Porting open source (UNIX) software to OpenVMS

A product of the OpenOffice.org on OpenVMS porting group

http://www.oooovms.dyndns.org/

Written by Martin Borgman and Ton van der Zwet

(guide 19-jul-2005)

This guide is a work in progress

The latest version can be found at http://www.oooovms.dyndns.org/reference

Table of Contents

Preface

OpenVMS and Open source both started using the Open prefix some time ago. Before that, all we had were a lot of operating systems, one of which was VAX VMS. Each operating system came with its own possibilities, strong and weak points. If you wanted a application to run on your computer you would buy or build the application yourself for your computer. The efforts needed to get the thing going, were often so great that people sought methods to make the process of implementing applications on a different platform (so called porting) easier. Especially software meant to be implemented on many different platforms and software developed for platforms not yet known had to take measures to make implementations as easy as possible. Conformation to open or industry standards helped a lot. If software was delivered including the sources so the client could modify the software to suit his (changing) needs with the obligation to make his changes available to other users of this software the term open source software comes to mind. Although open source software and open standards are strictly speaking not necessarily related, they are often mentioned in one breath, because people benefit the most if both are optimally available: easy (and thus cheap) deployment of applications.

The open in OpenVMS just means that people can rely on the conformation to open standards in the OpenVMS operating system. OpenVMS is available not only on VAX hardware but also on Alpha and since 2004 also on industry standard IA64 hardware.

Purpose

This guide will explain how to port opensource software to OpenVMS using the features introduced in the DII COE releases of OpenVMS (7.2-6C1 & 7.2-6C2).

The guide does not specifically target these versions of OpenVMS but the first public versions of OpenVMS with some of these new UNIX features. To be more specific, OpenVMS versions 7.3 7.3-1, 7.3-2 and up. The latest OpenVMS version (OpenVMS 8.2) has the most of these new features.

Because the development of the porting tooling is progressing fast and more portability features will be added to OpenVMS in the future, we will regularly update this guide. The latest version of this guide can be found at http://www.oooovms.dyndns.org/reference.

Many open source projects use UNIX shell scripts to build and install. We will be looking on how to do the same thing on OpenVMS using the UNIX commands and utilities provided by the GNV kit.

We will not look at porting open source software to older versions of OpenVMS, although some of these older versions are still supported. Porting to these older versions of OpenVMS can be done, but it is considerably more difficult.

Intended audience

Basically everybody with interest in open source software. If everything was perfect this would be the only requirement, but alas, things are not perfect. In the current versions of open source software OpenVMS is seldom recognized or considered as a target platform. We think (and hope) this will change soon. In the meanwhile porters will benefit from some programming experience. C and/or C++ experience on OpenVMS or UNIX systems is a big pre. Other languages often used in open source world are bash, make and perl.

System management experience is more a requirement because porting may require making adaptations to the porting environment on OpenVMS.

Acknowledgments

We would like to thank all the people who contributed in any form to this guide. There are many people who contributed material, thoughts or inside information. Without their contributions this guide wouldn't be as useful l as it is now. Steve Pitcher, Frank Ries and Brad McCusker are (or were)not

only working on GNV and the CRTL, but were also listening to our needs and sharing their knowledge with us.

Valuable support for our efforts continuously came from Sue Skonetski and our own VMS ambassador Gerrit Woertman. Without their support many doors wouldn't have opened at our knocking.

Part I Open source, Unix and OpenVMS

Open source versus propriety software

This guide will not start with a in depth philosophic discussion about the difference between Open source, Free ware, shareware and propriety software. Instead we prefer to follow the definitions given in the following publication:

Title:	The BUSINESS and ECONOMICS of LINUX and OPEN SOURCE
Written by:	Martin Fink
Printed by:	Prentice Hall
ISBN:	0-13-047677-3

These definitions and the legal implications the various license agreements have, are for our purpose not relevant. We will concentrate on the technical aspects of the process of the port of open source software to OpenVMS.

Unix, Linux, Open Unix standard

As the majority of the open source software originates from a form of UNIX, we will have to know the characteristics of this type of platform and the differences between these platform and the target platform: OpenVMS. This is simpler said as done, as there are as many different implementations of UNIX as there are suppliers of hardware and operating system software. Starting in the early days of UNIX, when the universities not only had access to UNIX, but also had access to the source of the operating system, different solutions to common problems emerged. This resulted in a family of related operating systems commonly known as UNIX. Ownership of the UNIX brand name passed hands and the situation became rather confusing. Who had the one and only UNIX? So, people started to define standards (alas not one standard...) by which a “real” UNIX could be recognized. In the reference section of our website (http://www.oooovms.dyndns.org/reference) we compare the following UNIX standards with the OpenVMS features: XSI, POSIX Base, Unix 98, Unix 95, P96 P92, C99, C89, SVID3, BSD, LSB 1.3

LINUX is a very popular UNIX like operating system, available in many different distributions, each with slightly different characteristics.

Making software for such a divers environment was a nightmare. But very inventive software developers found a solution. The solution was two fold:

- Define and enforce a minimum set of features through standardization, much like the RFC's for TCP/IP. On our website we have an set of tables listing a few of the most important UNIX standards and their content in terms of tools and required system services. (http://www.oooovms.dyndns.org/reference) item UNIX® System Interface Tables.
- Automate the build process by using a set of tools capable of finding and reacting (adapting) to differences between platforms. This led to the development of autoconf, automake and libtool. These tools are also known as the GNU autotools environment (http://sources.redhat.com/autobook/)

OpenVMS, open standards

OpenVMS on the other hand is a propriety operating system owned and maintained by HP. There are a few initiatives to make OpenVMS clones (for instance freeVMS (http://freevms.free.fr/indexGB.html)), but in this guide we will concentrate on the OpenVMS operating system from HP. As stated in the SPD (software product description) OpenVMS conforms itself to a set of Open Standards. open standards means in this respect that a standardization organization is responsible for the exact text of a standard. Full conformation to this standard means

in our opinion that a product will be adapted to the standard if a difference is found, not the other way around (adapting the standard to the behavior of a product, as is mostly the case with so-called industry-standards).

The open standards that are most interesting for open source to OpenVMS porters are (from the OpenVMS 8.2 SPD (http://h18000.www1.hp.com/info/XAV12X/XAV12XPF.PDF)):

- **Distributed Computing Environment (DCE) Support**
- **Support for OSF/Motif and X Window System Standards (X11R6 server and X11R5 client)**
- **ANSI X3.4-1986:** American Standard Code for Information Interchange
- **ANSI X3.41-1974:** Code Extension Techniques for Use with 7-bit ASCII
- **FIPS 1-2:** Code for Information Interchange, Its Representations, Subsets, and Extensions
- **ISO 646:** ISO 7-bit Coded Character Set for Information Exchange
- **ISO 2022:** Code Extension Techniques for Use with ISO 646
- **ISO 3307:** Representations of Time of the Day
- **ISO 4873:** 8-Bit Code for Information Interchange — Structure and Rules for Implementation
- **ISO 9660:** Information Processing—Volume and file structure of CD–ROM for information exchange

Open source on OpenVMS

The question here is not, can we have open source software on OpenVMS, because you're already using some open source software. CDSA is an open source security framework that is now used on OpenVMS. You cannot install current releases of OpenVMS without it. There are two CDSA add-ons, SSL and Kerberos. Both are open source. In the TCP/IP package you'll find BIND and DHCP from the Internet Software Consortium. And what about the SMTP, POP, IMAP, SSH.... These are just some packages that are part of the standard OpenVMS distribution. But there's more, there's Apache, Mozilla, Samba, MySQL, Tomcat, Perl, Python. We could continue for over an hour naming open source tools that are ported to OpenVMS.

Then there's the business rationale of open source. Why should you, or your company use open source software. This is not as simple as many people think. The fact that open source is free does not necessarily mean that it is cheaper to run for you or your company. Because of the complexity of the subject we would like to point you to a good book on the subject:

Title: The BUSINESS and ECONOMICS of LINUX and OPEN SOURCE

Written by: Martin Fink

Printed by: Prentice Hall

ISBN: 0-13-047677-3

OpenVMS-UNIX differences

The best way to compare UNIX and OpenVMS is probably by way of a table with a side by side comparison of the interesting aspects. In appendix 4 you'll find such a table.

This is but a partial list. It is probably best to point you to some good books on the subject:

From John Wisniewski, who was the driving force behind the OpenVMS Hobbyist program, is the excellent book:

Title: Linux and OpenVMS Interoperability

Written by: John Wisniewski

Printed by: Digital Press

ISBN: 1-55558-267-2

The following book is written to make life easier for OpenVMS literate people new to the UNIX environment:

Title: UNIX for VMS Users

Written by: Philip E. Bourne
Printed by: Digital Press
ISBN: 1-55558-034-3

Part II: build the open source porting environment on OpenVMS.

Build the OpenVMS porting system

The first step in the porting process is getting a suitable OpenVMS system up and running. Although OpenVMS is available on three hardware platforms, only two of them are sufficiently equipped to do serious opensource porting: Alpha and IA64. The VAX platform lacks support for some important features in a stand-alone configuration. VAX OpenVMS for instance lacks support for native ODS5 and GNV.

Because we have only access to Alpha systems and because IA64 OpenVMS is, as far as we know, functionally equal to Alpha OpenVMS, all examples are based on Alpha OpenVMS.

In the reference section of the www.oooovms.dyndns.org website are examples of steps involved in building a OpenVMS system. There are documents about upgrading firmware (http://www.oooovms.dyndns.org/reference/sw_install/firmware.html), initial installation of OpenVMS (http://www.oooovms.dyndns.org/reference/sw_install/install_os.html), first boot (http://www.oooovms.dyndns.org/reference/sw_install/firstboot.html) and initial configuration of OpenVMS (http://www.oooovms.dyndns.org/reference/sw_install/initial_configuration.html).

To make an optimal environment we need to give some attention to system parameters, some system file sizing and process quota and settings. Lets proceed with the system parameters.

System parameters

OpenVMS system parameters are very much like UNIX kernel parameters.

System parameters take effect at system boot time, although there are system parameters that can be changed and have immediate effect on a running system. So, unless you want to reboot often, you should take some time to set the system parameters to their correct values for your system.

New values for system parameters should be edited in the `sys$system:modparams.dat`. This file is node specific and should contain all changes you want to take effect to next time the system boots. This file is read by the autogen procedure. We recommend you comment on all changes in this file.

CHANNELCNT should be set to a value at least as big as the UAF FILLM value. A good practice is to set the value to the biggest value of: the current value, the largest UAF FILLM value and 4096. Note that the SDK process will have the *lower* value of the UAF quota FILLM or the SYSGEN parameter CHANNELCNT.

Part of a modparams.dat:

```
.
.
! Created during installation of OpenVMS AXP V7.3-1 26-JAN-2003 16:05:09.28
MIN_GBLSECTIONS=1000
!
ALLOCLASS=1             ! necessary for shadowing
SHADOWING=2             ! enable shadowing
SHADOW_SYS_DISK=1       ! enable shadowed system disk
SHADOW_SYS_UNIT=0       ! unit number system disk
SHADOW_MAX_COPY=4       ! max concurrent shadow copies
!
CHANNELCNT=8192         ! large compiles etc..
!
MAXPROCESSCNT=128       ! we don't need so many processes...
!
MIN_CTLPAGES=1536       ! performance tcpip processes?
!
SWAPFILE=0              ! do not adjust the swapfile sizes
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
.
.
```

After you made the necessary changes to `sys$system:modparams.dat` invoke the autogen procedure:

```
$ @sys$update:autogen savparams setparams feedback
```

You can check the results of this procedure as follows:

```
$ set terminal/width=132
$ differences/parallel sys$system:setparams.dat
------------------------------------------------------------------------------------------------------------------------
File SYS$SYSROOT:[SYSEXE]SETPARAMS.DAT;7                        |  File SYS$SYSROOT:[SYSEXE]SETPARAMS.DAT;6
------------------------------- 11 ---------------------------------------------------------------- 11 ----------------
set SYSMWCNT 2102                                               |  set SYSMWCNT 2092
set WSMAX 262144                                                |  set WSMAX 262144
set NPAGEDYN 4349952                                            |  set NPAGEDYN 3022848
set NPAGEVIR 19169280                                           |  set NPAGEVIR 12877824
set PAGEDYN 1794048                                             |  set PAGEDYN 1785856
------------------------------- 58 ---------------------------------------------------------------- 58 ----------------
set GBLPAGES 1118016                                            |  set GBLPAGES 1105978
------------------------------------------------------------------------------------------------------------------------

Number of difference sections found: 2
Number of difference records found: 6

DIFFERENCES /IGNORE=()/PARALLEL-
    SYS$SYSROOT:[SYSEXE]SETPARAMS.DAT;7-
    SYS$SYSROOT:[SYSEXE]SETPARAMS.DAT;6
```

If you are satisfied with the generated or changed system parameters, reboot the system when it suits you best:

```
$ @sys$update:autogen reboot
```

Other settings

PAGEFILESIZE
When you increase the PGFLQUO UAF parameter, you should also increase the system's page file size as needed to accommodate the new PGFLQUO parameter. Autogen (see the system parameters section) also calculates new values for the swap, page and sysdump files, unless instructed not to do so by specifying a directive in `sys$system:modparams.dat` or by not specifying the genfiles phase.
You can also use the `sys$update:swapfiles.com` procedure to set the pagefilesizemanually:

```
$ @sys$update:swapfiles
To leave a file size at its current value type a
carriage return in response to its size prompt.
Current file sizes are:

Directory SYS$SPECIFIC:[SYSEXE]

PAGEFILE.SYS;2          1056800
SYSDUMP.DMP;4            208583
SWAPFILE.SYS;3             16400

Total of 4 files, 1682393 blocks.

There are 10110396 available blocks on SYS$SYSDEVICE.

Enter new size for paging file:
Enter new size for system dump file:
Enter new size for swapping file:
$
```

Installing GNV

Introduction

GNV stands for GNU is not VMS. It is a set of open source commands and utilities that are ported to OpenVMS. The GNV kit contains a port of the bash shell, the gnu diff utilities, the gnu file utilities, the gnu find utilities, gawk, grep, gzip, less, gnu make, man, sed, gnu shell utilities, gnu text utilities, (un)zip, a vi wrapper for tpu, vms(un)tar and wrappers for ar, cc, gcc and cpp. This list is not complete and more UNIX utilities become available with every new version of the GNV kit.

GNV is an open source project with a web page <http://gnv.sourceforge.net> and several mailing lists.

If you are planning to add tools to GNV, join the gnv developer list.

Installation

First download the latest kit from the OpenVMS open source page at <http://www.openvms.compaq.com/opensource> or from the GNV project page at <http://gnv.sourceforge.net>.

Print the installation instructions and use them to install the kit. Please read it carefully because it contains some important information.

The requirements for installing and using GNV are:

- must install on an ODS-5 disk
- must use on an ODS-5 disk

To check the installation disk use the following command:

```
$ SHOW DEVICE DSA0: /FULL

Disk DSA0:, device type COMPAQ BB00911CA0, is online, mounted, file-oriented
    device, shareable, available to cluster, error logging is enabled, device
    supports bitmaps (no bitmaps active).

    Error count                    0    Operations completed           11118695
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count             1204    Default buffer size                512
    Total blocks            17773524    Sectors per track                  168
    Total cylinders             5290    Tracks per cylinder                 20
    Logical Volume Size     17773524    Expansion Size Limit           17793024

    Volume label          "ALPHASYS"    Relative volume number                0
    Cluster size                   3    Transaction count                  1249
    Free blocks             10110408    Maximum files allowed           2221690
    Extend quantity                5    Mount count                           1
    Mount status              System    Cache name             "_DSA0:XQPCACHE"
    Extent cache size             64    Maximum blocks in extent cache  1011040
    File ID cache size            64    Blocks in extent cache           237132
    Quota cache size               0    Maximum buffers in FCP cache        630
    Volume owner UIC           [1,1]    Vol Prot    S:RWCD,O:RWCD,G:RWCD,W:RWCD

  Volume Status:  ODS-5, subject to mount verification, protected subsystems
      enabled, write-through caching enabled, access dates enabled, hard links
      enabled.

Disk $1$DKA600:, device type COMPAQ BB00911CA0, is online, member of shadow set
    DSA0:, error logging is enabled.

    Error count                    0    Shadow member operation count  10118018
    Allocation class               1
```

And this is not all the information but the important bit is at the bottom of the example text. Right after the Volume Status: header it says ODS-5 in this case. So this disk should be OK for use. Also note that hard links are enabled. For UNIX portability it is generally a good idea to have this enabled. It is also a good idea to enable access dates on a volume because this is a POSIX requirement.

We also recommend using multi member shadow sets. First of all, shadow sets are more secure, i.e., you won't lose all your data when a disk breaks down, but also because shadow set names have no “$”-sign in them. The “$”-sign has a special meaning in UNIX shell scripts. To use a “$”-sign in a name you'll have to escape the “$”-sign with a “\”-sign.

By the way you can and should at least use single member shadow sets to prevent this got-ya.

To make a disk ODS-5 compatible use the following command:

```
$ SET VOLUME <device>: /STRUCTURE=5
```

To enable hard links and access dates use the following command:

```
$ SET VOLUME <device>: /VOLUME_CHARACTERISTICS=(HARDLINKS, ACCESS_DATES)
```

Be aware that enabling hard links can take a considerable amount of time!

Everything you ever wanted to know about shadowing and a description on how you can mount your disks as shadow sets is described in the OpenVMS documentation:HP Volume Shadowing for OpenVMS. The “help” command can help you as well.

Another thing I would like to note here is high-water marking. This is another typical OpenVMS security feature that is on by default. High-water marking guarantees that a user cannot read data that was not written by the user, by destroying all data in the disk block after the EOF marker. There is a small performance penalty in doing this. Although the penalty is only marginal these days, many people will advice you to turn it off. You can do so by giving the following command:

```
$ SET VOLUME <device>: /NOHIGHWATER_MARKING
```

Disabling high-water marking is not mandatory for the installation of GNV or the UNIX portability.

To do a default install of the current GNV software, follow these steps also described in the documentation:

1. Log in to the SYSTEM account (at the login prompt, enter user name SYSTEM and the appropriate password), or an account with equivalent privileges.
2. At the DCL prompt ($), go to the directory where you saved the executable you downloaded from the Internet and extract the PCSI kit by running the executable:

   ```
   $ RUN DEC-AXPVMS-GNV-V0106-002-1.PCSI_SFX_AXPEXE
   ```

3. Type the following command, as shown:

   ```
   $ PRODUCT INSTALL GNV
   ```

4. When you enter the PRODUCT INSTALL command, the system responds with a display similar to the following:

   ```
   The following product has been selected:
    DEC AXPVMS GNV V1.6             Layered Product


    Do you want to continue? [YES]
   ```

   Continue the procedure by pressing the ENTER key for the default answer (YES). The procedure might take several minutes and numerous messages might be displayed on the screen.
   In response to each prompt displayed by the system, choose the default answer by pressing the ENTER key.
5. To use GNV we need to do some system wide setup. To do this we start the following DCL command procedure:

   ```
   $ @SYS$STARTUP:GNV$STARTUP.COM
   ```

   To do this setup every time we boot our system we need to add the above command to the system startup procedure SYS$MANAGER:SYSTARTUP_VMS.COM . So fire up the editor:

   ```
   $ EDIT SYS$MANAGER:SYSTARTUP_VMS.COM
   ```

   Go to the bottom of the file (<Do>bottom) and enter the following lines before the $ EXIT:

   ```
   $!
   $! GNV
   $!
   $ file := SYS$STARTUP:GNV$STARTUP.COM
   $ IF F$SEARCH(file) .NES. "" THEN @'file'
   $!
   ```

   And exit the editor with <Crtl>Z.

Note: The `GNV$STARTUP.COM` procedure uses the procedure `sys$common:[sys$startup]gnv_destination.com to` defines the location where GNV was installed. You will have to modify this procedure if you move GNV to a different location by hand.

6. There's also some user setup that we'll need to do. For our current login session do the following:

```
$ IF F$TRNLNM(“GNU”,"LNM$SYSTEM_TABLE") .NES. “” THEN @GNU:[LIB]GNV_SETUP.COM
```

   And if we want this user setup to happen when you log in, we must add the above line to our SYS$LOGIN:LOGIN.COM. This counts for every user on your system who wants to use the GNV tools. If we want this to happen for every user on our system, we could add the line to the system wide login procedure SYS$MANAGER:SYLOGIN.COM ($ EDIT 'F$TRNLNM(“SYS$SYLOGIN”)'.COM). Just remember to add the line before the EXIT statement (login scrips usually do things depending on the current mode login (F$MODE), note that the gnv setup procedure must be made to work in interactive mode if you plan to use it on the command line).

And of course, no reboot is needed.

Process quotas

Managing process quotas is another OpenVMS stronghold. It allows the system to manage the system resources and maintain an acceptable performance for all processes on the system.

The following recommendation comes from the July 2003 issue of “Optimizing Java Technology Software Performance on HP OpenVMS”. You can find this document at the HP OpenVMS e-Business Technology page http://h71000.www7.hp.com/ebusiness/technology.html.

Please note that the suggested quota's are a minimum requirement for UNIX compatibility. These quota's are specified for user accounts running JAVA applications, but in general its a good idea to give porters and users running ported open source software the same quota's. The look and feel (performance) on OpenVMS will more closely resemble the look and feel on other platforms.

Account quotas

Log is as user SYSTEM and go to the SYS$SYSTEM directory:

```
$ SET DEFAULT SYS$SYSTEM:
```

Start the account management program:

```
$ MCR AUTHORIZE
```

Show your account settings (in the example below the DEFAULT account is shown, this is a special account used for account creation):

```
UAF> show default

Username: DEFAULT                           Owner:
Account:                                    UIC:    [200,200] ([DEFAULT])
CLI:      DCL                               Tables: DCLTABLES
Default:  [USER]
LGICMD:
Flags:  DisUser
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                     Sat Sun
No access restrictions
Expiration:            (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:         90 00:00    Pwdchange:      (pre-expired)
Last Login:            (none) (interactive),           (none) (non-interactive)
Maxjobs:          0  Fillm:       100  Bytlm:        64000
Maxacctjobs:      0  Shrfillm:      0  Pbytlm:           0
Maxdetach:        0  BIOlm:       150  JTquota:       4096
Prclm:            8  DIOlm:       150  WSdef:         2000
Prio:             4  ASTlm:       250  WSquo:         4000
Queprio:          4  TQElm:        10  WSextent:     16384
CPU:        (none)  Enqlm:      2000  Pgflquo:      50000
Authorized Privileges:
  NETMBX       TMPMBX
Default Privileges:
  NETMBX       TMPMBX
UAF>
```

We are going to change the following minimum account quotas:

Parameter Name	***Value***	***IA64 Value***
FILLM	4096	
WSDEF	2048	4096
WSQUOTA	4096	
WSEXTENT	16384	
PGFLQUO	2097152	
BYTLM	400000	
BIOLM	150	
DIOLM	150	
TQELM	100	

To change the values of the parameters do the following:

```
UAF> MODIFY <account> /<parameter>=<value>
```

You can change multiple parameters in th same command just add another /<parameter>=<value> for each value you need to change. To change the WSDEF, WSQUO and WSEXTENT for the DEFAULT account you would enter the following command:

```
UAF> MODIFY DEFAULT /WSDEF=2048 /WSQUO=4096 /WSEXTENT=16384
```

You exit the AUTHORIZE utility by typing EXIT, or <Ctrl>Z (It may be possible that the terminal

emulator you are using correctly maps the EXIT function to the F10 function key) .

As said before, the DEFAULT account is used as a template for the creation of new accounts. If you want to raise the default quotas for all new accounts you are going to create on your system, it may be a good idea to modify the DEFAULT account.

Changing the default account will not change any setting for existing accounts. Check and modify existing accounts as needed

Setting up porting accounts

So far we have set the correct environment for our porting attempts. Lets summarize the steps involved. To create porting accounts one has to follow a few steps:

- Setting up the OpenVMS porting system with the right system parameters
- Installing and starting GNV, compilers and other porting tools
- Setting up the default UAF account
- Creating additional porting accounts and/or modifying existing accounts.
- Creating and modifying home directories for porters.

After that, in order to use the changed account settings, you must log out and back in again.

creating additional accounts

To create additional porting accounts, you can use the following commands:

```
$ mcr authorize
UAF> add 'user' /uic=['group','member'] /device='userdevice': /directory=['user']
/passw='secret'/flag=nodisuser/nopwdexp
UAF> exit
$ create/dir 'userdevice':['user'] /owner='user'
$ create/dir 'userdevice':['user'.temp]/owner='user'
$
$ create 'userdevice':['user']login.com
$! login.com for OpenOffice portingroup member
$
$ set term/dev=vt300
$ set term/line/insert
$! start gnv
$ @GNU:[lib]GNV_SETUP.COM
$!
$! setup tools
$ set proc/parse_style=extended
$ set process /case_lookup=(blind)
$ define/job decc$pipe_buffer_size 65000
$
$
$ scratch = f$trnlnm("sys$login") - "]" + ".temp]"
$ define/job sys$scratch 'scratch'
$!
$ exit
<Ctrl>Z

$ create 'userdevice':['user'].bashrc
# .bashrc
#
PATH=$PATH:/usr/bin:/usr/local/bin
export PATH
```

```
export GNV_DISABLE_DCL_FALLBACK=1
<Ctrl>Z
$
```

A procedure for creating a number of porting users including a example run can be found in appendix 3

PART III: Hints and tips on using the porting environment and the GNV tools

Working with GNV

When you have installed GNV correctly you should be able to use UNIX commands like ls right from DCL. However if you have LSE installed the ls command will start the LSE editor instead.

When you type bash at the DCL prompt. The bash shell is started and from that point on everything works as if you were using a UNIX box. There are however some minor but important differences. You may know that UNIX systems use a hierarchical file-system and OpenVMS does not. However some of the important hierarchy is mimicked under OpenVMS. By default the / directory points to the OpenVMS GNU:[000000] directory. The /bin directory points to the OpenVMS GNU:[bin] directory. Etc. In short the / directory is not the root of a disk!

You can access other locations by typing */device-name* or *concealed device logical*/directory. I.E. DSA50:[kits.gnu] becomes /dsa50/kits/gnu.

When in bash you can enter most DCL commands, unless there is a name conflict like the ls - LSE problem. In that case you can enter the bash dcl command and enter the DCL command behind it. I.E. the DCL LS Readme command becomes dcl “ls Readme” in bash and opens the Readme file in LSE.

If you don't want bash to “Fall Back” to DCL to execute commands, do the following in BASH:

```
bash$ export GNV_DISABLE_DCL_FALLBACK=1
```

We recommend putting this command in the `.bashrc` procedure in your login directory. See the example procedure in the “creating additional accounts” section.

By the way you can always use the BASH dcl command to execute any DCL command!

Tips

- always work on ODS-5 disks
  `$ SET VOLUME <device>: /STRUCTURE=5`
- always enable ODS-5 extended filename parsing
  `$ SET PROCESS /PARSE_STYLE=EXTENDED`
  (you can put this line in your LOGIN.COM)
- because of the way bash handles pipes at the moment we need to do the following
  `$ DEFINE/JOB DECC$PIPE_BUFFER_SIZE=65000`
  (you can put this line in your LOGIN.COM)
- the best way to handle UNIX symbolic links is to use hard links on the disk you are working from
  `$ SET VOLUME <device>: /VOLUME_CHARACTERISTICS=(HARDLINKS)`
  Please note that this can take a considerable amount of time.
- it may be necessary to support POSIX style access dates
  `$ SET VOLUME <device>: /VOLUME_CHARACTERISTICS=(ACCESS_DATES)`
- when running configure scripts, it may be a good idea to disable DCL fall back
  `bash$ export GNV_DISABLE_DCL_FALLBACK=1`
  (you can put this in your .bashrc)

If you want to know more about using UNIX or bash, there is plenty of information available on the Internet. If you like books better, take a look at the offerings from O'Reilly <http://www.oreilly.com/>.

I suggest you do take some time to learn the UNIX environment before continuing.

Part IV: Building open source software

Most of you probably know that open source means that the software is usually distributed in source form. To use the software on your system you need to build it yourself.

To make this building much easer the GNU community developed the GNU build tools. By the way, these tools not only make your life much easer, they also make the life of the open source developers much easer.

Now let's look at how you would build some open source software on a UNIX or Linux box.

1. Download the source distribution from the Internet. Most of the time this will be a .tar.gz file but you'll also find .tgz, .tar.Z .tar.bzip2 or .zip files.
2. Create a directory
   ```
   % mkdir <name>
   ```
3. Go to the directory you just created
   ```
   % cd <name>
   ```
4. Unpack the file (some options may not work on OpenVMS yet)

   - .tar.gz
     ```
     % zcat <filename> | tar xvf -
     ```
     or
     ```
     % tar xvzf <filename>
     ```
     or
     ```
     % gunzip <filename>
     ```
     and
     ```
     % tar xvf <filename - .gz>
     ```
   - .tgz — see .tar.gz
   - .tar.Z — see .tar.gz
     or
     ```
     % uncompress <filename>
     ```
     and
     ```
     % tar xvf <filename - .Z>
     ```
   - .tar.bz2
     ```
     % bz2cat <filename> | tar xvf -
     ```
     or
     ```
     % bunzip2 <filename>
     ```
     and
     ```
     % tar xvf <filename - .bz2>
     ```
   - .zip
     ```
     % unzip <filename>
     ```
5. Look for the configure script. This script may be in the directory you're in, but it may also be in the directory that was created by unpacking the distribution file. I.e. If the file you downloaded from the Internet was called tar-1.2.4.tar.gz it is possible that you now have a directory named tar-1.2.4 in your current directory. Enter that directory to look for a configure file. If you don't find one look for files with uppercase names like README or INSTALL.
6. When you found the configure script, run it
   ```
   % ./configure
   ```
7. When all go's well you can build the executable(s)
   ```
   % make
   ```
8. And install the package (usually you must first become the root user before you can install something)
   ```
   % make install
   ```

And that's all folks.

Well, that was all when everything works the way it's supposed to. And even on a UNIX box things can go wrong. To analyze what went wrong do the following:

- read the files with uppercase names and find out if you meet the prerequisites, find out if you need to do something special for your UNIX (OpenVMS) version (after all UNIX (OpenVMS) != UNIX)
- try ./configure --help this may give you an idea about extra parameters you may have to specify

with the ./configure command

- check the files that were generated by the configure script: config.cache, config.log, config.status, config.h and Makefile(s)

If this didn't help you may need to make your hands dirty.

The GNU build system

Didn't I say before that the GNU build system would make your life much easer? Well it does. But let me first explain what may go wrong in the last example. The configure script and the Makefile.in file(s) may have been created before your UNIX (OpenVMS!) system or system version came into existence, so it doesn't know about your systems specifics.

Thinking about OpenVMS for a moment, probably none of the configure scripts and make files out there, know anything about our GNV environment on OpenVMS

But let's get back to the problem at hand.

To make the configure script and the Makefile.in file(s) aware of our systems specifics we will need to add some steps to our build example.

The following steps come between steps 5 and 6:

a. `% aclocal`

b. `% autoconf`

c. `% automake -a`

To make this work, your system should have these packages and GNU make, m4, texinfo and GNU tar readily installed. For most open source UNIX clones like Linux, FreeBSD, NetBSD and MacOS X, this is true. But for many commercial UNIX systems this isn't the case. As for OpenVMS, only GNU make is available at the moment.

The best thing for most commercial UNIX systems is to check out if the manufacturer of your UNIX system made the missing tools available for download. If not check out the latest versions of the missing tools from the GNU web site. If nothing worked out, you are in the same position as us with our GNV environment on OpenVMS.

At this point I think that it should be clear to us that we need all these tools a.s.a.p.

If you want to know more about the gnu build system, take a look at the following web site(s):

GNU Autoconf, Automake and Libtool: http://sources.redhat.com/autobook/

Recommendation for OpenVMS

Because of constant updates to GNV we recommend to get the latest GNV sources from the CVS repository and build the GNV kit yourself. See the build instructions on http://www.4ovms.dyndns.org

Don't forget to raise the pipe buffer size.

```
bash$ dcl “define/job DECC\$PIPE_BUFFER_SIZE 65000”
```

And don't forget to disable DCL fall back.

```
bash$ export GNV_DISABLE_DCL_FALLBACK=1
```

Always check the configure script for conftest.dir. Change the name of this file to something like conftest.ddd

```
bash$ mv configure configure.org
```

```
bash$ sed 's/conftest\.dir/conftest\.ddd/g' configure.org > configure
```

Use the configure option --build=<Processor>-<OS maker>-<OS name>. For OpenVMS Alpha the configure option is --build=alpha-hp-vms.

Because the GNV bash version is somewhat outdated, some shell scripts do not work. One of those scripts is called depcomp. This script is used to determine dependency's during compilation. To get rid of this dependency checking, you can add the --disable-dependency-tracking option to the configure command.

The OpenVMS C runtime library

The C Run Time Library or CRTL is an OpenVMS shared library containing most of the “standard” C functions. I quoted the word standard because a) there are a lot of standards and I didn't say witch one. b) every standard is bit of a moving target.

Since the DII COE initiative, HP intends to make the CRTL compliant to the latest X-Open standard. Because of the architectural differences between OpenVMS and UNIX this is not an easy task, especially if you also want to maintain backwards compatibility.

The OpenVMS CRTL developers decided to add these new features gradually with every new OpenVMS and CRTL release.

HP also decided that these new features will become available on both the Alpha and the new Itanium architecture. Some features may also become available on the VAX, but because ODS-5 extended filename support is not available on the VAX, the functionality that is essential to open source porting will probably never become available on the VAX.

The new CRTL features can be categorized in two groups:

1. existing functions with new and more standard functionality
2. completely new functions to make the CRTL more standard (X-Open v.6) compliant.

For the functions in group one the classic behavior is standard and to use the new functionality you need to set feature switches (more on those later).

The functions in group two can be divided in two more groups

1. functions that should work the same in both the classic OpenVMS environment as in the new UNIX like environment
2. functions that should not work the same in both environments

For this last group, think about functions that use or return a path. For this group the behavior of these functions is determined by the previously mentioned feature switches.

So what are these feature switches and how can I set them?

I will not explain what feature switches there are and what they do exactly because you can read all about them in the CRTL help, the CRTL release notes and the CRTL Reference manual. But I will show you some of the interesting new ones in OpenVMS 7.3-2. And I will show you how you can change their values.

So let's first get to the most interesting new feature switch, especially when you're new to porting. It is DECC$UNIX_LEVEL. The text below is from the CRTL reference manual (CRTL 1-38):

> With the DECC$UNIX_LEVEL logical name, you can manage multiple C RTL feature logical names at once. By setting a value for DECC$UNIX_LEVEL from 1 to 100, you determine the default value for groups of feature logical names. The value you set has a cumulative effect: the higher the value, the more groups that are affected. Setting a value of 20, for example, enables all the feature logicals associated with a DECC$UNIX_LEVEL of 20, 10 and 1.
>
> The principal logical names affecting UNIX like behavior are grouped as follows:
>
> | | |
> |---|---|
> | 1 | General corrections |
> | 10 | Enhancements |
> | 20 | UNIX style file names |
> | 30 | UNIX style file attributes |
> | 90 | Full UNIX behavior - No concessions to OpenVMS |

Level 30 is appropriate for UNIX like programs such as BASH and GNV.

The DECC$UNIX_LEVEL values and associated groups of affected feature logical names are:

```
General Corrections         (DECC$UNIX_LEVEL == 1)

DECC$FIXED_LENGTH_SEEK_TO_EOF             1
DECC$POSIX_SEEK_STREAM_FILE               1
DECC$SELECT_IGNORES_INVALID_FD            1
DECC$STRTOL_ERANGE                        1
DECC$VALIDATE_SIGNAL_IN_KILL              1

General Enhancements        (DECC$UNIX_LEVEL == 10)

DECC$ARGV_PARSE_STYLE                     1
DECC$EFS_CASE_PRESERVE                    1
DECC$STDIO_CTX_EOL                        1
DECC$PIPE_BUFFER_SIZE                     4096
DECC$USE_RAB64                            1

UNIX style file names       (DECC$UNIX_LEVEL == 20)

DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION   1
DECC$EFS_CHARSET                          1
DECC$FILENAME_UNIX_NO_VERSION             1
DECC$FILENAME_UNIX_REPORT                 1
DECC$READDIR_DROPDOTNOTYPE                1
DECC$RENAME_NO_INHERIT                    1
DECC$GLOB_UNIX_STYLE                      1

UNIX like file attributes   (DECC$UNIX_LEVEL == 30)

DECC$EFS_FILE_TIMESTAMPS                  1
DECC$EXEC_FILEATTR_INHERITANCE            1
DECC$FILE_OWNER_UNIX                      1
DECC$FILE_PERMISSION_UNIX                 1
DECC$FILE_SHARING                         1

UNIX compliant behavior     (DECC$UNIX_LEVEL == 90)

DECC$FILENAME_UNIX_ONLY                   1
DECC$POSIX_STYLE_UID                      1
DECC$USE_JPI$_CREATOR                     1
DECC$DETACHED_CHILD_PROCESS               1
```

Notes:

- Defining a logical name for an individual feature logical supersedes the default value established by DECC$UNIX_LEVEL for that feature.
- Future revisions of the C RTL may add new feature logicals to a given DECC$UNIX_LEVEL. For applications that specify that UNIX level, the effect is to enable those features by default.

Please note that not all the available feature switches are listed in the quote. The ones not listed are not part of any UNIX level! Read the CRTL help, release notes and documentation for a full list of supported feature switches and their intended use.

You may already have noticed one way of setting these feature switches. You can define a logical with the same name and appropriate value. And because the feature switches alter the behavior at runtime, you don't have to recompile while playing with the switches. But when you are finished determining the appropriate values for the switches you may want to set their values from within your program. You can of course do this by setting the logical from within your program, but there is a much better way.

The CRTL functions decc$feature_get_index, decc$feature_get_name, decc$feature_get_value, and decc$feature_set_value are specially designed for this task.

Below you'll see a small example (CRTL REF-82): program.

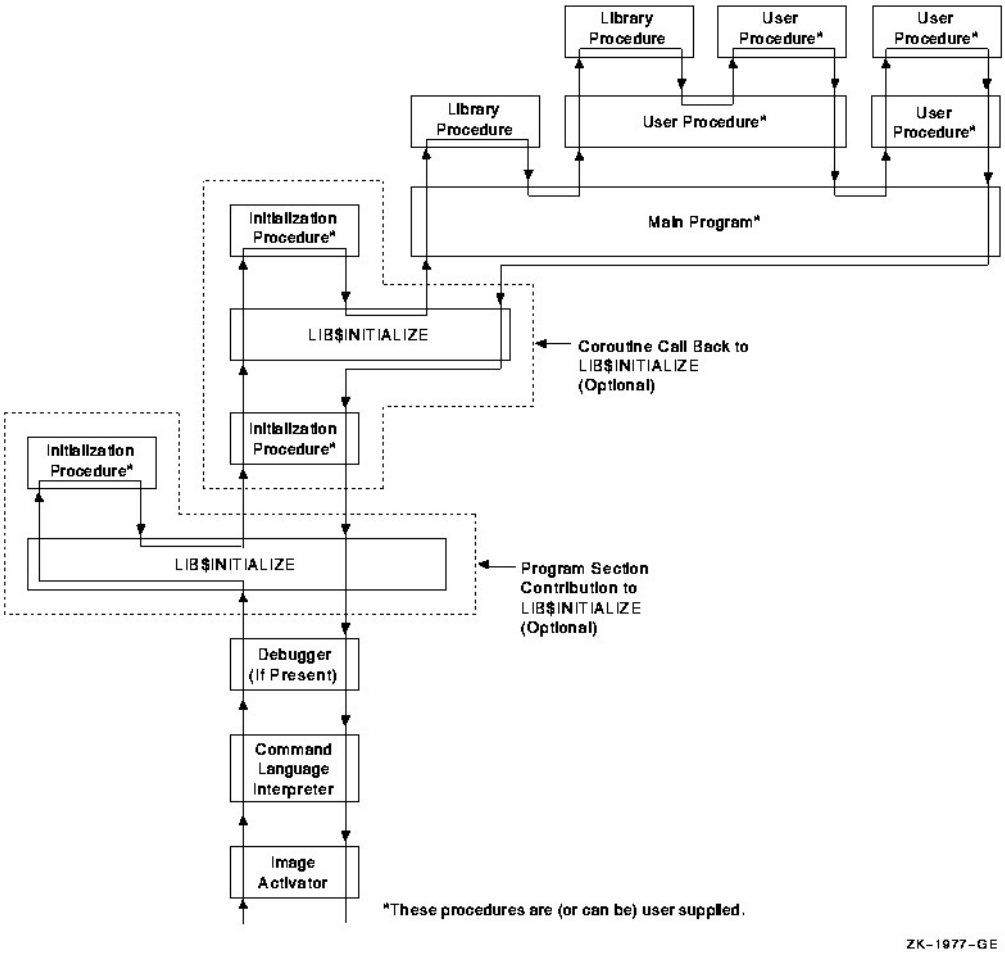
```
static int set_feature_default(char *name, int value)
{
     int index = decc$feature_get_index(name);
      if (index == -1 || decc$feature_set_value(index, 0, value) == -1)
      {
           perror(name);
           return -1;
      }
      return 0;
}

static void my_init( void)
{
      set_feature_default("DECC$POSIX_SEEK_STREAM_FILE" , TRUE);
      set_feature_default("DECC$ARGV_CASE_PARSE_STYLE" , TRUE);
      set_feature_default("DECC$EFS_CASE_PRESERVE" , TRUE);
      set_feature_default("DECC$FILE_SHARING" , TRUE);
}
```

It is not too difficult to add code like this to a program, but there's a catch.

Feature switches like DECC$ARGV_PARSE_STYLE need to be set before the arguments are being parsed and this happens somewhere between image activation and the call to the programs main function.

The process of what happens before your program starts is documented in the OpenVMS Programming Concepts Manual. In chapter 18 you'll find a complete flow of what happens and when (see picture below).



The important thing to note here is the LIB$INITIALIZE function. What we are going to do is create a source file that defines a LIB$INITIALIZE function that points to a function that sets the feature switches to their appropriate values.

Now let's go back to the example I used above. An enhanced version of the example on page CRTL REF-83 is presented below.

```
#pragma extern_model save
#pragma extern_model strict_refdef "LIB$INITIALIZE" nowrt, long
#if __INITIAL_POINTER_SIZE
#    pragma __pointer_size __save
#    pragma __pointer_size 32
#else
#    pragma __required_pointer_size __save
#    pragma __required_pointer_size 32
#endif
     /* Set our contribution to the LIB$INITIALIZE array */
void (* const iniarray[])() = {my_init, } ;
#if __INITIAL_POINTER_SIZE
#    pragma __pointer_size __restore
#else
```

```
#    pragma __required_pointer_size __restore
#endif
#pragma extern_model restore


/*
** Force a reference to LIB$INITIALIZE to ensure it
** exists in the image.
*/
int LIB$INITIALIZE();
globaldef int (*lib_init_ref)() = LIB$INITIALIZE;
```

Put both parts of the example in one source file. Compile and link with your program to make it work. The beauty of it is that you don't have to change the program you are porting to set the feature switches!

If you want to see a more complete example, take a look at GNU:[src.GNV.CRTLSUP.SRC]VMS_CRTL_INIT.C file. This file is used by some of the GNV tools.

Recommendation

A good starting point is UNIX level 30, but don't use the DECC$UNIX_LEVEL feature switch. Set the individual feature switches that make up UNIX level 30 and increase DECC$PIPE_BUFFER_SIZE to at least 8192. The reasons for not using the DECC$UNIX_LEVEL feature switch are compatibility with OpenVMS 7.3-1 and the simple fact that you may not need all the feature switches that make up level 30 or you want to use different values (see the recommendation for DECC$PIPE_BUFFER_SIZE).

Macros

Predefined macros

The list below is a list of predefined macros on OpenVMS 7.3-2 Alpha using the Standard DECC C compiler version 6.5.

```
These macros are in effect at the start of the compilation.
----- ------ --- -- ------ -- --- ----- -- --- ------------


 __G_FLOAT=1  __DECC=1  vms=1  VMS=1  __32BITS=1  __PRAGMA_ENVIRONMENT=1
 __CRTL_VER=70320000  __vms_version="V7.3-2  "  CC$gfloat=1  __X_FLOAT=1
 vms_version="V7.3-2  "  __DATE__="Feb  7 2004"  __STDC_VERSION__=199901L
 __DECC_MODE_RELAXED=1  __DECC_VER=60590001  __VMS=1  __ALPHA=1
 VMS_VERSION="V7.3-2  "  __IEEE_FLOAT=0  __VMS_VERSION="V7.3-2  "
 __STDC_HOSTED__=1  __TIME__="13:54:21"  __Alpha_AXP=1  __VMS_VER=70320022
 __BIASED_FLT_ROUNDS=2  __INITIAL_POINTER_SIZE=0  __STDC__=1
 __LANGUAGE_C__=1  __vms=1  __alpha=1  __D_FLOAT=0
```

The following macros are important for porting:

__VMS, __vms	Are the ones we can use for general OpenVMS specific stuff
__ALPHA, __Alpha_AXP, __alpha	Can be used for Alpha specific stuff
__VAXC, __VAX11C, __vaxc, __vax11c	Can be used for VAX specific stuff
__IA64, __ia64	Can be used for Itanium (ia64) specific stuff
__VMS_VER	Can be used for OpenVMS version specific stuff (please use __CRTL_VER instead)
__DECC_VER	Can be used for compiler version specific stuff
__CRTL_VER	Can be used for CRTL version specific stuff

Please note that I only included the preferred format of the macros, most of them also exist without the two dashes in front.

Other macros

There are also numerous other macros that change the behavior of one or more CRTL functions.

_XOPEN_SOURCE_EXTENDED	Makes visible XPG4-extended features, including traditional UNIX based interfaces not previously adopted by X/Open.
	Standard Selected: XPG4 V2
	Other Standards Implied: XPG4, ISO POSIX-2, ISO POSIX-1, ANSI C
_XOPEN_SOURCE	Makes visible XPG4 standard symbols and causes _POSIX_C_SOURCE to be set to 2 if it is not already defined with a value greater than 2.
	Standard Selected: XPG4
	Other Standards Implied: XPG4, ISO POSIX-2, ISO POSIX-1, ANSI C
_POSIX_C_SOURCE==199506	Header files defined by ANSI C make visible those symbols required by IEEE 1003.1c-1995.
	Standard Selected: IEEE 1003.1c-1995
	Other Standards Implied: ISO POSIX-2, ISO POSIX-1, ANSI C
_POSIX_C_SOURCE==2	Header files defined by ANSI C make visible those symbols required by ISO POSIX-2 plus those required by ISO POSIX-1.
	Standard Selected: ISO POSIX-2
	Other Standards Implied: ISO POSIX-1, ANSI C
_POSIX_C_SOURCE==1	Header files defined by ANSI C make visible those symbols required by ISO POSIX-1.
	Standard Selected: ISO POSIX-1
	Other Standards Implied: ANSI C
_STDC_VERSION__==199409_	Makes ISO C Amendment 1 symbols visible.
	Standard Selected: ISO C Amendment 1.
	Other Standards Implied: ANSI C
_ANSI_C_SOURCE	Makes ANSI C standard symbols visible.
	Standard Selected: ANSI C.
	Other Standards Implied: None.
_POSIX_EXIT	To enable the ISO POSIX-1 compatible exit function.
_BSD44_CURSES	This macro selects the Curses package from the 4.4BSD Berkeley Software Distribution.
_VMS_CURSES	This macro selects a Curses package based on the VAX C compiler. This is the default Curses package.
_SOCKADDR_LEN	This macro is used to select 4.4BSD-compatible and XPG4 V2-compatible socket interfaces. These interfaces require support in your underlying TCP/IP software. Contact your TCP/IP vendor to inquire if the version of TCP/IP software you run supports 4.4BSD sockets. (HP TCP/IP supports this feature, as far as I can tell, Process Softwares Multinet and TCPware don't)
_LARGEFILE	The C RTL provides support for compiling applications to use file sizes and offsets that are two gigabytes (GB) and larger. This is accomplished by allowing file offsets of 64-bit integers.
__USE_LONG_GID_T	To compile an application for 32-bit UID/GID support.
_DECC_SHORT_GID_T	To compile an application for 16-bit UID/GID support
_USE_STD_STAT	This macro is used to select the UNIX style stat structures. (New in OpenVMS 8.2)

Please read the CRTL help and/or CRTL Reference Manual for more information.

Recommendation

Please don't use general macros like VMS, vms , __VMS or_ _vms. Use __CRTL_VER instead. The C Runtime Library is becoming more “standard” with every new version of the CRTL. Fixes may no longer be necessary in newer versions of the CRTL. In some cases program functionality has been crippled with #ifndef VMS in a way that is no longer necessary.

Using __CRTL_VER will allow us to control more precisely what needs to be changed for the many different versions of the CRTL.

When the macro __CRTL_VER doesn't exist, define it with the value of __VMS_VER.

```
#  if defined(__VMS_VER) && !defined(__CRTL_VER)
#    define __CRTL_VER __VMS_VER
#  endif
```

When compiling open source programs in DCL, add the /DEFINE=_POSIX_EXIT to your CC command! This is the default when compiling under bash.

GCC wrapper

The GNV kit does not come with gcc but with a wrapper that uses HP C and C++ compilers to do its compilation work and it uses the standard OpenVMS Linker as a substitute for ld. By the way there is also a ar wrapper that uses the standard OpenVMS Librarian.

The GCC wrapper does quite a good job, but it isn't perfect. Also note that the HP compilers are very strict. By this I mean that they complain immediately when something may not be entirely correct. So we'll see a lot of warnings when compiling open source software.

You can see all the options the gcc wrapper supports by using the -h or -help option.

I would like to note the following on the behavior of the GCC wrapper, the wrapper adds the /DEFINE=_POSIX_EXIT to the compile statement by default.

The -Wc and -Wl gcc option may come in handy if you want to add a HP CC, CPP or LINK option to the gcc command line, that does not seem to be implemented.

One gcc option you sometimes need is -names_as_is_short. This will make the compiler case sensitive and it allows function names with more than 32 characters.

You may get into trouble with the gcc -g option in configure scripts. Many configure scripts use this option by default. This options is used to compile and/or link with debug information included in the program. The problem is that on UNIX systems a program compiled and linked with debug information doesn't start the debugger when executed. This is quite different from what we OpenVMS people are used to. If you see the debugger prompt pop up when running a configure script, either remove the -g option from the configure script or start configure with CFLAGS="".

Fork()

The fork issue is quite high on the issue list because it is quite common to Open Source programs and we don't have this function on OpenVMS.

We do have vfork() on OpenVMS, but the implementation of vfork() on OpenVMS is also not standard compliant. We can however use it to work around most of the fork() calls in opensource software.

Before we dig into the workarounds, lets first determine what fork() is supposed to do.

The fork() function is the UNIX standard way to create a new process. The new (child) process is an exact duplicate of the calling (parent) process except:

- the child process has a unique process ID
- the child process ID does not match any active process group ID
- the child process has a different parent process ID
- the child process has its own copy of the parent's open file descriptors
- the child process has its own copy of the parent's open directory streams
- the child process may have its own copy of the parent's message catalog descriptors
- the child process' values of tms_utime, tms_stime, tms_cutime and tms_cstime are set to 0
- the time left until an alarm clock signal is reset to 0
- all semadj values are cleared
- file locks set by the parent process are not inherited by the child process
- the set of signals pending for the child process is initialized to an empty set
- interval timers are reset in the child process

After fork(), both parent and child processes are capable of executing independently before either one terminates.

The fork() function returns 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, -1 is returned to the parent process, no child process is created and errno is set to indicate the error.

The vfork() function is on most systems identical to the fork() function. On some systems, child's created with vfork() can share data or code segments with their parent process.

On OpenVMS the vfork() function only provides the setup necessary for a subsequent call to an exec function. No child process is create by a vfork call!

When vfork is called:

- it saves the return address (the address of the vfork call) to be used later as a return address for the call to an exec function
- it saves the current context
- it returns the integer 0 the first time it is called (before the call to an exec function is made). After the exec call is made, the exec function returns control to the parent process, at the point of the vfork() call, and returns the process ID of the child as the return value.

Some of you may already see some similarities between fork() and OpenVMS vfork(). If fork() is closely followed by an exec call, we can use OpenVMS vfork() with a little work. If fork() stands on it's own, we're still screwed.

Example 1 (from GNU tar 1.15.1 lib/rmdir.c):

```
#ifndef __VMS
    cpid = fork ();
#else  /* VMS */
    cpid = vfork ();
#endif  /* VMS */
  switch (cpid)
    {
    case -1:              /* cannot fork */
      return -1;          /* errno already set */

    case 0:               /* child process */
      execl ("/bin/rmdir", "rmdir", dpath, (char *) 0);
      _exit (1);

    default:              /* parent process */
.
.
.
      return 0;
    }
```

Example 2 (from GNU tar 1.15.1 lib/rtapelib.c):

```
#ifndef __VMS
    status = fork ();
#else   /* VMS */
    status = vfork ();
#endif  /* VMS */
    if (status == -1)
      {
      int e = errno;
      free (file_name_copy);
      errno = e;
      return -1;
      }

    if (status == 0)
      {
      /* Child.  */
#ifdef __VMS
      save_stdin = dup (STDIN_FILENO);
      save_stdout = dup (STDOUT_FILENO);
#endif  /* VMS */
      close (STDIN_FILENO);
      dup (to_remote[remote_pipe_number][PREAD]);
      close (to_remote[remote_pipe_number][PREAD]);
#ifndef __VMS
      close (to_remote[remote_pipe_number][PWRITE]);
#endif  /* VMS */
      close (STDOUT_FILENO);
      dup (from_remote[remote_pipe_number][PWRITE]);
#ifndef __VMS
      close (from_remote[remote_pipe_number][PREAD]);
#endif  /* VMS */
      close (from_remote[remote_pipe_number][PWRITE]);

      sys_reset_uid_gid ();

      if (remote_user)
        execl (remote_shell, remote_shell_basename, remote_host,
               "-l", remote_user, rmt_command, (char *) 0);
      else
        execl (remote_shell, remote_shell_basename, remote_host,
```

```
              rmt_command, (char *) 0);

      /* Bad problems if we get here.  */

      /* In a previous version, _exit was used here instead of exit.  */
      error (EXIT_ON_EXEC_ERROR, errno, _("Cannot execute remote shell"));
      }

    /* Parent.  */
#ifdef __VMS
    {
      int status;
      status = dup2 (save_stdin, STDIN_FILENO);
      if (status < 0)
      {
        error (EXIT_ON_EXEC_ERROR, errno, _("Error resoring stdin"));
      }
      status = dup2 (save_stdout, STDOUT_FILENO);
      if (status < 0)
      {
        error (EXIT_ON_EXEC_ERROR, errno, _("Error resoring stdout"));
      }
    }
#endif  /* VMS */
    close (from_remote[remote_pipe_number][PWRITE]);
    close (to_remote[remote_pipe_number][PREAD]);
```

This second example may look a little intimidating but remember that on OpenVMS the “Child” code is actually executed by the parent. The “Child” process is created by the execl() function call not by the vfork() function call.

Note 1: By default the OpenVMS exec call's use LIB$SPAWN to create child processes. In most cases this is what you want, however in some cases you may want the child processes to be a detached processes. To achieve this you can enable the DECC$DETACHED_CHILD_PROCESS feature switch. This has some implications. See CRTL Reference Guide chapter 5.

Note 2: On OpenVMS the execlp() and execvp() functions search VAXC$PATH and not the PATH environment variable to obtain the location of the file to execute. This can be quite problematic.

RMS

RMS stands for Record Management System. The OpenVMS filesystem is built on RMS and all the files on an OpenVMS system are RMS files. An RMS file is record oriented and can contain keys for keyed access (RMS indexed files). There are many different kinds of records and countless record attributes in RMS. As with most record oriented systems, locking of files and records is something to be very aware of. UNIX doesn't have these features and on UNIX systems it is possible for two programs to have the same file open for writing at the same position in the file. On OpenVMS this is impossible and you have to deal with this problem when you run into it.

Another problem is the difference between stream-lf files and variable-length records. Both files contain records of variable length, but from a UNIX programs point of view Stream-lf files best resemble UNIX files. However very few OpenVMS programs know how to handle stream-lf files. OpenVMS programs typically handle files with the variable-length record type.

Known Porting Issues

Quite a few CRTL functions still don't quite work as their UNIX counterparts and some functions are still missing. To give you some examples:

The vfork function does not behave exactly like its UNIX counterpart (read the section on vfork in the CRTL reference manual). If a program uses vfork it will probably check the existence of vfork during configure and this is also a situation where configure will hang. The solution to the hang is quite simple. Log in using a different session and look for the child process of your previous session. Kill it and the configure script continues.

And while we're at it, OpenVMS unfortunately has no fork function yet, but there are many known workarounds available is only we know where.

This is not a definitive guide on how to port open source to OpenVMS. Things are changing quite rapidly. We should continue to keep this document up to date with the current state of OpenVMS and the GNV kit.

We found a list of known areas where compatibility problems occur when porting from UNIX to OpenVMS on comp.os.vms, This list is the basis for a list we maintain. This list can be found in appedix 2: List of known issues.

Legend:

issue#: sequential number (1...31 from cov list, rest locally generated)
sev/prio: severity to our port (VMS encoded..)/our priority rating (VMS encoded..)

The severity and priority are encoded from the OpenOffice porting group viewpoint.

(default = I/0)
status: VMS encoded activity from the OpenOffice group viewpoint.

CUR: actively being worked on.

COM: no local activity due to lack of local resources.

LEF: waiting for something under local control (f.i. Media).

CEF: waiting for something under external control (f.i. patch from VMS-engineering).

(default = LEF)
description: Short description of the issue.
explanation, status: Explanation of the issue and known workarounds, fixes and viewpoints.

We added some comments to some of the items (see (OO)).

Part V: Appendices

Appendix 1: Example port of m4

In this example I use GNU m4 1.4. By the way as you can see below I use POSIX.2 PAX to untar the tar file.

```
$ set process /parse_style=extended
$ gunzip m4-1.4.tar.gz
$ pax -rvf m4-1.4.tar
USTAR format archive assumed
m4-1.4/README
m4-1.4/NEWS
m4-1.4/TODO
m4-1.4/THANKS
m4-1.4/COPYING
m4-1.4/INSTALL
.
.
.
```

Let me not bore you with a long list of files.

Before I start bash I'll raise DECC$PIPE_BUFFER_SIZE:

```
$ define/job decc$pipe_buffer_size 65000
```

Start Bash:

```
$ bash
```

Disable DCL fallback:

```
bash$ export GNV_DISABLE_DCL_FALLBACK=1
bash$ ls
m4-1.4  m4-1.4.sav  m4-1.4.tar
bash$ cd m4-1.4
bash$ ls
BACKLOG    Makefile.in  TODO        checks        doc         mkinstalldirs
COPYING    NEWS         acconfig.h  config.h.in   examples    src
ChangeLog  README       aclocal.m4  configure     install-sh  stamp-h.in
INSTALL    THANKS       c-boxes.el  configure.in  lib
```

let's start configure with following options:

`--build=build'

Specifies the type of system on which the package will be built. If not specified, the default will be the same configuration name as the host.

The parameter for the build option is in the following form:

<processor>-<OS Manufacturer>-<OS name>

Note that we don't have to set the execute bit to start the configure script.

```
bash$ ./configure --build=alpha-hp-vms
```

```
creating cache ./config.cache
checking for mawk... no
checking for gawk... gawk
checking for gcc... gcc
.
.
.
creating Makefile
creating doc/Makefile
creating lib/Makefile
creating src/Makefile
creating checks/Makefile
creating examples/Makefile
creating config.h
```

Wow, no problems so far. Let's try make.

```
bash$ make
for subdir in doc lib src checks examples; do \
  echo making all in $subdir; \
  (cd $subdir && make CC='gcc' CFLAGS='-g' LDFLAGS='' LIBS='' prefix='/usr/local'
exec_prefix='/usr/local' bindir='/usr/local/bin' infodir='/usr/local/info' all)
|| exit 1; \
done
making all in doc
make[1]: Entering directory `/USER50/KITS/GNU/M4/m4-1.4/doc'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/USER50/KITS/GNU/M4/m4-1.4/doc'
making all in lib
make[1]: Entering directory `/USER50/KITS/GNU/M4/m4-1.4/lib'
gcc -c -DHAVE_CONFIG_H -I.. -I.  -g regex.c
gcc -c -DHAVE_CONFIG_H -I.. -I.  -g getopt.c

      if (optind != argc && !strcmp (argv[optind], "--"))
..........................^
%CC-I-IMPLICITFUNC, In this statement, the identifier "strcmp" is implicitly
declared as a function.
at line number 408 in file USER50:[KITS.GNU.M4.m4-1^.4.lib]getopt.c;1

        if (!strncmp (p->name, nextchar, nameend - nextchar))
............^
%CC-I-IMPLICITFUNC, In this statement, the identifier "strncmp" is implicitly
declared as a function.
at line number 484 in file USER50:[KITS.GNU.M4.m4-1^.4.lib]getopt.c;1

            if (nameend - nextchar == strlen (p->name))
....................................^
%CC-I-IMPLICITFUNC, In this statement, the identifier "strlen" is implicitly
declared as a function.
at line number 486 in file USER50:[KITS.GNU.M4.m4-1^.4.lib]getopt.c;1
gcc -c -DHAVE_CONFIG_H -I.. -I.  -g getopt1.c
gcc -c -DHAVE_CONFIG_H -I.. -I.  -g error.c

char *strerror ();
......^
%CC-W-FUNCREDECL, In this declaration, function types differ because one has no
argument information and the other has an ellipsis.
at line number 56 in file USER50:[KITS.GNU.M4.m4-1^.4.lib]error.c;1
gcc -c -DHAVE_CONFIG_H -I.. -I.  -g obstack.c

    abort ();
....^
%CC-I-IMPLICITFUNC, In this statement, the identifier "abort" is implicitly
```

```
declared as a function.
at line number 333 in file USER50:[KITS.GNU.M4.m4-1^.4.lib]obstack.c;1
gcc -c -DHAVE_CONFIG_H -I.. -I.  -g xmalloc.c
gcc -c -DHAVE_CONFIG_H -I.. -I.  -g xstrdup.c
rm -f libm4.a
ar cru libm4.a regex.o getopt.o getopt1.o error.o obstack.o xmalloc.o xstrdup.o
Warning: u unrecognized switch
%LIBRAR-W-COMCOD, compilation warnings in module ERROR file
USER50:[KITS.GNU.M4.m4-1^.4.lib]error.o;1
: libm4.a
make[1]: Leaving directory `/USER50/KITS/GNU/M4/m4-1.4/lib'
making all in src
make[1]: Entering directory `/USER50/KITS/GNU/M4/m4-1.4/src'
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g m4.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g builtin.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g debug.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g eval.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g format.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g freeze.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g input.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g macro.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g output.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g path.c
gcc -c -DHAVE_CONFIG_H -I.. -I. -I./../lib  -g symtab.c
gcc  -o m4 m4.o builtin.o debug.o eval.o format.o freeze.o input.o macro.o
output.o path.o  symtab.o ../lib/libm4.a
%LINK-W-WRNERS, compilation warnings
        in module ERROR file USER50:[KITS.GNU.M4.m4-1^.4.lib]libm4.a;1
%LINK-W-MULDEF, symbol DECC$GETOPT multiply defined
        in module DECC$SHR_EV56 file SYS$COMMON:[SYSLIB]DECC$SHR_EV56.EXE;1
%LINK-W-MULDEF, symbol DECC$GA_OPTARG multiply defined
        in module DECC$SHR_EV56 file SYS$COMMON:[SYSLIB]DECC$SHR_EV56.EXE;1
%LINK-W-MULDEF, symbol DECC$GL_OPTOPT multiply defined
        in module DECC$SHR_EV56 file SYS$COMMON:[SYSLIB]DECC$SHR_EV56.EXE;1
%LINK-W-MULDEF, symbol DECC$GL_OPTIND multiply defined
        in module DECC$SHR_EV56 file SYS$COMMON:[SYSLIB]DECC$SHR_EV56.EXE;1
%LINK-W-MULDEF, symbol DECC$GL_OPTERR multiply defined
        in module DECC$SHR_EV56 file SYS$COMMON:[SYSLIB]DECC$SHR_EV56.EXE;1
make[1]: Leaving directory `/USER50/KITS/GNU/M4/m4-1.4/src'
making all in checks
make[1]: Entering directory `/USER50/KITS/GNU/M4/m4-1.4/checks'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/USER50/KITS/GNU/M4/m4-1.4/checks'
making all in examples
make[1]: Entering directory `/USER50/KITS/GNU/M4/m4-1.4/examples'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/USER50/KITS/GNU/M4/m4-1.4/examples'
```

OK, we did see some warnings, but nothing too bad. let's try the m4 executable.

```
bash$ cd src
bash$ m4 --help
Usage: m4 [OPTION]... [FILE]...
Mandatory or optional arguments to long options are mandatory or optional
for short options too.

Operation modes:
      --help                   display this help and exit
      --version                output version information and exit
  -e, --interactive            unbuffer output, ignore interrupts
  -E, --fatal-warnings         stop execution after first warning
  -Q, --quiet, --silent        suppress some warnings for builtins
  -P, --prefix-builtins        force a `m4_' prefix to all builtins
```

```
Preprocessor features:
  -I, --include=DIRECTORY      search this directory second for includes
  -D, --define=NAME[=VALUE]    enter NAME has having VALUE, or empty
  -U, --undefine=NAME          delete builtin NAME
  -s, --synclines              generate `#line NO "FILE"' lines

Limits control:
  -G, --traditional            suppress all GNU extensions
  -H, --hashsize=PRIME         set symbol lookup hash table size
  -L, --nesting-limit=NUMBER   change artificial nesting limit

Frozen state files:
  -F, --freeze-state=FILE      produce a frozen state on FILE at end
  -R, --reload-state=FILE      reload a frozen state from FILE at start

Debugging:
  -d, --debug=[FLAGS]          set debug level (no FLAGS implies `aeq')
  -t, --trace=NAME             trace NAME when it will be defined
  -l, --arglength=NUM          restrict macro tracing size
  -o, --error-output=FILE      redirect debug and trace output

FLAGS is any of:
  t   trace for all macro calls, not only traceon'ed
  a   show actual arguments
  e   show expansion
  q   quote values as necessary, with a or e flag
  c   show before collect, after collect and after call
  x   add a unique macro call id, useful with c flag
  f   say current input file name
  l   say current input line number
  p   show results of path searches
  i   show changes in input files
  V   shorthand for all of the above flags

If no FILE or if FILE is `-', standard input is read.
bash$
```

And it works!?!

But we are far from ready.

- we need to set the necessary feature switches by linking a LIB$INITIALIZE routine to our program
- the CRTL feature switches are not the only way to make some functions behave more UNIX like. For example you can define _POSIX_EXIT to make the CRTL exit function behave POSIX compliant. Please note that the _POSIX_EXIT macro is already defined by the gcc wrapper!
- As an OpenVMS person, I really can't live with all these warnings

To fix the LIB$INITIALIZE problem, you could do the following (you're still in the src directory):

```
bash$ cp /src/GNV/CRTLSUP/SRC/VMS_CRTL_INIT.C .
```

Edit the Makefile in this directory to add the VMS_CRTL_INIT.C source file and VMS_CRTL_INIT.o object file to the following lines:

```
SOURCES = m4.c builtin.c debug.c eval.c format.c freeze.c input.c \
macro.c output.c path.c stackovf.c symtab.c
OBJECTS = m4$O builtin$O debug$O eval$O format$O freeze$O input$O \
macro$O output$O path$O $(STACKOVF) symtab$O
```

After the changes the lines should look like this:

```
SOURCES = m4.c builtin.c debug.c eval.c format.c freeze.c input.c \
macro.c output.c path.c stackovf.c symtab.c VMS_CRTL_INIT.C
OBJECTS = m4$O builtin$O debug$O eval$O format$O freeze$O input$O \
macro$O output$O path$O $(STACKOVF) symtab$O VMS_CRTL_INIT$O
```

To fix the compile warnings you'll need to make some changes to various files. Start with config.h in the root directory of the package. When you look at this file, you'll understand why.

The linker warnings are the result of the fact that the m4 sources contain replacement functions for the standard command-line argument parsing functions with the same name's as the standard functions. You can either disable these replacements or do something like the following:

Go to the end of the config.h file and add the following lines:

```
#define getopt my_getopt
#define optarg my_optarg
#define optopt my_optopt
#define optind my_optind
#define opterr my_opterr
```

Now run ./configure and make again.

So in short:

```
bash$ make clean
bash$ make
```

GNU m4 also comes with some scripts to check it functionality so let's take a look how well it does.

```
bash$ make check
.
.
.
cd checks && make check
make[1]: Entering directory `/USER50/KITS/GNU/M4/m4-1.4/checks'
PATH=`pwd`/../src:$PATH; export PATH; \
cd . && ./check-them *[0-9][0-9].*
GNU m4 1.4
Checking 01.define
.
.
.
Checking 30.include
../doc/m4.texinfo:2078: Origin of test
30.include: stderr mismatch
1c1
< 30.include:2: m4: Cannot open no-such-file: No such file or directory
---
> 30.include:2: m4: Cannot open no-such-file: no such file or directory
Checking 31.include
.
.
.
Checking 57.m4exit

Failed checks were:
  30.include:err
make[1]: Leaving directory `/USER50/KITS/GNU/M4/m4-1.4/checks'
bash$
```

The test that failed is only complaining about the error message not being exactly right.

All in all this looks quite good already. But things aren't always this easy.

Appendix 2: Porting Issues

Legend:

issue#: sequential number (1...31 from cov list, rest locally generated)
sev/prio: severity to our port (VMS encoded..)/our priority rating (VMS encoded..)

The severity and priority are encoded from the OpenOffice porting group viewpoint.

(default = I/0)
status: VMS encoded activity from the OpenOffice group viewpoint.

CUR: actively being worked on.

COM: no local activity due to lack of local resources.

LEF: waiting for something under local control (f.i. Media).

CEF: waiting for something under external control (f.i. patch from VMS-engineering).

(default = LEF)
description: Short description of the issue.
explanation, status: Explanation of the issue and known workarounds, fixes and viewpoints.

We added some comments to some of the items (see (OO)).

issue#	*sev/prio*	*status*	*description*	*explanation, status*

1.	I/0	CUR	stat and st_ino	On UNIX stat.h has:
				ino_t st_ino;
				but on OpenVMS it has
				ino_t st_ino[3];
				consequently the st_ino part of the stat structure is a value on UNIX and a pointer on OpenVMS. Generally this means you have to replace something like
				if(inode == foo.st_ino)
				with
				if(inode[0] == foo.st_ino[0] &&
				inode[1] == foo.st_ino[1] &&
				inode[2] == foo.st_ino[2[)
				(OO) This issue was fixed in OpenVMS 8.2, however you'll have to compile with -D_USE_STD_STAT (GNV) or /DEFINE=_USE_STD_STAT.

2.	I/0	COM	write() to tcp/ip socket	write() is supposed to send as much of a buffer as it can to the output device and then return that value. On OpenVMS write() will fail if it attempts to write more than 64k bytes to a socket. So even though write() is usually used in a loop that cycles through until a buffer is written out it won't work correctly on OpenVMS if the buffer is too big. The workaround is to put in an #ifdef that restricts the transfer to < 64k bytes at one time.
3.	I/0	COM	use of select()	OpenVMS select() only works on sockets, it does NOT work on files. Therefore code which uses select() to synchronize IO for both sockets and files will not function. The fix is reasonably complex. For an example see: http://seqaxp.bio.caltech.edu/pub/SOFTWARE/XTERM_VMS_122_TOP.ZIP
4.	E/16	CUR	foo.tar.gz	UNIX doesn't care what a file is, OpenVMS does. ODS2 will only accept one ".extension" in a file name. ODS5 will accept pretty much anything that UNIX will. Starting with OpenVMS 7.3-2 even the system disk can be ODS5 formatted. But be careful not all layered products support this yet. (OO) using new CRTL features promisses a (more) simple port when using only ODS5 disks. At this moment we plan to do a ODS5 only port.

issue#	*sev/prio*	*status*	*description*	*explanation, status*
5.	W/4	COM	A zillion warnings on compilation	Bad news Bucky, most software is crap and you're looking at it. The OpenVMS compilers are much pickier by default than those on other platforms and show you where the problems are. Your best bet in general is to fix the warnings you see, and make the compiler air out all the code's dirty laundry. Get your code to compile cleanly with /standard=ansi89/prefix=all/warn=enable=(level4,questcode) and you'll save yourself a lot of trouble later on. It will also make your code more portable. (OO) we know from the porting newsgroup and from direct contact that there will be lots of 64 bit issues and typical compiler version specific workarounds in the code. We'll try to produce clean code, but won't modify general code just for cleanliness.

issue#	*sev/prio*	*status*	*description*	*explanation, status*
6.	E/16	CUR	use of UNIX paths	Most UNIX code thinks that it can always take a path and add "/blah/foo.bar" to get a relative path. This will work on OpenVMS as well so long as the initial path is in UNIX format. However if somebody entered: "USRDISK:[JOE.TEST]" then the full file spec will be a hybrid: "USRDISK:[JOE.TEST]/blah/foo.bar" which won't fly. Compaq C provides functions for converting back and forth between OpenVMS and UNIX file/path specs. In general, you want to stick with UNIX file paths since compilers on some other non UNIX platforms support them, but OpenVMS file paths are only useful on OpenVMS. Ultimate fix: a standard set of C routines which define a file/path structure something like: char * nodename; struct ACCESSSTRU access; char * device; struct PATHSTRU path; char * name; char * type; char * version; which will hide all the delimiter information (so no more explicit testing of []:/\. etc. in the code). And even then programmers will have to stick to the routines which manipulate this structure. So long as all this information is lumped into one character string inside the code there will always be portability problems. This was a design flaw in the C language which should have been handled back when "fopen" was first defined. (OO) using the new CRTL features promises a (more) simple port when using only ODS5 disks. At this moment we plan to do a ODS5 only port.
7.	I/0	LEF	stream-lf and long writes	The default file type produced by the C RTL on OpenVMS is stream-lf. It is very similar to the text file format on UNIX EXCEPT that records can't go above 32767 bytes. Well, they can, but they get split and basically it doesn't quite work like it does on UNIX. Workaround: If possible, use binary file types. Also complain loudly and bitterly (and most likely, futilely) to HP and maybe someday they'll fix it. (OO) With the DECC$DEFAULT_UDF_RECORD feature switch enabled, file access mode defaults to RECORD instead of STREAM mode for all files except STREAMLF.

issue#	*sev/prio*	*status*	*description*	*explanation, status*
8.	I/0	LEF	what about X11?	That's a whole other subject. See: "X11/Motif portability concerns, UNIX to OpenVMS" at http://seqaxp.bio.caltech.edu/www/X11_VMS_NOTES.TXT
9.	W/4	COM	data types	Ironically this will get you most often when you port from Tru64. "long int" there is 64 bits, but it's only 32 bits on OpenVMS. Tru64 also has a long long type which is 128 bits. For "long int" you can usually #ifdef in "unsigned __int64" on the OpenVMS version. For "long long" best hope that the code doesn't really need 128 bits, because you can't get it on this platform! (OO) we know from the porting newsgroup and from direct contact that there will be lots of 64 bit issues and typical compiler version specific workarounds in the code. We'll try to produce clean code, but won't modify general code just for cleanliness.
10.	I/0	LEF	ioctl for terminals	If you encounter ioctl() calls being used to control a terminal you are out of luck. The only solution is to rewrite the code using QIO's. You will usually need to access the documentation from the source UNIX's, as ioctl isn't all that portable among UNIX's either. (OO) POSIX.! depreciated ioctl() a long time ago, because is isn't portable. But the POSIX.! recommended termios.h with tc*() and cf*() fuctions are not available on OpenVMS either.
11.	E/16	CUR	makefile	OpenVMS has no "make". You can rewrite makefile to descrip.mms and use either MMS (part of DECset) or MMK (free) and obtain similar functionality. Your other choice is to install GNV from http://gnv.sourceforge.net/. It has bash and gmake. (OO) We know we have to have DMAKE. This MAKE version is the Achilles-heel of this port. The whole project depends on the availability of DMAKE. We have a alpha version working. Continued improvements in the CRTL and the porting library will make DMAKE a more stable and reliable product.
12.	I/0	LEF	lines are too long to view	Many of the OpenVMS tools don't like the wide records which are often found in source code and other files originating on UNIX systems. For instance, you can't TYPE many files, and EDIT/EDT chokes on long records too. I usually use NEDIT http://www.decus.de:8080/www/vms/sw/nedit.htmlx to handle these sorts of file problems.

issue#	*sev/prio*	*status*	*description*	*explanation, status*
13.	I/0	LEF	why does it run so slowly?	After you complete your port you will often find that the OpenVMS version is much, much slower than the UNIX version if it is very IO intensive. The ratio can be as high as 100X faster for UNIX over OpenVMS. Most often this is due to the file caching speedup from UNIX, plus the 2-6X slowdown that RMS imposes on record oriented files (as measured on a RAMdisk.) Binary IO speeds will be closer, the IO slowdown is primarily a problem for text/record oriented files. You can improve performance by using the SET RMS command to increase the block and buffer sizes. Also increase the /extend value to avoid a flurry of short extends. If you know ahead of time the final size of the output file set extend to that. (You can also set these parameters in the code by using OpenVMS specific extensions.) Improvements in the OpenVMS file caching system are in late stages of development, and those should help once they appear on production systems. That said, watch out for the use of, umm, unusual file techniques on programs from UNIX. For instance, you may find frequent calls to freopen(), or other program methods which cause the disk heads to fly back and forth between the front of the file and the end. These sorts of bad design are masked on UNIX by the file caching system, but are exposed in all their glory on OpenVMS. Programs may also run slowly on OpenVMS, or not at all, if they expect to be able to grab vast amounts of memory. User accounts on OpenVMS are usually allocated a relatively small amount of virtual and physical memory. (OO) The recent OpenVMS ports of Apache 2.0 and Mozilla 1.5 prove that this no longer an issue.
14.	I/0	LEF	why do I get two (or more) copies of the output file?	Many UNIX programs try to determine if they can write an output file by doing an fopen, and if it succeeds, doing another. On UNIX the second one overwrites the first and no one's the wiser. On OpenVMS you get two versions. Sometimes scratch files are used in a similar manner, only for those they tend to be opened and closed many more times, resulting in a long string of files. Fix: find the offending second fopen() and #ifdef it and the fclose() before it out of the code. For scratch files, either reopen the existing file or be sure to delete the file after the fclose().
15.	I/0	LEF	fseek/ftell don't work right	These assume a UNIX like file organization which may or may not be present on files being read by a C program in OpenVMS. Fix: rewrite the code to use fgetpos() and fsetpos(). (OO) Resent changes to the CRTL solve this issue.

issue#	*sev/prio*	*status*	*description*	*explanation, status*
16.	W/4	CUR	linker can't resolve ReadDir	By default UNIX is case sensitive pretty much everywhere. OpenVMS is not. So the symbol "ReadDir" goes to "readdir" which is the standard function. You'll also see this a lot where people have a variable "foobar" and a function "Foobar". It's generally a problem any place code uses case to distinguish between variables and/or functions. (Constants, which only the compiler sees, tend not to cause so much havoc.) Code written this way is really a pain to port because you (usually) didn't write it, and so tend not to notice these minor case differences. The bugs which can result if the linker does manage to resolve things, but incorrectly, can be very messy. Preferred fix: rewrite the code so that all symbols are unique when uppercased. Other fix: use the compiler switch /names=as_is, which preserves case. (OO) Our preferred method is /names=(as_is,short).
17.	I/0	LEF		compiler gives an implicit function warning for a common function This happens either because the appropriate header has been omitted or because the function in question is #ifdef'd off by language standard specific defines. Most UNIX compilers are really sloppy about which function is in which language spec. Ok, they're not sloppy, but the programmers never put them into a mode where they check. So some function which is an XOPEN extension compiles in without warnings on UNIX but raises a warning on OpenVMS. Fix: Use /define=(_XOPEN_SOURCE,_XOPEN_SOURCE_EXTENDED,_POSIX_SOURCE) as required to get the right lines of the header files processed. (OO) Please read the Macro's chapter and The CRTL Reference Guide (1.5.2 Selecting Standard; page 1-19). The macro _XOPEN_SOURCE_EXTENDED already implies _XOPEN_SOUCE and _POSIX_SOURCE!
18.	W/2	COM	where are the standard header files?	They always live in the text library: SYS$COMMON:[SYSLIB]DECC$RTLDEF.TLB but may also have been expanded into SYS$SYSDEVICE:[SYS0.SYSCOMMON.DECC$LIB.REFERENCE.DECC$RTLDEF]*.h (OO) Be aware that the HP C/C++ compilers use the text libraries in SYS$COMMON:[SYSLIB]DECC$RTLDEF.TLB, and that the text version in SYS$SYSDEVICE:[SYS0.SYSCOMMON.DECC$LIB.REFERENCE.DECC$RTLDEF]*.h are only for reference purposes and may not be up-to-date!

issue#	*sev/prio*	*status*	*description*	*explanation, status*
19.	E/4	COM	what's wrong with #include <../../foo/woo/blah.h>?	Putting paths into includes is BAD, EVIL, YUCKY, etc., and unfortunately, fairly common. The example shown here is the worst case - you'll only ever have a hope of compiling this if the default directory is in exactly the right place. Preferred fix: Whenever possible eliminate the paths from the code and use /include=([-],[],[.foo]) and the like to tell the compiler where to find them. (These are the equivalent of using on UNIX: -I.. -I. -Ifoo) Desperate measures fix: For an include like you can define a concealed logical "foo" which points to the correct place in the directory structure on your system and Compaq C will be able to look in the [.woo.moo] subdirectory of it and find the file blah.h. (OO) Although I don't remember ever having a problem with include paths, the current version of the HP C/C++ compilers handles UNIX style include paths very well. In the current C standard, the extention (.h) shoold also be ommitted.
20.	I/0	LEF	the compiler includes "foo.h" but not "Foo.h"	OpenVMS is case insensitive. Even ODS5, which will preserve case, will not distinguish between these two files. Usually this shows up long before the compiler gets a shot at it when the archive is unpacked and complains/warns that it is overwriting FOO.H. Fix: rewrite the code to use case invariant file names. (OO) Starting with OpenVMS 7.3-1 you can turn on case sensitive behavior. In DCL you do this with: `$ SET PROCESS/CASE=SENSETIVE` But you should also compile with /names=as_is

issue#	*sev/prio*	*status*	*description*	*explanation, status*

21.	I/0	LEF	how do I find a memory access problem?	Often code which ran "correctly" on one platform will fail miserably on another if it access memory out of bounds. Such "working" code will usually fail when you port it to OpenVMS just because variables will be located in memory differently.
				
				One method of dealing with this is to do:
				
				`#ifdef __VMS`
				`#define free myfree`
				`int decc$free(void *ptr); /* prototype for decc$free */`
				`void myfree( void *ptr){`
				`#include #include if(decc$free(ptr) != 0){`
				`(void) printf("illegal free() operation\n");`
				`(void) lib$signal(SS$_ACCVIO);`
				`}`
				`}`
				`#endif /* __VMS */`
				
				decc$free will complain if you try to free a region of memory, whereas free() cannot warn you about the mayhem which is being committed. Many memory errors begin or end with an invalid free() and this will catch them.
				
				Here is a method suggested by Hoff (Stephen) Hoffman and posted in comp.os.vms
				
				Because of the likelihood of application programming errors -- trampling past the end of the allocated storage being most common -- I almost never call malloc and free directly. Further, by intercepting the memory allocation calls, I can call lib$get_vm and similar, particularly using VM zones, and can tailor the particular behavior most appropriate. With a "temporary mempory pool" VM zone, I can also flush all allocations in that zone in a single call, so that I can easily reset the pool... Also available with VM zones are the tools to traverse and report on the VM zones.
				
				Jacket your calls to malloc and free. Code the malloc jacket to add a quadword at the front of the allocated area and a quadword at the back and then call the actual malloc asking for the requested size plus sixteen bytes. Before returning the allocated storage -- the base address of the allocated block plus eight -- fill the front and back quadword with a known (and no zero bytes in the quadword) pattern, possibly based on the address and size, etc. Code the free jacket to check for the patterns and scream if it finds errors, and to call free if not -- remembering to back up the base of the buffer by eight bytes from what the caller passed in. (I typically refer to these quadwords as "fenceposts" -- straying from the expected behaviour is quite easy to track down.)
				
				If you were on more recent OpenVMS Alpha version, you could use the heap analyzer in the OpenVMS Debugger to poke around.
				
				(OO) Also note that there are several compiler options

issue#	*sev/prio*	*status*	*description*	*explanation, status*
22.	E/24	CEF	fork()	OpenVMS vfork() doesn't work like fork(). Search your code for fork() and if you find it, refer to gnuplot, or some other already ported application, for an example of how to replace the code. (OO) Also note that OpenVMS vfork() doesn't quite work like UNIX vfork(). See the fork() chapter in this document.
23.	I/0	LEF	where do I look for help?	Post to the newsgroup comp.os.vms Use dejanews power search at http://www.dejanews.com/home_ps.shtml to search that newsgroup. (Be aware that the default setting only goes back about a year, use the date fields on the bottom to look back further.) "Ask the wizard" at http://h71000.www7.hp.com/wizard/ Get a copy of the freeware CD or other already ported programs and scan them for solutions to the current problem. The freeware CD is on the net at: http://h71000.www7.hp.com/openvms/freeware/ (OO) The GNV http://gnv.sourceforge.net/ site, the HP Open Source Tools page http://h71000.www7.hp.com/opensource/opensource.html and our own sites http://www.4ovms.dyndns.org and http://www.oooovms.dyndns.org to name a few more.
24.	I/0	LEF	system()	system() passes a command line to a subprocess, executes it, and then returns. Since the command line is by definition OS specific, any instances of system() in ported code must inevitably be rewritten. Ideally they should be removed entirely - find some way to do perform the desired action in code rather than by running a subprocess. (OO) It isn't OS specific. It's shell specific. Also note that with GNV installed many UNIX commands work.
25(JEM).	I/0	LEF	getenv() behavior	The behavior of getenv() depends on the version and ECO of the DEC C RTL and possibly logical names defined on your system. Fix: use sys$trnlnm() instead. (DRM comment: I've also observed instances where symbols and logicals interfered with the expected behavior. This happened primarily when the symbol/logical involved was a "standard" symbol that getenv handled differently. For instance, if "term" was previously defined in DCL than getenv("term") would return the user defined value, which often was not at all the desired quantity, and the program would fail.) (OO) Recent versions of the CRTL may have fixed this issue.

issue#	*sev/prio*	*status*	*description*	*explanation, status*
26(JEM).	I/0	LEF	mmap() on text files	mmap() will open files in a binary mode. That is fine so long as the file is stream-lf or binary. However, if it is some other type of RMS file the program doing this will likely fail when it encounters the embedded RMS information. Fix (DRM): require input files to be either stream-lf or binary, or if that is not possible, do not use mmap(). Either way will likely require extensive recoding. (OO) Recent versions of the CRTL may have fixed this issue.
27(JEM).	I/0	LEF	/tmp and others map to locations, overridden by logical names	Apparently some versions of the C RTL map /tmp to SYS$SCRATCH /dev/null to NLA0:, and perhaps some others are translated as well. In the case of /tmp if you have defined a logical TMP and try to write to /tmp/foo.txt it will go to the directory set by the logical. This is consistent with the usage of logicals for paths which are not "special". Just be aware that the logical TMP or DEV may have been defined with some other meaning, so that /dev/null might end up creating a real file NULL in the directory pointed to by the logical DEV! (OO) There are several CRTL feature switches to tune this behavior.
28(JEM).	I/0	LEF	setuid()	The setuid() function is a stub and just returns 0 to indicate "success". That may not be optimal, since it really failed (the UID did not change.) User written setuid() is possible, but when the transfer is to an account which does not have read access to the calling processes job table, then SYS$SCRATCH and SYS$LOGIN will not be available. (OO) Starting with OpenVMS 7.3-2 you can enable POSIX style ID's, with the DECC$POSIX_STYLE_UID feature switch. With POSIX style ID's enabled setuid() is fully functional.
29(JEM).	I/0	LEF	uid_t/gid_t size inconsistencies	getuid() returns 16 bits (the member part of a UIC) with some compiler flags or on VMS 6.x and under, and 32 bits (the whole UIC) otherwise. stat() returns a 32 bit uid and a 16 bit gid. setuid() (user written) will need a 32 bit uid value. (OO) For some time now, you can compile with the macro __USE_LONG_GID_T to enable 32 bit gid and uid values.

issue#	*sev/prio*	*status*	*description*	*explanation, status*
30.	I/0	LEF	Files produced by the C RTL have invalid RMS attributes	This isn't a portability problem per se, but rather a general C RTL problem which will affects all ported programs. The problem is that sequential text files written by the C RTL default to having a Longest Record value of 32767, and this can cause havoc with some other programs which foolishly believe that the value is accurate. For instance, SORT will allocate 32767 bytes/record for any such file, so that it cannot easily sort even a relatively small file. (The work around in this instance is to either use SORT/PROCESS=tag or to figure out how big LRL is and repair the file with SET FILE/ATTRIB=(lrl:whatever) before sorting it.) Partial fix: (This is really a sad excuse for a solution but it's all there is) $ define/user decc$default_lrl 100 will set the LRL value to 100 instead of 32767. However, the 100 can be just as wrong as the 32767 - write a record of length 16k into that file and the RMS value will stay at 100. The C RTL should really just keep track of the records it writes and put the REAL value. Section 1.6 of the Compaq C Run-Time Library Reference Manual is the only place this logical is mentioned: In OpenVMS Version 7.0 the default LRL value on stream files was changed from 0 to 32767. This change caused significant performance degradation on certain file operations such as sort. This is no longer a problem. The Compaq C Run-Time Library now lets you define the logical DECC$DEFAULT_LRL to change the default record-length value on stream files. The Compaq C Run-Time Library first looks for this logical. If it is found and it translates to a numeric value between 0 and 32767, that value is used for the default LRL. I disagree emphatically that "this is no longer a problem". The value of LRL set by DECC$DEFAULT_LRL is every bit as wrong as the default LRL. Its only advantage is that it can be "small and wrong" rather than "large and wrong."

issue#	*sev/prio*	*status*	*description*	*explanation, status*
31.	I/0	LEF	Where do I obtain tools and libraries for porting?	bash,gmake, and many others are part of GNV http://gnv.sourceforge.net/ qt 2.1 http://www.lehrig.de/service/service2.php xforms 0.88 (warning, bugs in some versions of the C RTL and some graphics drivers are triggered by this software! Includes a port of XPM in shareable format) http://world.std.com/~xforms/ parallel programming: pvm, tcgmsg (maybe MPICH too, but it wasn't there as this was written) ftp://v36.chemie.uni-konstanz.de/ OpenVMS Porting Library from Compaq (September 2003) http://h71000.www7.hp.com/openvms/products/ips/porting.html Many libraries may be found in already completed ports see: http://h71000.www7.hp.com/openvms/freeware/index.html http://h71000.www7.hp.com/freeware/ Use advanced search from http://groups.google.com/ to look for posts in comp.os.vms concerning the package of interest. (OO) The HP Open Source Tools page http://h71000.www7.hp.com/opensource/opensource.htmland, The HP e-business technology page http://h71000.www7.hp.com/ebusiness/technology.html, Jouk Jansen's OpenVMS page http://nchrem.tnw.tudelft.nl/openvms/software2.html and our own sites http://www.4ovms.dyndns.org and http://www.oooovms.dyndns.org to name a few more.

issue#	*sev/prio*	*status*	*description*	*explanation, status*

32.			Use of the fdopen() call after a open() call fails while the same code on unix succeeds.	Depending on the mode-specification in the open() call the following fdopen() call can fail. Unless the mode parameters specify a “b” (binary), the file system assumes that the file isn't opened in binary mode. The following fdopen() will give a NULL back, indicating failure, because fdopen() expects a file in binary mode.
				
				To solve this problem you must add extra parameters to the open() call.
				
				Code fragment from BZIP2:
				
				/* Open an output file safely with O_EXCL and good permissions.
				This avoids a race condition in versions < 1.0.2, in which
				the file was first opened and then had its interim permissions
				set safely. We instead use open() to create the file with
				the interim permissions required. (--- --- rw-).
				
				For non-Unix platforms, if we are not worrying about
				security issues, simple this simply behaves like fopen.
				*/
				FILE* fopen_output_safely (Char* name, const char* mode)
				{
				# if BZ_UNIX
				FILE* fp;
				IntNative fh;
				fh = open(name, O_WRONLY|O_CREAT|O_EXCL, S_IWUSR|S_IRUSR);
				if (fh == -1) return NULL;
				fp = fdopen(fh, mode);
				if (fp == NULL) close(fh);
				return fp;
				# else
				return fopen(name, mode);
				# endif
				}
				
				We added "ctx=bin" and "fop=dfw" to the open() call. "ctx=bin" is the required part for the described problem. "fop=dfw" is added to improve performance. Look at the description in the manual for an explanation of these parameters.
				
				/* Open an output file safely with O_EXCL and good permissions.
				This avoids a race condition in versions < 1.0.2, in which
				the file was first opened and then had its interim permissions
				set safely. We instead use open() to create the file with
				the interim permissions required. (--- --- rw-).
				
				For non-Unix platforms, if we are not worrying about
				security issues, simple this simply behaves like fopen.
				*/
				FILE* fopen_output_safely (Char* name, const char* mode)
				{
				# if BZ_UNIX
				FILE* fp;
				IntNative fh;
				# if VMS
				fh = open(name, O_WRONLY|O_CREAT|O_EXCL, S_IWUSR|S_IRUSR,
				"ctx=bin",
				"fop=dfw");
				# else
				fh = open(name, O_WRONLY|O_CREAT|O_EXCL, S_IWUSR|S_IRUSR);
				# endif
				if (fh == -1) return NULL;
				fp = fdopen(fh, mode);
				if (fp == NULL) close(fh);
				return fp;
				# else
				return fopen(name, mode);
				# endif
				}

issue#	*sev/prio*	*status*	*description*	*explanation, status*
33.	E/3		unknown switch errors from gcc and ar wrappers	Because the gcc and ar executables are 'just' wrappers around the digital compilers and the standard linker and not all switches are implemented (yet), the executables will complain of 'unknown switches'. There are switches that can safely be ignored (for instance the 'u' switch used with ar). To do this permanently you can modify the ar wrapper: old gnu:[src.gnv.wrapper]ar.c: **Code:** <snip> case 'q': case 'r': break; <snip> new ar.c **Code:** <snip> case 'q': case 'r': case 'u': break; <snip> Then there switches that can be replaced with another command: gcc -dumpmachine (Display the compiler's target processor) as long as you figure out what part of the output is needed. Code fragment + output on a Linux machine: $ gcc -dumpmachine | awk -F- '{print $3}' Linux On OpenVMS you can replace this with: bash$ uname OpenVMS or bash$ uname -s OpenVMS A list of possible gcc switch replacements: -Wall -Wc/Warn -funsigned-char -Wc/UNSIGNED_CHAR The last category is the real error which has consequences for the compile or link. In this category fall -pipe and the various -print switches

Appendix 3: adduser.com

Procedure to add a number of users. This procedure must be run by a privileged account (system). Define a logical (define user dsa12:) if you want the user accounts created on an other disk as the system disk.

```
$!
$! p1 last UID value used
$! p2 number of user accounts to create
$!
$ if p1.eqs."" then p1 = 10 ! last UID value used
$ if p2.eqs."" then p2 = 16 ! count
$!
$ if f$trnlnm("user") .eqs ""
$ then
$       disk=f$trnlnm("sys$sysdevice")
$ else
$       disk=f$trnlnm("user")
$endif
$!       sh sym disk
$       create/dir 'disk'[user]
$       define/system/exec/trans=conc users 'disk'[user.]
$       create sys$manager:temp1.tmp
$DECK
$! login.com for OpenOffice portingroup member
$
$ set term/dev=vt300
$ set term/line/insert
$! start gnv
$ @GNU:[lib]GNV_SETUP.COM
$!
$! setup tools
$ set process /parse_style=extended
$ set process /case_lookup=(blind)
$ define/job decc$pipe_buffer_size 65000
$
$
$ scratch = f$trnlnm("sys$login") - "]" + ".temp]"
$ define/job sys$scratch 'scratch'
$!
$ exit
$!
$EOD
$       create sys$manager:.tmp
$DECK
# .bashrc
#
PATH=$PATH:/usr/bin:/usr/local/bin
export PATH
export GNV_DISABLE_DCL_FALLBACK=1
$EOD
$       open/write out loop.tmp
$       write out "$! generated procedure for account creation"
$       write out "$!"
$       write out "$ set def sys$system"
$       write out "$ define/sys/exec/trans=conc users ''disk'[user.]"
$       write out "$!"
$ x = p1
$ i = 0
$loop:
$ x = f$integer(f$fao("!OL", %o'x' +1))    ! Magic
$ i = i + 1
$ write sys$output i, " ", x
```

```
$       write out "$    mcr authorize"
$       write out "add
user''i'/uic=[237,''x']/dev=users:/dir=[user''i']/passw=user''i'/flag=nodisuser/n
opwdexp"
$       write out "$    create /dir users:[user''i']/own=user''i'"
$       write out "$    create /dir users:[user''i'.temp]/own=user''i'"
$       write out "$    copy/log sys$manager:temp1.tmp
users:[user''i']login.com"
$       write out "$    copy/log sys$manager:.tmp  users:[user''i'].bashrc"
$       write out "$!"
$ if i.lt.p2 then goto loop
$       write out "$! cleaning up a bit..."
$       write out "$ delete/noconf sys$manager:temp1.tmp;*"
$       write out "$ delete/noconf sys$manager:.tmp;* "
$       write out "$ exit"
$eof: close out
$!
$       exit
```

Appendix 4: OpenVMS – UNIX comparison

UNIX	*VMS*
root (privileged user)	SYSTEM or any username with sufficient (all?) privileges. Every user can be equipped with a set of privileges and/or process rights identifiers needed for the required task
hierarchical file-system directory paths are separated by the / symbol exact paths start with / a relative path can start with a directory name or with .. which means go one directory back and then go to the given directory path the directory . is the current directory / (root) /bin /etc /usr /dev /tmp /var	device oriented file system all absolute path names start with a node name, next a device name or logical name followed by a dot separated path between angular brackets logical:[dir.subdir.subsubdir] originally a directory tree could only be 8 levels deep the root directory is always logical:[000000] a relative path is a dot separated path starting with a “.” or a “-” between angular brackets [.dir.subdir] [-.dir.subdir] with one or more – within the angular brackets you specify you want to go one or more directory back in the directory structure a logical can represent a device, both actual devices, points in the file system we want to represent as devices (concealed), paths, path lists or other values or value lists SYS$SYSDEVICE: is an example of a device logical. It is the device holding the system software SYS$COMMON: is an example of a path that can be used as a device (you can do DIR SYS$COMMON:[000000]) SYS$SYSROOT: is an example of a path list SYS$ANNOUNCE: is an example of a simple value with ... at the end of a path you specify all subdirectories. You can use % as single character wild card character and * for multiple character wild card characters both in filenames and in file paths. XX%%YY.* *SMTP*.COM

UNIX	*VMS*
Numerous file systems: ufs (the original UNIX file system), ext, ext2, ext3 (Linux), adfs (True64), efs(IRIX), hfs, s5 (HP-UX), jfs (AIX) and let's not forget nfs.	The most important ones for this document are ODS-2 and ODS-5. ODS-5 is rather new and is a requirement for UNIX portability.
Symbolic links	Hard links or Soft links Hard links were introduced in 7.3-1 and only work when enabled on the disk. The advantage over soft links is that a reference count is maintained, to make sure that the file doesn't get deleted until the last link is deleted.
pwd	SHOW DEFAULT
df	SHOW DEV D
cd	SET DEFAULT
/bin $HOME or ~ $PATH	SYS$SYSTEM: SYS$LOGIN: DCL$PATH:
environment variables	symbols & logicals symbols are quite similar to UNIX environment variables logicals can also contain values but they are not necessarily limited to the scope of your process or job. Logicals are organized in tables that can be seen cluster wide, system wide, group wide, process wide or job specific. You can even create your own logical name tables.

case sensitive	OpenVMS is case insensitive by default. Since version 7.3 you can make your process behave case preservative on ODS-5 disks by enabling extended filename parsing. Since version 7.3-1 you can also make your process behave case sensitive on ODS-5 disks.
	`SET PROCESS /PARSE_STYLE=(TRADITIONAL/EXTENDED)`
	`SET PROCESS /CASE_LOOKUP=(BLIND/SENSITIVE)`

character devices, block devices	mailbox (qio/qiow) RMS ...

UNIX	*VMS*

/dev/..., makedev, mount, fstab.	VMS recognizes all new devices at boot time and automatically loads the necessary drivers.
	It is possible to load drivers and add new devices to a running system. OpenVMS 7.3-2 even allows you to enlarge disk volumes dynamically.
	To use a new disk you can use the following command to initialize it:
	`$ INITIALIZE <device> <label> - /NOHIGHWATER /STRUCTURE=5 - /VOLUME_CHARACTERISTICS= - (HARDLINKS,ACCESS_DATES)`
	Mount the disk disk with the following command:
	`$ MOUNT <device> <label> /SYSTEM - /NOASSIST`
	To mount a disk every time when the system boots add the above command to SYS$MANAGER:SYSTARTUP_VMS.COM.

sh, csh, tcsh, bash, ksh, zsh, ...	DCL
vi	EDIT (/TPU is the default)
/etc/profile	SYS$MANAGER:SYLOGIN.COM
~/.profile (for sh, csh, tcsh) ~/.bashrc (for bash shell)	SYS$LOGIN:LOGIN.COM
chmod +x to make a file executable	To be able to execute a executable or procedure one has to have at least execute access. No equivalent, see below
.<filename> will execute an executable file (script or program) when not in your PATH environment	@ to execute a DCL script RUN to execute a program not somehow known to your runtime environment. <command> to execute a program known to your runtime environment. There are various ways to extend your runtime environment. 1. define a symbol that points to the program to execute i.e. UNZIP :== SYSLOGIN:[TOOLS]UNZIP.EXE. UNZIP is now a command known to your environment. 2. add the directory containing the executable to your DCL$PATH i.e. DEFINE DCL$PATH SYS$LOGIN:[TOOLS],'F$TRNLNM("DCL$PATH")'. Now every executable program in the directory SYS$LOGIN:[TOOLS] is available as a command in your runtime environment. 3. Add a command definition to to your environment i.e. SET COMMAND.