Morning, Day #5 (9:00 am – 12:00 noon)

Chapter Fourteen - Program Development Utilities

The Program Development Cycle
The OpenVMS Set of Program Cycle Files
Program Development Utilities

Friday Afternoon, Day #5 (1:00 pm – 5:00 pm)

Chapter Fifteen - Introduction to Compaq Alpha Fortran

FORTRAN Language Syntax and Coding Structure
PFPC Coding Standards
Fortran Lab Assignment #1 - HELLO_WORLD

Monday Morning, Day #6 (9:00 am – 12:00 noon)

Chapter Sixteen - Handling FORTRAN Data Types - Intrinsic Types
- Handling FORTRAN Data Types - User Defined Structures

Monday Afternoon, Day #6 (1:00 pm – 5:00 pm)

Chapter Seventeen - Building Blocks of a Fortran Program
- The Program Unit
- Subroutines and Functions
- Getting Started With The Debugger

FORTRAN Lab Assignment #2 - Integer Manipulation

Tuesday Morning, Day #7 (9:00 am - 12:00 noon)

Chapter Eighteen - Part #1 - Conditionals
FORTRAN Lab Assignments #3 & #4 - Accepting and Parsing Input

Tuesday Afternoon, Day #7 (1:00 pm - 5:00 pm)

Chapter Eighteen - Part #2 - Loops and CASE Statements
FORTRAN Lab Assignment #5 - Accepting Input using a Loop

Wednesday Morning, Day #8 (9:00 am - 12:00 noon)

Chapter Nineteen: - SuRPAS WorkBench - Part 2
Chapter Twenty: - Part #1 - Input & Output - RMS I/O
FORTRAN Lab Assignment #6 - The Wishlist I/O Program

Wednesday Afternoon, Day #8 (1:00 pm - 5:00 pm)

Chapter Twenty: - Part #2 - SuRPAS I/O - FALIOS
FORTRAN Lab Assignment #7 - Searching SuRPAS Databases

Thursday Morning, Day #9 (9:00 am – 12:00 noon)

Chapter Twenty One - The SuRPAS Software Environment

The Code Management System Utility (CMS)

The Nightly Release Process

The Production Package (FTKPKG)

Thursday Afternoon, Day #9 (1:00 pm – 5:00 pm)

Chapter Twenty One - The SuRPAS Software Environment

The Release Process

Tape Backup (FTAPE)

Chapter Twenty Two - SuRPAS Workbench - Part III

Cross Reference

Conversion Utilities

Database Management

Examining Files

Friday Morning, Day #10 (9:00 am – 12:00 noon)

Chapter Twenty Three - The SuRPAS Workflow Process

Browsing The Intranet - accessing the SuRPAS Home Page

The SuRPAS Workflow Diagram and the Players

SuRPAS Document Management System (SDMS)

- The WAG and QA Estimate

Friday Afternoon, Day #10 (1:00 pm – 5:00 pm)

Chapter Twenty Three - The SuRPAS Workflow Process

Project Tracking

- FTKs
- WebFTK
- The FTK Document

Programming Procedures

- Creating Estimates
- Writing a Technical Specification
- Unit Test
- Code Reviews

Monday Morning, Day #11 (9:00 am – 12:00 noon)

Chapter Twenty Four - The Nightly Process

- The SuRPAS Nightly Process
- Standard and Non-standard Abends
- Problem Research and Troubleshooting
- The Trace Facility

Monday Afternoon, Day #11 (1:00 pm – 5:00 pm)

Chapter Twenty Five - Building Screens and Reports

The Course Manual

This course manual is intended to support and parallel the instruction flow of the SuRPAS Programmer's Training Course. The course manual is broken into twenty-five chapters. Each chapter corresponds to a lesson being taught by the instructor.

Course Organization

This course is organized into a series of lessons. Each lesson has its own objectives and covers a single topic or a closely-related set of topics. The course manual includes a chapter for each lesson. A chapter consists of an introduction, a set of objectives, the lesson text and a summary.

- The **introduction** describes the purpose of the lesson and outlines the contents of the lesson.

- The **objectives** identify the topics to be taught and the skills that will be learned in the lesson.

- The **lesson text** consists of:
    - A descriptive text organized by topic;
    - Illustrations that clarify the relationships among various elements of a topic or summarize steps of a particular process, command or language syntax.
    - Examples containing sample commands, code listings, or displayed output from actual interactive sessions on an OpenVMS system.
    - Interactive activities to help the students practice the skills as they are being learned in the classroom session.

- The **summary** reviews important concepts and commands or language syntax that was taught during the lesson.

There are twenty nine chapters, one for each of the twenty seven lessons, a set of student exercises and case studies, and a glossary of acronyms and terms.

Chapter twenty-eight provides interactive and laboratory exercises. These exercises help the student review and practice the skills learned during the lecture portion of the lesson. Chapter twenty-nine is intended to provide a student with a quick find capability when searching for definitions. It also further describes terms and provides additional background on some of the more difficult OpenVMS concepts.

Course Manual Conventions

The table on the following two pages describes the conventions used in the text, code listings, example notes, and command tables of this course.

Ctrl/C	A boxed letter grouping can represent a key on your keyboard. This key must be pressed to achieve a desired effect. A forward slash (/) within a group of letters acts as a separator between the keys represented on either side of it and indicates that a number of keys are to be pressed at the same time. In the example shown here, the Ctrl/C indicates that the user should press the Ctrl key and hold it down while pressing the C key.
UPPERCASE $ SHOW USER 9RB	A command or syntax that is displayed in upper case characters must be typed exactly as shown.
lowercase $ PRINT *wishlist.txt*	Lower case characters represent elements of a command or language syntax that you must replace according to the description of the text. Very often these lower case fields are also italicized for easier identification and differentiation.
Horizontal Ellipses $ TYPE file-spec...	Horizontal ellipses (three periods in succession) indicate that additional information can be supplied, such as parameters, qualifiers values, or data.
Vertical Ellipses $ TYPE myfile.txt • • • $	Vertical ellipses (a vertical set of dots) indicate that some of the data, example code, or commands that the system would normally display on the screen or in a printout is not shown. This information is most often omitted because of its lack of relevance to the topic being discussed.

Square Brackets ([ ])
$ SHOW USER [user-name]

Square brackets indicate that the enclosed item is optional in the command or language syntax.

Quotation Marks

Quotation Marks (“) refers to double quotation marks.

Apostrophes

Apostrophes (‘) refers to single quotation marks.

Example or figure notes.

Important shortcuts or tidbits of information that can make the difference between an easy fix and a tough struggle.

The Course Manual

Module One: OpenVMS and The Digital Command Language (DCL)

Chapter One:	**Course Introduction**
Chapter Two:	**Logging In To The System**
Chapter Three:	**Command Communications**
Chapter Four:	**Naming and Maintaining Files**
Chapter Five:	**Using OpenVMS Editors - EVE & AMS**
Chapter Six:	**Managing Files**
Chapter Seven:	**SuRPAS and SuRPAS Workbench – Part 1**
Chapter Eight:	**Customizing your Working Environment**
Chapter Nine:	**The SuRPAS File Environment**
Chapter Ten:	**OpenVMS File Security**
Chapter Eleven:	**Manipulating Data In Command Procedures**
Chapter Twelve:	**Handling Input and Output in a Command Procedure**
Chapter Thirteen:	**Executing Code and Commands**

Module Two: Compaq Alpha Fortran

Chapter Fourteen:	**Program Development Utilities**
Chapter Fifteen:	**Introduction to Compaq Alpha Fortran**
Chapter Sixteen:	**Handling Data In Fortran**
Chapter Seventeen:	**Simple Fortran Programs**
Chapter Eighteen:	**Conditionals and Loops**
Chapter Nineteen:	**SuRPAS WorkBench - Part 2**
Chapter Twenty:	**Input and Output**

Module Three: SuRPAS and its Utilities

Chapter Twenty One:	**The SuRPAS Spftware Environment**
Chapter Twenty Two:	**SuRPAS Workbench - Part 3**
Chapter Twenty Three:	**The SuRPAS Workflow Process**
Chapter Twenty Four:	**SuRPAS Nightly Process**
Chapter Twenty Five:	**Report Creation & Screen Generation**

Module Four: Appendices

Appendix A:	**Class Exercises and Solutions**
Appendix B:	**Command Procedures and Fortran Code Examples**
Appendix C:	**Glossary of Terms**

The OpenVMS Operating System

Overview

The OpenVMS operating system runs on the Compaq family of VAX and Alpha-based workstations, notebooks, laptops, and servers. The VAX processor is a 32-bit virtual address architecture processor and the Alpha is a 64-bit virtual address architecture processor. Both were originally designed by Digital Equipment Corporation. The term OpenVMS refers to the open architecture and standards of the operating system as well as its support of the virtual memory management capabilities built into the VAX and Alpha processor architecture.

Features

Software Environment

- OpenVMS is a multiuser, multifunction, virtual memory operating system that supports all VAX and Alpha series computers.
- OpenVMS supports timesharing processes and real-time processes concurrently.
- The design limitation on a single OpenVMS system is 8,192 users. However, the actual limitation is governed by the processor's capability, memory limitations and storage resources.
- OpenVMS provides the Digital Command Language (DCL), a comprehensive, user-friendly command line interface for the user.
- DCL is one of the simplest control languages ever developed for a Digital processor.
- DCL contains an extensive set of commands that allow a user to perform tasks such as the following:
    - Develop and execute programs
    - Manipulate files and directories
    - Work with disk and tape devices
    - Extract information on the process or system
    - Modify the user's work environment

Virtual Memory System

- The OpenVMS operating system is a virtual memory operating system.
- Virtual memory is an interaction between hardware and software that logically extends the physical memory of a system onto disk space.
- From the user's standpoint, the secondary storage locations are treated as if they were physical memory.
- The size of virtual memory depends on the amount of physical memory available plus the amount of disk storage used for nonresident code and data (that is, code and data not currently located in physical memory).
- This design allows users to write and execute arbitrarily large programs without concern for addressing limitations.

Program Development

The OpenVMS operating system provides a comprehensive set of tools for developing programs. Among these are:

- The EDT Line Editor
- The EVE Text Processor
- DSR Document Processing
- DIFFERENCES File Comparison Utility
- Various programming language compilers (more than 10 languages supported)
- Linker (converts object code into executable images)
- Librarian
- Symbolic Debugger
- Patch Utility
- Library of Digital-supported system services
- Library of Digital-supported linked-library services
- OpenVMS MACRO assembly programming language

Security and User Control

OpenVMS provides privilege, protection, and quota mechanisms to limit user access to system-controlled objects within the system.

- Generally, the system manager and operations control personnel have privileges to access all areas of the system.
- The system manager and operations control accounts may either have all ` privileges by default or the ability to grant the accounts those privileges needed.
- Access to the system, interactive, batch, network, or device, is accomplished through the user account.
- The user accounts are maintained by the system manager using the AUTHORIZE system utility.
- Each account requires a user identification and password to be validated before access is granted.
- An additional password may be placed on terminal ports, accounts, and specific files if necessary to protect data integrity.
- All user passwords are encoded using a one-way encryption algorithm, and cannot be displayed with any utility or program.

Network Services

- DECnet-VAX
- EtherNet (CSMA/CD protocol)
- VAXcluster communications support (SCS) including:
    - Connection manager.
    - MSCP server.
    - Distributed Lock manager.
    - Distributed Job Controller.
    - Distributed Operator Communications.
    - Distributed File System.
- SNA Gateway
- TCP/IP Protocol

The Open VMS Process

All activity within the OpenVMS operating system is achieved within the confines of a process. A ***process*** is the representation of a ***program*** (a linked executable image) in physical memory.

- The Operating System:
  - Swapper
  - Job Control
  - Queue managers
  - Network processes
- Users
  - Interactive processes
  - Batch processes
  - Detached processes (Jobs)

Jobs and Processes

Jobs are Independent or ***detached processes***. A detached process has its own resources:

- virtual memory
- quotas
- limits
- UIC (User Identification Code)
- Independent termination

Processes or ***subordinate processes*** are dependent on their creator process/job. A dependent process relies on the creator job to share it's resources with the set of dependent processes owned by the job. The subordinate process:

- shares the job's virtual memory, quotas, limits
- has the job's UIC
- must terminate before the job terminates
- subtracts from the process limit (PRCLM)

User Identification Code (UIC)

The User Identification Code (UIC) uniquely identifies all users and their created jobs and processes. The UIC is assigned by the system manager when a user account is created. It is made up of two 16-bit numbers (32-bit longword) with associated names:

- group number (group name)
- member number (member name)

The ***group number*** can range from 1 to 1024, (1-10 are reserved for operating system processes). The group number is intended to include 1 or more users that have common needs and activities. A specific group number may represent a company department or project group. A "group" of users can have access to each others' processes that members of other "groups" may not have. Each group number has an associated ***group name*** that usually describes the group.

The **member number** is unique to a specific user in the operating system and points to a record in the database of user accounts called the User Authorization File (UAF). All users (members) belong to a specific group and have an associated group number. There is a ***member name*** (usually the user name) associated with each member number.

A UIC is displayed using either the number format (as an octal number) or the name format, as follows:

1. ***[group name, member name]***

2. ***[group number, member number]***

The following are examples of these two formats:

- [123,13]
- [1,10]
- [MF1,9RB]
- [COMPROCS,TEST]

Quotas and Limits

Every job has a set of quotas and limits. These quotas and limits define the amount of system resources a process (or a job's processes) can use. These resources include pages of virtual memory (both private and global), access to system data structures, and rights to create new system-managed objects and gain access to existing system-managed objects.

Process Priority and Scheduling

Process-related operating system displays (SHOW...) provide status and characteristics including a process's priority. To better understand what this number means it is important to recognize that a process must have control of a processor before it can execute a line of code or access data. Only one process per processor can be executing at any given time. Processes gain control based on a number of criteria, one of which is the process's priority.

Process Priority

The ***priority*** of a process is a number from 0 to 31, where 31 is the highest priority and 0 is the lowest. The processes that we create will only use the first 16 priorities (0 to 15). The ***base priority*** of a process is dictated by the system manager (in the UAF). The base priority for interactive processes defaults to 4 (in most systems). The base priority of batch processes defaults to either 2 or 3 (system manager defined).

The process with the highest priority of all processes that are able to execute their next line of code (instruction) gains control of a processor. The priority of a process can be temporarily increased or decreased by the operating system, so it is not unusual to see the process priority raising and lowering.

Process States

A process's state refers to the state of execution of that process at any given point in time. These states are often displayed by the operating system and will tell you what your process is currently doing. The three major types of states are:

- Current
- Computable
- Waiting

Current state is the state of those processes (one process per processor) that are currently executing code. You will see CUR displayed in the various SHOW displays when a process is in current state.

There are two ***computable states***. A process that is ***computable*** has the ability to execute code but its priority is not high enough or the process's virtual memory is not in-swapped (not in physical memory). You will see COM or COMO displayed in the various SHOW displays when a process is in current state.

There are sixteen ***wait states***. A process that is waiting is not able to execute code because it is waiting for a system or process request to complete.

Threads

The OpenVMS operating system supports the ability to execute one or more computational entities on a processor. Each of these entities is referred to as a ***thread of execution*** or simply a ***thread***. A thread can be a program unit, a task, a procedure, a DO loop, a command file, or some other unit of computation.

All threads executing within a single process share the same address space and other process contexts, but have their own unique per-thread hardware context. This allows a multi-threaded process to execute different threads on different processors if the Alpha is a multiprocessor computer.

Command Files As Threads

There are two basic types of command files within the OpenVMS operating system. The simplest command file executes the DCL commands from top to bottom in a single stream of execution, or a single thread.

The second and more complicated type of command file creates secondary process threads that are executed simultaneously. This type of command file is called a multi-threaded command file.

In a multi-threaded command file each thread executes independently of the other threads on the same or different processors. The command file is not considered to have completed until all threads within the command file have completed. Under certain circumstances it is necessary to coordinate activities of two or more executing threads. This may be necessary when one thread requires information calculated by another thread before the first thread can continue. This synchronization requirement is also supported by OpenVMS.

An understanding of threads is critical when supporting the SuRPAS nightly process. Without this understanding error troubleshooting is very difficult. Threads are discussed in more detail in chapter twenty three.

Appendix 1a - A Brief History of The Alpha and The OpenVMS Operating System

Digital Equipment Corporation

The VAX and Alpha processors, as well as the OpenVMS operating system, were designed and developed by engineers of Digital Equipment Corporation (often referred to as DEC or just Digital). Considered the number two computer builder in the 1970's and 1980's (behind IBM), it was the largest manufacturer of minicomputers and the first serious manufacturer of network servers. Created as an offshoot of MIT in the early 1960's by graduate engineering student, Kenneth Olsen, Digital Equipment Corporation was formed in an old clothing mill in Maynard, Massachusetts. DEC's corporate headquarters remained there until the company was purchased by Compaq Corporation.

DEC's first family of minicomputers was the PDP series, consisting of 8-bit, 12-bit, and 16-bit computers. The PDP (*Programmed Data Processor*) family began with the PDP-1, which was actually built at MIT. The most popular PDP's were the PDP-8 and the PDP-11. The PDP-11 is a 16-bit computer which was built in various models. The last of these models, produced in the early 1990's, ran a number of operating systems, including RSTS, RSX-11, and RT-11. The VAX processor and the VMS operating system have their origins in the PDP-11 and the RSX-11 operating system.

VAX

The original VAX processor was developed in the mid 1970's and introduced in 1978. The first VAX was the VAX-11/780. Note that the name resembles the PDP-11. This similarity was intended to give comfort to those PDP-11 users who felt they were being abandoned by Digital for a newer, faster, more expensive model. In fact, Digital continued to produce computers in the PDP family until late in the 1990's. The number, 780, indicates the year it was introduced (1978), and was the basis for all future 700-series model numbers.

The VAX architecture is a 32-bit, virtual memory addressing architecture. It gets its name from *Virtual Address eXtension*. The VAX processor was the first to implement virtual addressing in the hardware of the processor as opposed to in the software of the operating system.

Alpha

The Alpha is a 64-bit virtual memory computer. As with its predecessor, the VAX, the Alpha also implements virtual addressing in the hardware of its processor. However, it can map a 64-bit address which allows access to considerably more memory than did the VAX's 32-bit addressing.

The VAX/VMS Operating System

The original VMS operating system designed for the VAX-11/780 in 1978 was called the VAX/VMS operating system. It fully supported the virtual addressing capabilities of the VAX architecture. In order to keep PDP-11 users happy, it also supported a 16-bit mode (called *compatibility mode*) that ran PDP-11 applications (with considerably reduced performance).

The Future of OpenVMS

On June 25, 2001 Compaq and Intel announced an agreement that provides for a migration path from the Alpha processor to the new Intel Itanium processor family. The migration will take several years to accomplish. It is estimated that the first OpenVMS Compaq server containing the Itanium processors will reach market in 2004. Compaq is committed to advancing the Alpha processor (in an OpenVMS configuration) at least through the delivery of the first Itanium processors. The company will continue to provide off-the-shelf Alpha systems for as long as customer demand continues.

Appendix 1b - Virtual Memory Management in the OpenVMS Operating System

The VAX and Alpha processors map virtual memory addresses to physical memory locations, allowing the OpenVMS operating system to use strictly virtual memory addresses. Each 8-bit ***byte*** is individually addressable. Virtual memory is divided into ***pages*** of 512 bytes (this is the default).

The OpenVMS virtual memory architecture differentiates memory used by the operating system from memory used by created processes. The architecture reserves 2 gigbytes (GB) of virtual memory address for processes and the rest for the operating system. Process memory is referred to as "***P space***" or ***process space*** and system memory is referred to as "***S space***" or ***system region***.

Process Space

The 2 gigabytes of process virtual memory are further segmented into two 1 gigabyte areas called the ***program region (P0 space)*** and ***control region (P1 space)***. Program image code (static code and data) is stored in P0 space and dynamic code and data (including procedure arguments, debugger information, and page tables) are stored in P1 space.

System Region

Anything not owned by a process must be managed by the operating system and is stored in the system region (S space). This includes:

- The operating system code
- Operating system parameters, data structures and global page tables
- Device drivers
- System libraries
- System services
- Run-time libraries
- Debugger code and data
- All shared code and data

The Pager and Swapper

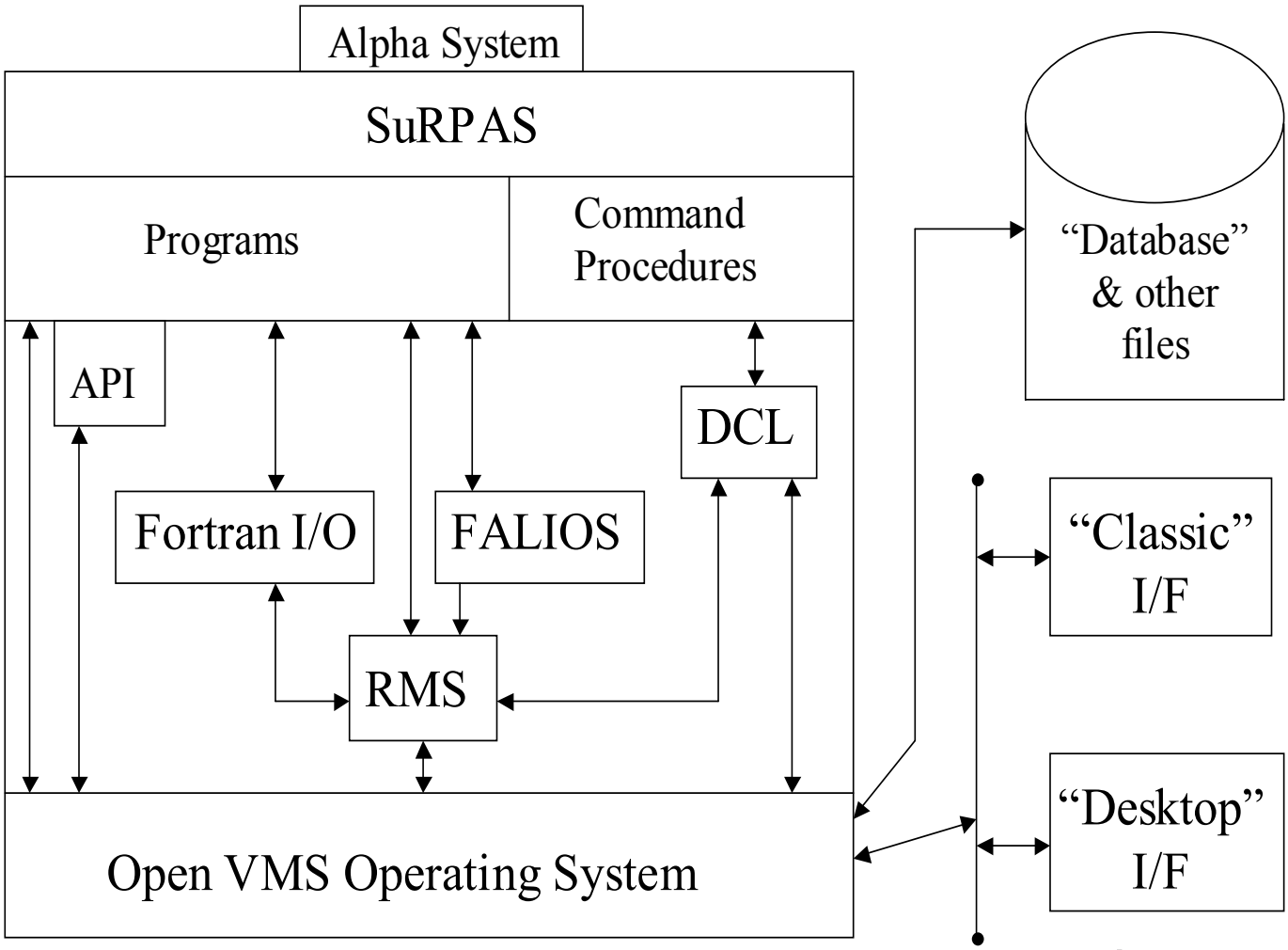
Virtual memory is often larger than the physical memory of a given Alpha system. It is necessary for some of the code and data mapped into virtual memory to reside on disk rather than in physical memory. It is the responsibility of the ***swapper*** process and the ***pager*** facility to decide what resides in physical memory and when to move the information either from disk to physical memory or from physical memory to disk.

Paging and Faulting

Many of the informational displays (SHOW...) within the operating system report page and fault activity. It is important to understand the basics of what is being reported. When code and data are moved from the disk into a process's virtual memory (physical memory) the action is called paging (or in-swapping). When code and data are moved out of a process's virtual memory, it is called ***page faulting*** (or simply ***faulting***). For every page that is moved out of a process's virtual memory a page fault is said to have occured.

Appendix 1c - SuRPAS Production System (at customer site)

Brokerage subaccounting system
- Shareholder record keeping
- Trade clearing
- Settlement across multiple funds
- Alternative to traditional NSCC process
  - NSCC = National Securities Clearing Corporation
  - Centralized clearance, settlement, and information services
- Typically runs on "Alpha" system at customer site



Appendix 1d - SuRPAS Development & Testing System