VMS Software

VSI OpenVMS

VSI OpenVMS Guide to Extended File Specifications

Operating System and Version: VSI OpenVMS IA-64 Version 8.4-1H1 or higher
VSI OpenVMS Alpha Version 8.4-2L1 or higher

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

VSI OpenVMS Guide to Extended File Specifications

Legal Notice

Preface vii
1. About VSI vii
2. Intended Audience vii
3. Document Structure vii
4. Related Documents vii
5. OpenVMS Documentation viii
6. VSI Encourages Your Comments viii
7. Conventions viii
Chapter 1. Overview of Extended File Specifications for OpenVMS 1
1.1. Benefits of Extended File Specifications 1
1.2. Features of Extended File Specifications 1
1.2.1. ODS-5 Volume Structure 2
1.2.1.1. Long File Names 2
1.2.1.2. More Characters Legal Within File Names 2
1.2.1.3. Preservation of Case 2
1.2.2. Deep Directory Structures 3
1.2.2.1. Directory Naming Syntax 3
1.3. Considerations Before Enabling ODS-5 Volumes 3
1.3.1. Considerations for System Management 4
1.3.2. Considerations for Users 4
1.3.2.1. Mixed-Version Support 4
1.3.3. Record Management Services Changes 5
1.3.3.1. Extended File Specification Support 5
1.3.3.2. Mixed-Architecture Support 6
1.3.4. Considerations for Applications 6
1.4. Recommendations for Using Extended File Specifications on OpenVMS Applications 7
Chapter 2. Managing Extended File Naming on OpenVMS Systems 9
2.1. Levels of Support for Extended File Specifications 9
2.1.1. Full Support 9
2.1.2. Default Support 9
2.1.3. No Support for Extended File Naming 10
2.1.4. No Support for ODS-5 10
2.2. Enabling Extended File Specifications on OpenVMS Alpha Systems 11
2.2.1. Using RMS Default Extended File Specifications Features 11
2.2.2. Enabling ODS-5 Volumes 11
2.2.2.1. Initializing a New ODS-5 Volume 12
2.2.2.2. Converting an Existing Volume to ODS-5 12
2.2.3. Converting from ODS-5 to ODS-2 14
2.3. Controlling Access to ODS-5 Volumes 17
2.3.1. Preventing VAX Users from Accessing an ODS-5 Volume 17
2.3.2. Preventing an Untested Application from Accessing an ODS-5 Volume 18
2.4. System Management Utility Changes 19
2.4.1. Analyze/Disk_Structure Utility 19
2.4.2. Backup Utility (Alpha Only) 19
2.4.3. Physical Backups of ODS-5 Volumes on VAX Systems 20
2.4.4. Mount Utility (Alpha Only) 20
Chapter 3. Extended File Naming Characteristics 21
3.1. File Specifications 21
3.1.1. Traditional (ODS-2) Syntax 21
3.1.2. Extended (ODS-5) Syntax 22

3.1.2.1. ISO Latin-1 Character Set 22
3.1.2.2. Special Characters 22
3.1.2.3. Interpretation of Period (.) 23
3.1.2.4. Expanded File Specification Length 23
3.1.2.5. Using Wildcards 24
3.1.2.6. Case Preservation 25
3.2. Directory Specifications 25
3.2.1. Deep Directory Structures 25
3.2.2. Directory Naming Syntax 26
3.2.2.1. Directory ID and File ID Abbreviation 26
3.3. Working in Mixed Environments 26
3.4. DCL Support for ODS-5 Volumes 26
3.4.1. Using the Extended File Specifications Parsing Feature in DCL 27
3.4.1.1. Enabling the Extended File Name Parsing Style 27
3.4.1.2. Resetting the Default File Name Parsing Style 27
3.4.1.3. Switching Between File Name Parsing Styles 27
3.4.2. Using Extended File Names in DCL Command Parameters 28
3.4.3. Command Procedure File Specification 28
3.4.4. Case Preservation and $FILE 29
3.4.5. Ampersand Versus Apostrophe Substitution 29
3.5. DCL Commands and Utilities 31
3.6. System Services Changes 33
3.6.1. New Services 33
3.6.2. Changed Services 39
3.6.2.1. $GETJPI System Service 39
3.6.2.2. $CREPRC System Service 39
3.6.2.3. $SETDDIR System Service 39
3.7. Displaying Extended File Names on a Terminal 40

Chapter 4. Extended File Naming Considerations for OpenVMS Application Developers 41

4.1. Evaluating Your Current Support Status 41
4.1.1. Default Support 41
4.1.2. No Support for Extended File Names 41
4.1.3. No Support for ODS-5 Volumes 42
4.2. Upgrading an Application to Support Extended File Specifications 42
4.2.1. Upgrading to Default Support 42
4.2.1.1. Providing Support for ODS-5 42
4.2.1.2. Providing Support for Extended File Naming 43
4.2.2. Upgrading to Full Support 44

Appendix A. Setting Users' Expectations of Extended File Specifications 45

A.1. New Extended File Specifications Characteristics 45
A.2. ODS-2 and ODS-5 Used Together 48
A.3. Architecture-Related Notes 51
A.4. Restrictions 52

Appendix B. Technical Information 53

B.1. System Services Changes 53
B.1.1. New Services 53
B.1.2. Changed Services 59
B.1.2.1. $GETJPI System Service 59
B.1.2.2. $CREPRC System Service 59
B.1.2.3. $SETDDIR System Service 59

B.2. Record Management Services (RMS) Changes 59
B.2.1. Overview of Record Management Services Changes 60
B.2.1.1. Extended File Specification Support 60
B.2.1.2. Additional Characters 60
B.2.1.3. Deeply Nested Directory Support 60
B.2.2. Syntax and Semantics Changes 61
B.2.2.1. Use of Hyphen as First File Name Character 61
B.2.2.2. Characters Accepted Directly 61
B.2.2.3. Characters That Require an Escape Character 61
B.2.2.4. Characters That Can Have an Escape Character 62
B.2.2.5. Reserved Escape Sequences 62
B.2.2.6. Canonical Form of File Specifications 62
B.2.2.7. DID Abbreviation 63
B.2.2.8. FID Abbreviation 63
B.2.3. RMS Data Structure Changes (Alpha Only) 64
B.2.3.1. NAM Block 64
B.2.3.2. NAML Block 65
B.3. Files-11 XQP Changes 70
B.3.1. File Naming and Format Changes 71
B.3.1.1. Specifying the Format of the Input File Name 71
B.3.1.2. Controlling the Format of Returned File Names 72
B.3.1.3. Wildcard Searches and Pseudonames 73
B.3.1.4. Compatibility with Unchanged Applications 74
B.3.2. File Attribute Changes 74
B.3.2.1. Modified File Attributes 74
B.4. Programming Utility Changes 76
B.4.1. File Definition Language (FDL) Routines 76
B.4.1.1. FDL$CREATE Routine (Alpha Only) 77
B.4.1.2. FDL$GENERATE Routine (Alpha Only) 77
B.4.1.3. FDL$PARSE Routine (Alpha Only) 77
B.4.1.4. FDL$RELEASE Routine (Alpha Only) 78
B.5. Run-Time Library Changes 78
B.5.1. LIB$CREATE_DIR 78
B.5.2. LIB$DELETE_FILE 78
B.5.3. LIB$FILE_SCAN 79
B.5.4. LIB$FIND_FILE 79
B.5.5. LIB$RENAME_FILE 79
B.5.6. LIB$FID_TO_NAME 80
Appendix C. Character Sets 81

Preface

This document provides an overview of Extended File Specifications and describes the impact of Extended File Specifications on system managers, application developers, and users of the traditional OpenVMS environment.

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. Intended Audience

This document is intended for system managers, application developers, and users who implement Extended File Specifications on one or more systems in an OpenVMS environment.

3. Document Structure

This manual consists of the following chapters and appendixes:

- Chapter 1 provides an overview of Extended File Specifications and its features.
- Chapter 2 describes the changes visible to OpenVMS system managers, provides instructions on how to enable and control user access to ODS-5 volumes, and describes the impact on functions such as backing up and restoring media.
- Chapter 3 describes the changes visible to OpenVMS users when using ODS-5 volumes.
- Chapter 4 describes how to evaluate the support for Extended File Specifications of OpenVMS applications.
- Appendix A contains guidelines for setting users' expectations about using the features of Extended File Specifications.
- Appendix B contains detailed technical information about the changes to the OpenVMS programming interface to support Extended File Specifications. Much of this material appears in other documents in the OpenVMS documentation.
- Appendix C describes the DEC Multinational character set and the ISO Latin-1 character set.

4. Related Documents

For related information about Extended File Specifications, see the following documents:

- *VSI OpenVMS Guide to OpenVMS File Applications*
- *VSI OpenVMS DCL Dictionary: A-M*
- *VSI OpenVMS DCL Dictionary: N-Z*
- *VSI OpenVMS RTL Library (LIB$) Manual*

- *VSI OpenVMS Record Management Services Reference Manual*
- *VSI OpenVMS System Manager's Manual, Volume 1: Essentials*
- *VSI OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems*
- *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L*
- *VSI OpenVMS System Management Utilities Reference Manual, Volume 2: M-Z*
- *VSI OpenVMS System Services Reference Manual: A-GETUAI*
- *VSI OpenVMS System Services Reference Manual: GETUTC-Z*
- *VSI OpenVMS Utility Routines Manual*

5. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

6. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: <docinfo@vmssoftware.com>. Users who have VSI OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product.

7. Conventions

The following conventions may be used in this manual:

Convention	Meaning
Ctrl/ *x*	A sequence such as **Ctrl/** *x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that you must enclose the options in parentheses if you choose more than one.

Convention	Meaning
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.
[|]	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are options; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold text	This typeface represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER= *name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Monospace type`	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

Chapter 1. Overview of Extended File Specifications for OpenVMS

OpenVMS Version 7.2 on Alpha implements Extended File Specifications, which consists of two major components:

- A new, optional, volume structure, ODS-5, which provides support for file names that are longer and have a greater range of legal characters than in previous versions of OpenVMS
- Support for deep directories

Taken together, these components provide much greater flexibility for OpenVMS Alpha systems (using *Advanced Server for OpenVMS 7.2*, formerly known as PATHWORKS for OpenVMS), to store, manage, serve, and access files that have names similar to those in a Windows 95/98 or Windows NT environment.

This chapter provides a brief overview of the benefits, features, and support for Extended File Specifications, as well as changes in OpenVMS behavior that occur under Extended File Specifications.

1.1. Benefits of Extended File Specifications

The deep directories and extended file names supported by Extended File Specifications provide the following benefits:

- Users of *Advanced Server for OpenVMS 7.2* (formerly known as PATHWORKS for OpenVMS) have the ability to store longer file names, preserve the case of file names, and use deeper directory structures. These new capabilities make the use of an OpenVMS file server more transparent to Windows 95/98 and Windows NT users.
- OpenVMS system managers can see files on OpenVMS systems with the names as specified by Windows 95/98 and Windows NT users.
- Applications developers who are porting applications from other environments that have support for deep directories can use a parallel structure on OpenVMS.
- Longer file naming capabilities and Unicode support enables OpenVMS Version 7.2 to act as a DCOM server for Windows NT clients, and ODS-5 provides capabilities that make the OpenVMS and Windows NT environment more homogeneous for DCOM developers.
- JAVA applications on OpenVMS will comply with JAVA object naming standards.
- General OpenVMS users can make use of long file names, new character support, and the ability to have lowercase and mixed-case file names.

These benefits result from the features described in Section 1.2.

1.2. Features of Extended File Specifications

Extended File Specifications consists of two main features, the ODS-5 volume structure, and support for deep directories. These features are described in the sections that follow.

1.2.1. ODS-5 Volume Structure

OpenVMS Version 7.2 implements On-Disk Structure Level 5 (ODS-5). This structure provides the basis for creating and storing files with extended file names. You can choose whether or not to convert a volume to ODS-5 on your OpenVMS Alpha systems.

The ODS-5 volume structure allows the following features:

- Long file names
- More characters legal within file names
- Preservation of case within file names

These features are described in the sections that follow.

1.2.1.1. Long File Names

On an ODS-5 volume, the name of a file (excluding the version number) can be up to 236 8-bit or 118 16-bit characters long. Complete file specifications longer than 255 bytes are abbreviated by RMS when presented to unmodified applications.

For more information on extended file names, see Section 3.1.2.

1.2.1.2. More Characters Legal Within File Names

A broader set of characters is available for naming files on OpenVMS. ODS-5 offers support for file names that use the 8-bit ISO Latin-1 character and 16-bit Unicode (UCS-2) character sets.

ISO LATIN-1 and Unicode (UCS-2) Character Sets

The ISO Latin-1 Multinational character set is a superset of the traditional ASCII character set used by versions of OpenVMS previous to Version 7.2. In extended file specifications, all characters from the 8-bit ISO Latin-1 Multinational character set are valid in file specifications, *except* the following:

C0 control codes (0x00 to 0x1F inclusive)
Double quotation marks (")
Asterisk (*)
Backslash (\)
Colon (:)
Left and right angle brackets (< >)
Slash (/)
Question mark (?)
Vertical bar (|)

To unambiguously enter or display certain special characters in an ODS-5 compliant file specification, such as a space, you must precede the character with a circumflex (^).

For more information on how these character sets are used in file names, see Section 3.1.2.

1.2.1.3. Preservation of Case

In prior versions of OpenVMS, DCL, RMS, and the file system converted all file specifications to uppercase. ODS-5 preserves the case of file specifications. For example:

```
$ CREATE x.Y
[Ctrl/Z]
$DIRECTORY

Directory  DISK1:[USER1]

x.Y;1

$
```

As you can see, the mixed-case of the file name is preserved.

For more information on case sensitivity, see Section 3.1.2.6.

1.2.2. Deep Directory Structures

Both ODS-2 and ODS-5 volume structures support deep nesting of directories, subject to the following limits:

- There can be up to 255 levels of directories.
- The name of each directory can be up to 236 8-bit or 118 16-bit characters long.

For example, a user can create the following deeply nested directory:

```
$ CREATE/DIRECTORY [.a.b.c.d.e.f.g.h.i.j.k.l.m]
```

A user can create the following directory with a long name on an ODS-5 volume:

```
$ CREATE/DIRECTORY
[.AVeryLongDirectoryNameWhichHasNothingToDoWithAnythingInParticular]
```

Complete file specifications longer than 255 bytes are abbreviated by RMS when presented to unmodified applications.

1.2.2.1. Directory Naming Syntax

On an ODS-5 volume, directory names conform to most of the same conventions as file names when using the ISO Latin-1 character set. Periods and special characters can be present in the directory name, but in some cases, they must be preceded by a circumflex (^) in order to be recognized as literal characters.

Section 3.2 contains more information about deep directories. $SET_PROCESS_PROPERTIESW System Service (Alpha Only) contains information about displaying long directory names.

1.3. Considerations Before Enabling ODS-5 Volumes

ODS-5 is being introduced primarily to provide enhanced file sharing capabilities for users of *Advanced Server for OpenVMS 7.2* (formerly known as PATHWORKS for OpenVMS), as well as DCOM and JAVA applications.

Once ODS-5 volumes are enabled, some of the new capabilities can potentially impact certain applications or layered products, as well as some areas of system management. The new syntax for file names that is allowed on ODS-5 volumes cannot be fully utilized on ODS-2 volumes. Because pre-

Version 7.2 Alpha systems cannot access ODS-5 volumes, and Open VMS Version 7.2 VAX systems have limited ODS-5 functionality, you must be careful where and how you enable ODS-5 volumes in mixed-version and mixed-architecture OpenVMS Clusters.

The following sections comprise a summary of how enabling ODS-5 volumes can impact system management, users, and applications.

1.3.1. Considerations for System Management

RMS access to deep directories and extended file names is available only on ODS-5 volumes mounted on OpenVMS Alpha V7.2 systems. VSI recommends that ODS-5 volumes be enabled only on a homogeneous OpenVMS Alpha V7.2 Cluster.

If ODS-5 is enabled in a mixed-version or mixed-architecture OpenVMS Cluster, the system manager must follow special procedures and be aware of specific restrictions on mixed-version and mixed-architecture OpenVMS Clusters with ODS-5 volumes enabled:

- Users must access ODS-5 files and deep directories from OpenVMS Alpha V7.2 systems only, because these capabilities are not supported on earlier versions.
- Users who have created deep directories can view those directories only from OpenVMS Alpha V7.2 systems.
- Pre-Version 7.2 systems cannot mount an ODS-5 volume nor read ODS-2 or ODS-5 file names on that volume.

Section 1.3.2 describes in greater detail the limitations of ODS-5 support for users in a mixed-version or mixed-architecture OpenVMS Cluster.

Most unprivileged applications will work with most extended file names, but some may need modifications to work with all extended file names. Privileged applications that use physical or logical I/O to disk and applications that have a specific need to access ODS-5 file names or volumes may require modifications and should be analyzed. Section 1.3.4 describes in greater detail the impact of ODS-5 on OpenVMS applications.

Chapter 2 contains more information for determining the levels of support for Extended File Specifications, and guidelines for managing a system with ODS-5 volumes enabled.

1.3.2. Considerations for Users

A user on an OpenVMS Alpha Version 7.2 system can take advantage of all Extended File Specifications capabilities on ODS-5 volumes mounted on an OpenVMS Alpha Version 7.2 system.

A user on a mixed-version or mixed-architecture OpenVMS Cluster is subject to some limitations in ODS-5 functionality. Section 1.3.2.1 lists those restrictions that exist on a mixed-version OpenVMS Cluster. Section 1.3.3.2 lists those restrictions that exist on a mixed-architecture OpenVMS Cluster.

1.3.2.1. Mixed-Version Support

Systems running prior versions of OpenVMS cannot mount ODS-5 volumes, correctly handle extended file names, or even see extended file names.

The following sections describe support on OpenVMS Version 7.2 and on prior versions of OpenVMS in a mixed-version cluster.

Users on OpenVMS Alpha Version 7.2 Systems

A user on an OpenVMS Alpha Version 7.2 system can continue to access pre-Version 7.2 files and directories; for example, a user can do all of the following:

- Create and access deep directory structures on ODS-2 volumes.
- Read a BACKUP saveset created on an earlier version of OpenVMS.
- Use DECnet to copy a file with an ODS-5 name to a file with an ODS-2 name on a system running an earlier version of OpenVMS.

Users on Pre-Version 7.2 Systems

On mixed-version clusters, some restrictions exist. Users on a version of OpenVMS prior to Version 7.2:

- Cannot access any files on an ODS-5 volume. This is true regardless of whether the volume is connected physically on a CI or SCSI bus, or by an MSCP or QIO server.
- Assumptions that file specifications are identical between RMS and the file system.

Note

All unmodified XQP applications running on an OpenVMS VAX or Alpha system that access an ODS-5 volume will see pseudonames returned in place of Unicode or ISO Latin-1 names that are not ODS-2 compliant. This can cause applications to act in an unpredictable manner.

Applications that specify or retrieve filenames with the XQP interface using ODS-5 disks must be modified in order to access files with extended names.

1.3.3. Record Management Services Changes

To support HFS, the Record Management Services (RMS) have been enhanced to provide the following functions through existing interfaces:

- Support for a wider range of characters in a file name, extension, and directory
- Access to file specifications with extended characters
- Support for directory structures deeper than eight levels
- Access to file specifications longer than 255 bytes through the NAM block with some restrictions in functionality
- Access and complete specification of file specifications longer than 255 bytes by callers who are aware of the new naming characteristics through a new interface (NAML block)

For more information about using RMS in an HFS environment, see Section B.2.

1.3.3.1. Extended File Specification Support

With ODS-5, RMS can manipulate filenames and subdirectory specifications of up to 255 8-bit or 16-bit characters in length. RMS can handle a total path name 512 8-bit or 16-bit characters in length.

Prior to OpenVMS Alpha Version 7.2, the NAM block interface could pass file specifications of up to 255 bytes each (including the resultant file specification). The following Cannot successfully create or restore an ODS-5 image saveset. However, these users can successfully restore ODS-2-compliant file names from an ODS-5 saveset.

1.3.3.2. Mixed-Architecture Support

Current ODS-2 volume and file management functions remain the same on both VAX and Alpha Version 7.2 systems; however, extended file naming and parsing are not available on VAX systems.

The following sections describe support on OpenVMS VAX and Alpha systems in a mixed-architecture cluster.

Limited Extended File Specifications Capabilities on VAX Systems

In mixed-architecture OpenVMS Version 7.2 clusters, OpenVMS Version 7.2 VAX systems are limited to the following Extended File Specifications functionality:

- Ability to mount an ODS-5 volume
- Ability to write and manage ODS-2-compliant files on an ODS-5 volume
- See pseudonames (\pISO_LATIN\.??? or \pUNICODE\.???) when accessing an ODS-5 file specification

BACKUP Limitations

From a VAX system, users cannot successfully create or restore an ODS-5 image saveset. However, these users can successfully restore ODS-2-compliant file names from an ODS-5 saveset.

1.3.4. Considerations for Applications

ODS-5 functionality can be selected on a volume-by-volume basis. If ODS-5 volumes have not been enabled on your system, all existing applications will continue to function as before. If ODS-5 volumes have been enabled, you need to be aware of the following changes:

- OpenVMS file handling and command line parsing have been modified to enable them to work with extended file names on ODS-5 volumes while still being compatible with existing applications. The majority of existing, unprivileged applications will work with most extended file names, but some may need modifications to work with all extended file names.
- Privileged applications that use physical or logical I/O to disk may require modifications and should be analyzed. Applications that have a specific need to access ODS-5 file names or volumes should be analyzed to determine if they require modification.

On ODS-5 volumes, existing applications and layered products that are coded to documented interfaces, as well as most DCL command procedures, should continue to work without modification.

However, applications that are coded to undocumented interfaces, or include any of the following, may need to be modified in order to function as expected on an ODS-5 volume:

- Internal knowledge of the file system, including knowledge of:

  Double quotation marks (")
  Asterisk (*)

Backslash (\)
Colon (:)
Left and right angle brackets (< >)
Slash (/)
Question mark (?)
Vertical bar (|)

Note that this explicitly includes both the C1 character set (hex 80-9F) as well as graphical and other characters between 9F and FF. This allows the entire ISO Latin-1 character set (with the 7-bit character exclusions noted above) and any defined Unicode character.

Note

Information about Unicode TBS.

- The data layout on disk
- The contents of file headers
- The contents of directory files
- File parsing tailored to a particular on-disk structure.
- Assumptions about the syntax of file specifications, such as the placement of delimiters and legal characters.
- Assumptions about the case of file specifications. Mixed and lowercase file specifications will not be converted to uppercase, which can affect string matching operations.
- Assumptions that file specifications are identical between RMS and the file system.

Note

All unmodified XQP applications running on an OpenVMS VAX or Alpha system that access an ODS-5 volume will see pseudonames returned in place of Unicode or ISO Latin-1 names that are not ODS-2 compliant. This can cause applications to act in an unpredictable manner.

Applications that specify or retrieve filenames with the XQP interface using ODS-5 disks must be modified in order to access files with extended names.

See Chapter 4 for further discussion of the support status of OpenVMS applications.

1.4. Recommendations for Using Extended File Specifications on OpenVMS Applications

It is essential that system managers perform the following steps before enabling ODS-5:

- Review all ODS-5 considerations.
- Understand the support levels for different OpenVMS applications.
- Segregate applications that do not support ODS-5 or have not been tested with ODS-5 names or volumes.

- Review the guidelines for setting users' expectations in Appendix A.

Note

VSI recommends that you enable ODS-5 disks in a homogeneous OpenVMS Version 7.2 Alpha cluster only.

Chapter 2. Managing Extended File Naming on OpenVMS Systems

Managing an OpenVMS system with Extended File Specifications requires an understanding of the support provided by different OpenVMS applications, how to enable and control the new environment, and the changes to OpenVMS system management utilities. This chapter contains the following topics:

- Levels of support provided by the current set of OpenVMS commands and utilities that support Extended File Specifications
- How to enable Extended File Specifications on an OpenVMS Alpha system
- How to control user access to ODS-5 volumes
- Changes to system management utilities

2.1. Levels of Support for Extended File Specifications

To help determine the expected behavior of OpenVMS utilities and commands for ODS-5, the following levels of support have been established. Each level outlines the acceptable behavior of a utility or command when it encounters an extended (ODS-5 compliant) file specification.

The levels of support for ODS-5, from full support to no support, are defined in Sections 2.1.1 through 2.1.4.

2.1.1. Full Support

OpenVMS utilities and commands that offer full support for ODS-5 have been specifically modified to take advantage of all the features of extended file naming. These utilities and commands should accept and handle extended file specifications without error while maintaining the case as created.[1]

In addition, OpenVMS commands and utilities that fully support Extended File Specifications can accept and produce long file specifications that exceed the traditional 255-byte limit in their original form[2]—without requiring them to be abbreviated in Directory ID (DID) or File ID (FID) format. For the list of OpenVMS components that fully support Extended File Specifications, see Section 3.4.

2.1.2. Default Support

OpenVMS utilities and commands with default support have had little or no modification to take advantage of Extended File Specifications. These utilities and commands are expected to handle most of the attributes of extended file specifications (such as new characters and deep directory structures) correctly. However, file names may be created or displayed with the wrong case.

[1] When creating the first version of a new file, the case of the new file matches that case specified by the user. When creating subsequent versions of an existing file, the case remains the same as the original version.
[2] If you are typing a long file specification on a DCL command line, DCL still limits the command line length to 255 bytes.

In contrast with utilities that have full support, utilities with default support rely on DID and FID abbreviation offered by RMS to handle long file specifications. As a result, these utilities are subject to the following restrictions related to DID and FID abbreviation:

- Matching operations in an environment where FID abbreviation is used may not always work as expected. For example, wildcard matching operations may not capture all target file names because the long file names may be represented in their numeric FID-abbreviated form. This restriction specifically applies to matching operations that are performed outside of RMS.
- Wildcards and sticky defaults cannot be used with a FID abbreviation. For example, the following commands are illegal:

  ```
  $ DIRECTORY a[1,2,3]*.txt $ COPY a[1,2,3].txt *.txt2
  ```

  Because a FID abbreviation is a unique numeric representation of one file, it cannot be used to represent or match any other file.
- Creating a file using a FID abbreviation is illegal.

For more information about DID abbreviations, see Section B.2.2.7. For more information about FID abbreviations, see Section B.2.2.8.

2.1.3. No Support for Extended File Naming

OpenVMS utilities and commands that do not support extended file names can function on ODS-5 volumes; however, they are restricted to operating with traditional file specifications only. These utilities and commands should be used carefully on ODS-5 volumes because VSI cannot ensure that they will function successfully when they encounter extended file specifications.

Table 2.1 lists the OpenVMS utilities and commands that do not support Extended File Specifications because of limitations with either handling extended file names or the ODS-5 volume structure.

2.1.4. No Support for ODS-5

OpenVMS utilities and commands that do not support the ODS-5 volume structure cannot handle extended file names. These utilities and commands should be used carefully on ODS-5 volumes because VSI cannot ensure that they will function successfully even when they only encounter traditional file specifications.

Table 2.1 lists the OpenVMS utilities and commands that do not support Extended File Specifications because of limitations with either handling extended file names or the ODS-5 volume structure.

Table 2.1. Non-Supported OpenVMS Components

Component	Notes
No ODS-5 Support	
Disk defragmenters	Unsupported unless a specific defragmentation tool documents that it has been updated to support an ODS-5 volume.[1]
System disk	Do not set to or initialize as an ODS-5 volume.
No Extended File Naming Support	
Code compilers	Cannot use extended file names for object files. However, code compilers can create applications that support extended names.

Component	Notes
INSTALL Known images	Do not install an image with an extended file name as a known image.
LINK	Cannot output an image with an extended file name.
MONITOR	Cannot reliably process extended file names.
Network files (NET*.DAT)	Do not rename to an extended file name.
Object modules (.OBJ)	Do not rename to an extended file name.
Page and swap files	Do not use an extended file name.
SYSGEN	Do not write a parameter file with an extended file name.
System startup files	Do not rename to an extended file name.

[1]Note that DFO has been modified to support ODS-5 volumes.

2.2. Enabling Extended File Specifications on OpenVMS Alpha Systems

Sections 2.2.1, 2.2.2, and 2.2.3 explain how to take advantage of Extended File Specifications on OpenVMS systems.

Note

Extended File Specifications is not available on systems running versions of OpenVMS Alpha prior to Version 7.2. On these systems, you cannot mount ODS-5 volumes nor take advantage of extended file names on an OpenVMS file system.

2.2.1. Using RMS Default Extended File Specifications Features

RMS allows you to use directory levels deeper than 8, as well as the new RMS API extensions on both ODS-2 and ODS-5 volumes. However, you can create extended file names only on ODS-5 volumes. Section 2.2.2 contains procedures for creating new ODS-5 volumes and for converting ODS-2 volumes to ODS-5 volumes.

On ODS-5 volumes, you – and also a program – can create a file with an extended name on an ODS-5 volume. However, by default, DCL (as well as some applications) does not accept all extended file names and capitalizes any lowercase file names entered on the command line. For DCL to accept all extended file names, you must enable the extended parsing style for DCL, as explained in Section 3.4.1.

Section B.2 contains detailed information about RMS Extended File Specifications features.

2.2.2. Enabling ODS-5 Volumes

To create an ODS-5 volume on an OpenVMS Alpha system, the system manager must do either of the following:

- Initialize a new volume as ODS-5

- Convert an existing volume from ODS-2 to ODS-5

Creating ODS-5 volumes allows you to take advantage of ODS-5 attributes for *Advanced Server for OpenVMS 7.2* (formerly known as PATHWORKS) clients; you can see and manage these attributes from OpenVMS.

Section 2.2.2.1 contains instructions for initializing a new ODS-5 volume. Section 2.2.2.2 contains instructions for converting an existing volume to ODS-5.

Note

If you plan to add a new volume to a volume set, the structure level of the new volume must match that of the volume set. If it does not, the Mount utility displays the following error message:

```
Structure level on device ... is inconsistent with volume set
```

2.2.2.1. Initializing a New ODS-5 Volume

You can initialize a new volume as an ODS-5 volume by issuing the INITIALIZE command in the following format. Note that once you initialize the volume, the current contents of the volume are lost.

```
$ INITIALIZE /STRUCTURE_LEVEL=5 device-name volume-label
```

For example:

```
$ INITIALIZE /STRUCTURE_LEVEL=5 DKA300: DISK1
$ MOUNT DKA300: DISK1 /SYSTEM
%MOUNT-I-MOUNTED, DISK1 mounted on _STAR$DKA300:
```

The first command initializes the DKA300: device as an ODS-5 volume and assigns the volume-label DISK1. The second command mounts the DISK1 volume as a public volume.

To verify that the volume has been initialized ODS-5, you can issue a command and see a display such as the following:

```
$ WRITE SYS$OUTPUT F$GETDVI ("DKA300:","ACPTYPE")
F11V5
```

F11V5 indicates that the volume is ODS-5.

2.2.2.2. Converting an Existing Volume to ODS-5

To convert an existing volume to ODS-5, follow these steps:

1. Dismount the volume throughout the cluster; for example:

   ```
   $ DISMOUNT /CLUSTER DKA300:
   ```

2. Mount the volume as a private volume; for example:

   ```
   $ MOUNT DKA300: DISK1
   %MOUNT-I-MOUNTED, DISK1 mounted on _STAR$DKA300:
   ```

   Omitting the /SYSTEM qualifier causes the system to mount the volume as a private, not a public, volume.

3. You can check that the volume is ODS-2 by issuing a command and seeing a display such as the following:

```
$ WRITE SYS$OUTPUT F$GETDVI ("DKA300:","ACPTYPE")
F11V2
```

F11V2 indicates that the volume is ODS-2.

4. VSI strongly recommends that you back up the volume. You cannot go back to ODS-2 format once you change to ODS-5 except by restoring a backup, as described in Section 2.2.3. For example:

```
$ BACKUP /IMAGE DKA300: SAV.BCK /SAVE_SET
```

5. Set the characteristics of the volume by using a command in the following format:

```
$ SET VOLUME /STRUCTURE_LEVEL=5  device-name
```

For example:

```
$ SET VOLUME /STRUCTURE_LEVEL=5 DKA300:
```

Note

You cannot use the SET VOLUME command to change a volume from ODS-5 to ODS-2. To reset a volume to ODS-2, see the instructions in Section 2.2.3.

If a failure occurs after you issue the SET VOLUME/STRUCTURE_LEVEL command, refer to the instructions following Step 5.

When you issue the SET VOLUME command, the system verifies that the volume can be converted by testing for the following:

- The device must be a disk, and its on-disk structure must be ODS-2 or ODS-5.

  If the volume fails these tests, the system displays messages similar to the following:

```
%SET-E-NOTMOD, DKA300: not modified
-SET-E-NOTDISK, device must be a Files-ll format disk

%SET-E-NOTMOD, DKA300: not modified
-SET-W-INVODSLVL, Invalid on-disk structure level
```

- The disk must be privately owned; the owner process-ID (PID) must be the same as the process that issues the SET VOLUME command.

  If the volume fails this test, the system displays a message similar to the following:

```
%SET-E-NOTMOD, DKA300: not modified
-SET-W-NOTPRIVATE, device must be mounted privately
```

- The mount count must indicate that the device has been mounted only once, which protects against anyone mounting the device over a cluster.

  If the volume fails this test, the system displays a message similar to the following:

```
%SET-E-NOTMOD, DKA300: not modified
-SET-W-NOTONEACCR, device must be mounted with only one accessor
```

6. Dismount the private volume DKA300: and remount the volume publicly by issuing commands such as the following:

```
$ DISMOUNT DKA300:
$ MOUNT /CLUSTER DKA300: DISK1
%MOUNT-I-MOUNTED, DISK1 mounted on _STAR$DKA300:
```

   To verify that the volume has been converted to ODS-5, you can issue a command and see a display such as the following:

```
$ WRITE SYS$OUTPUT F$GETDVI ("DKA300:","ACPTYPE")
F11V5
```

   F11V5 indicates that the volume is ODS-5.

If a Failure Occurs...

If a failure such as an I/O error or a system crash occurs after you enter the SET VOLUME/STRUCTURE_LEVEL command, but before the command completes, the volume might be only partially updated. If so, when you enter the MOUNT command, the Mount utility will display one of the following error messages:

```
Inconsistent file structure level on device ...
Structure level on device ... is inconsistent with volume set
```

If either condition is true, you can issue the MOUNT command only with the /NOSHARE qualifier (or with no qualifier, because /NOSHARE is the default). When you do, the system displays the same error message but only as a warning.

To recover from the error condition, reissue the SET VOLUME/STRUCTURE_LEVEL=5 command. Then dismount and remount the disk. As a last resort, you can restore the backup you made.

2.2.3. Converting from ODS-5 to ODS-2

Two types of BACKUP operations, file and image, support converting ODS-5 file names to ODS-2 file names. (File and image operations are described in the Backup chapter of the *VSI OpenVMS System Manager's Manual*.)

In the examples in the following descriptions, notice that when you perform a conversion to or from a save set, the “created as” or “copied as” message is displayed for the converted files.

- Conversions during image operations
  - Restoring an ODS-5 image save set to an ODS-2 disk

    You can use this method if you have an image backup of an ODS-5 disk, and you want to restore it to an ODS-2 disk.

    In the command line in the following example, IMAGE.BCK is the ODS-5 save set, and DKA200 is the ODS-2 disk. When you use this conversion method, you must pre-initialize the output disk to ODS-2 and then include the /NOINIT qualifier in your command line.

```
$ BACKUP/LOG/IMAGE/CONVERT DKA500:[000000]IMAGE.BCK/SAVE -
_$ DKA200:/NOINIT


%BACKUP-I-ODS5CONV, structure level 5 files will be converted to
 structure
```

```
        level 2 on DKA200:
-BACKUP-I-ODS5LOSS, conversion may result in loss of structure level
 5
        file attributes
%BACKUP-S-CREATED, created DKA200:[000000]000000.DIR;1
%BACKUP-S-CREATED, created DKA200:[000000]BACKUP.SYS;1
%BACKUP-S-CREATED, created DKA200:[000000]CONTIN.SYS;1
%BACKUP-S-CREATED, created DKA200:[000000]CORIMG.SYS;1
%BACKUP-S-CREATED, created DKA200:[000000]SECURITY.SYS;1
%BACKUP-S-CREATED, created MDA2:[000000]TEST_FILES.DIR;1
%BACKUP-S-CREATEDAS, created DKA200:[TEST_FILES]SUB^_^{DIR^}.DIR;1 as
        DKA200:[TEST_FILES]SUB$$DIR$.DIR;1
%BACKUP-S-CREATEDAS, created
        DKA200:[TEST_FILES.SUB^_^{DIR^}]SUB^&_~_FILE_~.DAT;1 as
        DKA200:[TEST_FILES.SUB$$DIR$]SUB$_$_FILE_$.DAT;1
%BACKUP-S-CREATEDAS, created
        DKA200:[TEST_FILES]THIS^_IS^_A^_TEST^{_FILE_^}.DAT;1 as
        DKA200:[TEST_FILES]THIS$IS$A$TEST$_FILE_$.DAT;1
%BACKUP-S-CREATED, created DKA200:[000000]VOLSET.SYS;1
```

- Saving an ODS-5 disk to an ODS-2 image save set

  You can use this method if you want to make an ODS-2 image save set of an ODS-5 disk that can be read by a system running a version of OpenVMS prior to Version 7.2.

  In the following example, DKA500: is an ODS-5 disk, and IMAGE.BCK is an ODS-2 save set on the DKA200: disk.

```
$ BACKUP/LOG/CONVERT/IMAGE DKA500: DKA200:[000000]IMAGE.BCK/SAVE
%BACKUP-I-ODS5CONV, structure level 5 files will be converted to
structure level 2 on DKA200:
-BACKUP-I-ODS5LOSS, conversion may result in loss of structure level
 5
file attributes
%BACKUP-S-COPIED, copied DKA200:[000000]000000.DIR;1
%BACKUP-S-COPIED, copied DKA200:[000000]BACKUP.SYS;1
%BACKUP-S-HEADCOPIED, copied DKA200:[000000]BADBLK.SYS;1 header
%BACKUP-S-HEADCOPIED, copied DKA200:[000000]BADLOG.SYS;1 header
%BACKUP-S-HEADCOPIED, copied DKA200:[000000]BITMAP.SYS;1 header
%BACKUP-S-COPIED, copied DKA200:[000000]CONTIN.SYS;1
%BACKUP-S-COPIED, copied DKA200:[000000]CORIMG.SYS;1
%BACKUP-S-HEADCOPIED, copied DKA200:[000000]INDEXF.SYS;1 header
%BACKUP-S-COPIED, copied DKA200:[000000]SECURITY.SYS;1
%BACKUP-S-COPIED, copied DKA200:[000000]TEST_FILES.DIR;1
%BACKUP-S-COPIEDAS, copied DKA200:[TEST_FILES]Sub^_^{Dir^}.DIR;1 as
DKA200:[TEST_FILES]SUB$$DIR$.DIR;1
%BACKUP-S-COPIEDAS, copied
DKA200:[TEST_FILES.Sub^_^{Dir^}]Sub^&_~_File_~.Dat;1 as
DKA200:[TEST_FILES.SUB$$DIR$]SUB$_$_FILE_$.DAT;1
%BACKUP-S-COPIEDAS, copied
DKA200:[TEST_FILES]This^_is^_a^_Test^{_File_^}.Dat;1 as
DKA200:[TEST_FILES]THIS$IS$A$TEST$_FILE_$.DAT;1
%BACKUP-S-COPIED, copied DKA200:[000000]VOLSET.SYS;1
```

- “Copying” the contents of an ODS-5 disk to an ODS-2 disk

You can use this method if you want to create an ODS-2 disk from an ODS-5 disk without creating an intermediate save set.

When you use this conversion method, you must pre-initialize the output disk to ODS-2 and include the /NOINIT qualifier in your command line. In the following example, DKA500 is the ODS-5 disk, and DKA200 is the ODS-2 disk.

```
$ BACKUP/LOG/CONVERT/IMAGE DKA500: DKA200:/NOINIT
%BACKUP-I-ODS5CONV, structure level 5 files will be converted to
structure level 2 on DKA200:
-BACKUP-I-ODS5LOSS, conversion may result in loss of structure level
 5
file attributes
%BACKUP-S-CREATED, created DKA200:[000000]000000.DIR;1
%BACKUP-S-CREATED, created DKA200:[000000]BACKUP.SYS;1
%BACKUP-S-CREATED, created DKA200:[000000]CONTIN.SYS;1
%BACKUP-S-CREATED, created DKA200:[000000]CORIMG.SYS;1
%BACKUP-S-CREATED, created DKA200:[000000]SECURITY.SYS;1
%BACKUP-S-CREATED, created DKA200:[000000]TEST_FILES.DIR;1
%BACKUP-S-CREATED, created DKA200:[TEST_FILES]SUB$$DIR$.DIR;1
%BACKUP-S-CREATED, created DKA200:[TEST_FILES]THIS$IS$A$TEST$_FILE_
$.DAT;1
%BACKUP-S-CREATED, created DKA200:[000000]VOLSET.SYS;1
```

- Conversions during file operations
  - "Copying" individual ODS-5 files to an ODS-2 disk

    This conversion method allows you to interchange files between ODS-5 and ODS-2 disks. You can, for example, select a directory tree for a disk-to-disk "copy" operation.

    In the following example, DKA500 is the ODS-5 disk, and DKA200 is the ODS-2 disk.

```
$ BACKUP/LOG/CONVERT DKA500:[*...]*.*;* DKA200:[*...]*.*;*


%BACKUP-I-ODS5CONV, structure level 5 files will be converted to
        structure level 2 on DKA200:
-BACKUP-I-ODS5LOSS, conversion may result in loss of structure level
 5
        file attributes
%BACKUP-S-CREDIR, created directory DKA200:[TEST_FILES.SUB$$DIR$]
%BACKUP-S-CREATED, created DKA200:[TEST_FILES]THIS$IS$A$TEST$_FILE_
$.DAT;1
```

  - Saving individual ODS-5 files in an ODS-2 save set

    You can use this method to save ODS-5 files in a save set that can be read on a system running a version of OpenVMS prior to Version 7.2.

    In the following example, DKA500 is an ODS-5 disk, and DKA200 is an ODS-2 disk; FILES.BCK is the ODS-2 save set.

```
$ BACKUP/LOG/CONVERT DKA500:[*...]*.*;* DKA200:FILES.BCK/SAVE


%BACKUP-I-ODS5CONV, structure level 5 files will be converted to
```

```
        structure level 2 on DKA200:
-BACKUP-I-ODS5LOSS, conversion may result in loss of structure level
        5 file attributes
%BACKUP-S-COPIED, copied DKA200:[000000]000000.DIR;1
%BACKUP-S-COPIED, copied DKA200:[000000]TEST_FILES.DIR;1
%BACKUP-S-COPIEDAS, copied DKA200:[TEST_FILES]Sub^_^{Dir^}.DIR;1 as
        DKA200:[TEST_FILES]SUB$$DIR$.DIR;1
%BACKUP-S-COPIEDAS, copied
        DKA200:[TEST_FILES.Sub^_^{Dir^}]Sub^&_~_File_~.Dat;1 as
        DKA200:[TEST_FILES.SUB$$DIR$]SUB$_$_FILE_$.DAT;1
%BACKUP-S-COPIEDAS, copied
        DKA200:[TEST_FILES]This^_is^_a^_Test^{_File_^}.Dat;1 as
        DKA200:[TEST_FILES]THIS$IS$A$TEST$_FILE_$.DAT;1
```

If BACKUP cannot convert a file name within its existing directory, it converts the file name and leaves it unconnected so that ANALYZE /DISK /REPAIR can connect it to the [SYSLOST] directory, where the file has an ODS-2-compliant name. BACKUP also displays messages such as the following:

```
%BACKUP-I-RECOVCNT, 5 files could not be converted into a directory on
 DKA100
-BACKUP-I-RECOVCMD, use the Analyze/Disk_Structure/Repair command to
 recover
     files
```

In this case, you need to move the file from [SYSLOST] to the appropriate directory. Refer to the "created as" log messages to see where the file logically would have been placed so that you can move it there manually.

2.3. Controlling Access to ODS-5 Volumes

A system manager may choose to enforce one or both of the following restrictions:

- Prevent users on a VAX from accessing files on an ODS-5 volume
- Prevent untested applications from accessing files on an ODS-5 disk (You can allow certain users to override this access control on an ODS-5 volume.)

The system manager can impose either of these restrictions by using normal OpenVMS discretionary controls. Refer to the *VSI OpenVMS Guide to System Security* for more information.

Sections 2.3.1 and 2.3.2 contain examples of restricting access to ODS-5 volumes.

2.3.1. Preventing VAX Users from Accessing an ODS-5 Volume

Follow these steps to prevent a user from accessing an ODS-5 volume from a VAX node:

1. Define an identifier (for example, VAX_NODE) to identify users running on an OpenVMS VAX node. For example:

   ```
   $ RUN SYS$SYSTEM:AUTHORIZE

   UAF> ADD /IDENTIFIER VAX_NODE

   %UAF-I-RDBADDMSG, identifier VAX_NODE value %X80010037 added to
   ```

```
        rights database
```

2. On each VAX node, add VAX_NODE to the system rights list. For example:

   ```
   $SET RIGHTS_LIST /ENABLE /SYSTEM VAX_NODE
   ```

   The /ENABLE qualifier in the command adds VAX_NODE to the system rights list.

   Also add this command to the SYSTARTUP_VMS.COM command procedure.

3. To prevent users on a VAX node from gaining access to an ODS-5 volume, place an Access Control Entry (ACE) on the volume that denies access to holders of the VAX_NODE identifier. For example:

   ```
    $ SET SECURITY /CLASS=VOLUME ODS5_DISK -
   _$ /ACL=(ID=VAX_NODE,ACCESS=NONE)
   ```

2.3.2. Preventing an Untested Application from Accessing an ODS-5 Volume

Follow these steps to prevent an untested application from accessing an ODS-5 volume:

1. Define an identifier (for example, ODS5_UNSAFE) to identify applications that you do not want to access an ODS-5 volume. For example:

   ```
   UAF> ADD /IDENTIFIER ODS5_UNSAFE /ATTR=SUBSYSTEM

   %UAF-I-RDBADDMSG, identifier ODS5_UNSAFE value %X80010039 added to
          rights database
   ```

2. Attach a protected subsystem ACE to the application with the ODS5_UNSAFE identifier. For example:

   ```
   $ SET SECURITY /CLASS=FILE SYS$SYSTEM:APPLICATION.EXE -
   _$ /ACL=(SUBSYSTEM,ID=ODS5_UNSAFE)
   ```

3. To each ODS-5 volume, attach an ACE denying access to the ODS-5 volume to holders of the ODS5_UNSAFE identifier. For example:

   ```
   $ SET SECURITY /CLASS=VOLUME ODS5_DISK/ -
   _$ ACL=(ID=ODS5_UNSAFE,ACCESS=NONE)
   ```

You can also override the restriction in the last step to allow trained users to access untested applications by following the remaining steps:

a. Create another identifier (for example, ODS5_UNTRAINED):

   ```
   UAF> ADD /IDENTIFIER ODS5_UNTRAINED

   %UAF-I-RDBADDMSG, identifier ODS5_UNTRAINED value %X80010038 added to
          rights database
   ```

b. Assign this identifier to all users. For example:

   ```
   UAF> GRANT/IDENTIFIER ODS5_UNTRAINED *

   %UAF-I-GRANTMSG, identifier ODS5_UNTRAINED granted to *
   ```

c. Instead of Step 3, place an Access Control Entry (ACE) on the volume that denies access to holders of the ODS5_UNTRAINED identifier. For example:

```
$ SET SECURITY /CLASS=VOLUME ODS5_DISK/ -
_$ ACL=(ID=ODS5_UNSAFE+ODS5_UNTRAINED,ACCESS=NONE)
```

This command prevents ODS5_UNTRAINED users from accessing the volume with ODS5_UNSAFE applications.

d. Remove the identifier from an individual user you are willing to let use any application on an ODS-5 volume. For example:

```
UAF> REVOKE/IDENTIFIER ODS5_UNTRAINED SHEILA_USER
%UAF-I-REVOKEMSG, identifier ODS5_UNTRAINED revoked from SHEILA_USER
```

After you complete these steps:

- An untrained user can use an untested application only to access ODS-2 volumes.
- A trained user can access ODS-5 volumes with any application.

2.4. System Management Utility Changes

The following sections describe changes made to OpenVMS system management utilities to support extended file names.

2.4.1. Analyze/Disk_Structure Utility

The Analyze/Disk_Structure utility (ANALYZE/DISK_STRUCTURE) now checks the readability and validity of Files-11 On-Disk Structure (ODS) Levels 1, 2, and 5 disk volumes.

The following new qualifier has also been added:

/STATISTICS

This qualifier produces statistical information about the volume under verification and creates a file, STATS.DAT, which contains per-volume statistics. The information placed in STATS.DAT is the following:

- The number of ODS-2 and ODS-5 headers on the volume
- The number of special headers on ODS-5 volumes
- The distribution of file name lengths
- The distribution of extension header chain lengths
- The distribution of header identification area free space
- The distribution of header map area and ACL area free space
- The totals of header space that is in use and header space that is not in use

2.4.2. Backup Utility (Alpha Only)

Following are new features the Backup utility (BACKUP) has implemented to support extended file names on Alpha systems:

- New BACKUP qualifier: /CONVERT

  If you convert an ODS Level 5 file to an ODS Level 2 file, some ODS-5 file attributes can be lost.

  Note

  Use the /CONVERT qualifier to perform image restores of ODS-5 save sets to ODS-2 volumes. To preserve the output volume as ODS-2, you must also use the /NOINIT qualifier.

- Deep directories

  Enhanced algorithms handle deep directories. Prior to OpenVMS Version 7.2, 32 levels of directories were supported by the Backup utility. In Version 7.2, the Backup utility supports as many levels of directories as RMS allows, which is currently 255 levels.

- Extended character set

  BACKUP can process file names that use characters from the following character sets:

  - DEC Multinational (MCS)
  - ISO Latin-1
  - Unicode (UCS-2)

2.4.3. Physical Backups of ODS-5 Volumes on VAX Systems

On OpenVMS VAX systems, BACKUP supports ODS-5 volumes only when you specify the /PHYSICAL qualifier to back up a volume. The BACKUP /PHYSICAL command causes BACKUP to make a block-by-block physical backup of the disk, ignoring the structured contents of the disk.

On Alpha systems, you can use either the BACKUP /IMAGE or BACKUP /PHYSICAL command.

See the Backup chapter of the *VSI OpenVMS System Manager's Manual* for more information about support for extended file names by the Backup utility on Alpha processors.

2.4.4. Mount Utility (Alpha Only)

The Mount utility has been modified to provide full support for ODS-5 volumes.

Chapter 3. Extended File Naming Characteristics

Extended File Specifications provides a wider variety of character set options and naming conventions, similar to those available on Windows NT. This chapter describes the impact of Extended File Specifications on the general user, and contains the following topics:

- Outlining the differences in file and directory specifications between ODS-2 and ODS-5
- Manipulating files with extended file names
- Using extended file names in DCL command procedures
- Displaying ODS-5 file specifications in DECwindows

3.1. File Specifications

On ODS-5 volumes, there are two possible naming styles for file specifications: **traditional** (ODS-2 compliant) and **extended** (ODS-5 compliant).

Section 3.1.1 describes ODS-2 compliant name syntax. Section 3.1.2 describes ODS-5 compliant name syntax.

3.1.1. Traditional (ODS-2) Syntax

The traditional (ODS-2) file name syntax is the syntax most OpenVMS users are familiar with. OpenVMS Versions 7.1 and earlier follow this syntax, which supports the following character set and naming conventions:

ODS-2 Character Set

Traditional (ODS-2-compliant) file names can use alphanumeric characters (A-Z, a-z, 0-9), dollar sign ($), underscore (_) and hyphen (-).

Case Insensitivity

Case preservation is not supported with traditional syntax. You can enter file names in uppercase, lowercase, or mixed case; however, all characters are stored in uppercase format.

Standard Delimiters

With traditional syntax, the file type is preceded by a period (.). The file version is separated from the type by a semicolon (;) or sometimes a period (.). (When the system displays file specifications, it displays a semicolon in front of the file version number.) Directories are enclosed by brackets ([]) or angle brackets (<>). Directory levels are separated by periods (.).

Limited File Length

Traditional file specifications follow the 39.39 format, supporting only a single period (.) separating the file name and file type.

3.1.2. Extended (ODS-5) Syntax

The extended file name syntax offered on ODS-5 volumes supports a larger character set, longer file names, and longer file specifications. This syntax allows OpenVMS systems to store and access files with Windows NT-style file specifications that use the following character set and naming conventions:

3.1.2.1. ISO Latin-1 Character Set

The ISO Latin-1 character set is a superset of the traditional ASCII character set used by versions of OpenVMS previous to 7.2. All characters from the 8-bit ISO Latin-1 character set are valid in ODS-5 file specifications *except* the following:

C0 control codes (0x00 to 0x1F inclusive)
double quotation marks (")
asterisk (*)
backslash (\)
colon (:)
left and right angle brackets (< >)
frontslash (/)
question mark (?)
vertical bar (|)
C0 control codes (0x00 to 0x1F inclusive)
Double quotation marks (")
Asterisk (*)
Backslash (\)
Colon (:)
Left angle bracket (<)
Right angle bracket (>)
Slash (/)
Question mark (?)
Vertical bar (|)

3.1.2.2. Special Characters

Some ISO Latin-1 characters require an escape character to precede them in a file specification in order to be interpreted as literal characters rather than special function characters. In extended file names, RMS and DCL interpret the circumflex (^) as an escape character. The following list contains rules for using the escape character:

- The escape character (^) followed by underscore (_) or by a space represents a space.
- The escape character (^) followed by any of the following characters means that the character is to be used as part of a file name rather than having any special meaning that it might otherwise have in a file specification:

  ```
  .  ,  ;  [  ]  %  ^  &
  ```

- A user can enter a literal period (.) with or without the escape character (^) in a file name. The system adds the escape character to any periods other than those that act as delimiters for the file type and version number. Literal periods (.) in directory names *must* be preceded by the escape character.
- An escape character followed by a hexadecimal digit requires a second hexadecimal digit. Interpret the two following characters as a hexadecimal value for an arbitrary 1-byte character.

For example, ^20 represents a space.

- An escape character followed by “U” within a file specification indicates that the four hexadecimal digits that follow are to be interpreted as Unicode. For example, ^U012F.

  All characters in file specifications that are not preceded by an escape character (^) are presumed to be ISO Latin-1.

Note

File names containing special characters cannot be accessed from a VAX system. See Section 3.3 for more information about mixed-architecture environments.

3.1.2.3. Interpretation of Period (.)

The use of the period (.) as a literal character in extended file names requires RMS to determine which periods are file name characters and which are delimiters.

When only one period (.) is used in an extended file name, that period is interpreted as the delimiter. As in previous versions of OpenVMS, this behavior also occurs if the single period is followed by a number:

```
$ CREATE Test.1
```

creates the file:

```
Test.1;1
```

Determination of Version Numbers

When there are multiple periods (.) in a file name, RMS looks at all the characters after the last period. If those characters are all numeric, or all numeric and preceded by a minus sign (-), the numeric string is determined to be a version number. However, if there are more than 5 numeric characters, RMS rejects the file name as illegal. If there is a nonnumeric character following the last period, then it is interpreted as a type delimiter.

For example, the following command:

```
$ CREATE Test4.3.2.1
```

creates the file:

```
Test4^.3.2;1
```

where *.2* is the file type and *1* is the file version.

A version number explicitly delimited by a semicolon (;) must also be 5 or fewer numeric characters, and can be preceded by a minus sign (-).

3.1.2.4. Expanded File Specification Length

On an ODS-5 volume, the file name together with the file type can be up to 236 8-bit characters or 118 16-bit characters in length. Unmodified programs and utilities may limit or abbreviate complete file specifications to 255 bytes.

```
$ CREATE This.File.Name.Has.A.Lot.Of.Periods.DAT
$ CREATE -
```

```
_$
 ThisIsAVeryLongFileName^&ItWillKeepGoingForLotsAndLotsOfCharacters.Exceed
 -
_$ ingThe39^,39presentInPreviousVersionsOfOpenVMS
$ DIRECTORY

Directory TEST$ODS5:[TESTING]

ThisIsAVeryLongFileName^&ItWillKeepGoingForLotsAndLotsOfCharacters.Exceeding
The39^,39presentInPreviousVersionsOfOpenVMS;1
This^.File^.Name^.Has^.A^.Lot^.Of^.Periods.DAT;1

Total of 2 files.
```

Section 3.6 discusses how RMS abbreviates file specifications when the full file specification exceeds the limit of 255 bytes.

3.1.2.5. Using Wildcards

Single- and multiple-character wildcards function as expected with ODS-5 files. A single-character wildcard represents exactly one character in either the file name or file type, but may not be used in the file version string. A multiple-character wildcard can represent any number of characters (including zero characters) in the file name or file type. A multiple-character wildcard can be used in place of a version string.

3.1.2.5.1. Wildcard Characters

The following characters are wildcard characters when working on any OpenVMS 7.2 volume:

- The asterisk (*) is a multiple-character wildcard.
- The percent sign (%) is a single-character wildcard.
- The question mark (?) is a single-character wildcard.
- The asterisk (*) is a multiple-character wildcard.
- The percent sign (%) is a single-character wildcard.
- The question mark (?) is a single-character wildcard.

The percent sign (%) continues to be a single-character wildcard to maintain compatibility with existing applications. The percent sign (%) may be used as a literal character when preceded by the circumflex (^) and is also a literal character in Windows NT file names. Therefore, in addition to the percent sign, RMS also recognizes the question mark (?) as a single character wildcard. The question mark functions identically to the percent sign as a wildcard character on OpenVMS 7.2 and later. The percent sign and the question mark each matches *exactly* one character in a search pattern.

Note

An escaped character (such as ^.) or an escape sequence (such as ^EF or ^U0101) is considered a single character for purposes of wildcard matching.

3.1.2.5.2. Wildcard Syntax

Although DCL preserves the case of extended file names, wildcard matching is case blind.

A search operation with wildcards continues to match only against the corresponding character in the same part of the target specification. Table 3.1 contains examples of some wildcard searches.

Table 3.1. Sample Wildcards and Matching Patterns

The pattern...	matches...	...but does not match
A*B;*	AHAB.;1	A.B;1
A.*.B*	A^.DISK.BLOCK;1	A^.C^.B.DAT;1
A?B.TXT;*	A^.B.TXT;5	A^.^.B.TXT;1
*.DAT	Lots^.of^.Periods.dat;1	DAT.;1
Mil?no.dat	Milano.dat;1	Millaano.dat;1
NAPOLI.?.DAT	napoli.q.dat;1	napoli.abc77.dat;1

3.1.2.6. Case Preservation

On an ODS-5 volume, the case for all versions of a file name is identical; the case is preserved as the file name was first created. When you create more than one file with the same name differing only in case, DCL treats the subsequent files as new versions, and converts them to the same case as the original file.

For example, the following sequence of commands:

```
$ CREATE CaPri.;1
$ CREATE CAPRI
$ CREATE capri
```

produces the resulting files:

```
CaPri.;1  CaPri.;2  CaPri.;3
```

In prior versions of OpenVMS, DCL and RMS converted all file specifications to uppercase. On ODS-5 volumes, the case of all file names is preserved as created by the user.

3.2. Directory Specifications

The following sections describe the deeper directory structures and extended naming syntax available on ODS-5 volumes. It is now possible to go beyond the eight levels of directories previously supported in OpenVMS.

3.2.1. Deep Directory Structures

OpenVMS 7.2 supports deep nesting of up to 255 directories with the restriction that the total directory specification can be no longer than 512 8-bit or 16-bit characters.

For example, a user can create the following directories on an ODS-2 or ODS-5 volume:

```
$ CREATE/DIRECTORY [a.b.c.d.e.f.g.h.i.j.k.l.m]
```

A user can create the following directory with a long name on an ODS-5 volume:

```
$ CREATE/DIRECTORY -
[.AVeryLongDirectoryNameWhichHasNothingToDoWithAnythingInParticular]
```

3.2.2. Directory Naming Syntax

On ODS-5 volumes, directory names conform to most of the same conventions as file names when using the ISO Latin-1 character set. Periods and special characters may be present in the directory name, but they must be preceded by the escape character (^) in order to be recognized as literal characters, as shown in Table 3.2.

Table 3.2. Directory Names on ODS-5 Volumes

CREATE/DIRECTORY. . .	Result
[Hi^&Bye]	Hi^&Bye.DIR;1
[Lots^.Of^.Periods^.In^.This^.Name]	Lots^.Of^.Periods^.In^.This^.Name.DIR;1

3.2.2.1. Directory ID and File ID Abbreviation

Under some circumstances, a full file specification may contain more characters than the 255 bytes allowed by unmodified applications. If a file specification that such an application needs exceeds 255 bytes in length, RMS generates a shorter file specification by abbreviating the directory to a DID, and if necessary, the filename to a FID.

When the file specification is too long, RMS first attempts to generate a shorter directory specification by identifying the directory with its directory ID. This shorter specification is referred to as a DID.

```
TEST$ODS5:[5953,9,0]Alghero.TXT;1
```

Note that this form of the directory name must have three numbers and two commas to avoid ambiguity with UIC format directory names. With the DIRECTORY command you can view the shorter DID version as well as the full version of a file specification. See Section 3.6 for more information on displaying long file specifications. See Section B.2.2.7 for more information about DID abbreviations. See Section B.2.2.8 for more information about FID abbreviations.

3.3. Working in Mixed Environments

When working in an environment that contains both OpenVMS Alpha and OpenVMS VAX systems, it is important for a user to know the following:

- The system type
- The volume type where the user's default directory resides
- The volume type where the user creates a new file

OpenVMS 7.2 allows VAX systems to mount ODS-5 volumes; however, users on OpenVMS VAX systems can access only files with ODS-2-compliant file names.

When working in a mixed environment of ODS-2 and ODS-5 volumes, keep in mind the restrictions of ODS-2 file names when creating files on ODS-5 volumes. If you copy a file that has special characters in its name from an ODS-5 to an ODS-2 volume, you must give it an ODS-2 compliant name.

3.4. DCL Support for ODS-5 Volumes

When using extended file names on the DCL command line, you need to set the parsing style to EXTENDED to accept and display extended file specifications. The default setting is TRADITIONAL. To set the parsing style, enter the command:

```
$ SET PROCESS/PARSE_STYLE=EXTENDED
```

Note

DCL lexical functions use the DEC Multinational character set, which is different from the ISO Latin-1 character set used for file names on an ODS-5 disk. This can lead to unexpected results if, for example, you use the DCL function F$EDIT to upcase a filename. F$EDIT will not upcase DEC MCS characters with hexadecimal values of F0, F7, FE, and FF.

See Section 3.4.1 for more information about changing the DCL name parsing style.

3.4.1. Using the Extended File Specifications Parsing Feature in DCL

Sections 3.4.1.1, 3.4.1.2, and 3.4.1.3 describe how to control the DCL name parsing style, both on the command line and in a command procedure.

3.4.1.1. Enabling the Extended File Name Parsing Style

On OpenVMS Alpha systems, you can tell DCL to accept ODS-5 file names on a per process basis by entering the following command:

```
$ SET PROCESS/PARSE_STYLE=EXTENDED
```

Note that this command has no effect on an OpenVMS VAX system.

After you enter the command, DCL accepts a file name such as the following:

```
$ CREATE MY^[FILE
```

The circumflex (^) character is used as an escape character to tell DCL to treat the next character (in this case, a left bracket) as a literal character in the name, rather than as a delimiter.

For additional information, see the description of the SET PROCESS/PARSE_STYLE command in the *VSI OpenVMS DCL Dictionary: N-Z*.

3.4.1.2. Resetting the Default File Name Parsing Style

The default DCL parsing style for file names is for ODS-2 style file names. To reset DCL to the default parsing style, enter the following command:

```
$ SET PROCESS/PARSE_STYLE=TRADITIONAL
```

After you enter this command, DCL accepts only ODS-2 file name formats.

3.4.1.3. Switching Between File Name Parsing Styles

A command procedure that requires a specific file name parsing style can include commands within the procedure to switch between styles. The following command procedure saves the current parsing style, sets the parsing style to TRADITIONAL, performs (unspecified) commands, then restores the saved parsing style.

```
$ original_style= f$getjpi("","parse_style_perm")
$ SET PROCESS/PARSE_STYLE=TRADITIONAL
```

```
    .
    .
    .
$ SET PROCESS/PARSE_STYLE='original_style'
```

The first command equates 'original_style' with the current parse style. The second command sets the parsing style to TRADITIONAL. The last command resets the parsing style to the original style.

3.4.2. Using Extended File Names in DCL Command Parameters

Command procedures that use file names as parameters can produce different results in an ODS-5 environment.

You can switch from the TRADITIONAL to the EXTENDED parsing style, and this section describes the following areas that may be affected if you choose to do so:

- Command procedure file specification
- Case preservation and $FILE
- Ampersand versus apostrophe substitution

See Section 3.4.1 for more information on switching between parsing styles.

3.4.3. Command Procedure File Specification

If indirect command procedures are used, you may need to put quotes around some procedure arguments.

The following examples show the differences in output between TRADITIONAL and EXTENDED parsing styles when using the same command file, SS.COM:

```
$ create ss.com
$ if p1 .nes. "" then write sys$output "p1 = ",p1
$ if p2 .nes. "" then write sys$output "p2 = ",p2
$ if p3 .nes. "" then write sys$output "p3 = ",p3
```

- Setting the parsing style to TRADITIONAL and running SS.COM produces the following output:

  ```
  $ set process/parse_style=extended
  $ @ss ^ parg2 parg3
  p1 = ^
  p2 = PARG2
  p2 = PARG3
  ```

  Note that the circumflex (^) is the first argument (not an escape character), and that case is not preserved for the p2 and p3 procedure arguments

- Setting the parsing style to EXTENDED produces the following output when running the same command procedure:

  ```
  $ set process/parse_style=extended
  $ @ss ^ parg2 parg3
  p1 = ^ PARG2
  p2 = PARG3
  ```

Note that the command procedure recognizes the circumflex (^) as the escape character that identifies the space as a literal character rather than an argument separator, and that "^ PARG2" is the first argument. Case is not preserved.

- Adding quotes to the circumflex (^) produces the following results:

```
$ @ss "^" parg2 "parg3"
p1 = ^
p2 = PARG2
p3 = PARG3
```

Because the circumflex (^) is within a quoted string, it is not treated as an escape character.

- Adding quotes to the p3 argument produces the following result:

```
$ @ss "^" parg2 "parg3"
p1 = ^
p2 = PARG2
p3 = PARG3
```

Note that case is preserved for the p3 procedure argument.

- When the parsing style is set to TRADITIONAL, the following command treats the circumflex (^) and the parg2 and parg3 strings as procedure arguments, and the command procedure produces the following results:

```
$ set process/parse_style=traditional
$ @ss^ parg2 parg3
p1 = ^
p2 = PARG2
p3 = PARG3
```

- When the parsing style is set to EXTENDED, the circumflex (^) is treated as an escape character that identifies the space as a literal character. DCL looks for the file "SS^_PARG2.COM" and produces the error shown in the following example:

```
$ set process/parse_style=extended
$ @ss^ parg2 parg3
%DCL-E-OPENIN, error opening SYS$ROOT:[SYSMGR]ss^_parg2.COM; as
 input
-RMS-E-FNF, file not found
```

3.4.4. Case Preservation and $FILE

DCL attempts to preserve the case of file specifications. It can do this only for commands defined with the Command Definition Utility (CDU). DCL preserves case for any item defined in the command definition file (.CLD) with the $FILE parse type.

Refer to the *VSI OpenVMS Command Definition, Librarian, and Message Utilities Manual* for more information.

3.4.5. Ampersand Versus Apostrophe Substitution

You can use ampersand (&) substitution, as opposed to apostrophe substitution, to preserve case during traditional parsing.

The following traditional parsing example shows a series of commands that change the case of a character string:

```
$ set process/parse_style=traditional
$ x = "string"
$ define y 'x'
$ sho log y
   "Y" = "STRING" (LNM$PROCESS_TABLE)
$ define y &x
%DCL-I-SUPERSEDE, previous value of Y has been superseded
$ sho log y
   "Y" = "string" (LNM$PROCESS_TABLE)
```

Note that the use of the ampersand (&) preserved the case of the character string assigned to the x variable.

Apostrophe substitution takes place before the command line is set to uppercase, and ampersand substitution takes place after the command line is set to uppercase.

The following extended parsing example shows the same series of commands:

```
$ set process/parse_style=extended
$ define y 'x'
%DCL-I-SUPERSEDE, previous value of Y has been superseded
$ sho log y
   "Y" = "string" (LNM$PROCESS_TABLE)
$ define y &x
%DCL-I-SUPERSEDE, previous value of Y has been superseded
$ sho log y
   "Y" = "string" (LNM$PROCESS_TABLE)
```

Note that both character strings for the y variable are returned lowercase. This happens because the DEFINE command uses $FILE, which preserves the case.

Ampersand substitution can therefore be used to specify EXTENDED file names even though the parsing style is set to TRADITIONAL, as shown in the following example:

```
$ set process/parse=extended
$ cre file^ name.doc
Contents of an ODS5 file
 Exit

$ set process/parse=traditional
$ a = "file^ name.doc"
$ type file^ name.doc
%DCL-W-PARMDEL, invalid parameter delimiter - check use of special
 characters
 \^NAME\
$ type 'a'
%DCL-W-PARMDEL, invalid parameter delimiter - check use of special
 characters
 \^NAME\
$ type &a
Contents of an ODS5 file
```

Note

Ampersand substitution does not work for foreign commands.

3.5. DCL Commands and Utilities

Some DCL commands and OpenVMS utilities have been modified to take advantage of all the features of extended file names. These utilities and commands accept and handle extended file specifications without error and without modifying their expected case.

Other DCL commands and OpenVMS utilities have had little or no modification to take advantage of extended file names. These utilities and commands are expected to handle most of the attributes of extended file specifications (such as new characters and deep directory structures) correctly.

See Table 3.3 for the new features in DCL to support Extended File Specifications.

Section 2.1 fully defines the different levels of support for extended file names provided by DCL commands and OpenVMS utilities in OpenVMS Version 7.2.

The following DCL commands and OpenVMS utilities provide full support for extended file names

ANALYZE /AUDIT
ANALYZE /DISK
ANALYZE /RMS
BACKUP
CONVERT
CONVERT /RECLAIM
COPY
CREATE /DIRECTORY
DELETE
DIRECTORY
DUMP
EDIT /ACL
EXCHANGE /NETWORK
FDL
PURGE
RECOVER/RMS
RENAME
SEARCH
SET SECURITY
SYSMAN
TYPE

Table 3.3 lists the new features in DCL to support Extended File Specifications.

Table 3.3. DCL New Features

DCL Command	New Features
COPY	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
DELETE	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
DIRECTORY	Added the following items: • Qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED

DCL Command	New Features
	• Display item to /FULL to display Client Attributes
DUMP	Added the following items: • Display item to /DIRECTORY to display Name type attribute • Display item to /HEADER to display new attributes • Qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
EXCHANGE NETWORK	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
F$FILE_ATTRIBUTES Lexical	Added new item codes: FILE_LENGTH_HINT, VERLIMIT, DIRECTORY
F$GETDVI Lexical	Added new type to the ACPTYPE item code.
F$GETJPI Lexical	Added new item codes: PARSE_STYLE_PERM and PARSE_STYLE_IMAGE
INITIALIZE	Added a new qualifier: /STRUCTURE=5 device-name[:] volume-label
PRINT	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
PURGE	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
RENAME	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
SEARCH	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
SET ACL	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
SET DEFAULT	Modified the directory-spec parameter to accept ODS-5-compliant file specifications.
SET DIRECTORY	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
SET FILE	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
SET PROCESS	Added a new qualifier: /PARSE_STYLE=(keyword), where keywords are TRADITIONAL and EXTENDED.
SET SECURITY	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
SET VOLUME	Added a new qualifier: /STRUCTURE_LEVEL=5
SHOW DEVICE/FULL	Updated the display information to show the disk structure level.

DCL Command	New Features
SUBMIT	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED
TYPE	Added new qualifier, /STYLE, with new keywords, EXPANDED and CONDENSED

For detailed information about the enhancements made to the OpenVMS operating system and utilities in support of Extended File Specifications, see the *VSI OpenVMS DCL Dictionary: A-M*, the *VSI OpenVMS DCL Dictionary: N-Z*, and the *VSI OpenVMS Utility Routines Manual*.

3.6. System Services Changes

This section describes the following system services:

- New services:
  - $SET_PROCESS_PROPERTIESW
  - $CVT_FILENAME
- Changed services:
  - $CREPRC
  - $GETJPI
  - $SETDDIR

3.6.1. New Services

$SET_PROCESS_PROPERTIESW System Service (Alpha Only)

$SET_PROCESS_PROPERTIESW System Service (Alpha Only) — The $SET_PROCESS_PROPERTIESW system service sets a simple value associated with a process.

Format

$SET_PROCESS_PROPERTIESW mbz1 ,mbz2 ,mbz3 ,property ,value, prev_value

C Prototype:

```
int sys$set_process_properties(
        unsigned int mbz1,
        unsigned int mbz2,
        unsigned int mbz3,
        unsigned int property,
        unsigned __int64 value,
        unsigned __int64 *prev_value);
```

Arguments

mbz1,mbz2,mbz3

Reserved for future use by VSI. Must be specified as 0.

property

OpenVMS usage:	integer
type:	longword (unsigned)
access:	read only
mechanism:	by value

A constant that selects which property to set.

Valid values for property are defined by the $PPROPDEF macro as shown in Table B.1.

Table 3.4. Property Code Descriptions

Property Code	Description
PPROP$C_PARSE_STYLE_TEMP:	The type of command parsing to use. This value will be set only for the life of the image. The value will revert to the permanent style on image rundown. Valid values are: PARSE_STYLE $C_TRADITIONAL and PARSE_STYLE $C_EXTENDED.
PPROP$C_PARSE_STYLE_PERM:	The type of command parsing to use. This value will be set for the life of the process unless the style is set again. Valid values are: PARSE_STYLE $C_TRADITIONAL and PARSE_STYLE $C_EXTENDED.

value

OpenVMS usage:	integer
type:	quadword (unsigned)
access:	read
mechanism:	by value

A quadword value to which to set the property.

prev_value

OpenVMS usage:	access_mode
type:	quadword (unsigned) address of a quadword value
access:	write
mechanism:	by reference

The address of a quadword that will receive the previous value of the property.

Required Access or Privileges

None.

Required Quota

None.

Related Services

$GETJPI

Condition Values Returned

SS$NORMAL	The service completed successfully.
SS$ACCVIO	Access violation.

$CVT_FILENAME System Service (Alpha Only)

$CVT_FILENAME System Service (Alpha Only) — Converts a string from RMS format to file-system (ACP-QIO) format or from file-system (ACP-QIO) format to RMS format.

Format

SYS$CVT_FILENAME cvttyp ,srcstr ,inflags ,outbuf ,outlen ,outflags

C Prototype:

```
int sys$cvt_filename (unsigned int cvttyp,
                      void *srcstr,
                      unsigned int inflags,
                      void *outbuf,
                      unsigned short int *outlen,
                      unsigned int *outflags);
```

Arguments

cvttyp

OpenVMS usage:	unsigned_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

Longword value that indicates whether the conversion is to be from RMS format to ACP-QIO format or vice versa.

There are two legal values for this parameter, represented by the symbols CVTFNM $C_ACPQIO_TO_RMS and CVTFNM$C_RMS_TO_ACPQIO, which are defined by the $CVTFNMDEF macro.

srcstr

OpenVMS usage:	string of bytes or words
type:	string of bytes or words
access:	read only
mechanism:	by 32-bit descriptor--fixed length string descriptor

String to be converted by the service.

If the conversion is to be from RMS format to ACP-QIO format, *srcstr* is an ISO Latin-1 or VTF-7-encoded character string. If the conversion is to be from ACP-QIO format to RMS format, *srcstr* is a string of byte-width or word-width characters.

The descriptor length field indicates the length of the input string in bytes, whether the characters are byte-width or word-width.

The *srcstr* argument is the 32-bit address of a descriptor that points to this string.

inflags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

Longword flag mask indicating characteristics of the input string.

For conversion from RMS format to ACP-QIO format, only the CVTFNM$V_NO_DELIMITERS flag is valid.

For conversion from ACP-QIO format to RMS format, legal flags are CVTFNM$V_WORD_CHARS and CVTFNM$V_NO_DELIMITERS (defined by the $CVTFNMDEF macro). The flag descriptions are shown in Table B.2.

Table 3.5. Flag Descriptions

Flag	Description
CVTFNM$V_WORD_CHARS	Input source string contains word-width UCS-2 characters (ACPQIO_TO_RMS).
CVTFNM$V_NO_DELIMITERS	Input source string should be treated as an arbitrary string (such as a subdirectory name) rather than as a filename that contains (or should contain) dots or semicolons as type and version delimiters.

outbuf

OpenVMS usage:	string of bytes or words
type:	string of bytes or words
access:	write only
mechanism:	by 32-bit descriptor--fixed-length string descriptor

The buffer into which the converted string is to be written.

If the conversion is from RMS format to ACP-QIO format, the string may consist of byte-width ISO Latin-1 characters or word-width UCS-2 characters, depending upon the characters in the source string. (If any character in the source string requires a word to represent, then all characters in the output buffer will be of word width.)

If the conversion is from ACP-QIO format to RMS format, then the output string will consist of ISO Latin-1 and VTF-7 characters, in RMS canonical form. (Refer to the *VSI OpenVMS Guide to OpenVMS File Applications*.)

For ACPQIO_TO_RMS conversion, if the output string is composed of word-width characters, the CVTFNM$V_WORD_CHARS flag in the *outflags* flag mask will be set.

The *outbuf* argument is the 32-bit address of a descriptor pointing to a buffer writable in the mode of the caller.

outlen

OpenVMS usage:	word_unsigned
type:	word (unsigned)
access:	write only
mechanism:	by 32-bit reference

The *outlen* argument is the 32-bit address of a (16-bit) word writable in the mode of the caller.

outflags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	write only
mechanism:	by 32-bit reference

Longword flag mask in which the service sets or clears flags to indicate characteristics of the output string.

For an RMS_TO_ACPQIO conversion, SYS$CVT_FILENAME sets the bit corresponding to CVTFNM $V_WORD_CHARS (defined by the $CVTFNMDEF macro) if the characters of the converted string are one word (rather than one byte) wide. If the characters of the converted string are one byte wide, the service clears the CVTFNM$V_WORD_CHARS bit. All other bits are cleared by an RMS_TO_ACPQIO conversion.

The *outflags* argument is the 32-bit address of a 32-bit flag mask writable in the mode of the caller.

Description

This service is intended to provide conversion of a filename(1) or of a subdirectory name(2) between the RMS format (as seen at the RMS interface) and ACP-QIO format (as stored on-disk). Prior to Version 7.2, these representations were the same. This is not necessarily the case for extended (ODS-5) filenames. (Refer to the *VSI OpenVMS Guide to OpenVMS File Applications* for details on ODS-5 file specifications.)

1. A filename consists of a file name, a file type, and a file version.

2. A subdirectory name is a string to which ".DIR;1" may be appended to form a directory file name, as stored on-disk.

Depending upon the value of *cvttyp*, the service will perform conversion of a string from RMS format to ACP-QIO format or from ACP-QIO format to RMS format.

The source string is described by the argument *srcstr*, the output buffer is described by the argument *outbuf*, and the resultant string length is written to the argument *outlen*.

If any of the source string falls within the address range of the output buffer, the output string is unpredictable.

RMS-to-ACPQIO Conversion:

A string described by the *srcstr* descriptor argument is converted to an ISO Latin-1 or UCS-2 string with each character represented in a form that can be passed to the ACP-QIO via the $QIO service.

If the CVTFNM$V_NO_DELIMITERS input flag is clear, the source string will be scanned, and, if necessary, a dot and a semicolon will be inserted or appended as though a $PARSE were done with no default name, type, or version fields supplied. If the scan detects any delimiters indicating the presence of fields other than name (without FID), type, or version, a syntax error will be returned.

If the CVTFNM$V_NO_DELIMITERS input flag is set, individual characters will be validated and converted to their on-disk form. However, no scan is done to determine if type and version delimiters are present, and no delimiters are added.

A percent sign (%) that is not preceded by the escape character (^) is converted to a question mark. An ISO Latin-1 character that is preceded by the escape character (^) is converted to the corresponding ISO Latin-1 character. A VTF-7 character (for example, ^U1234) that is preceded by the escape character (^) is converted to a UCS-2 character (for example, 0x1234).

If any character requires UCS-2 (word-width character) representation, all characters are represented in the output string with UCS-2. If no character requires word-width character representation, all characters are represented in the output string with ISO Latin-1 (byte-width) characters.

Valid characters are those that are legal in an RMS filename (file name, file type, and file version) or in an RMS subdirectory name. For example, directory delimiters “[” and “]” are not legal, unless preceded by the escape character (^).

ACPQIO-to-RMS Conversion:

The string described by the *srcstr* descriptor argument is converted to the RMS canonical form for that string.

If the CVTFNM$V_NO_DELIMITERS input flag is clear, the source string must contain at least one semicolon, and, to the left of the semicolon, at least one dot. If it does not, RMS$_SYN (syntax error) is returned. In the output string, all dots and semicolons other than those two are preceded by the RMS escape character (^).

If the CVTFNM$V_NO_DELIMITERS input flag is set, any dot or semicolon encountered is preceded in the output string by the RMS escape character (^).

The CVTFNM$V_WORD_CHARS flag of the *inflags* argument indicates whether the input string is to be interpreted as having byte-width (ISO Latin-1) or word-width (UCS-2) characters. If the argument indicates word-width characters, but the input length value is an odd number, a syntax error is returned.

Question marks are converted to percent signs; percent signs are preceded by the escape character (^). UCS-2 characters are converted to VTF-7 characters. All characters will be represented in RMS canonical form.

Required Access or Privileges

None.

Required Quota

None.

Related Services

None.

Condition Values Returned

SS$NORMAL	The service completed successfully.
SS$_BADPARAM	Unrecognized conversion type, extraneous input flags set, or zero-length input string.
SS$_INSFARG	Not enough arguments provided.
SS$_TOO_MANY_ARGS	Too many arguments provided.
RMS$_SYN	The service could not translate one or more characters in the strings described by the srcstr argument, the input string has word-width characters but odd byte-length (ACPQIO_TO_RMS only), or the CVTFNM $V_NO_DELIMITERS input flag was clear and the input string did not contain both type and version delimiters.
SS$_BUFFEROVF	The output buffer was not large enough to accommodate the converted string.

3.6.2. Changed Services

3.6.2.1. $GETJPI System Service

There are two new item codes for this system service. They are:

- JPI$_PARSE_STYLE_PERM
- JPI$_PARSE_STYLE_IMAGE

These return the values that were set by $SET_PROCESS_PROPERTIES, that can be either PARSE_STYLE$C_TRADITIONAL or PARSE_STYLE$C_EXTENDED. Return length is one byte for each one.

3.6.2.2. $CREPRC System Service

There is a new flag allowed in the *stsflg* parameter:

PRC$M_PARSE_EXPANDED

This sets the PARSE_STYLE_PERM and the PARSE_STYLE_IMAGE properties for the new process to EXPANDED.

3.6.2.3. $SETDDIR System Service

The following text has been added to the system service description:

On Alpha systems, the Set Default Directory service attempts to replace the default directory string with a DID if the length of the resulting default directory exceeds 255 bytes. If this happens, then in addition to the normal syntax check, the entire path to that specification, including the device, is verified and must exist for the call to succeed.

3.7. Displaying Extended File Names on a Terminal

When you want to display extended file names on a terminal, you must specify ISO Latin-1 as the character set for the terminal to display. Otherwise, the characters displayed on the terminal may not match those shown by a PC. The characters that differ between the DEC MCS and the ISO Latin-1 character sets are listed in Figure C.1.

To display the ISO Latin-1 character set correctly on a DECterm, select `UPSS ISO Latin` 1 from the General submenu on the Options menu.

To display the DEC Multinational character set correctly on a DECterm, select `UPSS DEC Supplemental` from the General submenu on the Options menu.

To display the ISO Latin-1 character set correctly on a VT320 or VT420, select `UPSS ISO Latin 1` from the General submenu on the Setup menu.

To display the DEC Multinational character set correctly on a VT320 or VT420, select `UPSS DEC Supplemental` from the General submenu on the Setup menu.

Chapter 4. Extended File Naming Considerations for OpenVMS Application Developers

This chapter describes how to evaluate an application's support for Extended File Specifications.

4.1. Evaluating Your Current Support Status

As part of testing OpenVMS Alpha Version 7.2, OpenVMS application developers should evaluate and test all existing applications to determine their current level of support for Extended File Specifications and whether that level is appropriate. See Section 2.1 for a description of the levels of support.

Any applications that are coded to undocumented interfaces may not provide support for either deep directories or extended file names. Section 4.1.2 lists additional application attributes that may prevent an application from supporting extended file names. Section 4.1.3 lists additional application attributes that may prevent an application from supporting ODS-5 volumes.

You can choose either to modify these applications to support Extended File Specifications or not to use them under Extended File Specifications. For information on how to modify an application to provide default support for Extended File Specifications, see Section 4.2.1. For information on how to upgrade an application to full support, see Section 4.2.2.

4.1.1. Default Support

Most unmodified OpenVMS applications fall into the default support category. Specifically, these applications use the traditional API rather than the new API when making RMS calls (see Section B.2 for details about the new RMS API). Applications that use high-level language calls to perform file operations will also fit into this category unless the language run-time libraries have been modified to full support. [1] In most cases, you will not need to modify these applications for them to function successfully under Extended File Specifications.

4.1.2. No Support for Extended File Names

An application that does any of the following may not support extended file names:

1. Uses the QIO interface to specify file names. Developers should examine all layered products and applications and evaluate any file name interaction between the RMS and the XQP interfaces. The format for extended file names varies for each interface. As a result, an application can no longer assume that it can use the same file name for both RMS and the XQP. In addition, the XQP does not allow an unmodified application to use extended file names. For more information about the changes made to the XQP to support extended file names, see Section B.3. Valid file names could differ between interfaces.

2. Makes assumptions about the syntax of file specifications, such as the placement of delimiters and legal characters.

[1] As of OpenVMS Version 7.2, no language RTLs have been upgraded to full support.

3. Makes assumptions about the case of file specifications. RMS no longer converts mixed and lowercase file specifications to uppercase in all cases. This could affect string matching operations.

4. Depends on the traditional directory depth (fewer than 8 levels).

4.1.3. No Support for ODS-5 Volumes

An application that uses internal knowledge of the file system, including knowledge of the contents of a directory and how file header data is structured on a disk cannot work correctly on an ODS-5 volume.

4.2. Upgrading an Application to Support Extended File Specifications

The following sections describe the changes necessary to upgrade the level of support for Extended File Specifications. Note that you must first ensure that the application meets the default support level before you can upgrade it to the full support level.

Note

If you are *not* using the RMS or QIO interfaces to perform disk I/O, the Extended File Specifications support level of your application depends on whether the interface you are using (such as a language run-time library) provides full support.

4.2.1. Upgrading to Default Support

To upgrade an application to provide default support for Extended File Specifications, you must ensure that it minimally supports both the ODS-5 volume structure and extended file naming as recommended in Sections 4.2.1.1 and 4.2.1.2, respectively. Default support is defined in Section 2.1.2.

4.2.1.1. Providing Support for ODS-5

Applications that do not support the new ODS-5 volume structure do not operate successfully on these volumes even if they encounter only traditional file specifications.

If an application does not work properly on an ODS-5 volume, examine the application for the following:

- *Does the application use physical or logical I/O to bypass the file system when accessing the volume, or does it access metadata files such as BITMAP.SYS directly?* These applications are usually system programs, such as disk defragmenters, or programs that try to avoid overhead by accessing the disk directly. These applications rely on specific knowledge of the file or directory structure on the disk, which has changed with introduction of the ODS-5 structure.

  Recommendation: Applications should use documented interfaces and structures whenever possible.

- *Does the application access and interpret the contents of directory files directly?* If so, the application may fail when it encounters a directory that contains extended file names.

  Recommendation: Modify the application to use the search functions provided with the RMS [2] or QIO interface, or with LIBRTL routines such as LIB$FIND_FILE.

[2]RMS directory caching size has drastically increased on OpenVMS Alpha Version 7.2, greatly improving performance of the $SEARCH system service with large directories.

4.2.1.2. Providing Support for Extended File Naming

If an application does not handle extended names successfully, examine the application for any the following:

- *Does the application attempt to parse or assume knowledge of the syntax of a file specification?* For example, the application might search for a bracket ([) to locate the beginning of a directory specification, or for a space character to mark the end of a file specification.

  Recommendation: The application should rely on RMS to determine whether a file specification is legal rather than pretesting the actual name. Use the NAML_NODE, NAML_DEV, NAM L_DIR, NAML_TYPE, and NAM$L_VER fields of the NAM block or SYS$FILESCAN to retrieve this information.

- *Does the application attempt to determine if two file names are the same by doing a string comparison?* Because file names are case-insensitive, and because there are several ways to represent some characters, a string compare may fail even though two strings represent the same file.

  Recommendation: See the example program [SYSHLP.EXAMPLES]FILENAME_COMPARE.C for a way to use the new system service $CVT_FILENAMES to compare filenames.

- *Does the application depend on the NAM$V_DIR_LVLS bits in the NAM$L_FNB field to determine how many directory levels there are in the current file specification?* Because there are only three bits in this field, it can only specify a maximum of eight levels. Applications seldom use these bits; they are mainly used by RMS when a NAM is specified as a related file specification.

  Recommendation: Starting with OpenVMS Version 7.2, there is a new larger field available in both the NAM and the NAML blocks, NAM$W_LONG_DIR_LEVELS. Use this field to locate the correct number of directory levels.

- *Does the application rely on the NAM$V_WILD_UFD and SFD1 - SFD7 bits to determine where there are wildcard directories?* Because there are only eight of these bits, they can only report wildcards in the first eight directory levels. Applications seldom use these bits; they are mainly used by RMS when a NAM is specified as a related file specification.

  Recommendation: Starting with OpenVMS Version 7.2, there is a new field available in both the NAM and NAML block, NAML$W_FIRST_WILD_DIR. Use this field to locate the highest directory level where a wildcard is to be found.

- *Does the application use the QIO interface to the file system and specify or request a file name from QIO directly?* The QIO interface requires that an application specify explicitly that it understands extended file names before it will accept or return the names. In addition, the file name format for extended file names is not identical between RMS and the QIO interface. Additionally, some file names may be specified in 2-byte Unicode (UCS-2) characters. Your application must be capable of dealing with 1 character that spans 2 bytes.

  Recommendations: Most applications that use the QIO interface also use RMS to parse file specifications and retrieve the file and directory ID for the file. They then use these ID values to access the file with the QIO interface. This method of access continues to work with extended names. VSI recommends changing to this method to fix the problem.

  You can also obtain the name that the QIO system uses from the NAML$L_FILESYS_NAME field of a NAML block, or use the new system service (SYS$CVT_FILENAME) to convert between the RMS and the QIO file name. In this case, you will also need to provide an expanded FIB block to

the QIO service to specify that your application understands extended names, expand your buffers to the maximum size, and prepare to deal with 2-byte Unicode characters.

4.2.2. Upgrading to Full Support

Some OpenVMS applications, such as system or disk management utilities, may require full support for Extended File Specifications. Typically, these are utilities that must be able to view and manipulate all file specifications without DID or FID abbreviation. To upgrade an application so that it fully supports all the features of Extended File Specifications, do the following:

1. Convert all uses of the RMS NAM block to the new NAML block.

2. Expand the input and output file name buffers used by RMS. To do this, use the NAML long_expanded and long_resultant buffer pointers (NAML$L_LONG_EXPAND and NAML $L_LONG_RESULT) rather than the short buffer pointers (NAML$L_ESA and NAML$L_RSA), and increase the buffer sizes from NAM$C_MAXRSS to NAML$C_MAXRSS.

3. If long file names (greater than 255 bytes) are specified in the FAB file name buffer field (FAB $L_FNA), use the NAML long_filename buffer field (NAML$L_LONG_FILENAME) instead. If long file names are specified in the FAB default name buffer field (FAB$L_DNA), use the NAML default name buffer field (NAML$L_LONG_DEFNAME) instead.

4. If you use the LIB$FIND_FILE, LIB$RENAME or LIB$DELETE routines, set LIB $M_FIL_LONG_NAMES in the `flags` argument (`flags` is a new argument to the LIB $DELETE routine). Note that you can use the NAML block in place of the NAM block to pass information to LIB$FILE_SCAN without additional changes.

5. If you use the LIB$FID_TO_NAME routine, the descriptor for the returned file specification may need to be changed to take advantage of the increased maximum allowed of 4095 (NAML $C_MAXRSS) bytes.

6. If you use the FDL$CREATE, FDL$GENERATE, FDL$PARSE, or FDL$RELEASE routine, you must set FDL$M_LONG_NAMES in the *flags* argument.

7. Examine the source code for any additional assumptions made internally that a file specification is no longer than 255 8-bit bytes.

Appendix A. Setting Users' Expectations of Extended File Specifications

Extended File Specifications enables users to use Windows-style file specifications in an OpenVMS environment. Among the ways you can help users become accustomed to Extended File Specifications is to explain some differences they might see between ODS-2 and ODS-5 file names. These differences become most apparent when users change from ODS-2 to ODS-5 styles.

Following are usage notes that you, as system manager, might pass on to users. These notes have been divided into the following categories:

- New Extended File Specifications characteristics
- ODS-2 and ODS-5 used together
- Architecture-related notes

A.1. New Extended File Specifications Characteristics

The following notes discuss issues related to new HSF characteristics that are unfamiliar to users.

Be Aware of Volume Structure

So that you can place ODS-5 files on ODS-5 volumes, make sure you know whether a disk is an ODS-2 or ODS-5 volume.

You can display the type of volume by issuing commands like the following:

```
$ SHOW DEVICE DKA500:/FULL

  Disk AABOUT$DKA500:, device type RZ25, is online, allocated, deallocate
  on dismount, mounted, file-oriented device, shareable.

    Error count                    0    Operations completed 155
    .
    .
    .
  Volume Status:  ODS-5, subject to mount verification, file high-water
  marking, write-back caching enabled.

$ SHOW DEVICE DKA200:/FULL

  Disk AABOUT$DKA200:, device type RZ25, is online, allocated, deallocate
  on dismount, mounted, file-oriented device, shareable.

    Error count                    0    Operations completed 232
    .
    .
```

```
    .

  Volume Status:  ODS-2, subject to mount verification, file high-water
  marking, write-back caching enabled.
```

After each command, the “Volume Status:” displayed indicates whether the volume is ODS-5 or ODS-2.

Do Not Use Extended File Names on ODS-2 Volumes

You cannot create a file with an extended file name on an ODS-2 volume.

In the following example, DKA200 is an ODS-2 volume, and the parse style is EXTENDED, which causes RMS to return an error message.

```
$ SET DEFAULT DKA200:[TEST]
$ CREATE x.x.x.x
%CREATE-E-OPENOUT, error opening DAK200:[TEST]X^.X^.X.X; as output
-RMS-E-CRE, ACP file create failed
-SYSTEM-W-BADFILEVER, bad file version number
```

Case Is Determined by the First Instance of an Extended File Name

On an ODS-5 volume, the case for all versions of a file name is identical; the case is preserved as the file name was first created.

In the following example, DKA500 is an ODS-5 disk.

```
$ SET DEFAULT DKA500:[TEST]
$ SET PROCESS /PARSE_STYLE=EXTENDED
$ CREATE myfile.txt
Ctrl/Z
$ CREATE MYFILE.TXT
Ctrl/Z
$ DIRECTORY

Directory DKA500:[TEST]

myfile.txt;2        myfile.txt;1
```

Be Aware of Extended File Specifications Case Preservation and Case Blindness

Although an ODS-5 volume preserves the case of a file as it is first entered, file searches are performed in a case-blind manner. Similarly, users must also be careful when they do comparisons, such as when they use DCL string functions such as .EQS. and F$LOCATE in a DCL command procedure.

The following example demonstrates the importance of case-blind matching of file names in DCL. In the program, notice that you specify no argument to do a case-sensitive match but that you specify an argument to do a case-blind match.

This program uses F$SEARCH to find all files that have a file type of “.TXT.” Because RMS (and thus F$SEARCH) does case-blind matching, F$SEARCH also finds files with the file type “.txt.” F $SEARCH then uses F$LOCATE to search the file name for “TEST.” Because F$LOCATE does case-sensitive comparisons, it fails to match unless you first change the string to uppercase.

```
$ case_blind = 0
$ if p1 .nes. "" then case_blind = 1 ❶
$loop:
$  file = f$search("*.TXT;") ❷
$  if file .eqs. "" then goto not_found
$  write sys$output "Search returns " + file
$  if case_blind .eq. 1 then file = f$edit(file,"UPCASE") ❸
$  if (f$locate("TEST",file) .ne. f$length(file)) then goto found ❹
$  goto loop
$found:
$   write sys$output "Found a file matching TEST"
$   exit
$not_found:
$   write sys$output "Did not find file matching TEST"
$   exit
```

Following are explanations of the callouts in the example.

❶ Set “case_blind” to 1 if there is an argument (which requests the program to do a case-blind comparison).
❷ Get a file ending in “.TXT” or “.txt” (because F$SEARCH is case-blind).
❸ If a case-blind comparison was selected in Step 1, change the file name to uppercase to make a case-blind comparison.
❹ If F$LOCATE finds a file, it will go to “found:.”

In the following example, the search program performs a case-sensitive search and does not find a match.

```
$ @test
Search returns DKA300:[FISHER]test.txt;1
Did not find file matching TEST
```

In the following example, the search program performs a case-blind search and does find a match.

```
$ @test case-blind
Search returns DKA300:[FISHER]test.txt;1

Found a file matching TEST
```

Abbreviated and Full Directories Listed Separately with CONDENSED File Names

Some system utilities and DCL commands, such as DIRECTORY, have a style switch to control how they display file names. If the style is CONDENSED, file names up to 255 bytes in length are displayed. When a file specification reaches the 255-byte limit, the directory name is abbreviated to a directory ID (DID).

The following example shows a CONDENSED directory name. The DIRECTORY command considers a DID abbreviated directory name as different from the unabbreviated directory name and therefore generates a separate header when the abbreviation occurs.

```
$ DIR/STYLE=CONDENSED

Directory DKA300:
[DEEPER.aaaa.bbbb.cccc.dddd.eeee.ffff.gggg.hhhh.iiii._ten.aaaa.
bbbb.cccc.dddd.eeee.ffff.gggg.hhhh.iiii._ten.aaaa.bbbb.cccc.dddd.eeee.ffff.gggg.
```

```
hhhh.iiii._ten.aaaa.bbbb.cccc.dddd.eeee.ffff.gggg.hhhh.iiii._ten]❶

aaaa.txt;1

Total of 1 file.

Directory DKA300:[528,7036,0]❷

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.txt;1

Total of 1 file.

Grand total of 2 directories, 2 files.❸
```

❶ With the CONDENSED style, if the combination of the directory name and file name does not exceed 255 bytes, the directory name is not shortened to a DID.
❷ With the CONDENSED style, if the combination of the directory name and file name exceeds 255 bytes, the directory name is shortened to a DID.
❸ When you issue a DIRECTORY command that displays both a full and an abbreviated directory format for the same directory name, DIRECTORY counts these as two different directories.

For more information about DIRECTORY commands, see the *VSI OpenVMS DCL Dictionary*.

Be Aware of Extended File Specifications as Equivalence Names

The Extended File Specifications escape character (^), is not used in a logical name equivalence string. When you define a logical name for an extended file name that requires escape characters, omit the escape characters from the extended file name in the DEFINE command. For example:

```
$ define xxx a&b
$ dir xxx

Directory DKA500:[EXTENDED_FILES]

a^&b.txt;1

Total of 1 file.
```

A.2. ODS-2 and ODS-5 Used Together

The following notes discuss issues related to using ODS-2 and ODS-5 together in a cluster.

Use Traditional File Names in a Mixed-Volume Environment

To avoid ODS-2 and ODS-5 file name incompatibility if you are working with both ODS-2 and ODS-5 volumes, and to assure backward compatibility with prior versions of OpenVMS, use only ODS-2 traditional file names.

Error Messages Can Vary Depending on Parse Style

Error messages displayed to users might vary depending on the parse style. Syntax errors that were formerly detected at the DCL-level are now passed on to the file system level, RMS and XQP, for

example, if the parse style is EXTENDED. As a result, the messages users receive for file syntax errors might be slightly different depending on the parse style and volume structure.

Following are examples of varying error messages.

- Examples of TRADITIONAL and EXTENDED styles on an ODS-5 volume:

```
$ SHOW DEVICE DKA500:/FULL

  Disk AABOUT$DKA500:, device type RZ25, is online, allocated,
 deallocate
  on dismount, mounted, file-oriented device, shareable.

    Error count                    0    Operations completed 155
    .
    .
    .
  Volume Status:  ODS-5, ❶ subject to mount verification, file high-
water
  marking, write-back caching enabled.

$ SET PROCESS /PARSE_STYLE=TRADITIONAL ❷
$ OPEN /WRITE FILE z.z.z.z
%DCL-W-PARMDEL, invalid parameter delimiter - check use of special
characters \.Z\ ❸
$ SET PROCESS /PARSE_STYLE=EXTENDED ❹
$ OPEN /WRITE FILE z.z.z.z
$ ❺
```

❶ The volume is ODS-5.
❷ The parse style is set to TRADITIONAL.
❸ DCL returns an error on some ODS-5 file names such as this one.
❹ The parse style is set to EXTENDED.
❺ DCL creates the file.

- Examples of TRADITIONAL and EXTENDED styles on an ODS-2 volume:

```
  Disk AABOUT$DKA200:, device type RZ25, is online, allocated,
 deallocate
  on dismount, mounted, file-oriented device, shareable.

    Error count                    0    Operations completed 232
    .
    .
    .

  Volume Status:  ODS-2, ❶ subject to mount verification, file high-
water
  marking, write-back caching enabled.

$ SET PROCESS /PARSE_STYLE=TRADITIONAL ❷
$ OPEN /WRITE FILE z.z.z.z
%DCL-W-PARMDEL, invalid parameter delimiter - check use of special
characters \.Z\ ❸
$ SET PROCESS /PARSE_STYLE=EXTENDED ❹
$ OPEN /WRITE FILE z.z.z.z
%DCL-E-OPENIN, error opening
-RMS-E-CRE, ACP file create failed ❺
```

```
-SYSTEM-W-BADFILEVER, bad file version number
```

❶ The volume is ODS-2.
❷ The parse style is set to TRADITIONAL.
❸ DCL returns an error message.
❹ The parse style is set to EXTENDED.
❺ DCL allows the file name, but XQP returns an error.

- Examples of different error messages for the same syntax error:

```
$ SHOW DEVICE DKA500:/FULL

  Disk AABOUT$DKA500:, device type RZ25, is online, allocated,
 deallocate
  on dismount, mounted, file-oriented device, shareable.

    Error count                    0    Operations completed 155
    .
    .
    .
  Volume Status:  ODS-5, ❶ subject to mount verification, file high-
water
  marking, write-back caching enabled.

$ SET PROCESS /PARSE_STYLE=TRADITIONAL ❷
$ CREATE a^<b.c

%DCL-W-PARMDEL, invalid parameter delimiter - check use of special
characters
 \^\ ❸

$ SET PROCESS /PARSE_STYLE=EXTENDED ❹
$ CREATE a^<b.c

%CREATE-E-OPENOUT, error opening a^<b.c as output
-RMS-F-SYN, file specification syntax error ❺
```

❶ The volume is ODS-5.
❷ The parse style is set to TRADITIONAL.
❸ DCL returns an error message for a syntax error.
❹ The parse style is set to EXTENDED.
❺ RMS returns a different error message for the same syntax error ("<" is not allowed even in an extended file name).

Be Aware of Implicit File Name Output

Be wary of defaults when you allow utilities to create output files based on the file name being processed. Be sure you know where a file is being placed so you will not inadvertently try to place an extended file on an ODS-2 volume.

The following examples show files being placed somewhere you might not expect:

- An error results if an application or a utility attempts to write an ODS-5 extended file name to an ODS-2 (DKA200:) volume; for example:

```
$ SHOW DEFAULT
```

```
  DKA200:[DOREO]
$ DUMP /OUTPUT DKA500:[DOREO]This^_is^_a^_file.Dat
%DUMP-E-OPENOUT, error opening DKA200:[DOREO]THIS^_IS^_A^_FILE.DMP;as
output
-RMS-E-CRE, ACP file create failed
-SYSTEM-W-BADFILENAME, bad file name syntax
```

The output file specified with the /OUTPUT qualifier defaults to the same name as the input file, with .DMP as the file type, in the default directory. While the input file specification is an extended name on an ODS-5 volume, the .DMP file must have a traditional name, because it will be written to an ODS-2 volume. As a result, an error occurs.

- A batch command file will fail to execute if the following conditions apply:
  - a log file was requested implicitly or explicitly and
  - the implicit or explicit log file specification would have an extended file name (that is, the name would be non-ODS-2-compliant) and
  - the log file would be created on an ODS-2 volume.

  The batch command file does not execute because a log file cannot be created. Most frequently, this situation occurs when the logical name SYS$LOGIN refers to an ODS-2 volume; this is because log files are implicitly created on the SYS$LOGIN device. In addition, if notification is disabled, you are not notified that your batch job did not execute.

  To avoid the problem, use the /LOG= qualifier and an ODS-2-compliant log file specification when you submit command files with extended file names.

A.3. Architecture-Related Notes

The following notes discuss Extended File Specifications issues related to system architecture.

Extended File Names Are Not Visible from a VAX System

Although you can mount ODS-5 volumes on a VAX, if you log in to a VAX system, extended file names are not visible. In their place, you will see a pseudoname:

- On VAX, if you attempt to display a file name that contains 2-byte Unicode characters, the pseudoname displayed is \PUNICODE\.???
- Any other name that is not a legal ODS-2 name is displayed as \PISO_LATIN\.???

For example, following are listings of the same directory as they appear on Alpha and VAX systems:

- On an Alpha system:

  ```
  $ DIRECTORY DPA100:[TEST]

  Directory DPA100:[TEST]

  Accounting^_data.lis;1                  atest.txt;1
  ```

- On an VAX system:

```
$ DIRECTORY DPA200:[TEST]

Directory DPA200:[TEST]

\PISO_LATIN\.???                          ATEST.TXT
```

In addition, the directory depth on a VAX is limited to 8 (or 16, using rooted logicals).

A.4. Restrictions

The following topic describes a restriction when using extended file names.

Tilde (~) As First Character in a File Name

The VSI C Run Time Library (CRTL) allows a programmer to specify both Unix-style and VMS-style file specifications to routines such as creat() and fopen().

In Unix file specifications, a tilde (~) in the first character of a pathname represents the user's home directory. However, in an OpenVMS extended file name, a tilde is legal anywhere in a file name or directory name.

To preserve backward compatibility, the CRTL will continue to interpret a leading tilde (~) to mean the user's home directory. To pass an OpenVMS file name that begins with a tilde (~) to a CRTL routine that accepts Unix-style file specifications, specify the tilde preceded by the escape character (^). For example, ^~.

The following VSI CRTL functions accept OpenVMS extended file names and require this syntax for a leading tilde (~) in the file specification:

.create
.fopen
.freopen
.open
.stat

Appendix B. Technical Information

This appendix duplicates technical information that appears in other parts of the OpenVMS documentation.

B.1. System Services Changes

This section describes the following system services:

- New services:
  - $SET_PROCESS_PROPERTIESW
  - $CVT_FILENAME
- Changed services:
  - $CREPRC
  - $GETJPI
  - $SETDDIR

B.1.1. New Services

$SET_PROCESS_PROPERTIESW System Service (Alpha Only)

$SET_PROCESS_PROPERTIESW System Service (Alpha Only) — The $SET_PROCESS_PROPERTIESW system service sets a simple value associated with a process.

Format

$SET_PROCESS_PROPERTIESW mbz1 ,mbz2 ,mbz3 ,property ,value, prev_value

C Prototype:

```
int sys$set_process_properties(
        unsigned int mbz1,
        unsigned int mbz2,
        unsigned int mbz3,
        unsigned int property,
        unsigned __int64 value,
        unsigned __int64 *prev_value);
```

Arguments

mbz1,mbz2,mbz3

Reserved for future use by VSI. Must be specified as 0.

property

OpenVMS usage:	integer
type:	longword (unsigned)

access:	read only
mechanism:	by value

A constant that selects which property to set.

Valid values for property are defined by the $PPROPDEF macro as shown in Table B.1.

Table B.1. Property Code Descriptions

Property Code	Description
PPROP$C_PARSE_STYLE_TEMP:	The type of command parsing to use. This value will be set only for the life of the image. The value will revert to the permanent style on image rundown. Valid values are: PARSE_STYLE $C_TRADITIONAL and PARSE_STYLE $C_EXTENDED.
PPROP$C_PARSE_STYLE_PERM:	The type of command parsing to use. This value will be set for the life of the process unless the style is set again. Valid values are: PARSE_STYLE $C_TRADITIONAL and PARSE_STYLE $C_EXTENDED.

value

OpenVMS usage:	integer
type:	quadword (unsigned)
access:	read
mechanism:	by value

A quadword value to which to set the property.

prev_value

OpenVMS usage:	access_mode
type:	quadword (unsigned) address of a quadword value
access:	write
mechanism:	by reference

The address of a quadword that will receive the previous value of the property.

Required Access or Privileges

None.

Required Quota

None.

Related Services

$GETJPI

Condition Values Returned

SS$NORMAL	The service completed successfully.
SS$ACCVIO	Access violation.

$CVT_FILENAME System Service (Alpha Only)

$CVT_FILENAME System Service (Alpha Only) — Converts a string from RMS format to file-system (ACP-QIO) format or from file-system (ACP-QIO) format to RMS format.

Format

```
SYS$CVT_FILENAME   cvttyp ,srcstr ,inflags ,outbuf ,outlen ,outflags
```

C Prototype:

```
int sys$cvt_filename (unsigned int cvttyp,
                      void *srcstr,
                      unsigned int inflags,
                      void *outbuf,
                      unsigned short int *outlen,
                      unsigned int *outflags);
```

Arguments

cvttyp

OpenVMS usage:	unsigned_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

Longword value that indicates whether the conversion is to be from RMS format to ACP-QIO format or vice versa.

There are two legal values for this parameter, represented by the symbols CVTFNM $C_ACPQIO_TO_RMS and CVTFNM$C_RMS_TO_ACPQIO, which are defined by the $CVTFNMDEF macro.

srcstr

OpenVMS usage:	string of bytes or words
type:	string of bytes or words
access:	read only
mechanism:	by 32-bit descriptor--fixed length string descriptor

String to be converted by the service.

If the conversion is to be from RMS format to ACP-QIO format, *srcstr* is an ISO Latin-1 or VTF-7-encoded character string. If the conversion is to be from ACP-QIO format to RMS format, *srcstr* is a string of byte-width or word-width characters.

The descriptor length field indicates the length of the input string in bytes, whether the characters are byte-width or word-width.

The `srcstr` argument is the 32-bit address of a descriptor that points to this string.

inflags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

Longword flag mask indicating characteristics of the input string.

For conversion from RMS format to ACP-QIO format, only the CVTFNM$V_NO_DELIMITERS flag is valid.

For conversion from ACP-QIO format to RMS format, legal flags are CVTFNM$V_WORD_CHARS and CVTFNM$V_NO_DELIMITERS (defined by the $CVTFNMDEF macro). The flag descriptions are shown in Table B.2.

Table B.2. Flag Descriptions

Flag	Description
CVTFNM$V_WORD_CHARS	Input source string contains word-width UCS-2 characters (ACPQIO_TO_RMS).
CVTFNM$V_NO_DELIMITERS	Input source string should be treated as an arbitrary string (such as a subdirectory name) rather than as a filename that contains (or should contain) dots or semicolons as type and version delimiters.

outbuf

OpenVMS usage:	string of bytes or words
type:	string of bytes or words
access:	write only
mechanism:	by 32-bit descriptor--fixed-length string descriptor

The buffer into which the converted string is to be written.

If the conversion is from RMS format to ACP-QIO format, the string may consist of byte-width ISO Latin-1 characters or word-width UCS-2 characters, depending upon the characters in the source string. (If any character in the source string requires a word to represent, then all characters in the output buffer will be of word width.)

If the conversion is from ACP-QIO format to RMS format, then the output string will consist of ISO Latin-1 and VTF-7 characters, in RMS canonical form. (Refer to the *VSI OpenVMS Guide to OpenVMS File Applications*.)

For ACPQIO_TO_RMS conversion, if the output string is composed of word-width characters, the CVTFNM$V_WORD_CHARS flag in the `outflags` flag mask will be set.

The `outbuf` argument is the 32-bit address of a descriptor pointing to a buffer writable in the mode of the caller.

outlen

OpenVMS usage:	word_unsigned
type:	word (unsigned)
access:	write only
mechanism:	by 32-bit reference

The *outlen* argument is the 32-bit address of a (16-bit) word writable in the mode of the caller.

outflags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	write only
mechanism:	by 32-bit reference

Longword flag mask in which the service sets or clears flags to indicate characteristics of the output string.

For an RMS_TO_ACPQIO conversion, SYS$CVT_FILENAME sets the bit corresponding to CVTFNM $V_WORD_CHARS (defined by the $CVTFNMDEF macro) if the characters of the converted string are one word (rather than one byte) wide. If the characters of the converted string are one byte wide, the service clears the CVTFNM$V_WORD_CHARS bit. All other bits are cleared by an RMS_TO_ACPQIO conversion.

The *outflags* argument is the 32-bit address of a 32-bit flag mask writable in the mode of the caller.

Description

This service is intended to provide conversion of a filename(1) or of a subdirectory name(2) between the RMS format (as seen at the RMS interface) and ACP-QIO format (as stored on-disk). Prior to Version 7.2, these representations were the same. This is not necessarily the case for extended (ODS-5) filenames. (Refer to the *VSI OpenVMS Guide to OpenVMS File Applications* for details on ODS-5 file specifications.)

1. A filename consists of a file name, a file type, and a file version.
2. A subdirectory name is a string to which ".DIR;1" may be appended to form a directory file name, as stored on-disk.

Depending upon the value of *cvttyp*, the service will perform conversion of a string from RMS format to ACP-QIO format or from ACP-QIO format to RMS format.

The source string is described by the argument *srcstr*, the output buffer is described by the argument *outbuf*, and the resultant string length is written to the argument *outlen*.

If any of the source string falls within the address range of the output buffer, the output string is unpredictable.

RMS-to-ACPQIO Conversion:

A string described by the *srcstr* descriptor argument is converted to an ISO Latin-1 or UCS-2 string with each character represented in a form that can be passed to the ACP-QIO via the $QIO service.

If the CVTFNM$V_NO_DELIMITERS input flag is clear, the source string will be scanned, and, if necessary, a dot and a semicolon will be inserted or appended as though a $PARSE were done with no

default name, type, or version fields supplied. If the scan detects any delimiters indicating the presence of fields other than name (without FID), type, or version, a syntax error will be returned.

If the CVTFNM$V_NO_DELIMITERS input flag is set, individual characters will be validated and converted to their on-disk form. However, no scan is done to determine if type and version delimiters are present, and no delimiters are added.

A percent sign (%) that is not preceded by the escape character (^) is converted to a question mark. An ISO Latin-1 character that is preceded by the escape character (^) is converted to the corresponding ISO Latin-1 character. A VTF-7 character (for example, ^U1234) that is preceded by the escape character (^) is converted to a UCS-2 character (for example, 0x1234).

If any character requires UCS-2 (word-width character) representation, all characters are represented in the output string with UCS-2. If no character requires word-width character representation, all characters are represented in the output string with ISO Latin-1 (byte-width) characters.

Valid characters are those that are legal in an RMS filename (file name, file type, and file version) or in an RMS subdirectory name. For example, directory delimiters “[” and “]” are not legal, unless preceded by the escape character (^).

ACPQIO-to-RMS Conversion:

The string described by the *srcstr* descriptor argument is converted to the RMS canonical form for that string.

If the CVTFNM$V_NO_DELIMITERS input flag is clear, the source string must contain at least one semicolon, and, to the left of the semicolon, at least one dot. If it does not, RMS$_SYN (syntax error) is returned. In the output string, all dots and semicolons other than those two are preceded by the RMS escape character (^).

If the CVTFNM$V_NO_DELIMITERS input flag is set, any dot or semicolon encountered is preceded in the output string by the RMS escape character (^).

The CVTFNM$V_WORD_CHARS flag of the *inflags* argument indicates whether the input string is to be interpreted as having byte-width (ISO Latin-1) or word-width (UCS-2) characters. If the argument indicates word-width characters, but the input length value is an odd number, a syntax error is returned.

Question marks are converted to percent signs; percent signs are preceded by the escape character (^). UCS-2 characters are converted to VTF-7 characters. All characters will be represented in RMS canonical form.

Required Access or Privileges

None.

Required Quota

None.

Related Services

None.

Condition Values Returned

SS$NORMAL	The service completed successfully.

SS$_BADPARAM	Unrecognized conversion type, extraneous input flags set, or zero-length input string.
SS$_INSFARG	Not enough arguments provided.
SS$_TOO_MANY_ARGS	Too many arguments provided.
RMS$_SYN	The service could not translate one or more characters in the strings described by the srcstr argument, the input string has word-width characters but odd byte-length (ACPQIO_TO_RMS only), or the CVTFNM $V_NO_DELIMITERS input flag was clear and the input string did not contain both type and version delimiters.
SS$_BUFFEROVF	The output buffer was not large enough to accommodate the converted string.

B.1.2. Changed Services

B.1.2.1. $GETJPI System Service

There are two new item codes for this system service. They are:

JPI$_PARSE_STYLE_PERM JPI$_PARSE_STYLE_IMAGE

These return the values that were set by $SET_PROCESS_PROPERTIES, that can be either PARSE_STYLE$C_TRADITIONAL or PARSE_STYLE$C_EXTENDED. Return length is one byte for each one.

B.1.2.2. $CREPRC System Service

There is a new flag allowed in the *stsflg* parameter:

PRC$M_PARSE_EXPANDED

This sets the PARSE_STYLE_PERM and the PARSE_STYLE_IMAGE properties for the new process to EXPANDED.

B.1.2.3. $SETDDIR System Service

The following text has been added to the system service description:

On Alpha systems, the Set Default Directory service attempts to replace the default directory string with a DID if the length of the resulting default directory exceeds 255 bytes. If this happens, then in addition to the normal syntax check, the entire path to that specification, including the device, is verified and must exist for the call to succeed.

B.2. Record Management Services (RMS) Changes

OpenVMS Record Management Services (RMS) has been modified to support Extended File Specifications. The following sections describe syntax and semantics changes and RMS data structure changes.

B.2.1. Overview of Record Management Services Changes

To support Extended File Specifications, the Record Management Services (RMS) have been enhanced to provide the following functions through existing interfaces:

- Support for a wider range of characters in a file name, extension, and directory
- Access to file specifications with extended characters
- Support for directory structures deeper than eight levels
- Access to file specifications longer than 255 bytes through the NAM block with some restrictions in functionality
- Access and complete specification of file specifications longer than 255 bytes by callers who are aware of the new naming characteristics through a new interface (NAML block)

B.2.1.1. Extended File Specification Support

On ODS-5 volumes, RMS can manipulate file names and subdirectory specifications of up to 255 8-bit or 16-bit characters in length. RMS can handle a total path name 512 8-bit or 16-bit characters in length.

Prior to OpenVMS Alpha Version 7.2, the NAM block interface could pass file specifications of up to 255 bytes each (including the resultant file specification). The following sections describe the changes that allow for passing longer file specifications and that provide compatibility with applications using the NAM block interface prior to this release.

B.2.1.2. Additional Characters

On ODS-5 volumes, RMS supports access to files and directories with names that contain arbitrary 8-bit characters, except for the C0 control set (hexadecimal 00 through 1F) and the following characters:

Double quotation marks (")
Asterisk (*)
Backslash (\)
Colon (:)
Left and right angle brackets (< >)
Slash (/)
Question mark (?)
Vertical bar (|)

Note that this explicitly includes both the C1 character set (hex 80-9F) as well as graphical and other characters between 9F and FF. This allows the entire ISO Latin-1 character set (with the 7-bit character exclusions noted above) and any defined Unicode character.

B.2.1.3. Deeply Nested Directory Support

Under Extended File Specifications on Alpha, RMS supports deep nesting of up to 255 directories, with the restriction that the total directory specification must be no longer than 512 8-bit or 16-bit characters. The deep nesting of directories is also supported on ODS-2 disks.

B.2.2. Syntax and Semantics Changes

The following sections describe new RMS file specification syntax and semantics features. See the *VSI OpenVMS Guide to OpenVMS File Applications* for more information about using RMS with Extended File Specifications.

B.2.2.1. Use of Hyphen as First File Name Character

Prior to OpenVMS Version 7.2, RMS documentation recommended against creating files with names that begin with a hyphen (minus sign).

On Alpha systems, the Extended File Specifications changes to RMS allow you to use a hyphen anywhere in a file name or a directory name. If a directory name containing a hyphen is ambiguous, that is, if it could be interpreted as referring to a parent directory, you must prefix the hyphen with the escape character (^) to accurately specify the file or directory.

B.2.2.2. Characters Accepted Directly

The set of characters valid through the RMS interface in a file specification (without any special escape character) is extended according to the following list. Note that these characters must not be preceded by the escape character circumflex (^)

- Upper and lowercase alphanumeric characters:

  ```
  A - Z, a - z, 0 - 9
  ```

- Special ASCII (7-bit) characters:

  ```
  $ -  _
  ```

- ISO Latin-1 characters in the range (hex) A0 to FF.

B.2.2.3. Characters That Require an Escape Character

- Escape (^) followed by a hexadecimal digit requires a second hexadecimal digit. The two following characters represent a hexadecimal value for an arbitrary 8-bit character.

  For example, ^20 represents a space.

- Escape followed by underscore (^_) or by a space represents one space.

- Escape followed by uppercase U (^U) indicates that the next four characters represent a hexadecimal value for an arbitrary 16-bit character.

- Each 2-byte Unicode character must be represented as a ^Uxxxx sequence.

- The following characters require an escape character when they are used as part of a file name on input to RMS and DCL. For the period (.) [1] and tilde (~) [2], the escape character is only required under some circumstances, detailed in their respective footnotes.

  ```
  Exclamation point (!)
  Pound sign (#)
  Ampersand (&)
  ```

[1]The escape character is required before a period in a directory name, is optional before a period in a file name, and must not be used for the period that delimits the file type. A period is not permitted in a file type.
[2]The tilde that is the leading character in a file name or directory may require an escape character.

```
Apostrophe (')
Left parenthesis (()
Right parenthesis ())
Plus sign (+)
Atsign (@)
Left brace ({)
Right brace (})
Period (.)[1]
Comma (,)
Grave accent (`)
Semicolon (;)
Left bracket ([)
Right bracket (])
Percent sign (%)
Circumflex (^)
Equal sign (=)
Tilde (~)[2]
```

B.2.2.4. Characters That Can Have an Escape Character

The following characters can be preceded by the escape character (^) on input to RMS or DCL, but need not be. For the period (.) [1] and tilde (~) [2] the escape character is only required under some circumstances, detailed in their respective footnotes.

```
Dollar sign ($)
Minus sign (-)
Period (.)[1]
Tilde (~)[2]
```

B.2.2.5. Reserved Escape Sequences

Sequences consisting of the escape character followed by any character not mentioned previously are reserved.

B.2.2.6. Canonical Form of File Specifications

In some cases, there are multiple ways to write the same characters. For example, ^20, ^ , and ^_ are all equivalent. When RMS outputs a file specification (as a resultant name, for example), it follows these rules to determine which form to use:

- Any character that cannot be represented with eight bits is represented as ^Uxxxx, where xxxx is four hexadecimal digits.
- Space is represented as the escape character followed by an underscore (^_).
- Other ISO Latin-1 8-bit characters that have no graphical representation or which are used for control functions by other OpenVMS software or by terminals is represented by an escape character followed by two hexadecimal digits (^xx). Otherwise, it is represented by its own character. The following 8-bit values are output as an escape character followed by two hexadecimal digits.

  ```
  7F (rubout)
  80-9F (C1 control characters)
  A0 (nonbreaking space)
  FF (Latin small letter y diaeresis)
  ```

- If the file specification is longer than 255 bytes and must be output through a NAM block, a DID or FID abbreviation is used.

- The following characters are output preceded by the escape character (^):

```
Exclamation point (!)
Pound sign (#)
Ampersand (&)
Apostrophe (')
Grave accent (`)
Left parenthesis (()
Right parenthesis ())
Plus sign (+)
Atsign (@)
Left brace ({)
Right brace (})
Period (.)
Comma (,)
Semicolon (;)
Left bracket ([)
Right bracket (])
Percent sign (%)
Circumflex (^)
Equal sign (=)
```

B.2.2.7. DID Abbreviation

With extended file names, some legal names will be too long for unmodified RMS applications or for DCL to handle, either because of many levels of long directory names or because of a long file name with escape characters in it. To maintain compatibility with applications using the traditional (or pre-Version 7.2) interfaces, shorter names that RMS can output to the application and that an application can input to RMS through the traditional interface will be generated.

DID abbreviation is the first step that RMS uses to create a generated name if the file specification is longer than 255 bytes. RMS attempts to generate a short enough name by abbreviating the directory with its directory ID (DID). For example:

```
DKA100:[5953,9,0]FOO.TXT;1.
```

Restrictions on DID Abbreviations

When RMS is processing a file specification that does not fit into a traditional (pre-Version 7.2) short output buffer, RMS can attempt to abbreviate the root or directory component by replacing the component with the directory ID of the lowest-level subdirectory in the component.

There are circumstances in which RMS will not be able to generate a DID-abbreviated root or directory component. For example, if the path to the lowest-level subdirectory includes a wildcard, RMS does not have a particular directory ID to use. (And RMS does not attempt to replace a portion of a root or directory component with a DID.)

When RMS is unable to sufficiently shorten a file specification by generating a DID-abbreviated root or directory, it proceeds to the next step, FID abbreviation.

B.2.2.8. FID Abbreviation

If the file specification is still too long after DID abbreviation, RMS next attempts to generate a short enough name by abbreviating the file with its File ID (a comma-separated sequence of decimal digit strings, surrounded by brackets) in the file name field.

In cases where the extension field is normally presented, a generated name includes the complete extension or drops the extension (including the period (.)), depending on whether there is space.

In cases where the version number is normally presented, a FID abbreviated file name includes the version. As a human-readable aid in recognizing files, when a FID abbreviation is generated, the name field also contains an initial subset of the actual file name. The subset consists of the first 38 characters of the file name (where escape counts as a character) followed by the tilde (~). No attempt is made to resolve ambiguities for files that differ only after the first 38 characters of their names.

Here is an example of a FID abbreviation:

```
LookAtWhatWeHave^!ThisIsAVery_long^.fi~[7254,30,0].txt;1
```

Restrictions on FID Abbreviations and DID Abbreviations

A FID abbreviated file name can be used for input to RMS, but only the FID abbreviation itself is significant to RMS. The subset file name, the type field, and the version are all ignored on input.

A FID cannot be used as input to the CREATE command (or any other file creation command or API) when creating a file. A DID cannot be used to create a directory. When a FID is used in a file specification, the file specification either initially contains a device and directory or it acquires them from defaults processing in RMS. If a file with the specified FID exists on the volume, it must exist within the specified directory; otherwise, RMS acts as though the file were not found.

B.2.3. RMS Data Structure Changes (Alpha Only)

This section lists the changes to the name (NAM) block. It also describes the new name (NAML) block that is used to specify file specifications longer than 255 bytes.

B.2.3.1. NAM Block

The file name block options field, NAM$B_NOP, has a new flag as shown in Table B.3.

Table B.3. New NAM$B_NOP Flag

Flag	Meaning
NAM$V_NO_SHORT_UPCASE	Set by the user to tell RMS not to convert the directory and file specification in the NAM $L_ESA buffer to uppercase.

The file name status bits field, NAM$L_FNB, has the new flags as described in Table B.4.

Table B.4. New NAM$L_FNB Flags

Flag	Meaning
NAM$V_DIR_LVLS_G7	Indicates that the number of directory levels are greater than 7. If this is set, NAM$V_DIR_LVLS is set to 7.
NAM$V_WILD_SFDG7	Indicates that a subdirectory greater than 7 contains a wildcard character. This field offset is summarized in the NAM$V_WILDCARD field offset.

The NAM has three new fields: NAMB_NMC, NAMW_FIRST_WILD_DIR, and NAM $W_LONG_DIR_LEVELS. The NAM$B_NMC field returns the flags described in Table B.5.

Table B.5. NAM$B_NMC Flag

Flag	Meaning
NAM$V_DID	Set by RMS if it found a DID-abbreviated directory in the root or directory name component of an input directory.
NAM$V_FID	Set by RMS if it found a FID-abbreviated file name in an input file specification.
NAM$V_RES_DID	Set by RMS if there is a DID-abbreviated directory in the short resultant or expanded buffer.
NAM$V_RES_FID	Set by RMS if there is a FID-abbreviated name in the short resultant or expanded buffer.
NAM$V_RES_ESCAPE	Set by RMS if there are any escape characters (^) in the short resultant or expanded buffer.
NAM$V_RES_UNICODE	Set by RMS if there is one or more ^U sequences in the short resultant or expanded buffer.

B.2.3.2. NAML Block

The NAML is a new block that can optionally take the place of a NAM block. The NAML has all the fields of the NAM, and additionally contains new fields to allow filespecs to be specified that are longer than 255 bytes.

Table B.6 describes the new fields for the NAML Block.

Table B.6. New Fields for the NAML Block

Extended Field Name	Size (bytes)	Meaning
NAML$L_FILESYS_NAME	4	File system name buffer address specified by the user. If RMS sets the NAML $V_FILESYS_NAME_UCS2 output flag, the output filename is in 2-byte characters, and everything that is in the file name including ASCII characters and delimiters are 2-byte characters. Otherwise, they are single byte characters.
NAML $L_FILESYS_NAME_ALLOC	4	File system name buffer allocated size specified by the user.
NAML $L_FILESYS_NAME_SIZE	4	File system name length returned by RMS.
NAML $L_LONG_DEFNAME_SIZE	4	Long default file specification string size specified as input. Equivalent to FAB$B_DNS (input only). Used only if FAB $L_DNA is set to -1, and FAB $B_DNS is set to 0.

Extended Field Name	Size (bytes)	Meaning
NAML$L_LONG_DEFNAME	4	Long default file specification string address specified as input. Equivalent to FAB$L_DNA (input only). Used only if FAB $L_DNA is set to -1, and FAB $B_DNS is set to 0.
NAML $L_LONG_FILENAME_SIZE	4	Long file specification string size. Equivalent to FAB$B_FNS (input only). Used only if FAB $L_FNA is set to -1, and FAB $B_FNS is set to 0.
NAML$L_LONG_FILENAME	4	Long file specification string address. Equivalent to FAB $L_FNA (input only). Used only if FAB$L_FNA is set to -1, and FAB$B_FNS is set to 0.
NAML$L_LONG_NODE_SIZE	4	Long node name string length.
NAML$L_LONG_NODE	4	Long node name string address.
NAML$L_LONG_DEV_SIZE	4	Long device string length.
NAML$L_LONG_DEV	4	Long device string address.
NAML$L_LONG_DIR_SIZE	4	Long directory string length.
NAML$L_LONG_DIR	4	Long directory string address.
NAML$L_LONG_NAME_SIZE	4	Long file name string length.
NAML$L_LONG_NAME	4	Long file name string address.
NAML$L_LONG_TYPE_SIZE	4	Long file type string length.
NAML$L_LONG_TYPE	4	Long file type string address.
NAML$L_LONG_VER_SIZE	4	Long file version string length.
NAML$L_LONG_VER	4	Long file version string address.
NAML $L_LONG_EXPAND_ALLOC	4	Long expanded string area size. Set by the caller to specify the size of the long expanded buffer.
NAML $L_LONG_EXPAND_SIZE	4	Long expanded string length. Set by RMS to show the length of the long expanded string.
NAML$L_LONG_EXPAND	4	Long expanded string area address. Set by the caller to point to the long expanded buffer.
NAML $L_LONG_RESULT_ALLOC	4	Long resultant string area size. Set by the caller to specify the size of the long resultant buffer.
NAML $L_LONG_RESULT_SIZE	4	Long resultant string length. Set by RMS to show the length of the long resultant string.

Extended Field Name	Size (bytes)	Meaning
NAML$L_LONG RESULT	4	Long resultant string area address. Set by the caller to point to the long resultant buffer.
NAML$L_INPUT_FLAGS	4	Additional flags specified as input to RMS, including NAML $V_NO_SHORT_OUTPUT, which is defined the table below.
NAML$L_OUTPUT_FLAGS	4	Additional status bits passed as output by RMS, including NAML $V_LONG_RESULT_ESCAPE and NAML $V_FILESYS_NAME_UCS2, which are defined in the table below.
NAML$Q_USER_CONTEXT	8	Caller can use this for any purpose. Will not be read or modified by RMS.

RMS reads the following flag from the NAML$L_INPUT_FLAGS field:

Flag	Meaning
NAML$V_NO_SHORT_OUTPUT	Set by the user to tell RMS not to fill in the NAM $L_ESA or NAM$L_RSA buffer.

RMS writes the following flags to the NAML$L_OUTPUT_FLAGS field:

Flag	Meaning
NAML$V_FILESYS_NAME_UCS2	Set by RMS if name pointed to by NAML $L_FILESYS_NAME consists of 6 2-byte Unicode characters.
NAML$V_LONG_RESULT_DID	Set by RMS if there is a DID-abbreviated directory in the long resultant or expanded buffer.
NAML$V_LONG_RESULT_ESCAPE	Set by RMS if there are any escape characters (^) in the long resultant or expanded buffer.
NAML$V_LONG_RESULT_FID	Set by RMS if there is a FID-abbreviated name in the long resultant or expanded buffer.
NAML$V_LONG_RESULT_UNICODE	Set by RMS if there is one or more ^U sequences in the long resultant or expanded buffer.

B.2.3.2.1. Validating the NAML Block

If the name block passed to RMS (see FAB$L_NAM) contains a block identifier (see NAML$B_BID) equal to NAML$C_BID, RMS performs the following validation checks:

1. NAML$B_BLN field is exactly equal to NAML$C_BLN.
2. NAML$L_LONG_RESULT_ALLOC and NAML$L_LONG_EXPAND_ALLOC are less than or equal to NAML$C_MAXRSS.

3. All unused fields (which have a symbolic name containing MBZ) contain zero. You can clear the entire structure before initializing any fields to meet this requirement.

If any of these validation checks fail, a RMS$_NAML error status is returned.

B.2.3.2.2. Using the NAM and NAML Block

The NAML has fields that are equivalent to all the NAM fields, plus 28 additional fields to accommodate longer file specifications. There are no FDL attributes for the NAML fields.

Many of the additional fields in the NAML correspond to NAM fields but allow longer names. For example, the fields NAML$L_LONG_EXPAND, NAML$L_LONG_EXPAND_ALLOC, and NAML $L_LONG_EXPAND_SIZE correspond to NAML_ESA, NAMB_ESS, and NAM$B_ESL, but allow names that are longer than 255 bytes. When there are fields that correspond this way, the original field is referred to as a "short field." The corresponding field is referred to as a "long field."

When RMS is writing information into fields in a NAML that have both a short and long version, RMS normally writes the equivalent information into both fields. If either the short field or the long field is too small to contain the information, RMS returns an error, though RMS first attempts to abbreviate specifications to allow them to fit in the short fields. You can prevent the error on the short fields by setting the flag NAML$V_NO_SHORT_OUTPUT, which instructs RMS not to write into the short fields. However, if you are using a NAML, RMS always uses the long fields. If you do not want RMS to use the long fields, you must use a NAM rather than a NAML.

When RMS is reading information from fields in a NAML that has both a short and a long version, RMS always reads from the long version. To cause RMS to read from the short fields, use a NAM rather than a NAML. The most common time that RMS reads from these fields is during a $SEARCH operation following a $PARSE, when RMS reads from the buffer pointed to by NAML $L_LONG_EXPAND for a NAML and NAM$L_ESA for a NAM. In addition, if a NAM or NAML is used as a related name block, RMS reads information from the buffer pointed to by NAML $L_LONG_RESULT for a NAML, or NAM$L_RSA for a NAM.

Because of these differences in the way RMS processes a NAM and a NAML, it is important that any code that might come in contact with the NAML be aware that it is a NAML and not a NAM. Several operations that a routine might do on a NAM will not work as expected on a NAML. For example, if a routine makes a copy of a NAML but uses the NAM$C_BLN constant as the length to copy, the result is an illegal NAML. If a routine replaces the buffers pointed to by NAM$L_ESA and NAM$L_RSA with the expectation that it can use the NAM without affecting the calling routine, it misses the buffers pointed to by NAML$L_LONG_EXPAND and NAML$L_LONG_RESULT.

For this reason, any API supplied by OpenVMS adheres to the rule that if it returned a NAM (either directly or indirectly through a FAB) in previous versions, it will not now start returning a NAML without some explicit action by the caller (usually setting a flag bit). We recommend that other APIs use the same rule. Further, if a NAML-aware application passes a NAML to an API, it must be prepared for that API to use only the NAM section (for example, it should not set the NAML $V_NO_SHORT_OUTPUT bit).

If you are writing a routine that is to accept either a NAM or a NAML, you should check the NAM $B_BID field to determine whether you have a NAM or a NAML; if you have a NAML, and you wish to read information that RMS has left in the NAML, look at the information in the long fields. In addition, if you wish to copy that NAM or NAML block to another location, you must be careful to use the length that is stored in the structure itself to determine how much to copy. You should use the NAM $B_BLN field in the structure you are copying rather than the NAM$C_BLN constant, because NAM $B_BLN contains the actual length of the structure. If you use the symbol NAM$C_BLN, which is the length of a NAM, it would be too short for a NAML.

B.2.3.2.3. Condition Values Returned

Table B.7 shows the additional condition values returned for various RMS services when using the NAML block.

Table B.7. RMS Condition Values Returned When Using NAML Block

RMS Service	Condition Value Returned
$CREATE	RMS$_NAML RMS$_NAMLESS RMS$_NAMLFSINV RMS$_NAMLFSSIZ RMS$_NAMLRSS
$DISPLAY	RMS$_NAMLESS RMS$_NAMLFSINV RMS$_NAMLFSSIZ
$ENTER	RMS$_NAML RMS$_NAMLFSINV RMS$_NAMLFSSIZ RMS$_NAMLRSS
$ERASE	RMS$_NAML RMS$_NAMLESS RMS$_NAMLFSINV RMS$_NAMLFSSIZ RMS$_NAMLRSS
$OPEN	RMS$_NAML RMS$_NAMLESS RMS$_NAMLFSINV RMS$_NAMLFSSIZ RMS$_NAMLRSS
$PARSE	RMS$_NAML RMS$_NAMLESS RMS$_NAMLFSINV RMS$_NAMLFSSIZ

RMS Service	Condition Value Returned
$REMOVE	RMS$_NAML RMS$_NALFSINV RMS$_NAMLFSSIZ RMS$_NAMLRSS
$RENAME	RMS$_NAML RMS$_NAMLESS RMS$_NAMLFSINV RMS$_NAMLFSSIZ RMS$_NAMLRSS
$SEARCH	RMS$_NAML RMS$_NAMLFSINV RMS$_NAMLFSSIZ RMS$_NAMLRSS

B.3. Files-11 XQP Changes

Note

The information about the file system contained in this section currently appears only in this document.

Files-11 Extended QIO Processor (XQP) file system has been enhanced to support extended file names through the $QIO interface. Note that in some cases, XQP file format rules differ from those that apply to other system services that accept file names, such as those provided by RMS. For a description of the new syntax and semantics used by RMS, see Section B.2.2.

The XQP enhancements support the following features of Extended File Specifications:

- Use of more characters, including those from the 8-bit ISO Latin-1 character set, and the 16-bit Unicode (UCS-2) character set
- Longer file names
- Preservation of case (as first created) within file names

The rest of Section B.3 describes the changes made to the Files-11 XQP file system and $QIO interface in more detail.

B.3.1. File Naming and Format Changes

Prior to OpenVMS Version 7.2, valid file names supported by the Files-11 XQP were limited to 85 ASCII characters [3] with both the file name and file type limited to 39 characters. In support of extended file names, these restrictions have been relaxed to allow the following:

- Use of most characters in the 8-bit ISO Latin-1 multinational character set (of which ASCII is a subset) in file names with the following exceptions:

  The C0 control set (hexadecimal 00 through 1F)
  Left angle bracket (<)
  Right angle bracket (>)
  Colon (:)
  Slash (/)
  Backslash (\
  Vertical bar (|)
  Question mark (?)
  Asterisk (*)

  Note that this explicitly *includes* both the C1 character set (hex 80-9F) as well as graphical and other characters between 9F and FF.

- Periods (.) within file names
- File and directory specifications encoded using 16-bit Unicode characters (UCS-2)
- Longer lengths for file names and file types, with the restriction that the file name and file type can total 236 8-bit or 118 16-bit characters, including a 1-character delimiter (.)
- File specifications up to 242 8-bit characters [4] or 124 16-bit characters [5]
- Mixed or lowercase input file specifications stored on disk without conversion to uppercase (also known as case preservation)

These changes apply only to those volumes that have been initialized or converted to the Files-11 ODS-5 format. Applications that rely on the semantics and behavior currently exhibited by ODS-2 volumes should continue to function as expected.

B.3.1.1. Specifying the Format of the Input File Name

File specifications are passed to the file system by descriptor by using the QIO P2 parameter. The descriptor contains a pointer to the text of the specification and a length field, which is the total length in bytes of the file specification.

The format of the specification can be identified in the new FIB$B_NAME_FORMAT_IN field, which can take one of the values listed in Table B.8.

Table B.8. FIB Constants for File Formats

Format Value	Format Type
FIB$C_ODS2	ODS-2 Format

[3] 39-character file name + 1-character delimiter (.) + 39-character file type + 1-character delimiter (;) + 5-character version number = 85 characters.
[4] 236-character file name, delimiter (.), file type + 1-character delimiter (;) + 5-character version number = 242 characters.
[5] 118-character file name, delimiter (.), file type + 1-character delimiter (;) + 5-character version number = 124 characters.

Format Value	Format Type
FIB$C_ISO_LATIN	ISO Latin-1 Format
FIB$C_UCS2	Unicode (UCS-2) Format

If the format specified is not one of those recognized by the file system, an SS$_BADPARAM error is returned. Otherwise, the file system attempts to parse the file specification according to the rules defined for the specified format. If the attempt to parse the name fails, an SS$_BADFILENAME or SS $_BADFILEVER error is returned.

If the FIB passed to the file system does not include the FIB$B_NAME_FORMAT_IN field, the file system assumes that the file specification supplied is in ODS-2 format. This is done to ensure compatibility with unchanged programs.

Before storing file specifications on the volume, the file system converts them to the simplest compatible format. For example, specifications supplied in Unicode (UCS-2) format that do not contain character values greater than 0x00FF are converted to ISO Latin-1 format before being stored on the volume.

B.3.1.2. Controlling the Format of Returned File Names

When returning a file specification, the file system writes the file format into the new FIB $B_NAME_FORMAT_OUT field. The value used will be one of those listed in Table B.8.

However, not all programs may be able to handle all available naming formats. Callers of the QIO system service can select which formats are returned to them using the new FIB$W_NMCTL flags described in Table B.9.

Table B.9. New FIB$W_NMCTL Flags

Flag Name	Interpretation
FIB$V_NAMES_8BIT	Caller can accept (8-bit) ODS-2 and ISO Latin-1 formats
FIB$V_NAMES_16BIT	Caller can accept (16-bit) Unicode (UCS-2) format.

These new flags control the format of returned file specifications as follows:

- Both flags clear

  Only ODS-2 format names are returned. Note that this includes specifications that were originally in ISO Latin-1 format or Unicode (UCS-2) format but converted to ODS-2 format before being stored on the volume. All specifications are converted to uppercase before being returned.

- FIB$V_ NAMES_8BIT set FIB$V_ NAMES_16BIT clear

  Only those file specifications stored in ODS-2 and ISO Latin-1 formats are returned. The value in the FIB$B_NAME_FORMAT_OUT field indicates the format of the particular name being returned. ODS-2 format file specifications are not converted to uppercase before being returned.

- FIB$V_ NAMES_8BIT clear FIB$V_ NAMES_16BIT set

  All file specifications are returned in Unicode (UCS-2) format.

- Both flags set

File specifications are returned in the format stored on the volume. This is the simplest format compatible with the file name syntax and the characters it contains. For example, a specification originally in Unicode format that only contains characters that are part of the ISO Latin-1 character set, are returned in ISO Latin-1 format.

B.3.1.3. Wildcard Searches and Pseudonames

The file specification returned by a file system operation is normally in a format that the calling program understands. This is not necessarily the case for operations where the input specification contains wildcard characters. For example, the wildcard in the following ODS-2 compliant file specification:

```
A*.DOC
```

could now correspond to the following ISO Latin-1 file specification:

```
A sample name with periods.and.other;punctuation#in the name.doc;1
```

Applications that assume that returned file specifications contain only one delimiting period could fail to perform correctly. Rather than return a file specification that would cause the calling program to fail, the file system returns a pseudoname in its place. The actual pseudoname returned depends on the type of name it represents, as shown in the following table.

File Format	**Sample Pseudoname**
ISO Latin-1 (FIB$C_ISL1)	`.???`
Unicode (FIB$C_UCS2)	`.???`

The file system determines which formats the calling program can understand from the settings of the FIB$V_NAMES_8BIT and FIB$V_NAMES_16BIT flags. These flags control the format of the returned name as shown in Table B.10.

Table B.10. FIB Flag Settings and Format of Related Returned Names

FIB Flag Settings		File Formats		
8BIT	16BIT	ODS-2	ISO Latin-1	Unicode
false	false	ODS-2	pseudoname	pseudoname
true	false	ODS-2	ISO Latin-1	pseudoname
false	true	UCS-2	UCS-2	UCS-2
true	true	ODS-2	ISO Latin-1	UCS-2

When returning a pseudoname, the file system notifies the user or calling application of the file without allowing direct file access. For this reason, pseudonames include characters that are *not* legal for input file specifications. Any attempt to use a pseudoname to manipulate a file will return a SYSTEM-F-BADFILENAME error.

Buffer Sizes

Table B.11 shows the minimum size that each buffer must be to contain all possible returned file specifications.

Table B.11. Safe Buffer Sizes for Each File Format (in Bytes)

File Format	**QIO Minimum**	**XQP Minimum**
ODS-2	86	86

File Format	QIO Minimum	XQP Minimum
ISO Latin-1	264	243
Unicode	538	486

The limit for a particular application depends on which formats it supports. Note that the minimum for the XQP is lower than the general limit for other file systems that use the QIO interface. This is because of the 236-byte size restriction for file specifications imposed by XQP.

If a file specification is longer than the supplied buffer, the file system truncates the returned specification without generating an error. If the file specification is shorter than the supplied buffer, the additional space from the end of the specification to the end of the buffer is filled with zeros.

B.3.1.4. Compatibility with Unchanged Applications

Any application that is not modified to take advantage of the new features of the QIO interface will, by default, receive only ODS-2 compatible file specifications or pseudonames provided they:

- Leave the FIB$V_NAMES_8BIT and _16BIT flags clear, or
- Supply a FIB that does not include the new FIB$B_NAME_FORMAT_IN and FIB $B_NAME_FORMAT_OUT fields.

File specifications that contain lowercase characters, which would otherwise be ODS-2 legal, are converted to uppercase before being returned.

File specifications supplied as input parameters by unchanged applications are interpreted as 8-bit ODS-2 names. The name is validated using the existing ODS-2 parsing rules. On an ODS-5 volume, the file specification is not converted to uppercase before it is stored on the disk. Unchanged applications will see the file specification in uppercase because of the conversion described above. Applications that set one of the new FIB flags will, however, see the specification in mixed case.

B.3.2. File Attribute Changes

The following sections describe the new file attributes introduced with ODS-5 and any changes to the semantics of existing attributes.

B.3.2.1. Modified File Attributes

Table B.12 shows the attributes that are modified or restricted for files on ODS-5 volumes.

Table B.12. Modified Attribute Codes

Attribute Name	Max Size (bytes)	Meaning
ATR$C_ASCNAME	252	File specification stored in file header
ATR$C_FILE_SPEC	4098	Device, best try path, and file specification
ATR$C_FILNAM	10	Radix-50 file name
ATR$C_FILTYP	4	Radix-50 file type
ATR$C_FILVER	2	Radix-50 file version

ATR$C_ASCNAME

The ATR$C_ASCNAME attribute allows the file specification stored in a file's primary file header to be read and written.

Reading the ATR$C_ASCNAME Attribute

For ODS-2 volumes, the ASCNAME attribute is returned as before. For ODS-5 volumes, the file specification is returned in the supplied buffer, and the name format is returned in the new FIB $B_ASCNAME_FORMAT cell.

The format in which the name is returned is controlled by the settings of the FIB$V_NAMES_8BIT and FIB$V_NAMES_16BIT flags in the same way as the output file specification parameter. A pseudoname can be returned in place of the actual file specification if the format is not one of those the calling program can accept.

Unlike the output file specification parameter, the length of a file specification contained in the ASCNAME attribute is not passed back explicitly. To determine the length of the file specification, the calling program must search the attribute buffer for the first occurrence of the padding character. If neither the FIB$V_NAMES_8BIT nor the FIB$V_NAMES_16BIT flag is set, the buffer is padded with space (note that only ODS-2 format names are returned in this case). If one or more of the flags are set, the attribute buffer is padded with zeros.

Note

The file system does not enforce a minimum length on the attribute buffer. If the file specification is longer than the attribute buffer, the value returned is truncated without signaling an error or warning.

In contrast, the file system does enforce a maximum size for the attribute buffer. Supplying a larger buffer returns a BADPARAM error.

Writing the ATR$C_ASCNAME Attribute

The ASCNAME attribute can only be written for files on ODS-2 or ODS-5 volumes provided that the FIB$V_NAMES_8BIT and FIB$V_NAMES_16BIT flags are clear.

The ability to write this attribute is only intended to provide compatibility with existing applications that do so. New and modified programs should not write this attribute. Changing its value can prevent a file from being permanently deleted.

In those cases where it is legal to write the attribute, the contents of the attribute buffer (up to 252 bytes) are copied to the file name field in the file header. For ODS-5 headers, the format is set to ODS-2, and the file name length is set to the offset of the first space character. This can be 252 bytes or the length of the supplied buffer, whichever is the least.

ATR$C_FILE_SPEC

The FILE_SPEC attribute is a read-only attribute that returns the physical file specification in the form:

```
DDnn:[DIR1.DIR2_DIRn]name.type;1
```

The file name returned is that from the file header, which may be different from that in the directory. The specification may be incomplete if any errors are encountered while reading the file headers of any of the directories in the path.

For files on ODS-5 volumes, the path may contain file names that are in any of the three name formats. This creates a number of problems; for instance, the presence of periods in a directory name could return

an ambiguous path specification. To avoid this and other problems, the file system makes use of services provided by RMS to translate the file specification and the components of the path to their escaped form.[6]

If the escaped form of the path is longer than can be accommodated by the buffer for the attribute, one or more directories in the path may be replaced by the DID of the rightmost of those replaced. This process is identical to that performed by RMS and is described in more detail in Section B.2. However, if the file specification, even after DID abbreviation, is longer than can be accommodated by the buffer, the file name is truncated. The file specification string returned to the user buffer has a 2-byte count prefix. The count contains the number of bytes for the untruncated file specification. If the count is greater than the size of the user buffer (minus the two bytes that contain the count), the user can conclude that the returned file specification has been truncated.

ATRC_FILNAM, ATRC_FILTYP, and ATR$C_FILVER

The first two of these attributes allow the file name and file type to be read and written using Radix-50 encoding. This encoding scheme enables 3 characters to be packed into a 16-bit word. Only 38 characters in the ODS-2 format set are valid for Radix-50 names, with the exceptions being dash (-) and underscore (_).

The maximum component lengths of a Radix-50 encoded file specification are:

- File name: 15 characters (10 bytes)
- File type: 6 characters (4 bytes)

As a result of the additional character and length restrictions, only a subset of legal ODS-2 file names is expressible in the Radix-50 encoding.

The file system only attempts to read or write the three attributes if the format of the existing file name in the file header is ODS-2. If this is not the case, a NORAD50 error will be returned. If the existing file name is in ODS-2 format, but is incompatible with the Radix-50 encoding or the length limits on Radix-50 file names, a BADFILENAME error will be returned.

The ATR$C_FILVER attribute allows the file version number in the file header to be read or written as a 2-byte integer. As the process requires the existing file name to be converted into a Radix-50 file name, the above restriction also applies to this attribute.

B.4. Programming Utility Changes

The following sections describe changes specific to OpenVMS programming utilities and their routines to support Extended File Specifications.

B.4.1. File Definition Language (FDL) Routines

The File Definition Language (FDL) routines have been enhanced on OpenVMS Version 7.2 for Alpha to support Extended File Specifications. A new flag, FDL$V_LONG_NAMES, has been added to the *flags* argument of the following routines:

[6]When you access files on an ODS-5 volume from a VAX system in a mixed architecture OpenVMS system, no escaped forms are returned. For an ODS-2 or ISO Latin-1 file format, the name stored in the file header is returned. For a UCS-2 file format, a pseudoname is returned, followed by the file identifier in parentheses. For example:

```
      DKA100:[ABC]
pUNICODE
.??? (10095,5,0)
```

FDL$CREATE
FDL$GENERATE
FDL$PARSE
FDL$RELEASE

The following sections describe the FDL$V_LONG_NAMES flag as it relates to each FDL routine.

B.4.1.1. FDL$CREATE Routine (Alpha Only)

The following new flag has been added to the *flags* argument:

Flag	Function
FDL$V_LONG_NAMES	Returns the RESULT_NAME using the long result name from a long name access block (NAML). By default, the RESULT_NAME is returned from the short fields of a name access block (NAM) and thus may have a generated specification. This flag is valid for OpenVMS Alpha only.

B.4.1.2. FDL$GENERATE Routine (Alpha Only)

The following new flag has been added to the *flags* argument:

Flag	Function
FDL$V_LONG_NAMES	Returns the FDL_FILE_RESNAM using the long result name from a long name access block (NAML). By default, the FDL_FILE_RESNAM is returned from the short fields of a name access block (NAM) and thus may have a generated specification. This flag is valid for OpenVMS Alpha only.

B.4.1.3. FDL$PARSE Routine (Alpha Only)

The following new flag has been added to the *flags* argument:

Flag	Function
FDL$V_LONG_NAMES	Allocates and returns a long name access block (NAML) linked to the returned RMS file access block (FAB). The appropriate values are set in the NAML and FAB blocks so that the long file name fields of the NAML block will be used. By default, a name block is not allocated and the file name fields of FAB are used. If the FDL$V_LONG_NAMES flag is set, then the FDL$V_LONG_NAMES bit must also be set in the *flags* argument to the FDL$RELEASE routine to ensure that memory allocated for the NAML block is deallocated properly.

Flag	Function
	This flag is valid for OpenVMS Alpha only.

B.4.1.4. FDL$RELEASE Routine (Alpha Only)

The following new flag has been added to the *flags* argument:

Flag	Function
FDL$V_LONG_NAMES	Deallocates any virtual memory used for a long name access block (NAML) created by the FDL $PARSE routine. This flag is valid for OpenVMS Alpha only.

B.5. Run-Time Library Changes

To enable the use of extended file names, several routines in the LIB$ Run-Time Library have been modified so that they can optionally accept or return a NAML block rather than a NAM block. The following routines have been modified:

- LIB$CREATE_DIR
- LIB$DELETE_FILE
- LIB$FILE_SCAN
- LIB$FIND_FILE
- LIB$RENAME_FILE
- LIB$FID_TO_NAME

B.5.1. LIB$CREATE_DIR

The maximum size of argument *device-directory-spec* is now 255 characters on VAX, and 4095 characters on Alpha.

B.5.2. LIB$DELETE_FILE

The format of LIB$DELETE_FILE is now:

```
LIB$DELETE_FILE     filespec [,default-filespec] [,related-filespec]
                    [,user-success-procedure] [,user-error-procedure]
                    [,user-confirm-procedure] [,user-specified-argument]
                    [,resultant-name] [,file-scan-context] [,flags])
```

The *flags* argument is new, and has the following format:

```
flags
 OpenVMS usage:   mask_longword
 type:            longword (unsigned)
 access:          read only
 mechanism:        by reference
```

User flags. The *flags* argument is the address of an unsigned longword containing the user flags.

The flag bits and their corresponding symbols are described in the following table:

Bit	Symbol	Description
0		Reserved to VSI.
1		Reserved to VSI.
2	LIB$M_FIL_LONG_NAMES	(Alpha only) If set, LIB $DELETE_FILE can process file names with a maximum length of NAML$C_MAXRSS. If clear, LIB$DELETE_FILE can process file specifications with a maximum length of 255 bytes (default).

On Alpha systems, if you specify the *user-confirm-procedure* in the call to LIB $DELETE_FILE, and the LIB$M_FIL_LONG_NAMES flag is set, the FAB referenced by the *fab* argument to the confirm-procedure routine references a NAML block rather than a NAM block. The NAML block supports the use of long file names with a maximum length of NAML$C_MAXRSS. See the *VSI OpenVMS Record Management Services Reference Manual* for information on NAML blocks.

LIB$DELETE_FILE has an additional condition value returned. LIB$INVARG indicates that an unspecified bit was set in the *flags* argument.

B.5.3. LIB$FILE_SCAN

The *fab* argument to LIB$FILE_SCAN can now reference either a NAM or NAML block.

B.5.4. LIB$FIND_FILE

The *flags* argument to LIB$FIND_FILE has the following new bit:

Bit	Symbol	Description
2	LIB$M_FIL_LONG_NAMES	(Alpha only) If set, LIB $FIND_FILE can process file specifications with a maximum length of NAML$C_MAXRSS. If clear, LIB$FIND_FILE can process file specifications with a maximum length of 255 bytes (default).

On Alpha systems, support for file specifications longer than 255 bytes is provided only when the LIB $M_FIL_LONG_NAMES flag is set in the *flags* argument. When this flag is set, a NAML block (rather than a NAM block) is part of the context, and file specifications can have a maximum length of NAML$C_MAXRSS. See the *VSI OpenVMS Record Management Services Reference Manual* for information on NAML blocks.

B.5.5. LIB$RENAME_FILE

The *flags* argument to LIB$RENAME_FILE has the following new bit:

Bit	Symbol	Description
2	LIB$M_FIL_LONG_NAMES	(Alpha only) Controls whether to accept file specifications greater than 255 bytes in length. If this bit is set, LIB $RENAME_FILE can process file specifications with a maximum length of NAML $C_MAXRSS characters. If this bit is clear, LIB $RENAME_FILE can process file names with a maximum length of 255 bytes.

On Alpha systems, if you specify the *user-confirm-procedure* in the call to LIB $RENAME_FILE, and the LIB$M_FIL_LONG_NAMES flag is set, the FAB referenced by the *old-fab* argument to the confirm-procedure routine references a NAML block rather than a NAM block. The NAML block supports the use of long file names with a maximum length of NAML$C_MAXRSS. See the *VSI OpenVMS Record Management Services Reference Manual* for information on NAML blocks.

B.5.6. LIB$FID_TO_NAME

If the file specification is longer than can be accommodated by the *filespec* buffer, a directory in the path may be replaced by a DID abbreviation (see Section 3.2.2.1). If the file specification after DID abbreviation is longer than can be accommodated by the buffer, the file specification is truncated and, as in prior versions, LIB$_STRTRU is returned as an alternate success status. In the case of a dynamic descriptor, the maximum allows a string as large as 4095 bytes to be returned.

Appendix C. Character Sets

The DEC Multinational Character Set (MCS) consists of a definition of the characters identified by hexadecimal values 00 through FF, inclusive, that was created and used by Digital Equipment Corporation. The DEC MCS is divided into two parts, the ASCII 7-bit character set (identified by hexadecimal values 00 through 7F, inclusive), and the set of 8-bit characters identified by hexadecimal values 80 through FF, inclusive. The DEC MCS is familiar to most users of software created and sold by DIGITAL.

The Unicode Standard Character Set (UCS-2) is a definition, by The Unicode Consortium, of the set of 16-bit characters that can be identified by hexadecimal values 0000 through FFFF, inclusive.

The ISO Latin-1 character set is the UCS-2 definition of the 8-bit characters identified by hexadecimal values 00 through FF, inclusive. The ISO Latin-1 character set definition differs slightly from the DEC MCS definition of the hexadecimal values 80 through FF.

Table C.1 contains the DEC Multinational Character Set (MCS). Table C.1 indicates the characters that differ between the two character sets, and Figure C.1 shows the differing characters.

See *The Unicode Standard*, published by The Unicode Consortium, for details about the Unicode (UCS-2) character set.

Table C.1. DEC Multinational Character Set

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name
ASCII Control Characters [1]		
00	NUL	null character
01	SOH	start of heading (Ctrl/A)
02	STX	start of text (Ctrl/B)
03	ETX	end of text (Ctrl/C)
04	EOT	end of transmission (Ctrl/D)
05	ENQ	enquiry (Ctrl/E)
06	ACK	acknowledge (Ctrl/F)
07	BEL	bell (Ctrl/G)
08	BS	backspace (Ctrl/H)
09	HT	horizontal tabulation (Ctrl/I)
0A	LF	line feed (Ctrl/J)
0B	VT	vertical tabulation (Ctrl/K)
0C	FF	form feed (Ctrl/L)
0D	CR	carriage return (Ctrl/M)
0E	SO	shift out (Ctrl/N)
0F	SI	shift in (Ctrl/O)
10	DLE	data link escape (Ctrl/P)
11	DC1	device control 1 (Ctrl/Q)
12	DC2	device control 2 (Ctrl/R)

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name
13	DC3	device control 3 (Ctrl/S)
14	DC4	device control 4 (Ctrl/T)
15	NAK	negative acknowledge (Ctrl/U)
16	SYN	synchronous idle (Ctrl/V)
17	ETB	end of transmission block (Ctrl/W)
18	CAN	cancel (Ctrl/X)
19	EM	end of medium (Ctrl/Y)
1A	SUB	substitute (Ctrl/Z)
1B	ESC	escape
1C	FS	file separator
1D	GS	group separator
1E	RS	record separator
1F	US	unit separator
ASCII Special and Numeric Characters		
20	SP	space
21	!	exclamation point
22	"	quotation marks (double quote)
23	#	number sign
24	$	dollar sign
25	%	percent sign
26	&	ampersand
27	’	apostrophe (single quote)
28	(	opening parenthesis
29	)	closing parenthesis
2A	*	asterisk
2B	+	plus
2C	,	comma
2D	–	hyphen or minus
2E	.	period or decimal point
2F	/	slash
30	0	zero
31	1	one
32	2	two
33	3	three
34	4	four
35	5	five

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name
36	6	six
37	7	seven
38	8	eight
39	9	nine
3A	:	colon
3B	;	semicolon
3C	<	less than
3D	=	equals
3E	>	greater than
3F	?	question mark
ASCII Alphabetic Characters		
40	@	commercial at sign
41	A	uppercase A
42	B	uppercase B
43	C	uppercase C
44	D	uppercase D
45	E	uppercase E
46	F	uppercase F
47	G	uppercase G
48	H	uppercase H
49	I	uppercase I
4A	J	uppercase J
4B	K	uppercase K
4C	L	uppercase L
4D	M	uppercase M
4E	N	uppercase N
4F	O	uppercase O
50	P	uppercase P
51	Q	uppercase Q
52	R	uppercase R
53	S	uppercase S
54	T	uppercase T
55	U	uppercase U
56	V	uppercase V
57	W	uppercase W
58	X	uppercase X
59	Y	uppercase Y

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name
5A	Z	uppercase Z
5B	[	left bracket
5C	\	backslash
5D	]	right bracket
5E	^	circumflex
5F	_	underscore
60	`	grave accent
61	a	lowercase a
62	b	lowercase b
63	c	lowercase c
64	d	lowercase d
65	e	lowercase e
66	f	lowercase f
67	g	lowercase g
68	h	lowercase h
69	i	lowercase i
6A	j	lowercase j
6B	k	lowercase k
6C	l	lowercase l
6D	m	lowercase m
6E	n	lowercase n
6F	o	lowercase o
70	p	lowercase p
71	q	lowercase q
72	r	lowercase r
73	s	lowercase s
74	t	lowercase t
75	u	lowercase u
76	v	lowercase v
77	w	lowercase w
78	x	lowercase x
79	y	lowercase y
7A	z	lowercase z
7B	{	left brace
7C	|	vertical line
7D	}	right brace (ALTMODE)
7E	~	tilde (ALTMODE)

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name
7F	DEL	rubout (DELETE)
Control Characters		
80		[reserved]
81		[reserved]
82		[reserved]
83		[reserved]
84	IND	index
85	NEL	next line
86	SSA	start of selected area
87	ESA	end of selected area
88	HTS	horizontal tab set
89	HTJ	horizontal tab set with justification
8A	VTS	vertical tab set
8B	PLD	partial line down
8C	PLU	partial line up
8D	RI	reverse index
8E	SS2	single shift 2
8F	SS3	single shift 3
90	DCS	device control string
91	PU1	private use 1
92	PU2	private use 2
93	STS	set transmit state
94	CCH	cancel character
95	MW	message waiting
96	SPA	start of protected area
97	EPA	end of protected area
98		[reserved]
99		[reserved]
9A		[reserved]
9B	CSI	control sequence introducer
9C	ST	string terminator
9D	OSC	operating system command
9E	PM	privacy message
9F	APC	application
Other Characters		
A0		[reserved] [2]

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name
A1	¡	inverted exclamation point
A2	¢	cent sign
A3	£	pound sign
A4		[reserved] [2]
A5	¥	yen sign
A6		[reserved] [2]
A7	§	section sign
A8	¤	general currency sign [2]
A9	©	copyright sign
AA	ª	feminine ordinal indicator
AB	«	angle quotation mark left
AC		[reserved] [2]
AD		[reserved] [2]
AE		[reserved] [2]
AF		[reserved] [2]
B0	°	degree sign
B1	±	plus/minus sign
B2	²	superscript 2
B3	³	superscript 3
B4		[reserved] [2]
B5	µ	micro sign
B6	¶	paragraph sign, pilcrow
B7	·	middle dot
B8		[reserved] [2]
B9	¹	superscript 1
BA	º	masculine ordinal indicator
BB	»	angle quotation mark right
BC	¼	fraction one-quarter
BD	½	fraction one-half
BE		[reserved] [2]
BF	¿	inverted question mark
C0	À	uppercase A with grave accent
C1	Á	uppercase A with acute accent
C2	Â	uppercase A with circumflex
C3	Ã	uppercase A with tilde
C4	Ä	uppercase A with umlaut (diaeresis)

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name
C5	Å	uppercase A with ring
C6	Æ	uppercase AE diphthong
C7	Ç	uppercase C with cedilla
C8	È	uppercase E with grave accent
C9	É	uppercase E with acute accent
CA	Ê	uppercase E with circumflex
CB	Ë	uppercase E with umlaut (diaeresis)
CC	Ì	uppercase I with grave accent
CD	Í	uppercase I with acute accent
CE	Î	uppercase I with circumflex
CF	Ï	uppercase I with umlaut (diaeresis)
D0		[reserved] [2]
D1	Ñ	uppercase N with tilde
D2	Ò	uppercase O with grave accent
D3	Ó	uppercase O with acute accent
D4	Ô	uppercase O with circumflex
D5	Õ	uppercase O with tilde
D6	Ö	uppercase O with umlaut (diaeresis)
D7	OE	uppercase OE ligature [2]
D8	Ø	uppercase O with slash
D9	Ù	uppercase U with grave accent
DA	Ú	uppercase U with acute accent
DB	Û	uppercase U with circumflex
DC	Ü	uppercase U with umlaut (diaeresis)
DD	Y	uppercase Y with umlaut (diaeresis)
DE		[reserved] [2]
DF	ß	German lowercase sharp s
E0	à	lowercase a with grave accent
E1	á	lowercase a with acute accent
E2	â	lowercase a with circumflex
E3	ã	lowercase a with tilde
E4	ä	lowercase a with umlaut (diaeresis)
E5	å	lowercase a with ring

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name
E6	æ	lowercase ae diphthong
E7	ç	lowercase c with cedilla
E8	è	lowercase e with grave accent
E9	é	lowercase e with acute accent
EA	ê	lowercase e with circumflex
EB	ë	lowercase e with umlaut (diaeresis)
EC	ì	lowercase i with grave accent
ED	í	lowercase i with acute accent
EE	î	lowercase i with circumflex
EF	ï	lowercase i with umlaut (diaeresis)
F0		[reserved] [2]
F1	ñ	lowercase n with tilde
F2	ò	lowercase o with grave accent
F3	ó	lowercase o with acute accent
F4	ô	lowercase o with circumflex
F5	õ	lowercase o with tilde
F6	ö	lowercase o with umlaut (diaeresis)
F7	oe	lowercase oe ligature [2]
F8	ø	lowercase o with slash
F9	ù	lowercase u with grave accent
FA	ú	lowercase u with acute accent
FB	û	lowercase u with circumflex
FC	ü	lowercase u with umlaut (diaeresis)
FD	ÿ	lowercase y with umlaut (diaeresis) [2]
FE		[reserved] [2]
FF		[reserved] [2]

[1]The ALTMODE and DELETE characters (decimal 125, 126, and 127) are also control characters.

[2]Different character in ISO Latin-1. See Figure C.1.

Figure C.1. Differences Between DEC Multinational Character Set and ISO Latin-1 Character Set

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name	Isolatin-1 Char	Isolatin-1 Character Name
A0		[reserved]		nonbreaking space
A4		[reserved]	¤	currency sign
A6		[reserved]	¦	broken vertical bar
A8	¤	currency sign	¨	spacing diaeresis
AC		[reserved]	¬	not sign
AD		[reserved]		soft hyphen
AE		[reserved]	®	registered trademark sign
AF		[reserved]	¯	spacing macron
B4		[reserved]	´	spacing acute
B8		[reserved]	¸	spacing cedilla
BE		[reserved]	¾	fraction three quarters
D0		[reserved]	Ð	Latin capital letter eth
D7	Œ	uppercase OE ligature	×	multiplication sign
DE		[reserved]	Þ	Latin capital letter thorn
F0		[reserved]	ð	Latin small letter eth
F7	œ	lowercase oe ligature	÷	division sign
FD	ÿ	lowercase y with umlaut, (diaeresis)	ý	Latin small letter y acute
FE		[reserved]	þ	Latin small letter thorn
FF		[reserved]	ÿ	Latin small letter y diaeresis