DIGITAL COBOL
Multiplatform Overview & Compatibility Guide

Part Number: AA–RD7XA–TK

September 1998

This book explains the close relationships among the DIGITAL COBOL family members on all supported platforms, and provides information that can help you plan for the migration to the Alpha platform.

Revision/Update Information:	This is a new document.
Operating Systems:	DIGITAL UNIX OpenVMS Alpha OpenVMS VAX Windows NT Alpha

Contents

Preface

1 Why You Need This Book...

2 ...And Why We Wrote This Book

3 Our DIGITAL COBOL Genealogy

4 Developing, Debugging, and Maintaining COBOL Programs

5 Compiling, Linking, and Running

6 Modifying the COBOL Command

7 Useful Summaries and Tables

Index

Preface

Who Should Read This Book

Engineers and Engineering Managers who are familiar with COBOL and are involved in the decision process of migrating or maintaining enterprise software should be familiar with this book. Programmers directly involved in the maintenance and migration of COBOL applications will use this book as an adjunct to the Reference Manual and User Manual.

Whether you are maintaining applications, developing new programs, or migrating software between DIGITAL platforms, you will find valuable reference material and techniques here.

This book is a companion to the documentation sets for all four members of the DIGITAL COBOL family:

- DIGITAL VAX COBOL (formerly called VAX COBOL)
- DIGITAL COBOL for OpenVMS Alpha (formerly called DEC COBOL)
- DIGITAL COBOL for DIGITAL UNIX (formerly called DEC COBOL)
- DIGITAL COBOL for Windows NT Alpha

This book is organized as follows:

Chapter	Description
One – Why You Need This Book...	Explains what benefits you can derive from using this book
Two –...And Why We Wrote This Book	Briefly describes the family of COBOL compilers from Digital Equipment Corporation
Three – Our DIGITAL COBOL Genealogy	Outlines the significant features of each member of our DIGITAL COBOL family.

Chapter	Description
Four – Developing, Debugging, and Maintaining COBOL Programs	Outlines the similarities—and distinctive differences—in the actual steps and typical tools used to manipulate COBOL code across its life cycle on four platforms.
Five – Compiling, Linking, and Running	Compares the commands for program builds for all DIGITAL COBOL platforms.
Six – Modifying the COBOL Command	Tables showing the qualifiers, options, and flags you can apply as modifiers to the compile command line on all DIGITAL COBOL platforms.
Seven – Useful Summaries and Tables	This chapter concentrates helpful information into one handy location.

Conventions

We use the following conventions in this book:

Convention	Description
`Special Type`	This special type in examples indicates system output in interactive examples.
Boldface, Red	Boldface or red type in examples indicates user input in interactive examples.
<Return>	Indicates that you should press the Return key in interactive examples.
[optional-item]	Square brackets in syntax denote that the enclosed item is optional.
{mandatory-item}	Braces in syntax denote that the enclosed item must be provided.
...	An ellipsis shows where a syntax item can be repeated zero or more times.
/LIS[=*filespec*]	On the command line, the italicized portion of a syntax structure (*filespec* in this example) represents substituted user input. In this case, you would type an actual file specification (=mylisting.txt, for example) in place of *filespec*.
http://www.digital.com, Why You Need This Book...	When viewed online, blue cross-references are hot links to World Wide Web URLs or specific locations in this book.

Acknowledgment

The authors and copyright holders of the copyrighted material used in our DIGITAL COBOL documentation are as follows: FLOW-MATIC (trademark of Unisys Corporation), Programming for the UNIVAC ® I and II, Data Automation Systems, copyrighted 1958, 1959, by Unisys Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

Chapter 1

Why You Need This Book...

Reassurance

If you are a COBOL user who is contemplating a hardware upgrade, and you are (naturally) apprehensive about the cost of rewriting your software for the new system, we believe there is information in this book that can reassure you.

This book provides you with descriptions of the close compatibility you will find within our DIGITAL COBOL product family.

For example, Chapter 2 shows the standard and extended features across platforms; Chapter 4 discusses the development environments on the various systems; Chapter 5 shows the similarity between build procedures, and Chapter 6 provides compile-command modifiers for *all* the platforms. Chapter 7 provides summary information such as file I/O, functions, environment control, and more.

Bear in mind that the most complete resource for all this information is the full documentation set. For example,

language extensions are integrated with the syntax diagrams in Chapter 6 of the DIGITAL COBOL Reference Manual, and compatibility details are in the User Manual.

Support

If you are a DIGITAL VAX COBOL user, you will find information in this book that can help you migrate your COBOL applications to the Alpha platform. While your ultimate source of information is the detailed DIGITAL COBOL technical manuals, we have brought some key information here for you.

Compatibility information, upgrade information, comparison of commands and features... We've included these here and in our documentation set not only to convince you that migrating to Alpha is easier than you might believe, but also to provide you with information to make the decision to make the move. Plus, you can contract for DIGITAL's technical support network to help you over any rough spots you might encounter.

Accessibility

You know you've seen the EBCDIC code for the bell character somewhere, but where was that? Try the Character Codes section in the *Useful Summaries and Tables* chapter of this book.

Your build procedures make use of DIGITAL VAX COBOL command qualifiers—so how do you modify the compile command on Alpha? Chapter 6, *Modifying the COBOL Command* provides the command modifiers for all four platforms covered by this book.

This guide provides a single source for valuable quick-reference programming information. And, if you're reading this book online, that next piece of information is only a mouse click away.

Migration and Compatibility

With DIGITAL COBOL, migrating between VAX, Alpha, and other vendors' COBOL isn't just a vision of the future, it's an opportunity that exists for you now. While some parts of your applications may perform differently across platforms, the basic logic of your applications will migrate more easily than you might have thought possible, because of the similarity of the COBOL compilers from DIGITAL. (See Chapter 6, *Modifying the COBOL Command*, for proof.)

We've brought a kernel of migration and porting information into this book to peak your attention. You'll find this in Migration—What's Involved? in Chapter 3. We've also included additional information in the Cross-Platform Compatibility table in Chapter 7, that vividly shows a list of COBOL features that are important to you, along with the compatibility of each feature across the platforms.

The broadest field of information that will help you migrate to Alpha will be found in the DIGITAL COBOL technical manuals, specifically in *DIGITAL VAX COBOL and DIGITAL COBOL Compatibility and Migration*, Appendix B of the User Manual .

Chapter 2

...And Why We Wrote This Book

DIGITAL COBOL: Continuous Progress, and Stability

Hundreds of millions of lines of COBOL code currently exist in our world, representing an untold number of applications. These have been written by customers who need the advantages provided by our COBOL compilers.

Computer technology is a constantly evolving, dynamic environment, matching the dynamics of the needs of computer users. Legacy COBOL applications were written around the computer system architecture and user scenarios that were then prevalent. Mainframe computers and minicomputers sat on raised computer floors out of the view of office staff. Desktop and point-of-sales devices consisted of data terminals with

minimal storage and no onboard intelligence. *Tomorrow's* demands call for multiple desktop systems running graphical user interfaces (GUIs) to enterprise-wide software accessing a centralized information bank.

Computing Styles and Working Styles

From Timeshare to Interactive

Computing style and work style have evolved considerably. Instead of time-share terminals hardwired to central computing systems, intelligent workstations now network with enterprise computers by cable, RF data link, and orbiting satellite,.

Instead of rudimentary on-screen forms, users now have smart interactive windows. In many cases the workstations are running complex, sophisticated *client* applications that do much processing before ever communicating with central data *servers*. Two-tier and three-tier client/server software is now becoming commonplace.

Client-Server

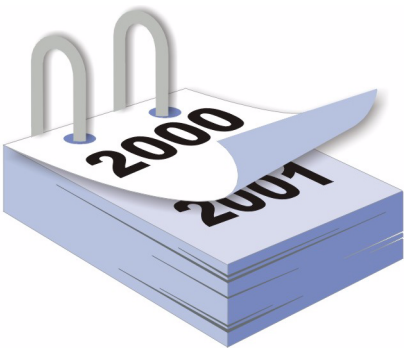
In one possible two-tier client/server application, DIGITAL COBOL client code might exist on a system running Windows NT Alpha. A GUI written in one of the graphics-rich visual languages might then use middleware to access a data file on a large system, through DIGITAL COBOL software running on an OpenVMS (VAX *or* Alpha) server.

In three-tier client/server programming, the first tier might include a desktop system running a client program that has either a GUI or a more traditional user interface written in DIGITAL COBOL for Windows NT Alpha. The second-tier client program uses middleware to access a back-room DIGITAL COBOL server running OpenVMS Alpha. The server accesses the third tier, a large, enterprise-wide database that is running on a central computer complex, perhaps many miles away, and produces the desired result.

...And Beyond

Digital Equipment Corporation has been meeting the needs of commerce for decades, through the evolution of the hardware and software as well as the proliferation of uses for computers. DIGITAL was there in the beginning with PDP-11 COBOL, and today we are in the forefront of the technology wave with high-performance compilers that run on the most popular, most efficient, most powerful, and most effective systems available.

What's more, we are hard at work on advanced compiler technology to meet and anticipate our customers' COBOL needs of *tomorrow*.



The Year 2000—Are Your Programs Ready?

It's important to note that the ultimate responsibility for ensuring that your programs will correctly deal with dates is firmly in *your* hands. To help you address this issue, DIGITAL has included several features in DIGITAL COBOL:

- Informational messages at compile time (when 2-digit year ACCEPT date coding is encountered during compile)
- 4-digit year ACCEPT FROM DATE
- 4-digit year ACCEPT FROM DAY
- 4-digit year functions: CURRENT-DATE, DATE/DAY-OF-INTEGER, INTEGER-OF-DATE/DAY, WHEN-COMPILED

There is additional information in the user manual for your platform. In addition, the Year 2000 (sometimes called Y2K) program at DIGITAL is extensive, including third party tools and the Year 2000 web site that is only a mouse click away, at: http://ww1.digital.com/year2000/e_home.html.

About Goals

The DIGITAL COBOL family meets a number of your needs and goals:

- Application migration
- Superior reliability
- Excellent run-time performance
- Integration with base system debuggers and tools such as LSE, DECset, and FUSE; the VMS Debugger, Ladebug, and WINDBG
- Integration with ISAM packages such as DIGITAL RMS, Informix, and Sector 7
- Integration with database and transaction environments such as Oracle CDD/Repository, DBMS, ACMS, and Encina
- Extensions for compatibility with X/Open and implementations from other COBOL vendors

The goal of our products is to provide industry-standard COBOL support—and beyond—for OpenVMS, DIGITAL UNIX, and Windows NT Alpha systems.

One Picture=How Many Words?

On the next pages of this chapter you will find drawings that pack a great deal of information into a small space. These drawings show many of the standard and extended features of DIGITAL COBOL. They dramatically depict the specific platforms on which the indicated features are available to help you meet your goals.

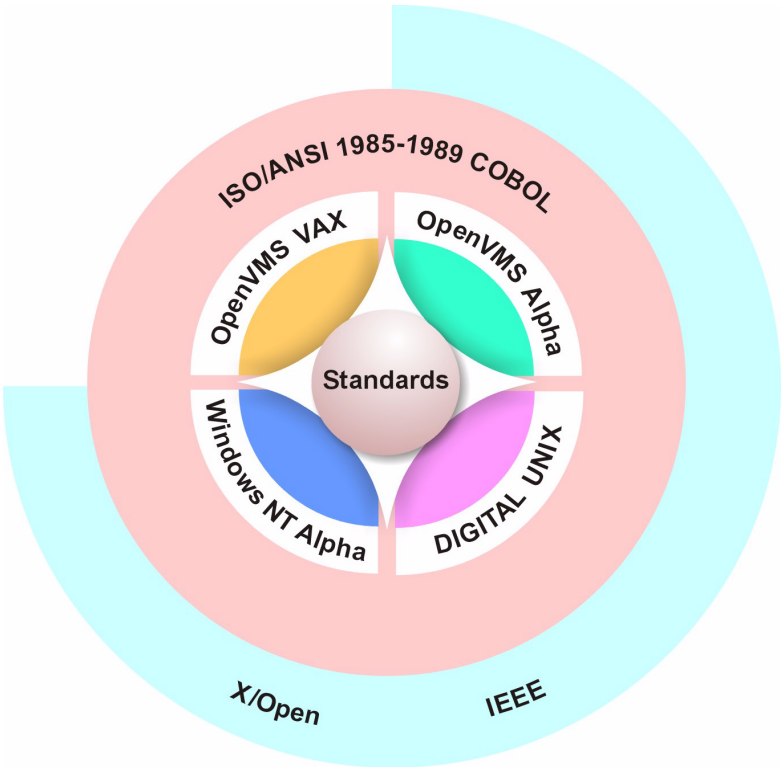
In general, each drawing shows an area where we at DIGITAL have something important to offer you, our COBOL customer. This might be adherence to standards, for example, or sophisticated screen handling. We distinguish, by way of cutouts or coverage, the platforms on which these features are available. These drawings can be your window to seeing your potential use of the whole DIGITAL COBOL product, to increase both your productivity and your profitability.

The specific details of the various items (standards; file handling; screen handling; language extensions; tools; compiler options; and information resources) are extensively covered in the user manual and reference manual for the respective platforms.

Our High Standards

DIGITAL COBOL is fully validated at the high level on all platforms addressed in this book. It is based on the ANSI-85 COBOL Standard with ANSI-89 addenda. On Alpha, DIGITAL COBOL also contains appropriate subsets of the extensions specified by external standards. For example, X/Open file sharing and record locking, screen handling, and IEEE floating point handling.

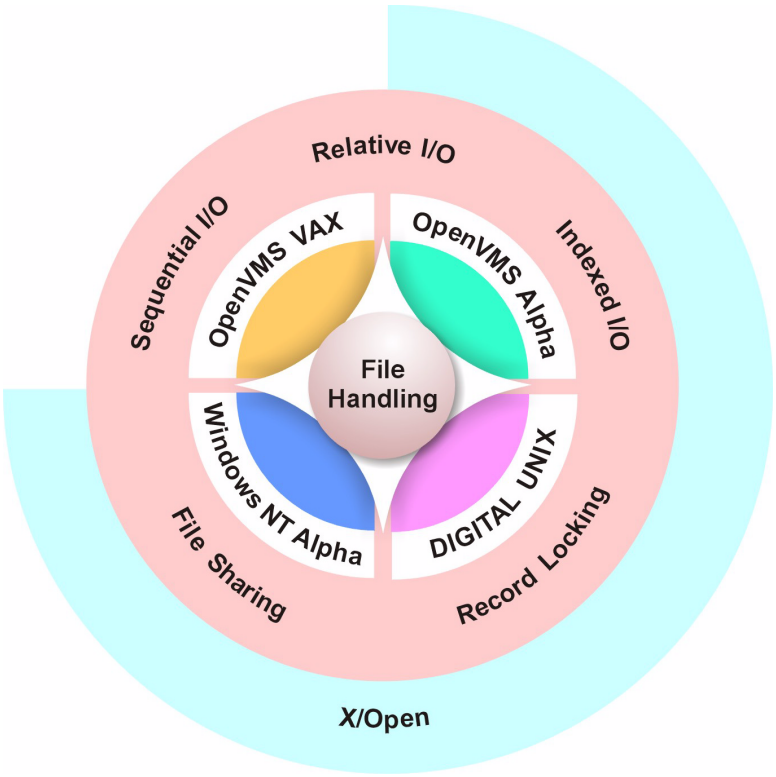
DIGITAL COBOL provides powerful extensions above and beyond the functionality in ISO/ANSI standards. When you combine these complementary extensions and standards you have extensive capabilities that you can migrate across our platforms... and take to the bank. The many similarities across all platforms bring power to your programs, and the differences are few and completely manageable. The details of these differences are documented in *DIGITAL VAX COBOL and DIGITAL COBOL Compatibility and Migration*, Appendix B of the DIGITAL COBOL User Manual.



Your Valuable Data

Efficient file handling is vital to your programs and to your data assets. Whether you are migrating your applications from VAX to Alpha or from another vendor's COBOL to DIGITAL COBOL, we have your needs covered. The essential functions related to file sharing and record locking provide shared access to your enterprise data on all platforms. The extended functions provided in X/Open file handling are available on Alpha, and we have tested third-party tools that provide ISAM support on DIGITAL UNIX (Informix C-ISAM) andWindows NT Alpha (Sector 7 ISAM).

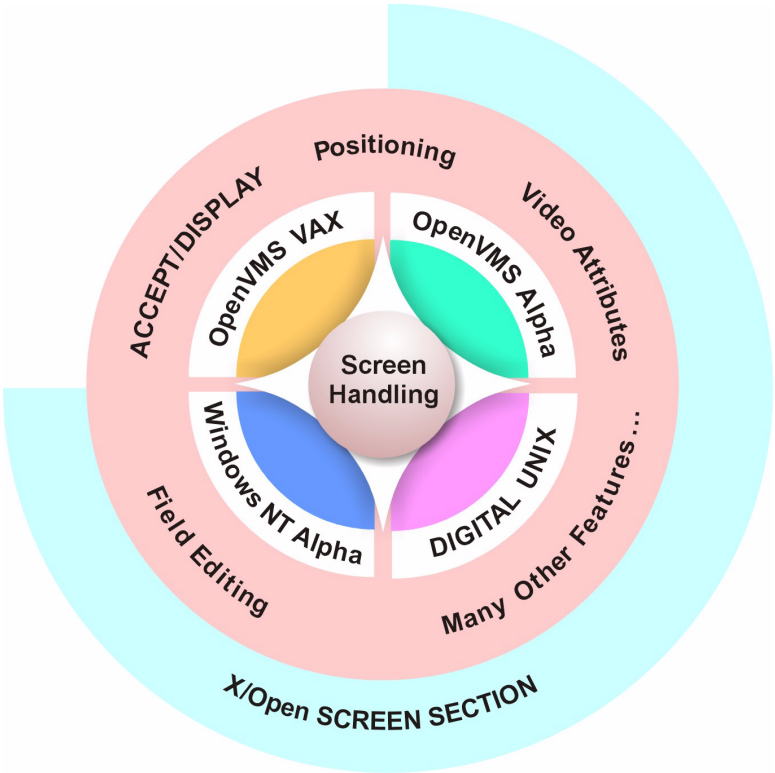
Efficient file handling, the *heart* of COBOL, is well supported across our four platforms. (Refer to *I/O Statements* in the *Useful Summaries and Tables* chapter.)



Your Valuable Users

DIGITAL COBOL naturally supports screen handling features that let your users access gateways to your enterprise data. Beyond standard terminal I/O, DIGITAL COBOL extensions let you use ACCEPT and DISPLAYstatements to control the position (line and column) and presentation (for example, using bold and blink) of prompts and displayed data on four platforms, and X/Open SCREEN SECTION on our three Alpha systems.

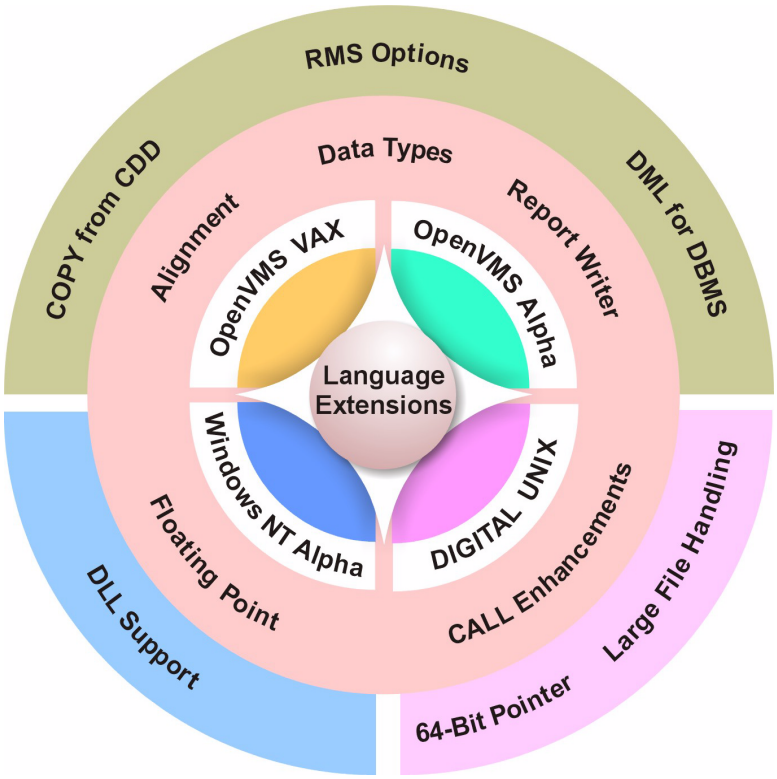
DIGITAL COBOL programs also integrate well with tools such as DECforms for your screen forms applications. Plus, you have a built-in subset of DEC Forms capability on Alpha; screen extensions allow you to use COBOL procedures to directly create forms on screen (and clear the screen) for user interaction. Your applications can "walk the fields" and collect data for later processing.



Extensions

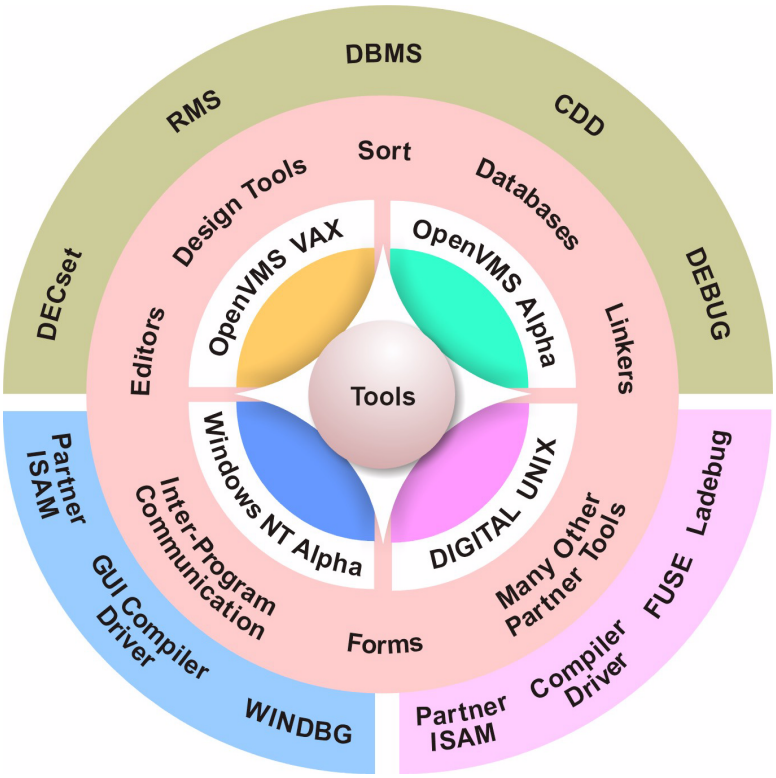
Our DIGITAL VAX COBOL users know that we've burst the seams of the standards envelope with such features as language verbs for DBMS, RMS coupling, and data repository links. These features are also there on Alpha. If you've been accessing indexed files with RMS, you can use third-party ISAM tools (which we have tested) on DIGITAL UNIX and Windows NT Alpha. CALL BY *reference* and CALL BY *content* is standard; we also give you CALL BY *descriptor* on OpenVMS, and CALL BY *value* on all platforms. We provide floating point and data alignment extensions. Extension such as these are documented in Chapter 6 of the Reference Manual.

Have you used shared objects on UNIX and shared images on OpenVMS? You'll find that creating a DLL on Windows NT Alpha gives you the same flexibility and memory advantages. And, our floating point and data alignment extensions provide you with powerful capabilities. Large files? We've tested with files >40Gb.



Tools—More Power for your Programming

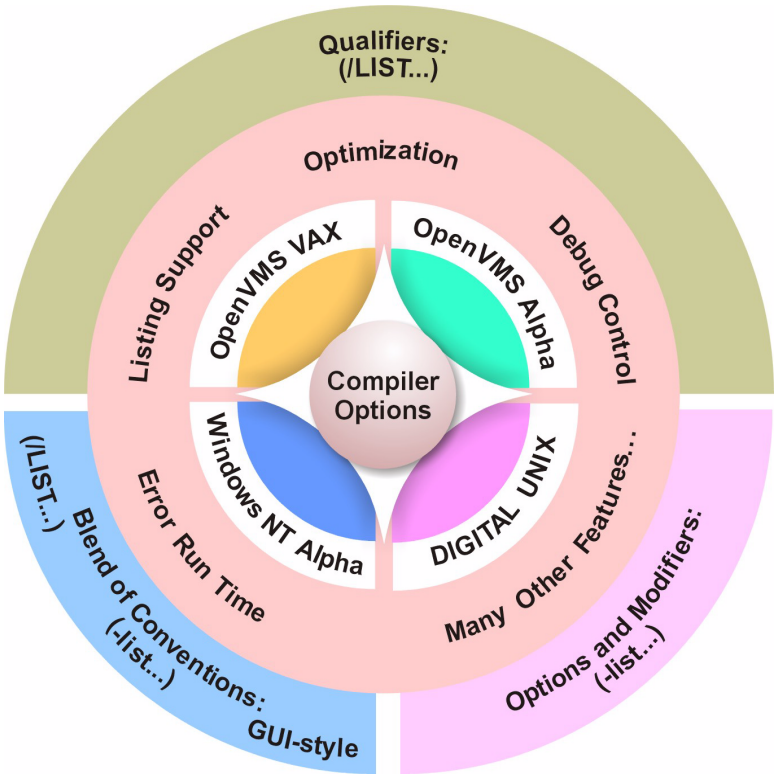
In addition to our own direct feature support, we work closely with partners outside of DIGITAL to provide tools that ensure both compatibility and added value. ISAM and our own RMS give you broad file access powers on all platforms. Editors and debuggers make your development more efficient. Your source library and build functions, via CMS and MMS on OpenVMS, are available on Windows NT Alpha with the CMS Client (element fetch and replace) and MMS Client (triggering OpenVMS builds). The functions of the base compiler driver on OpenVMS are provided for you by compiler drivers on Windows NT Alpha and DIGITAL UNIX, and these drivers combine the compile and link phases to save time. Plus, we've provided a GUI Interactive Compiler Driver (ICD) on Windows NT Alpha. There's more tools information in Chapter 4 and the user manuals. Regardless of development platform or deployment platform, the DIGITAL COBOL family has you covered.



Options

The COBOL command line qualifiers on OpenVMS, paralleled by the flags on DIGITAL UNIX, can *both* be used on Windows NT Alpha; the command modifiers are accepted in either format. Your command-line build procedures and favorite compile command files can be used on Alpha with little modification, and you can also use the Interactive Compiler Driver (ICD) to build complex command lines and store them for repeated use.

The full range of DIGITAL COBOL compiler options on all platforms (refer to *Qualifiers, Flags, Options...*, Chapter 6) bring power, efficiency, and productivity to your development projects.



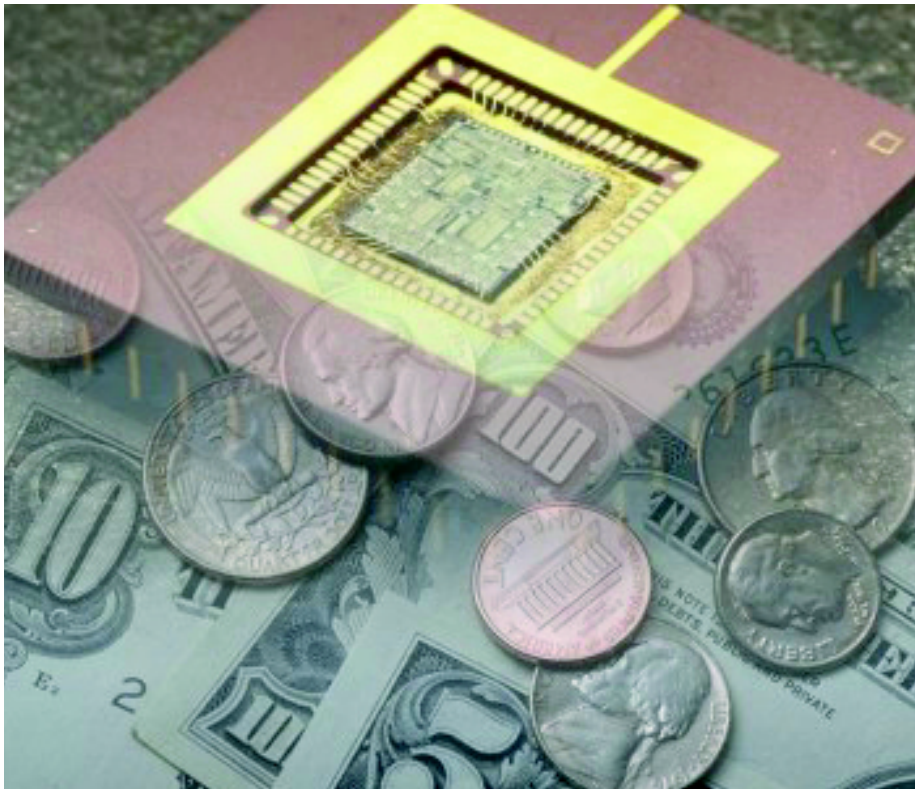
Resources

DIGITAL has an array of information resources available for you. You can contract for telephone support (and many of you do). Our award-winning technical documentation comes in several flavors to satisfy your appetite for information. Online Help and reference pages are on all four platforms. Bookreader documents are available on OpenVMS and DIGITAL UNIX, and PDF files are readable on Windows NT Alpha and DIGITAL UNIX. DIGITAL is also increasing its presence on the World Wide Web with HTML-based information resources. Try accessing http://www.openvms.digital.com/commercial/cobol for COBOL information, and http://www.digital.com/services for information about support services.

Hardcopy, online, and on the Internet, we've got the information you want!

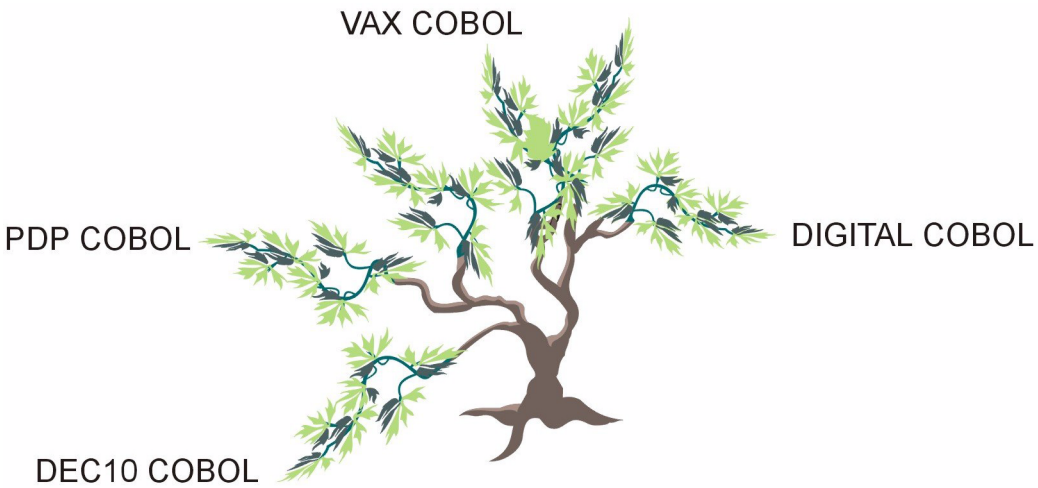
The Final "Why"

DIGITAL wants you to know that as your needs grow, we have a platform and a compiler to meet those needs—and it's neither exceedingly difficult nor budget-busting expensive to move your COBOL applications to the DIGITAL COBOL compiler on your next platform.



Chapter 3

Our DIGITAL COBOL Genealogy



Our DIGITAL COBOL family tree has strong, full branches and solid roots.

First, a Quick History of COBOL

The history of the COBOL programming language begins in 1943. Dr. Grace Murray Hopper, a professor in the mathematics faculty at Vassar College, decided to forego academia and join the Navy. She received orders to the Navy's Bureau of Ordnance computation project that was then under way at Harvard University. She was initially assigned to write coded instructions for the Mark I, the world's first large-scale digital computer.

At first the coding team wrote a complete set of instructions for each job that had to be done. Dr. Hopper found that many basic operations were being repeated for each job, and concluded that it would be far more efficient to write a set of instructions "...that would do a lot of the basic work over and over again for you." This was quite possibly the conception of the idea of compilers.

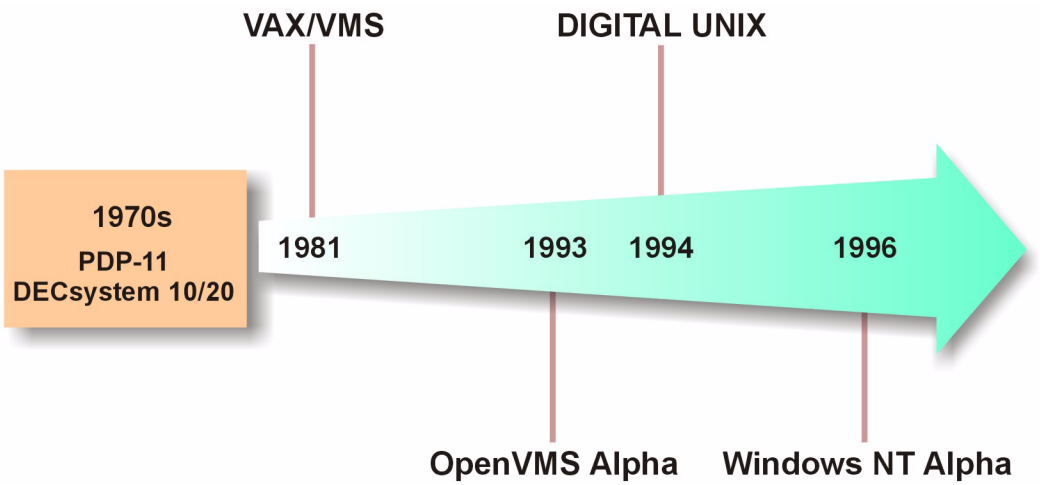
As an employee of Remington Rand in 1952 Dr. Hopper developed the A-0 compiler, which is considered to be the first formal program to accept generically coded input and produce machine instructions. A year later she developed the A-1, the first compiler to accept and resolve symbolic references. Armed with this experience and her knowledgeable opinions on programming languages, Dr. Hopper developed the first English language compiler, which was ultimately named B-0 (Flow-matic). This compiler was completed in 1957.

Not long after this accomplishment, Dr. Hopper met with other computer scientists, with the goal of developing the specifications for a common business language for computers. This meeting led to "Initial Specifications for a Common Business Language," a Department of Defense document. Thus was born COBOL.

In later years Dr. Hopper would be recalled to Navy service twice. In 1985 she received a promotion to Admiral, and finally retired from the Navy in 1986. That year she became a senior consultant for Digital Equipment Corporation. Dr. Hopper contributed significantly to the company until her death in 1992.

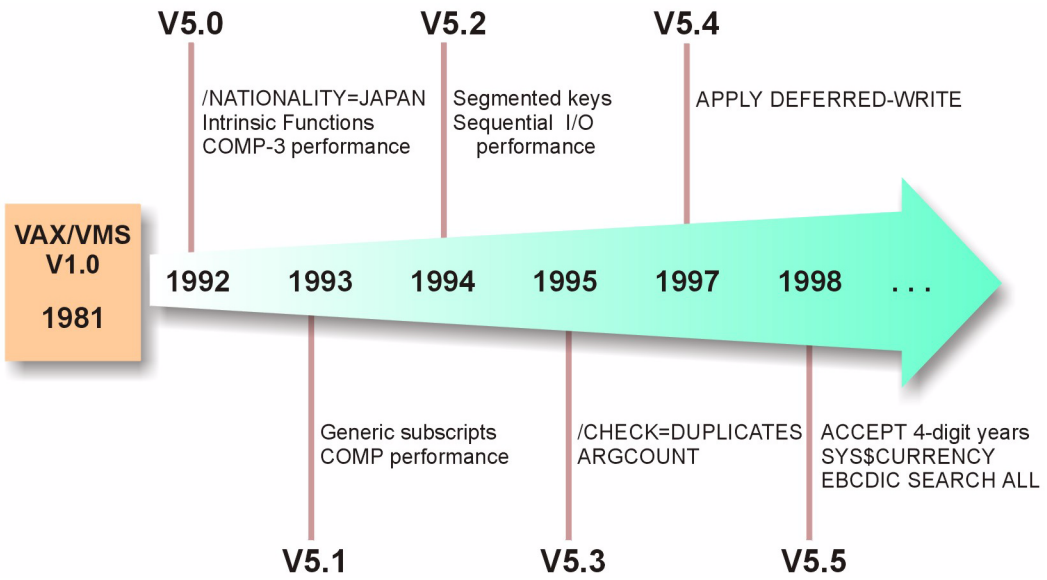
Our Family History

Our DIGITAL COBOL history began decades ago. Today's DIGITAL COBOL technology is built upon the best features and strengths of our earlier products. This fosters reliability, and provides a feature-rich COBOL for you. Meet the members of the DIGITAL COBOL family:



The family has run comfortably on OpenVMS VAX, DIGITAL UNIX, OpenVMS Alpha, and Windows NT Alpha for some time. The various family members might have been introduced to you as VAX COBOL, DEC COBOL, DIGITAL COBOL, or earlier product titles. By whatever name you recognize, this is a *stable* family.

DIGITAL VAX COBOL



Development

VAX COBOL design and development began in 1978 with the goal of bringing to the market a new high-performance COBOL product that ran on the VAX-11/780 hardware and VAX/VMS operating system. The original engineering team came from a background of PDP-11 COBOL and DECsystem 10/20 COBOL, and had early experience with other programming languages on VAX.

The design center for this new product was an early draft ANSI 1985 standard. Our efforts culminated in the release of VAX COBOL V1.0 in 1981, four years *before* the completion of the new ANSI standard.

The Product

VAX COBOL was renamed DIGITAL VAX COBOL at the time of the V5.5 release. The product takes full advantage of the OpenVMS operating system facilities and the VAX architecture.

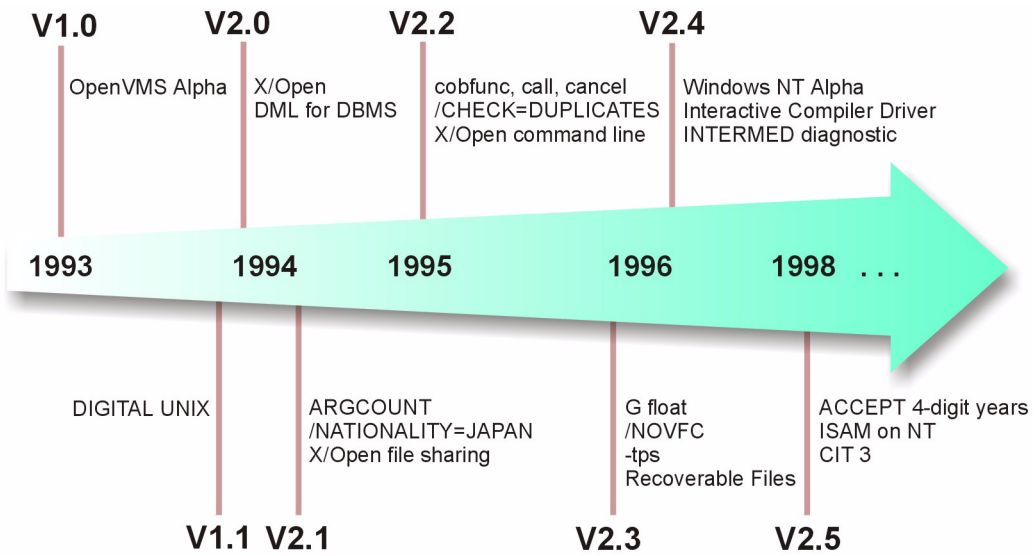
With the recent releases on OpenVMS VAX, DIGITAL VAX COBOL has extended functionality and improved performance with a continued focus on robustness and reliability. With the addition of intrinsic functions and generic subscripts for intrinsic functions, the product is now ANSI-85/-89 compliant at the HIGH level.

DIGITAL VAX COBOL also includes integrated support for multi-byte character handling. New I/O enhancements such as segmented key support and more complete checking of ISAM key declarations give the product more compatibility with file handling when used in conjunction with other programming languages and in other implementations of COBOL.

Numerous performance improvements have been made, primarily in the areas of arithmetic computation (namely, COMP-3 packed decimal and COMP binary) and sequential I/O (specifically, APPLY DEFERRED WRITE). In response to customer requests, DIGITAL VAX COBOL now includes more capabilities for currency sign handling and for handling 4-digit years (targeting increasingly important year 2000 issues).

The DIGITAL VAX COBOL V5.5 Release Notes include a complete summary of the new functionality and performance improvements introduced over the six (V5.x) releases during the last seven years. Detailed product information can be found in the Software Product Description which you can obtain by calling 1-800-DIGITAL, and at http://www.openvms.digital.com/commercial/cobol/ .

DIGITAL COBOL



Development

DIGITAL COBOL for Alpha made the highly successful DIGITAL COBOL compiler technology available on the new high-performance Alpha architecture. Significant performance gains were made by the use of GEM, the highly advanced code generator and optimizer that DIGITAL uses in its family of languages. This extended family includes Ada, BASIC, COBOL, C, C++, Fortran 77, Fortran 90, and Pascal.

The Product

DIGITAL COBOL emerged first on the Alpha architecture in 1993 (as DEC COBOL) on OpenVMS Alpha, then in 1994 on DIGITAL UNIX. In 1997, in response to customer demand, it was introduced on Windows NT Alpha.

Beginning in 1993, features, functionality, and performance improved with each release of DIGITAL COBOL. Building upon the feature-rich VAX COBOL product that evolved into DIGITAL VAX COBOL V5.5 (as described in the previous section, DIGITAL VAX COBOL), we added X/Open functionality, and more. The X/Open extensions include file sharing and record locking, screen handling, command line and environment variable / logical (OpenVMS) processing. An Interactive Compiler Driver was provided in 1997 to supplement the conventional command-line compiler driver (refer to Interactive Compiler Driver (ICD) in Chapter 4).

To give you more control over certain arithmetic operations, we added CIT3, an option thta provides increased arithmetic compatibility with DIGITAL VAX COBOL. On Windows NT Alpha we provide ORGANIZATION INDEXED (by using third-party ISAM products). On all platforms we give you four-digit year support for the ACCEPT statement.

Detailed product information can be found in the Software Product Description, which you can obtain by calling 1-800-DIGITAL, and at:

http://www.openvms.digital.com/commercial/cobol/

Migration—What's Involved?

With offerings on two architectures and three different operating systems, DIGITAL COBOL is naturally *designed* with compatibility and migration in mind. Whether you are considering migrating from VAX to Alpha, or between operating systems, you will be happy to know that DIGITAL COBOL products have the features and capabilities you need to accomplish this.

From the beginning, DIGITAL COBOL on OpenVMS Alpha was designed to duplicate as many VAX COBOL features as possible. Our Alpha-based COBOL contains all the RMS- and language extensions you might have used in your VAX applications. Naturally, there are some architectural differences, but the history of successful ports by many of our customers has proven that the differences are *manageable* and *worth it*.

There is information in the book you are reading to help you get started on the path of migration. For additional information on migration or compatibility, be sure to check out *VAX COBOL and DIGITAL COBOL Compatibility and Migration*, Appendix B of the User Manual.

Migration from VAX to Alpha

Many applications will migrate from VAX to Alpha with no changes needed! Generally, it's as simple as copying, unchanged, your COBOL data (sequential, relative, indexed...) and source code files to the new platform and recompiling there. Most of DIGITAL's COBOL extensions are available on the VAX and Alpha architectures (see Chapter 7 for a table that summarizes the available extensions).

Most of the familiar DIGITAL tools you expect (SORT, DECforms, Oracle CDD/Repository, Oracle CODASYL DBMS, Oracle Rdb...) are also there on OpenVMS Alpha. As an added feature, the compiler commands you use in your build system are provided in a parallel model (see Chapter 6).

Platform-specific differences do exist. For example, Alpha's double precision (D_float) data type is not 100% compatible with the D_float data type on VAX. However, Alpha supports *other* floating point data types, so if your VAX application requires float, and if it utilizes D_float, you can recode that section and go ahead with your development. Also, Alpha is a 64-bit system, so data alignment differences (for example, COMP alignment in record structures) may require attention.

Some other key OpenVMS (VAX to Alpha) migration areas that you might need to examine include:

- Subtle differences in ACCEPT/DISPLAY handling
- /STANDARD=V3 (pre-ANSI-85 results)
- Arithmetic operations (overflow and intermediate precision)

You will find more details on these and other migration topics in *VAX COBOL and DIGITAL COBOL Compatibility and Migration*, Appendix B of the User Manual.

Migration between Our Operating Systems

On DIGITAL COBOL's three Alpha platforms, OpenVMS, DIGITAL UNIX, and Windows NT Alpha, you will find a maximum amount of compatibility, despite the differences inherent in these three operating systems.

Our compilers support the full ANSI-85/89 definition of the COBOL language. In addition, if your needs include file sharing, terminal I/O extensions, report writer, and other such features, we have them for you. (See the Cross-Platform Compatibility matrix in Chapter 7.)

It's a fact that different operating systems have underlying differences that can impact your application. For example, DIGITAL UNIX and Windows NT Alpha do not offer support for RMS (the OpenVMS file system), and DIGITAL UNIX and Windows NT Alpha do not *natively* provide ISAM.

To meet your ISAM needs on those two platforms, we point you to third party ISAM's, that are tested and reliable. We've built in compatibility for sequential and relative files that enables you to move these files from OpenVMS under RMS to DIGITAL UNIX and Windows NT Alpha. You can use the files immediately—*without* need for data file conversion. In short, we recognize the differences, and we help you to deal with them.

Some other key platform (OpenVMS to DIGITAL UNIX or Windows NT Alpha) migration areas that you will want to consider will be:

- Unavailability of specific tools such as Oracle CDD/Repository and DBMS
- Differences in debugging tools and source code management tools
- CALL USING BY DESCRIPTOR

You will find more differences in *VAX COBOL and DIGITAL COBOL Compatibility and Migration*, Appendix B of the User Manual, but these are manageable.

Migration from Other Vendors

When we engineered DIGITAL COBOL on Alpha we included numerous X/Open COBOL extensions to make it easier to migrate from non-DIGITAL environments. These extensions include:

- X/Open SCREEN SECTION
- X/Open File sharing and record locking
- X/Open LINE SEQUENTIAL
- X/Open command line and environment variable–(VMS) logical processing
- X/Open RETURN-CODE

This means that DIGITAL COBOL on Alpha provides you with compatibility with DIGITAL VAX COBOL extensions, as well as compatibility with X/Open extensions—all usable in the same program or across a suite of programs.

No company has the capacity to support all extensions of all other vendors of COBOL. Recognizing this, DIGITAL COBOL provides the FOREIGN_EXTENSIONS option to the RESERVED_WORDS qualifier. This enables diagnostic reporting (as well as suggested changes) of various COBOL extensions from other vendors that are not currently implemented in DIGITAL COBOL (see *Porting to DIGITAL COBOL from Other Compilers*, Appendix D of the User Manual for more details).

Where Do We Go from Here?

The world of COBOL is a complex one with a long, rich history of innovation and improvement. In today's rapidly evolving global economy, this history creates opportunities, but it also presents problems for which no *single* COBOL product can provide all the solutions.

The ability to migrate within our DIGITAL COBOL family offers your business the best access to the tools for taking advantage of the opportunities and resolving the problems. In the following chapters, you will find more information on qualifiers, I/O, switches, and other tools DIGITAL COBOL provides, to help you utilize migration to achieve your goals.

The following chapters in this book will also provide you with valuable reference material you can apply on the platform you are using as your development and deployment home.

Chapter 4

Developing, Debugging, and Maintaining COBOL Programs

Depending on your target platform, you have a variety of tools and environmental features available to you for work on the life cycle of your COBOL programs. In general, the basic life cycle tools that you need are an editor, a librarian, and a debugger. Optional tools include a data dictionary, code analyzer, and graphical development environment. We provide you with all of these.

OpenVMS Features: DIGITAL COBOL on VAX and Alpha

The OpenVMS operating system, whether on Alpha or VAX, offers an extensive and mature programming environment. This section summarizes some of its important features and tools.

Language-Sensitive Editor (LSE)

The Language-Sensitive Editor (LSE) is a powerful and flexible text editor designed specifically for software development. It provides the following:

- Templates, which are formatted language constructs, for easy and accurate COBOL coding
- The COBOL keywords and correct punctuation are provided, and placeholders to indicate where you can add text
- Commands to compile, review, and correct compilation errors from within the editor
- Display of diagnostic messages, identifying the location of each error, if you specify /DIAGNOSTICS when you compile to produce a .DIA file
- Integration with the Code Management System (CMS) for efficient source file management
- View support, which provides a reverse-design facility with commands to compress code into overview line summaries, which you can edit to modify your code easily
- A callable report tool that can print views, standard reports, and customized

reports

To invoke LSE, issue the following command at the DCL prompt:

```
$ LSEDIT USER.COB
```

Source Code Analyzer (SCA)

The Source Code Analyzer (SCA) is an interactive tool for program analysis, closely integrated with LSE. You can invoke SCA through LSE.

SCA provides cross-referencing of program symbols and source files, and does static analysis of the relationships between subprograms, symbols, and files. SCA can easily find all references to a data item and identify program interdependencies.

The /ANALYSIS_DATA qualifier on the LSE COMPILE command (or the DCL COBOL compile command) causes the compiler to generate a data analysis file, which it stores in an SCA library that you have created.

For more information on LSE and SCA, see the LSE/SCA documentation.

Debugger

The OpenVMS Debugger is a powerful, mature symbolic debugger that you can use with COBOL programs on OpenVMS systems to track down logical or data errors. As you execute a program (which you have compiled and linked with the /DEBUG qualifier), the Debugger allows you to manipulate the flow of control by stepping through instructions, setting breakpoints, and so forth, and to examine and modify variables. You can also use the Debugger to examine a failed machine code instruction. Screen mode is a useful option.

The Debugger uses:

- Traceback information, for an ordered list of active blocks
- A symbol table of al variables, with names and locations
- A source line correlation table to associate lines in your source file with lines in your executable program

See HELP COBOL/DEBUG, HELP DEBUG, and the user manual for your platform for additional information. For complete information, see the OpenVMS Debugger Manual.

Code Management System (CMS)

The Code Management System (CMS) efficiently manages libraries of source code files. As programmers on your team access various source files to develop or maintain applications, CMS provides automatic security, easy but controlled accessibility, prevention of file conflicts, and maintenance of file histories. To use CMS, individual

programmers use simple fetch, reserve, and other commands to get the most recent files, and discover instantly whether another team member is currently modifying a file.

Performance and Coverage Analyzer (PCA)

The Performance and Coverage Analyzer (PCA) can be valuable when you are seeking to improve run-time performance of your program. With PCA, you can target specific areas of programs that require unusual amounts of CPU time. For example, if PCA tells you that 80% of the processing time is used by COBOL routines that use the COMP-3 data type, which only makes up 20% of your routines, you may consider converting these routines to COMP.

Record Management Services (RMS)

On OpenVMS systems, Record Management Services (RMS) is used by the DIGITAL COBOL or DIGITAL VAX COBOL Run-Time Library (RTL) to perform input/output (I/O). You can also access the RMS special registers, which contain the primary and secondary RMS completion codes of an I/O operation, to analyze errors.

Several DIGITAL extensions to the APPLY clause are available to tune RMS access to files for improved run-time performance.

Refer to the OpenVMS RMS documentation for complete information about RMS.

System Services

The callable OpenVMS System Services (which use the prefix SYS$) are prewritten routines with many uses. System services provide basic operating system functions, interprocess communication, and various control resources to OpenVMS users.

Oracle CDD/Repository

On OpenVMS systems (VAX or Alpha), you can use Oracle CDD/Repository to maintain shareable data definitions, such as record and field definitions. In large organizations, many data repositories can be linked to form one logical repository, so that users across an organization can access the common data definitions, using various languages.

In COBOL, the COPY FROM DICTIONARY statement copies member definitions in CDD/Repository.

Oracle CDD/Repository can store dependency information so that, for example, if a record definition needs to be changed, you can analyze the impact the change will have on the programs that use it.

Oracle CDD/Repository is an optional product available under a separate license from Oracle Corporation. (It is not available for UNIX or Windows NT systems.)

Oracle DBMS

Oracle DBMS, including Data Manipulation Language (DML) subschema access, is supported by DIGITAL COBOL on OpenVMS systems.

Oracle DBMS syntax includes the following language elements:

SUB-SCHEMA, DB, DB-EXCEPTION, LD, COMMIT, CONNECT, ERASE, FETCH, FIND, FREE, GET, KEEP, MODIFY, READY, RECONNECT, ROLLBACK, STORE, RETAINING, WHERE, EMPTY, MEMBER, OWNER, and TENANT.

Online Documentation

Complete information is easily accessible for DIGITAL COBOL and DIGITAL VAX COBOL on line, as follows:

- Online help provides instant, detailed information about COBOL, including the complete Release Notes, from the DCL command line. Just type HELP COBOL.
- The complete DIGITAL COBOL and DIGITAL VAX COBOL user and reference manuals are on line in Bookreader format for users with workstations.

DIGITAL COBOL for DIGITAL UNIX

FUSE

DIGITAL COBOL for DIGITAL UNIX systems supports DEC FUSE, the DIGITAL integrated software development environment for DIGITAL UNIX workstations. FUSE helps the programmer with coding, building, debugging, performance analysis, and code management. Some FUSE tools are layered on commonly used UNIX tools including make, prof, rcs, and sccs. FUSE tools have easy-to-use Motif graphic interfaces, including one for its text editor (the emacs and vi editors are also integrated).

The DEC FUSE integration framework allows tools to invoke one another and trigger operations. Programming tasks are automated and streamlined, reducing some operations to a single mouse button click.

With DEC FUSE you can:

- Distribute, build, and run compilations in parallel
- Create and use nested (recursive) makefiles
- Use default configurations to automatically set up tools for a new session
- Create custom language templates for use with the DEC FUSE editor

- Specify preprocessing and postprocessing scripts for code management operations
- Tailor your work environment, selecting your editor and debugger, as well as personal favorite (and meaningful) colors and fonts, for example
- Save your work context, and reestablish that work context from session to session.
- Automatically start and configure your chosen development productivity tools

See the DEC FUSE handbook for additional, detailed information.

Ladebug

Ladebug is a symbolic source code debugger with a command-line interface. dxladebug uses a graphical user interface. The debugger is also integrated as one of the DEC FUSE tools. It supports most DIGITAL COBOL data types.

With the debugger you can:

- Control the execution of individual source lines in a program
- Set stops (breakpoints) at specific source lines or under various conditions
- Change the value of variables within the debugging environment
- Refer to program locations by their symbolic names, using the debugger's store of information on the DIGITAL COBOL language to determine the proper scoping rules and how the values should be evaluated and displayed
- Print the values of variables and set a trace (tracepoint) to notify you when the value of a variable changes
- Execute shell commands, examine core files, examine the call stack, display registers, and perform various other functions

The DIGITAL COBOL user manual provides in an appendix a sample Ladebug debugging session. For complete information, refer to the Ladebug Debugger manual, part of the DIGITAL UNIX documentation set.

Online Documentation

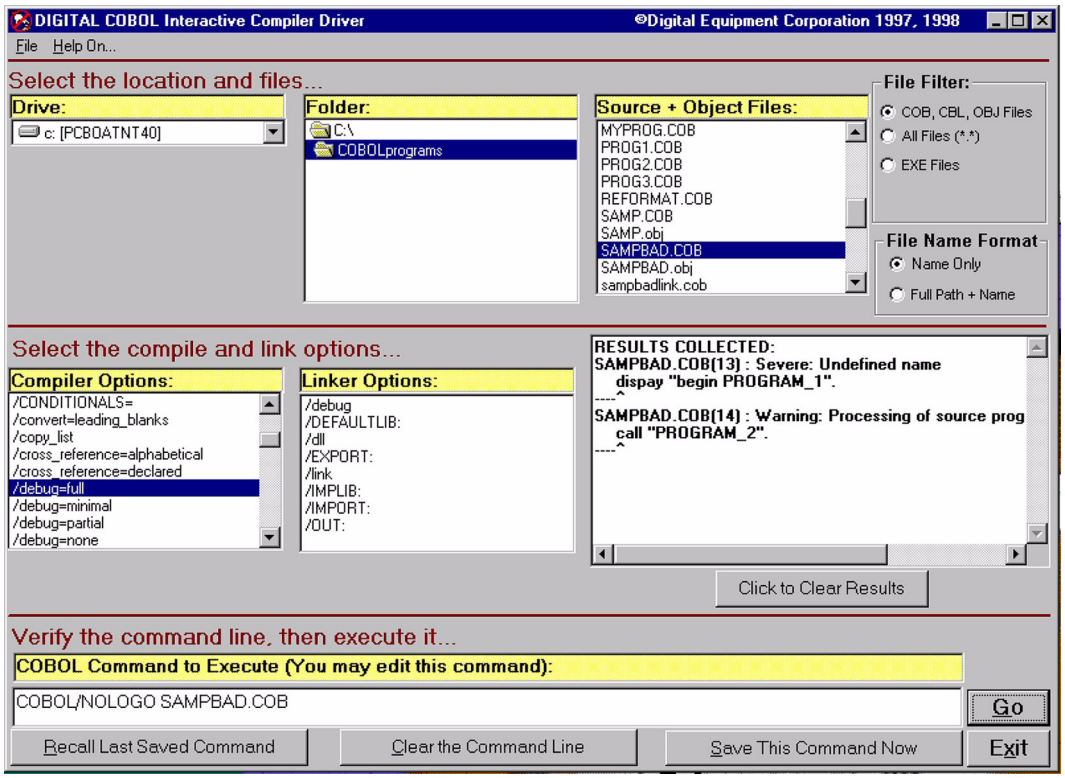
You can access online information about DIGITAL COBOL for DIGITAL UNIX by typing man cobol at the system prompt.

DIGITAL COBOL for Windows NT Alpha

Interactive Compiler Driver (ICD)

The Interactive Compiler Driver (ICD) is a graphical tool provided with the DIGITAL COBOL kit. With it, you can use simple mouse clicks to select:

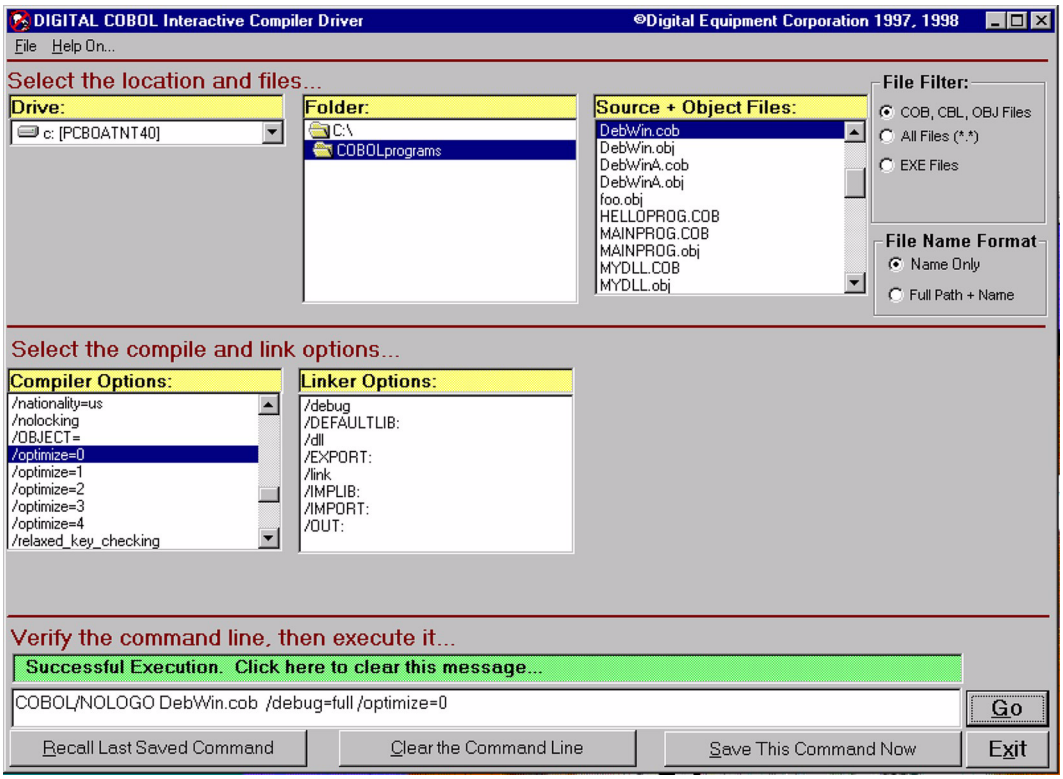
- Working device
- Working directory
- Filtered file lists and the file to compile
- Compiler options
- Link options



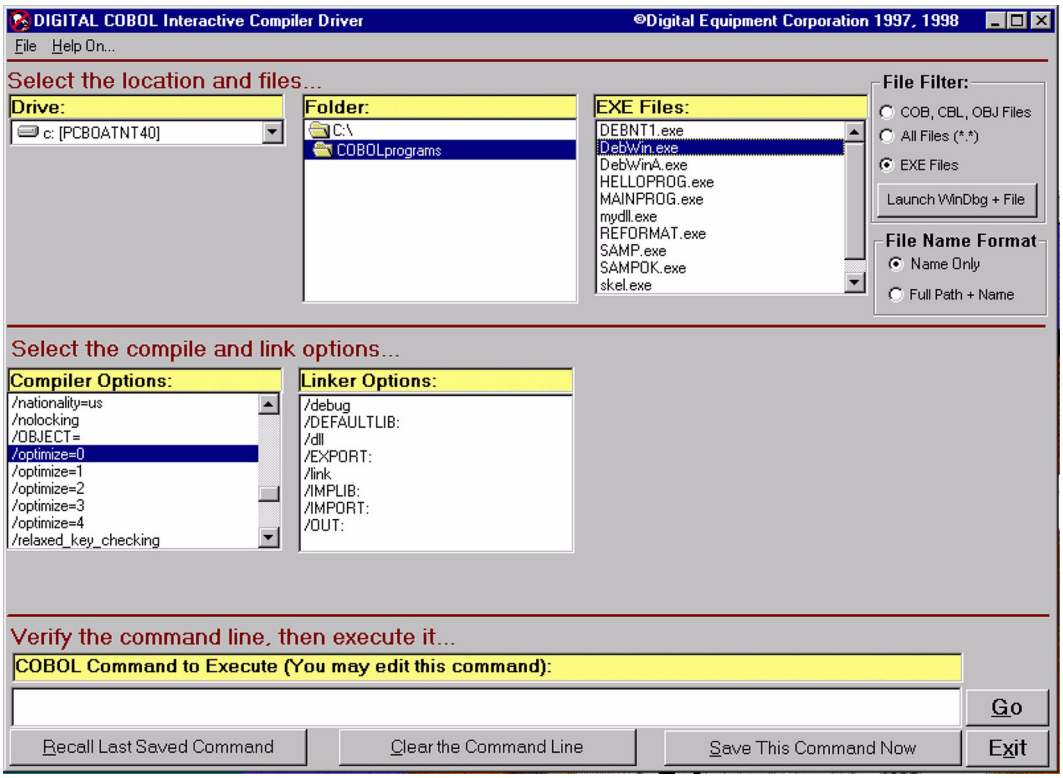
In the previous illustration the file SAMPBAD.COB has been compiled. The source contains an error that resulted in a compile error message. It appears that line 13 of the source has a typographical error. You can right-mouse click on the source file that contains the error to open that file in the Notepad editor and make the necessary correction.

Features of the ICD include:

- Construction and execution of command lines without extensive typing
- Display of compiler and linker messages
- Access to the Notepad editor to open source files and other files
- Access to the DIGITAL COBOL online *User Manual* and *Online Reference*
- Context retention (subsequent launches returns you to the device and directory where you were last working)



In the preceding illustration the file DEBWIN.COB has been compiled with debug symbols enabled and no optimization. There were no errors in the compilation.



The preceding illustration shows that the EXE file filter has been applied, so that only files with the EXE extension will be shown. This step enables a control that has not previously been seen here: the "Launch WinDbg + File" button. Since the DEBWIN.EXE file has been selected, if you click on the "Launch..." button now, the debugger will be launched with the DEBWIN.EXE filespec passed to it.

The ICD is a native Alpha image, and requires no translator or interpreter levels.

A sample ICD session is shown in the appendix on tools in the DIGITAL COBOL *User Manual*.

WinDbg

The Microsoft WinDbg debugger is a windows-based debugger provided on the DIGITAL COBOL kit. It is installed in the \MS_Tools directory under the other DIGITAL COBOL directories.

WinDbg has the following features:

- Easy setting of breakpoints by double clicking on a line in the source file to position the text cursor, then selecting Debug|Breakpoints|Add|OK
- Color highlighting: yellow for current line of source code and red for breakpoints, for example
- Examining and watching variables, in a debugger window that shows variable values, with watchpoints on variables that you select
- Customized positioning and sizing of various WinDbg windows to display debugging information as needed
- An option to select case insensitivity, useful for the case-insensitive COBOL language

For additional information on WinDbg, refer to the Microsoft WinDbg documentation.

Microsoft Developer Studio Debugger

DIGITAL COBOL for Windows NT Alpha Systems produces code that can be debugged with the Microsoft Developer Studio debugger (msdev.exe). This powerful debugger can be used to step through source code, set breakpoints, examine variables, access memory, and other functions.

CMS Client for Windows NT

The Windows NT connection with CMS is the CMS Client for Windows NT Alpha. The CMS Client allows you to access the CMS server (on an OpenVMS system) where CMS has been installed, using a Windows-style interface. CMS is the online library component of DECset that helps track software development and maintenance, as well as providing a useful repository for source code, binary data, and documentation source files.

Contact your DIGITAL sales office for additional information.

Online Documentation

The COBOL /? command provides online help. In addition, the DIGITAL COBOL manuals are available in Adobe Acrobat-readable .PDF format. The Acrobat Reader is provided on the kit.

Chapter 5

Compiling, Linking, and Running

What do you do with your program code? You compile the source, link the objects, and run the executables. Plus, your users may well run your program on a continuing basis. Your compile/link/run command can convey explicit directions for the actions to be taken upon your source files, object files, listing files, and executable files. You can direct actions of the compiler parser, engine, and back end. You can carry a password to an executable that enables operations, specifies a file or files on which to operate at run time, and directs internal operations of program logic.

DIGITAL COBOL supports command-line interface on OpenVMS, DIGITAL UNIX, and Windows NT Alpha. The Interactive Compiler Driver provides you with a graphical interface for compiling, linking, running, editing, and debugging.

Command Sequence and Format

Your command sequence can be as follows:

Simple Compile-Link-Run Commands

```
COBOL MYPROGRAM
LINK MYPROGRAM
RUN MYPROGRAM (or simply: MYPROGRAM)
```

Note: The command-line prompt (for example, $, %, C:) associated with your system would ordinarily appear before these commands.

You can convey extra instructions and information to the commands by adding modifiers to the command line. These modifiers are relatively consistent across the platforms, although there are some exceptions. The *Modifying the COBOL Command* chapter lists all of the command modifiers.

Compilers on OpenVMS receive command lines directly from your input device (terminal, file, and so forth). Compilers on DIGITAL UNIX and Windows NT Alpha have a compiler driver as an intermediate layer between your commands and the compiler engine.

Command syntax for the various platforms on which our COBOL family runs as follows:

Compile-Link-Run on OpenVMS VAX

```
COBOL[/COMMAND-QUALIFIER]... {FILE-SPEC [/FILE-QUALIFIER] ...}...
LINK[/COMMAND-QUALIFIER] ... {FILE-SPEC [/FILE-QUALIFIER] ...}...
RUN[/[NO]DEBUG] {FILE-SPEC}
```

Compile-Link-Run on OpenVMS Alpha

```
COBOL[/COMMAND-QUALIFIER]... {FILE-SPEC [/FILE-QUALIFIER] ...}...
LINK[/COMMAND-QUALIFIER] ... {FILE-SPEC [/FILE-QUALIFIER] ...}...
RUN[/[NO]DEBUG] {FILE-SPEC}
```

Compile-Link-Run on DIGITAL UNIX

```
cobol [-flags [options]]... {file-spec}
file-spec
```

Compile-Link-Run on Windows NT Alpha

```
COBOL [COMPILER OPTIONS] {FILE-SPEC}...  [/LINK [LINKER OPTIONS]]
FILE-SPEC
```

Note that compiling, linking, running and debugging on Windows NT Alpha can also be accomplished with the Interactive Compiler Driver (ICD). The ICD is described in *Developing, Debugging, and Maintaining COBOL Programs,* Chapter 4.

Chapter 6

Modifying the COBOL Command

Qualifiers, Flags, Options...

Regardless of what you call them, you can add *modifiers* to the COBOL command. These modifiers will perform specific functions during the lifetime of your compilation, and will directly affect the results of the compilation.

Modifiers offer you a variety of options for compiling, debugging, and documenting your programs. Some modifiers have optional keywords that you can add, to further define the action that you desire. In the following example, ALIGNMENT is a command modifier, and the PADDING keyword is added:

Example COBOL /ALIGNMENT=PADDING myprogram.cob

Note: When a modifier accepts multiple keywords, you can specify more than one and separate them with commas (except on DIGITAL UNIX, where the modifier is repeated for reach keyword). Refer to the syntax of the specific modifiers for guidance on format (for example, the use of parentheses).

The tables on the following pages show modifiers for the COBOL command on four supported platforms. **D** indicates the default, and **NA** indicates that that particular modifier does not apply to the indicated platform. Square brackets (**[]**) indicate that the enclosed item is optional, and braces (**{}**) surround required items. You provide *italicized* items as required (such as file specifications). The modifiers appear in alphabetical order.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/?** **equivalent to /HELP**
Display the COBOL HELP file on Windows NT. On other platforms Help is displayed by entering an operating system command, rather than by a COBOL command modifier.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**/ALIGNMENT [=[NO]PADDING] /NOALIGNMENT (D)**	**-align -align padding**	**/ALIGNMENT [=[NO]PADDING] /NOALIGNMENT (D)**
Specifies the alignment of binary data items within record structures. Specifying ALIGNMENT uses OpenVMS Alpha data alignment to increase performance and conformity to the Alpha Calling Standard. ALIGNMENT specifies natural alignment; it aligns all COMP, COMP-1, COMP-2, INDEX, and POINTER data along natural boundaries. A natural boundary is the smallest boundary at which data can be aligned without crossing the next boundary for that type. For example, longword is the natural boundary for four byte integers. Specifying ALIGNMENT is equivalent to using the SYNCHRONIZED clause or using ALIGNMENT compiler directives. (Refer to the reference manual for your platform for information about the SYNCHRONIZED clause.) ALIGNMENT=PADDING indicates Alpha natural alignment and padding of records according to the DIGITAL Alpha Calling Standard. The default, NOALIGNMENT, specifies OpenVMS VAX-compatible data alignment. This modifier aligns data on byte boundaries for compatibility with DIGITAL VAX COBOL and other OpenVMS VAX languages. The alignment you specify remains in effect throughout a given compilation, except as modified by ALIGNMENT compiler directives. For more information about the behavior of the ALIGNMENT modifier and ALIGNMENT compiler directives, refer to the Directives section (in online HELP) and the DIGITAL COBOL User Manual.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/ANALYSIS_DATA [=*filespec*] /NOANALYSIS_DATA (D)	**/ANALYSIS_DATA [=*filespec*] /NOANALYSIS_DATA (D)**	**NA**	**NA**
Indicates whether an ANA file is created during compilation. If you have the DEC Source Code Analyzer (SCA) installed on your system, you can use ANALYSIS_DATA to generate an output file of source-code analysis information. The source-code information file generated by ANALYSIS_DATA has a default file name of the primary source file and a default file type of ANA. SCA uses the ANA file to display information about program symbols and source files. The default, NOANALYSIS_DATA, does not generate an ANA file. For more information about SCA, see the Guide to Source Code Analyzer for OpenVMS Systems.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/ANSI_FORMAT **/NOANSI_FORMAT (D)**	**/ANSI_FORMAT** **/NOANSI_FORMAT (D)**	**-ansi**	**/ANSI_FORMAT** **/NOANSI_FORMAT (D)**
Indicates that the source program is in conventional ANSI format. The compiler then expects 80-character card image records with optional sequence numbers in character positions 1 through 6, indicators in position 7, Area A beginning in position 8, Area B beginning in position 12, and the identification area in positions 73 through 80. The default, NOANSI_FORMAT, indicates that the source program is in DIGITAL terminal format, where Area A begins in position 1, Area B begins in position 5, and the source program records do not have line numbers.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/AUDIT **[=("*string*",...)]** **/NOAUDIT (D)**	**/AUDIT** **[=("*string*",...)]** **/NOAUDIT (D)**	**NA**	**NA**
Controls whether user-supplied text is included in a CDD/Repository history list entry if a compilation accesses the dictionary. You can specify from 1 to 64 strings with AUDIT. If you specify more than one string, separate them with commas and enclose the list in parentheses. If you specify AUDIT without a string, the compiler creates standard history list entries with no additional text in the dictionary for COPY FROM DICTIONARY records and for information put in the dictionary as a result of specifying DEPENDENCY_DATA. Only one user-supplied string is included in these entries, even though up to 64 can be specified. The default, NOAUDIT, suppresses the creation of history list entries.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/BRIEF_HELP** **/NOBRIEF_HELP (D)**
Produces a syntax-only version of the HELP file. The default is NOBRIEF_HELP			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-C**	**NA**
Verifies the range of subscripts and reference modifiers. This is the equivalent of -check bounds			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-c**	**/COMPILE_ONLY** **/NOCOMPILE_ONLY (D)**
Does not invoke the linker. Creates .\file1.obj for default /OBJECT.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-call_shared**	**NA**
Produces a dynamic executable that uses shareable objects during run time. The linker searches for unresolved references in shared library (.so) files before searching in archive library (.a) files. The run-time loader is invoked to bring in all required shareable objects and resolve any symbols that remained undefined during static link time. This is the default.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/CHECK[=(*keyword*,...)] **/NOCHECK (D)**	**/CHECK[=(*keyword*,...)]** **/NOCHECK (D)**	**-check *keyword*** **-check none (D)**	**/CHECK[=(*keyword*,...)]** **/NOCHECK (D)**

Specifies conditions to be checked at execution time. Controls whether the system checks the validity of numeric digits or PERFORM statements, indexes, subscripts, reference modification, and the OCCURS DEPENDING ON depending item for specific run-time errors. If you specify any single modifier keyword, the default keywords do not change unless they are individually modified.

You can select one or more of the following keywords:

[NO]PERFORM	Verifies that PERFORM statement rules are met. The default is NOPERFORM.
[NO]BOUNDS	Verifies the range of subscripts and reference modifiers. The default is NOBOUNDS.
[NO]DECIMAL	Validates numeric digits when using display numeric items in a numeric context. The default is NODECIMAL.
[NO]DUPLICATE_KEYS (OpenVMS only)	Verifies that the duplicate key specification for indexed file keys in the source program matches the duplicate key specification in the physical file. The default is NODUPLICATE_KEYS.
ALL	Same as CHECK with no keywords.
NONE	Same as NOCHECK. Default if CHECK is not used.

Specifying CHECK PERFORM controls whether the system checks PERFORM statements. Incorrect use of PERFORM statements can produce unpredictable results and, when used with the PERFORM keyword, causes the system to generate a run-time message and abort the program.

Specifying CHECK BOUNDS controls whether the system checks the range of subscripts, indexes, and the depending item in the DEPENDING ON phrase of the OCCURS clause. The system generates a run-time message and aborts the program if it detects one of these errors:

- If DEPENDING ON is not specified and a subscript or index is greater than the upper bound or less than or equal to zero
- If DEPENDING ON is specified and a subscript or index is greater than the depending item or less than or equal to zero
- If a depending item is less than the low bound or greater than the upper bound, and either a subscripted or indexed item references a table or a group containing the table is referenced as a sending item

Specifying /CHECK=DECIMAL controls whether the system checks for numeric characters when using numeric display items in a numeric context and generates an error if any digit is invalid (not numeric).

(Continued next page)

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
(CHECK continued) Specify CHECK DECIMAL to validate data produced by other systems that might use a different internal representation for numeric data. You can also use this modifier keyword to detect logic errors in programs that result in text data being moved to numeric data items. Specifying CHECK produces extra instructions to perform these checks, which may result in slightly larger images and slightly longer execution times than the CHECK NODECIMAL modifier keyword. The default, NOCHECK, is the equivalent of CHECK NONE. Specify CHECK with the NOOPTIMIZE modifier, as optimization can make finding error found by CHECK more difficult.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/CONDITIONALS {=(*selector,...*)} **/NOCONDITIONALS (D)**	**/CONDITIONALS {=(*selector,...*)}** **/NOCONDITIONALS (D)**	**-conditionals *selector***	**/CONDITIONALS {=*selector*}** **/NOCONDITIONALS (D)**
Controls whether the conditional compilation lines in a source program are compiled or treated as comments. Specifying CONDITIONALS=(selector,...), where a selector is a list of one or more characters from A to Z, results in the selected conditional compilation lines being compiled. If you specify more than one selector, separate them with commas and enclose the list in parentheses. The default, NOCONDITIONALS, results in all conditional compilation lines being treated as comments during compilation.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**/CONVERT=*keyword*** **/NOCONVERT [D]**	**-convert *keyword***	**/CONVERT=*keyword*** **/NOCONVERT [D]**
The keyword options are: LEADING_BLANKS and NOLEADING_BLANKS Instructs the compiler to check for and change blanks to zeros in numeric display items. This modifier is particularly useful for porting your existing VAX COBOL programs to an OpenVMS Alpha system, because at run time blanks in the data will be changed to zeros. Specifying CONVERT LEADING_BLANKS produces extra instructions to perform these data conversions, which may result in slightly larger images and slightly longer execution times.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/COPY_LIST **/NOCOPY_LIST (D)**	**/COPY_LIST** **/NOCOPY_LIST (D)**	**-copy** **-copy_list** **-show copy**	**/COPY_LIST** **/NOCOPY_LIST (D)**

Controls whether source statements included by COPY statements are printed in the listing file.

COPY_LIST has no effect unless you also specify LIST.

The default, NOCOPY_LIST, suppresses the listing of text copied from library files; only the COPY statement appears in the listing file.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/CROSS_REFERENCE [=(*keyword*[,...])] **/NOCROSS_ REFERENCE (D)**	**/CROSS_REFERENCE [=(*keyword*[,...])]** **/NOCROSS_ REFERENCE (D)**	**-cross_reference,** **-show xrefkeyword**	**/CROSS_REFERENCE [=*keyword*]** **/NOCROSS_ REFERENCE (D)**

Controls whether the source listing file includes a cross-reference listing.

CROSS_REFERENCE has no effect unless you also specify LIST.

You can select one or both of the following keywords:

ALPHABETICAL (D)	The compiler sorts user-defined names in alphabetical order and lists them with the source program line numbers on which they appear. CROSS_REFERENCE=ALPHABETICAL is the equivalent of CROSS_REFERENCE.
DECLARED	Produces a listing of user-defined names in order of declaration.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/DEBUG [=(*keyword*[,...])] /NODEBUG (D)	**/DEBUG [=(*keyword*[,...])] /NODEBUG (D)**	**-g [*n*]**	**/DEBUG [=*keyword*] /NODEBUG (D)**

Controls the amount and type of debugging information in the object file. The keyword names vary by platform as follows:

OpenVMS VAX *keyword*	**OpenVMS Alpha *keyword***	**DIGITAL UNIX *n***	**Windows NT Alpha *keyword***	**Effect**
SYMBOLIC	SYMBOLIC	-g2	NA	Provides local symbol table information, allowing debugger access to all data items by data name. On OpenVMS, paragraph names and section names are also included for debugger access.
TRACEBACK	TRACEBACK	-g1	PARTIAL	Provides PC to line number correlation, allowing line by line stepping and tracing. On OpenVMS, prohibits source code viewing. On DIGITAL UNIX this is the default.
ALL	ALL	-g	FULL	Provides symbolic and traceback information as described above.
NA	NA	NA	MINIMAL	Provides entry point and assembler code debugging only. Source code is not viewable.
NODEBUG	NODEBUG	-g0	NONE, NODEBUG	No debugging information is provided. This is the default (except for DIGITAL UNIX).

Debugging and optimization are interrelated as follows:

Platform	**Default Optimization Action**
OpenVMS VAX	NA
OpenVMS Alpha	Optimization defaults are unchanged when DEBUG is specified. Specifying NOOP with DEBUG is recommended.
DIGITAL UNIX	Optimization is turned off when debugging information is requested. Specify -g3 if DEBUG with optimization is desired.
Windows NT Alpha	Optimization is turned off when debugging information is requested. Specify OPTIMIZATION if debug with optimization is desired.

Debugging and COPY files require some explanation. On OpenVMS and Windows NT Alpha, you can view source lines from simple COPY files as well as from the main source file. However, the debugger cannot reference source lines from COPY statements which reference CDD/Repository or any line in which text has been replaced. On DIGITAL UNIX, source lines from COPY files are not available to the debugger.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/DEPENDENCY_DATA **/NODEPENDENCY_** **DATA (D)**	**/DEPENDENCY_DATA** **/NODEPENDENCY_** **DATA (D)**	**NA**	**NA**
Controls whether or not a compiled module entity is stored in CDD/Repository. If you have CDD/Repository installed on your system, you can use DEPENDENCY_DATA to store CDD/Repository relationship information in the dictionary. The information generated will correlate the DIGITAL COBOL program with: The object file created by the compilation CDD/Repository entities specified in COPY FROM DICTIONARY statements CDD/Repository entities explicitly referenced in the RECORD DEPENDENCY statements			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/DIAGNOSTICS **[=*filespec*]** **/NODIAGNOSTICS**	**/DIAGNOSTICS** **[=*filespec*]** **/NODIAGNOSTICS**	**NA**	**NA**
Creates a diagnostic file containing compiler messages and diagnostic information. The diagnostic file is reserved for use by DIGITAL. The Language-Sensitive Editor (LSE) uses the diagnostic file to display diagnostic messages and to position the cursor on the line and column where a source error exists. The default file type for a diagnostic file is DIA. The default, NODIAGNOSTICS, suppresses the creation of a diagnostic file.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/DLL=*filespec*** **/NODLL (D)**
Specifies that the program should be linked as a dynamic-link library (DLL) and named "filespec". The default filespec is *filespec*.DLL The default is NODLL			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-o *filespec***	**/EXE** **/EXE=*filespec*.exe (D)**
On Windows NT Alpha, specifies that the program should be linked as an executable image (.EXE) and given the name contained in filespec. The default is filespec.EXE. On DIGITAL UNIX, names the object file if -c is specified; otherwise, names the executable file.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/EXTCOB=*extension***
Specifies file extensions to be processed as COBOL source programs. The default extensions are .cob and .cbl.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/EXTLNK=*extension***
Specifies file extensions to be processed by the Linker. The default file extensions are: .obj, .lib, .o, .def, .rbj, .res, .exe, and .dll			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/FIPS=74 **/NOFIPS (D)**	**/FIPS=74** **/NOFIPS (D)**	**-fips 74**	**/FIPS=74** **/NOFIPS (D)**

Supports the Federal Information Processing Standards Publication 21–1 (FIPS-PUB 21–1), issued by the U.S. National Bureau of Standards, interpretation of file status.

FIPS-PUB 21–1 specifies that a file status of 10 be returned when reporting AT END conditions. The FIPS=74 modifier returns a file status of 10 when reporting AT END conditions.

The following table compares the file status values that are returned when you use or do not use the FIPS=74 modifier. AT END file status values apply to any file organization accessed sequentially.

The default, NOFIPS, ensures version to version compatibility for COBOL.

FIPS=74 and NOFIPS modifier only apply when you also specify STANDARD=V3.

FILE STATUS VALUES WITH STANDARD=V3:		
	FIPS=74	NOFIPS
The file has no next logical record	10	13
An optional file was not present	10	15
The program did not establish a valid next record	10	16

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/FLAGGER [=(*keyword*,...)] /NOFLAGGER (D)	**/FLAGGER [=(*keyword*,...)] /NOFLAGGER (D)**	**-flagger *keyword***	**/FLAGGER [=(*keyword*,...)] /NOFLAGGER (D)**

Allows you to specify a FIPS level of COBOL syntax, in accordance with the Federal Information Processing Standards Publication 21-3 (FIPS-PUB 21-3) issued by the U.S. National Bureau of Standards, beyond which the compiler generates informational messages. To receive these informational messages, you must also specify WARNINGS=ALL or WARNINGS=INFORMATIONAL. Use the FLAGGER modifier when you know that your target system's compiler has a low level of FIPS syntax support.

You can select one or more of the following keywords:

HIGH_FIPS	Flags language constructs that are above the FIPS high validation level, such as DIGITAL extensions to ANSI COBOL.
INTERMEDIATE_FIPS	Flags language constructs that are above the FIPS intermediate validation level, such as language constructs that are within the FIPS high validation level or DIGITAL extensions to ANSI COBOL.
MINIMUM_FIPS	Flags language constructs that are above the FIPS minimum validation level, such as language constructs that are within the FIPS high and intermediate validation levels or DIGITAL extensions to ANSI COBOL.
OBSOLETE	Flags language constructs that the ANSI1985 COBOL Standard identifies as obsolete. If a language construct is within the selected FIPS validation level or optional module and is also on the obsolete list, the compiler generates only the obsolete informational message.
OPTIONAL_FIPS	Flags language constructs that are within FIPS optional modules, including Report Writer and Segmentation.
REPORT_WRITER	A subset of OPTIONAL_FIPS that flags language constructs that are within the FIPS optional module Report Writer.
SEGMENTATION	A subset of OPTIONAL_FIPS that flags language constructs that are within the FIPS optional module Segmentation.
SEGMENTATION_1	A subset of OPTIONAL_FIPS that flags language constructs that are above the level 1 of the FIPS optional module Segmentation.

You can use any combination of modifier options. if you specify more than one validation level, the compiler uses the lowest level. If you use FLAGGER without specifying a FIPS level and with another option, the compiler assumes FLAGGER=HIGH_FIPS.

You cannot specify FLAGGER without also specifying STANDARD=V3.

For additional information about the required and optional modules for the COBOL language, refer to the American National Standard Programming Language – COBOL, ANSI X3.23-1895, ISO 1989-1985. For more information about the FIPS validation levels, see Federal Information Processing Standards Publication 21-3.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**/FLOAT =[*keyword*]**	**NA**	**NA**

Specifies the floating-point data format to be used in memory for single and double precision data items.

You can use FLOAT at compile time to specify either VAX F_floating or IEEE S_floating formats for single precision data items or VAX D_floating or VAX G_floating or IEEE T_floating formats for double precision data items. You cannot mix VAX and IEEE formats in the same compilation unit.

You can select one of the following keywords:

D_FLOAT	Specifies that the memory format for COMP-1 data is VAX F_floating and for COMP-2 data is VAX D_floating. The default is D_FLOAT.
G_FLOAT	Specifies that the memory format for COMP-1 data is VAX F_floating and for COMP-2 data is VAX G_floating.
IEEE_FLOAT	Specifies that the memory format for COMP-1 data is IEEE S_floating and for COMP-2 data is IEEE T_floating.

The IEEE standard for binary floating-point arithmetic, ANSI/IEEE 754-1985, defines four floating-point formats in two groups, basic and extended, each group having two widths, single and double. The Alpha architecture supports the basic single and double formats. Refer to the Alpha Architecture handbook for more information about using floating-point data types with the Alpha architecture.

Because the Alpha architecture is IEEE-compliant, you can run existing COBOL programs containing IEEE floating-point data formats on DIGITAL COBOL.

Specifying FLOAT without a keyword is equivalent to specifying FLOAT=D_FLOAT.

Operations on D-FLOAT on Alpha are not 100% compatible with operations on D-DLOAT on VAX.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**/GRANULARITY [=*keyword*]**	**-granularity {*keyword*}**	**/GRANULARITY [=*keyword*]**

Specifies the minimum size of a memory access. Use GRANULARITY if different processes sharing memory attempt to update different parts of the same aligned quadword concurrently.

The keyword can be as shown in the following table:

BYTE	Used if concurrent processes sharing memory may be updating different bytes within the same quadword.
LONG	Used if concurrent processes sharing memory may be updating different longwords within the same quadword.
QUAD	(default) Used for best performance if you are sure that no concurrent updates will occur within the same aligned quadword.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/HELP** **/NOHELP (D)**
Displays the help file on the standard output device. Equivalent to /?.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-K**	**/KEEP**
Does not remove temporary files created during compilation and linking. On DIGITAL UNIX, K does not affect the naming or location of temporary files. On Windows NT Alpha, KEEP uses the current directory, and uses the program name to name object files. To see the names and locations of the temporary files, specify -v.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-L**	**NA**
Prevents the linker from searching for libraries in the standard directories.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-L*dir***	**NA**
Directs the linker to search for libraries in *dir* before searching the standard directories.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/LINKMAP** **[=*filespec*]** **/NOLINKMAP (D)**
Specifies that the linker should produce a map file. Default filespec: file1.MAP			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/LIST **[=*filespec*]** **/NOLIST (D)**	**/LIST** **[=*filespec*]** **/NOLIST (D)**	**-list** **equivalent to -V**	**/LIST** **[=*filespec*]** **/NOLIST (D)**
Controls whether the compiler produces an output listing file. When you specify LIST, you can control the defaults applied to the output file specification by your placement of the modifier in the command (OpenVMS only). The listing file always contains a listing of the source program, including any diagnostics. Other portions of the listing file are optionally produced under the control of MACHINE_CODE, MAP, and CROSS_REFERENCE. LIST is required when you want to use CROSS_REFERENCE, COPY_LIST, FLAGGER, MACHINE_CODE, or MAP. In interactive mode NOLIST is the default. In batch mode, LIST is the default. If a filespec is not specified, the compiler will create an output listing file with the same file name as the input source file and with a default file type of LIS (on OpenVMS), lis (on DIGITAL UNIX), or LST (on Windows NT Alpha).			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-l*string***	**NA**
Searches for *string* library for ld. This flag should be placed at the end of the command line.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/MACHINE_CODE **/NOMACHINE_CODE (D)**	**/MACHINE_CODE** **/NOMACHINE_CODE (D)**	**-mach,** **-machine_code,** **-show code**	**/MACHINE_CODE** **/NOMACHINE_CODE (D)**
Controls whether the listing file contains a list of compiler generated machine code. MACHINE_CODE has no effect unless you also specify LIST. The default, NOMACHINE_CODE, suppresses compiler generated machine code in the listing file. If you specify MACHINE_CODE with OPTIMIZE, the machine code listing may not reflect the actual code executed for a given statement nor the order of execution.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/MAP **[=*keyword*]** **/NOMAP (D)**	**/MAP** **[=*keyword*]** **/NOMAP (D)**	**-map {*keyword*}**	**/MAP** **[=*keyword*]** **/NOMAP (D)]**

Controls whether the compiler produces the following maps in the listing file:

a) Data names, procedure names, file names, and their attributes

b) External references such as user-called routines or Run-Time Library routines

You can control the format of the data name, procedure name, and file name maps in the listing file by specifying one or both of the following keywords:

ALPHABETICAL	Produces an alphabetical list of map items MAP=ALPHABETICAL is the equivalent of MAP.
DECLARED	Produces a list of map items in the order in which they were declared

If you specify MAP=(ALPHABETICAL,DECLARED), the compiler produces both alphabetical and declared map listings.

MAP has no effect unless you also specify LIST.

The default, NOMAP, suppresses the creation of maps in the listing file.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**/MATH_INTERMEDIATE** **[={*keyword*,...}]**	**-math_intermediate** **{*keyword*}**	**/MATH_INTERMEDIATE** **[={*keyword*,...}]**

Specifies the intermediate data type to be used when the result of an arithmetic operation can not be represented exactly. The default is MATH_INTERMEDIATE=FLOAT. The keywords are as follows:

FLOAT	Selects double-precision floating-point for the intermediate data type. Intermediate values are truncated to the most significant 53 bits, providing approximately 15 decimal digits of precision.
CIT3	Selects Cobol Intermediate Temporary for the intermediate data type. Intermediate values are truncated to the most significant 18 decimal digits.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-names {keyword}**	**/NAMES =*keyword***

Specifies how source code identifiers and literals that are externally visible are interpreted, in accordance with the following.keywords:

LOWERCASE	Causes the compiler to force all external names (program ids, literal entrynames in CALL statements, external data items) to be lowercase. This is the default.
UPPERCASE	Causes the compiler to force all external names (program id's, literal entrynames in CALL statements, external data items) to be uppercase.
AS_IS	Causes the compiler to *not* change the case of literals used in CALL literal statements. Other external data names are treated as if NAMES LOWERCASE had been specified, whereas external program ids are treated as if NAMES UPPERCASE had been specified.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/NATIONALITY [=*keyword*] /NATIONALITY=US (D)	**/NATIONALITY [=*keyword*] /NATIONALITY=US (D)**	**-nationality {*keyword*}**	**/NATIONALITY =*keyword***

Controls whether national language features are selected, as shown in the following table:

US	The default currency sign and symbol are the Dollar sign, and Japanese language support features are disabled.
JAPAN	The default currency sign and symbol are the Yen sign, and Japanese language support features are enabled. NODIAGNOSTICS and NOANALYSIS_DATA are specified implicitly. The specific features enabled include: • Yen currency-sign • National character user-defined-words • National data items (PIC N) • National literal (N"")

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-nolocking**	**/NOLOCKING**
Turns off default file locking. Must be used when you compile programs that access files served from NFS or UCX file systems.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/NOLOGO**
Suppresses product banner. Default: display product banner.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-non_shared** **-call_shared (D)**	**NA**
Produces a static executable image. The linker will search regular archive library files (.a) to resolve undefined references;.so files are not searched. Object files (.o) from archives are included in the executable produced. The default is call_shared, which will produce a dynamic executable image			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/OBJECT **[=*filespec*]** **/NOOBJECT**	**/OBJECT** **[=*filespec*]** **/NOOBJECT**	**-o filespec** **-noobject**	**/OBJECT** **[=*filespec*]** **/NOOBJECT**
Specifies the name of the object file. On DIGITAL UNIX, names the object file if -c is specified; otherwise, names the executable file. NOOBJECT does not produce an object file and may be useful for syntax checking only. (See also KEEP and COMPILE_ONLY on DIGITAL UNIX and Windows NT Alpha.)			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**/OPTIMIZE [=LEVEL=*n*] (D)** **/NOOPTIMIZE**	**-O*level***	**/OPTIMIZE[=*n*]** **/NOOPTIMIZE**

Controls whether the compiler optimizes the program to generate more efficient code for optimum run-time performance. Specifying NOOPTIMIZE instructs the compiler not to produce optimized code.

You can select one of the levels (value of n) shown in the following table:

0	Has the same effect as /NOOPTIMIZE. All optimizations are turned off.
1	Has some optimizations (such as instruction scheduling).Enables local optimizations and recognition of common subexpressions. The call graph determines the order of compilation of procedures.
2	Adds more optimizations (such as loop unrolling or split lifetime analysis) to those in level 1. Enables global optimization and all level 1 optimizations. This includes code motion, strength reduction and test replacement, split lifetime analysis, code scheduling, and inlining of arithmetic statement functions.
3	Adds more optimizations (such as decimal shadowing) to those in level 2. All optimizations are turned on. Enables additional global optimizations that improve speed (at the cost of extra code size), for example: integer multiplication and division expansion (using shifts), loop unrolling, and code replication to eliminate branches. Also performs all level 2 optimizations.
4	Is identical to level 3. OPTIMIZE=LEVEL=4 is the equivalent of OPTIMIZE or not specifying OPTIMIZE.

Specify NOOPTIMIZE if you specify DEBUG when compiling a program. NOOPTIMIZE expedites and simplifies your debugging session by putting the machine code in the same order as the lines in the source program. Optimizations can cause unexpected and confusing behavior in a debugging session. For more information about debugging your program with the NOOPTIMIZE modifier, see the user manual for your platform, online HELP or the reference page.

Specify NOOPTIMIZE if you specify MACHINE_CODE when compiling a program to ensure that the machine code listing reflects the actual code executed for a given statement as well as the order of execution.

NOOPTIMIZE is also useful in conjunction with CHECK, as optimization can make checking more difficult.

Specifying OPTIMIZE, the default, usually makes programs run more efficiently. However, using OPTIMIZE produces extra instructions to perform the optimization, which may result in larger object modules and longer compile times than NOOPTIMIZE.

To speed compilations during program development, you may want to compile with NOOBJECT when you want to check syntax, with NOOPTIMIZE when you check for correct execution, and later with OPTIMIZE for your final check. For more information about optimizing your program with OPTIMIZE, see the user manual for your platform, online HELP or the reference page.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-p*n***	**NA**

Controls profiling according to the value of *n* as shown in the following table:

0	Does not permit any profiling. If loading occurs, the standard run-time startup routine (crt0.o) is used, and the profiling libraries are not searched. This is the default.
1	Sets up profiling by periodically sampling the value of the program counter. This flag affects only the loading. When loading occurs, this flag replaces the standard run-time startup routine with the profiling run-time startup routine (mcrt0.o) and searches the level 1 profiling library (libprof1.a). When profiling occurs, the startup routine calls monstartup (3) and produces the file mon.out, which contains execution-profiling data for use with the postprocessor prof (1).

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA (See /CHECK=DUPLICATE)	**NA (See /CHECK=DUPLICATE)**	**-relax_key_checking -rkc**	**/RELAX_KEY_ CHECKING /NORELAX_KEY _CHECKING (D)**

Specifies that a file can be opened with fewer keys (not more, however) than the number with which the file was created; and keys need not match on whether duplicates are allowed. This keyword will provide correct results only in those cases where the unspecified keys are USAGE DISPLAY PIC X. The checks for key type, size, and offset are unaffected by this keyword.

Default: NORELAX_KEY_CHECKING

<table>
<tr><th>OpenVMS VAX</th><th>OpenVMS ALPHA</th><th>DIGITAL UNIX</th><th>Windows NT Alpha</th></tr>
<tr><td>NA</td><td>/RESERVED_WORD
keyword</td><td>-rsv keyword
-rsv xopen (D)</td><td>/RESERVED_WORD
keyword</td></tr>
<tr><td colspan="4">Controls whether the compiler recognizes certain COBOL words as reserved words. The two options are XOPEN and FOREIGN_EXTENSIONS, which are explained as follows:
<table>
<tr><td>[NO]XOPEN</td><td colspan="2">Controls whether or not the compiler recognizes reserved words defined by the COBOL X/Open Portability Guide. Use /RESERVED_WORDS=NOXOPEN if your program uses one or more of the X/Open reserved words as an identifier. The default is /RESERVED_WORDS=XOPEN.
The X/Open reserved words are as follows:</td></tr>
<tr><td></td><td>AUTO
BACKGROUND-COLOR
BELL
BLINK
EOL
EOS
ERASE
FOREGROUND-COLOR
FULL</td><td>HIGHLIGHT
LOWLIGHT
REQUIRED
RETURN-CODE
REVERSE-VIDEO
SCREEN
SECURE
UNDERLINE</td></tr>
<tr><td>[NO]FOREIGN_EXTENSIONS</td><td colspan="2">Controls whether or not the compiler recognizes reserved words used by foreign extensions (language constructs that are not part of DIGITAL COBOL). Use /RESERVED_WORDS=FOREIGN_EXTENSIONS if you want the compiler to output specific diagnostics of foreign extensions to assist you in porting to DIGITAL COBOL from other COBOL dialects. Do not use this option if your program uses any of the foreign_extensions reserved words as user-defined words.
The reserved words that are recognized for foreign_extensions are:</td></tr>
<tr><td></td><td>ADDRESS
CHANGED
CORD-INDEX
DBCS
DISP
DISPLAY-1
EJECT
ENTRY
EXAMINE
EXHIBIT
GOBACK
ID
KANJI
NAMED
NOTE</td><td>OTHERWISE
PASSWORD
POSITIONING
RECORDING
RECORD-OVERFLOW
RELOAD
REORG-CRITERIA
REURNING
SERVICE
SKIP1
SKIP2
SKIP3
TRACE
TRANSFORM</td></tr>
</table></td></tr>
</table>

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/SEQUENCE_CHECK **/NOSEQUENCE_CHECK (D)**	**/SEQUENCE_CHECK** **/NOSEQUENCE_CHECK (D)**	**-seq** **-sequence_check**	**/SEQUENCE_CHECK** **/NOSEQUENCE_CHECK (D)**
Controls whether the compiler initiates an ascending-order sequence check on the line numbers in columns 1 through 6 of the source program. Source programs written in terminal format always pass the sequence check. The default, NOSEQUENCE_CHECK, suppresses sequence checking.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-shared**	**/DLL [=*filespec*]** **/NODLL (D)**
Produces a dynamic shareable object for inclusion in a shared library. The linker will produce a shareable object that other dynamic executables can use at run time. If you also specify the -c option, shared is ignored and a .o file is created; otherwise, a .so file is created. The default is to produce a dynamic executable image. See also call_shared.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-show *keyword***	**NA**

Controls whether the listing will include specified information. The keyword values are shown in the following table:

code	Include a machine code listing. Equivalent to -mach.
copy	Include source statements included by COPY statements. Equivalent to -copy.
xref	Include a cross-reference of all symbols used in the source program, with user-defined names arranged alphabetically, along with line numbers of definitions and uses. Equivalent to -cross_reference.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/SOURCE** **[=*filespec]***
Names a source file. Useful when the source file specification does not end in a standard extension (.cob, .cbl). Default: Only files ending in the recognized extension are passed to the compiler.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/STANDARD [(=*keyword*,...)] /STANDARD=85 (D)	**/STANDARD [(=*keyword*,...)] /STANDARD=85 (D)**	**-std *keyword***	**/STANDARD [=*keyword*] /STANDARD=85 (D)**

Controls whether the compiler generates code according to either the ANSI 1974 or 1985 COBOL standard and produces informational messages associated with specific language features. To receive these informational messages, you must also specify WARNINGS=ALL or WARNINGS=INFORMATIONAL.

You can select one or more of the following keywords:

[NO]85	Produces code according to the 1985 ANSI standard for certain constructs. The default is STANDARD=85.
[NO]V3	Produces code in the manner of version 3.4 of DIGITAL VAX COBOL in specific instances, and issues informational messages for language constructs that would cause different run-time results if STANDARD=85 had been specified. The default is NOV3.
[NO]SYNTAX	Produces informational diagnostics on language features that point out DIGITAL extensions to the ANSI 1985 COBOL Standard. The default is NOSYNTAX.
[NO]XOPEN (Alpha only)	Produces code for the ASSIGN clause and default file-sharing behavior in the manner of the X/Open CAE specification for the COBOL language. The default is NOXOPEN.
[NO]MIA	Issue informational diagnostics for the language elements which do not conform to the MIA (Multivendor Integration Architecture). The default is NOMIA.
[NO]OPENVMS_AXP (DIGITAL VAX COBOL only)	The compiler generates informational messages to flag language constructs that are not available in DIGITAL COBOL. Using this information on the coding incompatibilities, you can modify your code before running the program on an OpenVMS Alpha system.
[NO]PDP11 (DIGITAL VAX COBOL only)	The compiler produces informational diagnostics on language features that are specific to the PDP11.

DIGITAL VAX COBOL and DIGITAL COBOL are based on the ANSI 1985 COBOL standard (1989 Addendum). As such,DIGITAL COBOL provides full support for the STANDARD=85 modifier keyword as well as support for some features of the STANDARD=V3 modifier keyword that were available with VAX COBOL Version 4.0 and higher.

When you specify STANDARD=V3 in specific instances, DIGITAL COBOL exhibits behavior that is consistent with the ANSI 1985 COBOL standard.

When you specify STANDARD=V3, DIGITAL VAX COBOL exhibits behavior consistent with V3 VAX COBOL.

For further information about the implementation of the STANDARD=V3 modifier keyword, see the user manual for your platform.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-taso**	**NA**
Tells the linker that the executable should be loaded in the lower 31-bit addressable virtual address range. The -T and D modifiers to the ld(1) command can also be used to ensure that the text and data segments addresses, respectively, are loaded into low memory. The flag, however, in addition to setting default addresses for text and data segments, also causes shared libraries linked outside the 31-bit address space to be appropriately relocated by the loader. If you specify taso and also specify text and data segment addresses with T and D, those addresses override the default addresses. The taso flag is useful for porting 32-bit programs to DIGITAL UNIX.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**/TIE** **/NOTIE (D)**	**NA**	**NA**
Generates code that allows native OpenVMS Alpha images to call translated VAX images and translated OpenVMS VAX images to call native OpenVMS Alpha images. This modifier is supported on OpenVMS Alpha systems only. Specifying TIE when you want to use native, compiled code with shared translated VAX images, either because the code might call into a translated VAX image or because it might be called from a translated VAX image. If you specify TIE, you should link the object module using the LINK command modifier NATIVE_ONLY. (See the OpenVMS Linker Utility Manual for information about the NATIVE_ONLY modifier.) Specifying NOTIE, the default, indicates that your compiled code is not associated with a translated VAX image. For more information about translated images, see Migrating to an Alpha VMS System: Translating Images. For information about interoperability, see *Migrating to an Alpha VMS System: Recompiling and Relinking Applications.*			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-T*num***	**NA**
Tells ld 1 to set the text segment origin. If *num* starts with a hexadecimal letter, precede it with the digit 0.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-tps**	**NA**
Specifies that files are part of a transaction processing system and enables Encina SFS record storage for applicable files			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/TRUNCATE /NOTRUNCATE (D)	**/TRUNCATE /NOTRUNCATE (D)**	**-trunc**	**/TRUNCATE /NOTRUNCATE (D)**
Controls how the compiler stores values in COMPUTATIONAL receiving items if high-order truncation is necessary. If you specify TRUNCATE, the compiler truncates values according to the number of decimal digits specified by the PICTURE size. Specifying TRUNCATE increases program execution time. The default, NOTRUNCATE, instructs the compiler to truncate values according to the hardware storage unit (word, longword, or quadword) allocated to the receiving item.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/USAGE**
Same as BRIEF_HELP.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-V**	**NA**
Creates a listing file of the source file with various compile-time information appended. The name of the listing file is the base name of the source file with .lis substituted for the .cob, .COB, .cbl, or .CBL. Equivalent to list. If you compile several files together, a separate listing file is created for each input file (named with the base name of the input file and the .lis suffix).			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-v**	**/VERBOSE /NOVERBOSE (D)**
Prints the passes as they execute with their arguments and their input and output files.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA ***See note below**	**/VFC (D)** **/NOVFC**	**NA**	**NA**
Generates VFC record format for the following types of files: • LINAGE • REPORT WRITER • APPLY PRINT-CONTROL • WRITE ADVANCING • ORGANIZATION SEQUENTIAL with GLOBAL • ORGANIZATION SEQUENTIAL with EXTERNAL If you specify NOVFC, or if these files are created on DIGITAL UNIX or Windows NT Alpha, these output files are generated with Stream_LF format. Note that VFC record format is available on DIGITAL VAX COBOL without the need for a modifier.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-w**	**NA**
Suppresses all warning messages. Equivalent to nowarn and warn none.			

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
/WARNINGS [=(*keyword*[,...])] /NOWARNINGS (D)	**/WARNINGS [=(*keyword*[,...])] /NOWARNINGS (D)**	**-warn [*keyword*] -nowarn (D)**	**/WARNINGS [=*keyword*] /NOWARNINGS (D)**

Causes the compiler to print warning and informational messages as well as error and severe error messages. You can select one or more of the keywords shown in the following table:

[NO]INFORMATION	Produces additional informational messages. The default is NOINFORMATION.
[NO]OTHER	Produces warning messages. The default is OTHER.
ALL	Provides the messages produced by the INFORMATION and OTHER keywords. WARNINGS=ALL is the equivalent of WARNINGS.
NONE	Omits the listing of all messages. WARNINGS=NONE is the equivalent of NOWARNINGS.
STANDARD	Produces informational diagnostics on language features that are DIGITAL extensions (DIGITAL VAX COBOL only).

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**NA**	**/WHAT /NOWHAT [D]**

Displays version information. Default: NOWHAT

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-xref**	**NA**

Directs the DIGITAL COBOL compiler to generate a data file that the DEC FUSE Database Manager uses to create a cross-reference database file. This improves the performance of the DEC FUSE Call Graph Browser and Cross-Referencer, which use the database file for their operations. See the DEC FUSE Handbook for more information on the FUSE cross-reference database and the DEC FUSE Cross-Referencer and Call Graph Browser.

OpenVMS VAX	OpenVMS ALPHA	DIGITAL UNIX	Windows NT Alpha
NA	**NA**	**-xref_stdout**	**NA**
Directs the compiler to output the DEC FUSE data file to standard output.			

Chapter 7

Useful Summaries and Tables

Whether you are migrating applications or maintaining code, this chapter brings the following helpful information to one handy location:

- Cross-platform compatibility
- COB$SWITCHES, cob_switches information and examples
- Intrinsic functions
- Allowable I/O statements
- File status values
- ASCII and EBCDIC character codes
- Related documentation
- Corresponding with us
- Online services
- How to order additional documentation

Cross-Platform Compatibility

The following table shows cross-platform compatibility as of the date of publication of this book. You should refer to the Release Notes and technical documentation for your platform for the most current state of compatibility of these features.

The key for this table is as follows:

Key	Meaning
!	Available now
N	Not available
P	Partial support

FEATURE	COMPATIBILITY			
	VAX COBOL	DIGITAL COBOL ON:		
		OpenVMS Alpha	DIGITAL UNIX	Windows NT Alpha
ANSI-85/-89 (more than 120 verbs)	!	!	!	!
Report writer	!	!	!	!
Tape handling	!	!	!	N
ORGANIZATION INDEXED	!	!	!	!
Segmented keys	!	!	!	!
/CHECK=DUPLICATE_KEYS	!	!	N	N
Relaxed key checking	N	N	!	!
RMS segmented keys	!	!	N	N
RMS special registers	!	!	N	N
RMS APPLY extensions	!	!	N	N
/STANDARD=V3	!	P	P	P
VFU	!	N	N	N
Print control files with VFC	!	!	N	N
Print control files without VFC	N	!	!	!
ISAM READ PRIOR/START LESS	N	!	!	!
VAX/DIGITAL COBOL ACCEPT/DISPLAY extensions	!	!	!	P
VAX/DIGITAL COBOL ACCEPT/DISPLAY screen handling	!	!	!	P
DISPLAY WITH CONVERSION	!	!	!	!
ACCEPT/DISPLAY VT52 support	!	N	N	N
ACCEPT/DISPLAY VT{1,2,3,4}xx support	!	!	!	N
VAX/DIGITAL COBOL file sharing and record locking	!	!	P	P
UCX/NFS support (nolocking)	N	N	!	!
FUNCTION ARGCOUNT	!	!	N	N
Little-endian COMP data	!	!	!	!

FEATURE	COMPATIBILITY			
	VAX COBOL	DIGITAL COBOL ON:		
		OpenVMS Alpha	DIGITAL UNIX	Windows NT Alpha
F,D floating	!	!	N	N
G floating	N	!	N	N
IEEE S,T floating	N	!	!	!
Floating point “E” literal	!	!	!	!
Pointer data	!	!	!	!
64-bit pointers	N	N	!	N
/CHECK=DECIMAL	P	!	!	!
X/Open RETURN-CODE	N	!	!	!
X/Open COMP-5/COMP-X	N	!	!	!
X/Open LINE SEQUENTIAL	N	!	!	!
X/Open ASSIGN TO	N	!	!	!
X/Open SCREEN SECTION	N	!	!	P
X/Open SCREEN SECTION screen handling	N	!	!	P
X/Open file sharing and record locking	N	!	!	!
X/Open environment variables	N	!	!	!
X/Open command line	N	!	!	!
/CHECK=(PERFORM,BOUNDS)	!	!	!	!
VAX-compatible alignment	!	!	!	!
Alpha natural alignment and padding	N	!	!	!
/[NO]SEPARATE_COMPILATION	P	!	N	N
I18N (PIC N, etc.)	!	!	!	!
CALL USING BY DESCRIPTOR	!	!	N	N
cobfunc, cobcall, cobcancel	N	!	!	N
Reformat	!	!	!	!
Terminal source format	!	!	!	!
Lowercase, -/_ in source	!	!	!	!

FEATURE	COMPATIBILITY			
	VAX COBOL	DIGITAL COBOL ON:		
		OpenVMS Alpha	DIGITAL UNIX	Windows NT Alpha
IDENT	!	!	N	N
Sample Application - Client/Server	N	N	N	!
Oracle CDD/Repository,DML support	!	!	N	N
Transarc Encina (-tps) support	N	N	!	N
DECset PCA,LSE/SCA support	!	!	N	N
DECset PDF support	!	N	N	N
FUSE support	N	N	!	N
Symbolic debugger support	!	!	!	!
Docs - UM/RM/IG/RN (IG in UM for NT)[1]	!	!	!	!
Docs - Client/Server	N	N	N	!
Docs - DML	!	!	N	N
Docs - online help	!	!	!	!
Docs - UM/RM .DECW$BOOK or .PDF	!	!	!	!

1. IG: Installation Guide; UM: User Manual; RM: Reference Manual; NT: Windows NT Alpha

COB$SWITCHES, cob_switches

COBOL has switches that can be set and tested from within an application. These switches can be extended beyond the application's internal environment at run time by using the environment variables COB$SWITCHES (on OpenVMS platforms) and COBOL_SWITCHES (on DIGITAL UNIX and Windows NT Alpha platforms). You can access and manipulate these switches from the command line and from within your programs.

Setting Switches Inside Your Program

To set switches from within an application, define them in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION and use the SET statement in the PROCEDURE DIVISION to specify switches ON or OFF, as in the following example:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
```

```
SPECIAL-NAMES.
SWITCH 10 IS MY-SWITCH
ON IS SWITCH-ON
OFF IS SWITCH-OFF.
.
.
.
PROCEDURE DIVISION.
000-SET-SWITCH.
SET MY-SWITCH TO ON.
IF SWITCH-ON
THEN
DISPLAY "SWITCH 10 IS ON"
.
.
.
```

To change the status of internal switches during execution, turn them on or off from within your program. However, be aware that this information is not saved between runs of the program.

Setting Switches Outside Your Program

COB$SWITCHES and COBOL_SWITCHES exist in your programming environment as environment variables. How you set or read them depends upon your platform.

OpenVMS Switches

To set switches for your process, use the DEFINE or ASSIGN DCL command to change the status of program switches as follows:

```
$ DEFINE COB$SWITCHES "SWITCH-LIST"
```

SWITCH-LIST can contain up to 16 switches separated by commas. Set a switch ON by specifying it in the switch-list. A switch is OFF (the default) if you do not specify it in the switch-list.

For example, to set the named switches on:

```
$ DEFINE COB$SWITCHES "1,5,13"
$ DEFINE COB$SWITCHES "9,11,16"
```

To set all switches off:

```
$ DEFINE COB$SWITCHES " "
```

DIGITAL UNIX Switches

To set switches from outside the image, use the SETENV command to change the status of program switches, as follows:

```
% setenv cobol_switches "switch-list"
```

To remove switch settings:

```
% unsetenv cobol_switches
```

To check switch settings, enter the following command:

```
% printenv cobol_switches
```

The switch-list can contain up to 16 switches separated by commas. To set a switch on, specify it in the switch-list.

A switch is off (the default) if you do not specify it in the switch-list.

For example, to set the named switches on:

```
% SETENV COBOL_SWITCHES "1,5,13"
% SETENV COBOL_SWITCHES "9,11,16"
```

To set all switches off:

```
% SETENV COBOL_SWITCHES " "
```

Windows NT ALPHA Switches

To set switches from outside the image, use the set command to change the status of program switches, as follows:

```
C:  SET COBOL_SWITCHES=SWITCH-LIST
```

Where switch–list is a list of current values from 1 to 16

To remove switch settings use set with no argument:

```
C:  SET COBOL_SWITCHES=
```

To check switch settings, enter the following command:

```
C:  ECHO %COBOL_SWITCHES%
```

The switch-list can contain up to 16 switches separated by commas. To set a switch on, specify it in the switch-list. A switch is off (the default) if you do not specify it in the switch-list.

For example, to set the named switches on:

```
C:  SET COBOL_SWITCHES=1,5,13
C:  SET COBOL_SWITCHES=9,11,16
```

To set all switches off:

```
C:  SET COBOL_SWITCHES=
```

Four-Platform Example to Evaluates Switches

The following example program tests for the state of switches 1 through 16:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SWITCHTEST.
*
*      THIS PROGRAM EXERCISES SWITCH-STATUS CONDITION TESTING.
*      ASSUMPTION: SWITCHES 1, 2, 3, 4 ON.
*      ASSUMPTION: SWITCHES 13, 14, 15, 16 OFF.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. VAXORALPHA.
OBJECT-COMPUTER. VAXORALPHA.
SPECIAL-NAMES.
      SWITCH 1  ON STATUS IS S1ON
      SWITCH 2  ON STATUS IS S2ON, OFF STATUS IS S2OFF
      SWITCH 3  OFF STATUS IS S3OFF
      SWITCH 4  OFF STATUS IS S4OFF, ON STATUS IS S4ON
      SWITCH 13 ON STATUS IS S13ON,
      SWITCH 14 ON STATUS IS S14ON, OFF STATUS IS S14OFF
      SWITCH 15 OFF STATUS IS S15OFF
      SWITCH 16 OFF STATUS IS S16OFF, ON STATUS IS S16ON.
*
DATA DIVISION.
PROCEDURE DIVISION.
S0 SECTION.
P0.
      DISPLAY " SWITCHTEST".
      IF S1ON GO TO P1.
      DISPLAY "? 1".
P1.   IF S2ON GO TO P2.
      DISPLAY "? 2".
P2.
      IF S2OFF DISPLAY "? 3"
      IF S3OFF DISPLAY "? 4".
      IF S4OFF DISPLAY "? 5".
      IF S4ON GO TO P3.
      DISPLAY "? 6".
P3.
      IF S13ON DISPLAY "? 7".
      IF S14ON DISPLAY "? 8".
      IF S14OFF GO TO P4.
      DISPLAY "? 9".
P4.
      IF S15OFF GO TO P5.
      DISPLAY "? 10".
P5.
      IF S16OFF GO TO P6.
      DISPLAY "? 11".
P6.
      IF S16ON DISPLAY "? 12".
```

```
*
      DISPLAY "*****END*****".
*
      STOP RUN.
```

Intrinsic Functions

DIGITAL COBOL intrinsic functions are powerful programming tools that dynamically derive a data value at run time. They relieve you of having to include code for tasks that are performed frequently. The returns are treated as temporary elementary data items.

There are six basic categories of intrinsic functions as shown in the following table:

Category	**Functions**
Scientific/Mathematical	ACOS, ASIN, ATAN, COS, FACTORIAL, LOG, LOG10, MOD, REM, SIN, SQRT, SUM, TAN
Relational	MAX, MIN, ORD-MAX, ORD-MIN
String Manipulation	LOWER-CASE, NUMVAL, NUMVAL-C, REVERSE, UPPER-CASE
Date Manipulation	CURRENT-DATE, DATE-OF-INTEGER, DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY, WHEN-COMPILED
Statistical/Accounting	ANNUITY, MEAN, MEDIAN, MIDRANGE, PRESENT-VALUE, RANGE, STANDARD-DEVIATION, VARIANCE
Programming Aids	ARGCOUNT *(OpenVMS only)*, CHAR, INTEGER, INTEGER-PART, LENGTH, ORD, RANDOM

The functions, their input arguments, types, and returns for the DIGITAL COBOL functions are arranged alphabetically in the following table:

Function	**Arguments** **A: Alphabetic** **AN: Alphanumeric** **I: Integer** **N: Numeric**	**Function Type**	**Returns**
ACOS	1N, *num*	Numeric	Arccosine of *num*
ANNUITY	1N, *num*; 1I, int	Numeric	Ratio of annuity paid for each of *int* periods at interest of *num* to initial investment of one monetary unit
ARGCOUNT (OpenVMS only)	None	Integer	Number of arguments passed to the COBOL program
ASIN	1N, *num*	Numeric	Arcsine of *num*
ATAN	1N, *num*	Numeric	Arctangent of *num*
CHAR	1I, *int*	Alphanumeric	Character in position *int* of program collating sequence
COS	1N, num	Numeric	Cosine of num
CURRENT-DATE	None	Alphanumeric	Current date and time
DATE-OF-INTEGER	1I	Integer	Standard date equivalent (YYYYMMDD) of integer date
DAY-OF-INTEGER	1I	Integer	YYYYDDD date equivalent of integer date
FACTORIAL	1I, *int*	Integer	Factorial of *int*
INTEGER	1N, *num*	Integer	The greatest integer not greater than *num*
INTEGER-OF-DATE	1I	Integer	Integer date equivalent of standard date
INTEGER-OF-DAY	1I	Integer	Integer date equivalent of date in YYYYDDD format
INTEGER-PART	1N, *num*	Integer	Integer part of *num*

(Continued) LENGTH	1A or N or AN, or 1NN	Integer	Length of argument
LOG	1 N, *num*	Numeric	Natural logarithm of *num*
LOG10	1 N, *num*	Numeric	Logarithm to base 10 of *num*
LOWER-CASE	1 A or 1 AN	Alphanumeric	All letters in the argument set to lowercase
MAX	1 or more A and/or AN, or 1 or more I and/or N	Depends on arguments (see manual)	Value of maximum argument
MEAN	1 or more N	Numeric	Arithmetic mean of arguments
MEDIAN	1 or more N	Numeric	Median of arguments
MIDRANGE	1 or more N	Numeric	Mean of minimum and maximum arguments
MIN	1 or more A and/or AN, or 1 or more I and/or N	Depends on arguments (see manual)	Value of minimum argument
Mod	1 I, *int*-1 and *int-2*	Integer	Value of *int-1* modulo *int-2*
NUMVAL	1 AN	Numeric	Numeric value of simple numeric string
NUMVAL-C	1 OR 2 AN	Numeric	Numeric value of numeric string with optional commas and currency sign
ORD	1 A or 1 AN	Integer	Ordinal position of the argument in collating sequence
ORD-MAX	1 or more A, or 1 or more N, or 1 or more AN	Integer	Ordinal position of maximum argument
ORD-MIN	1 or more A, or 1 or more N, or 1 or more AN	Integer	Ordinal position of minimum argument
PRESENT-VALUE	1 N, *num-1;* and 1 or more additional N, *num-2*	Numeric	Present value of a series of future period-end amounts, *num-2*, at a discount rate of *num-1*
RANDOM	1 I or none	Numeric	Random number

(Continued) RANGE	1 or more I, or 1 or more N	Depends on arguments	Value of maximum argument minus value of minimum argument
REM	2 N, *num-1* and *num-2*	Numeric	Remainder of *num-1/num-2*
REVERSE	1A or 1 AN	Alphanumeric	Reverse order of the characters of the argument
SIN	1 N, *num*	Numeric	Sine of *num*
SQRT	1 N, *num*	Numeric	Square root of *num*
STANDARD-DEVIATION	1 or more N	Numeric	Standard deviation of arguments
SUM	1 or more I, or 1 or more N	Depends on arguments	Sum of arguments
TAN	1 N, *num*	Numeric	Tangent of *num*
UPPER-CASE		1 A or AN	All letters in the argument set to uppercase
VARIANCE	1 or more N	Numeric	Variance of argument
WHEN-COMPILED	None	Alphanumeric	Date and time program was compiled

I/O Statements

The following tables show allowable I/O statements for sequential, line sequential, relative, and indexed files.

File Organization	Access Mode	Statement	Open Mode			
			INPUT	OUTPUT	I/O	EXTEND
SEQUENTIAL	SEQUENTIAL	READ REWRITE WRITE UNLOCK	Yes No No Yes	No No Yes Yes	Yes Yes No Yes	No No Yes Yes

File Organization	Access Mode	Statement	Open Mode			
			INPUT	OUTPUT	I/O	EXTEND
LINE SEQUENTIAL	SEQUENTIAL	READ	Yes	No	No	No
		REWRITE	No	No	No	No
		WRITE	No	Yes	No	Yes
		UNLOCK	Yes	Yes	No	Yes

File Organization	Access Mode	Statement	Open Mode			
			INPUT	OUTPUT	I/O	EXTEND
RELATIVE	SEQUENTIAL	DELETE	No	No	Yes	No
		READ	Yes	No	Yes	No
		REWRITE	No	No	Yes	No
		START	Yes	No	Yes	No
		WRITE	No	Yes	No	Yes
		UNLOCK	Yes	Yes	Yes	Yes
	RANDOM	DELETE	No	No	Yes	No
		READ	Yes	No	Yes	No
		REWRITE	No	No	Yes	No
		WRITE	No	Yes	Yes	No
		UNLOCK	Yes	Yes	Yes	No
	DYNAMIC	DELETE	No	No	Yes	No
		READ	Yes	No	Yes	No
		READ NEXT	Yes	No	Yes	No
		REWRITE	No	No	Yes	No
		START	Yes	No	Yes	No
		WRITE	No	Yes	Yes	No
		UNLOCK	Yes	Yes	Yes	No

File Organization	Access Mode	Statement	Open Mode			
			INPUT	OUTPUT	I/O	EXTEND
INDEXED	SEQUENTIAL	DELETE	No	No	Yes	No
		READ	Yes	No	Yes	No
		REWRITE	No	No	Yes	No
		START	Yes	No	Yes	No
		WRITE	No	Yes	No	Yes
		UNLOCK	Yes	Yes	Yes	Yes
	RANDOM	DELETE	No	No	Yes	No
		READ	Yes	No	Yes	No
		REWRITE	No	No	Yes	No
		WRITE	No	Yes	Yes	No
		UNLOCK	Yes	Yes	Yes	No
	DYNAMIC	DELETE	No	No	Yes	No
		READ	Yes	No	Yes	No
		READ NEXT	Yes	No	Yes	No
		READ PRIOR	Yes	No	Yes	No
		REWRITE	No	No	Yes	No
		START	Yes	No	Yes	No
		WRITE	No	Yes	Yes	No
		UNLOCK	Yes	Yes	Yes	No

File Status Values

The file status codes that you receive may vary, depending upon the presence and setting of the STANDARD command modifier. See /STANDARD in *Modifying the COBOL Command*, Chapter 6, and the user manual for your platform.

File Status	I/O Statements	File Organization	Access Mode	Meaning
00	All	All	All	Successful
02	REWRITE WRITE	Ind	All	Created duplicate alternate key
02	READ	Ind	All	Detected alternate duplicate key

File Status	I/O Statements	File Organization	Access Mode	Meaning
04	READ	All	All	Record not size of user's buffer
05	OPEN	All	All	Optional file not present
07	CLOSE OPEN	All	All	Invalid file organization or device
10	READ	All	Seq	No next logical record or option file not present (at end)
14	READ	Rel	All	Relative record number too large
21	REWRITE	Ind	Seq	Primary key changed after READ
21	WRITE	Ind	Seq	Attempted nonascending key value (invalid key)
22	REWRITE	Ind	All	Duplicate alternate key (invalid key)
22	WRITE	Ind, Rel	Ran	Duplicate key (invalid key)
23	DELETE READ REWRITE START	Ind, Rel	Ran	Record not in file; optional file not present (invalid key)
24	WRITE	Ind, Rel	All	Boundary violation or relative record number too large (invalid key)
30	All	All	All	All other permanent errors
34	WRITE	Seq	Seq	Boundary violation
35	OPEN	All	All	File not found
37	OPEN	All	All	Inappropriate device type
38	OPEN	All	All	File previously closed with lock
39	OPEN	All	All	Conflict of file attributes
41	OPEN	All	All	File already opened
42	CLOSE	All	All	File not opened
43	DELETE REWRITE	All	Seq	No previous READ or START
44	REWRITE WRITE	All	All	Invalid record size
46	READ	All	Seq	No valid next record (at end)

File Status	I/O Statements	File Organization	Access Mode	Meaning
47	READ START	All	All	File not open, or incompatible open mode
48	WRITE	All	All	File not open, or incompatible open mode
49	DELETE REWRITE	All	All	File not open, or incompatible open mode
90	All	All	All	Record locked by another user (record available)
91	OPEN	All	All	File locked by another user
92	DELETE READ REWRITE START WRITE	All	All	Record locked by another user (record not available)
93	UNLOCK	All	All	No current record
94	UNLOCK	All	All	File not open, or incompatible open mode
95	OPEN	All	All	No file space on device
98	CLOSE	All	All	Any other CLOSE error

Character Codes

Character	ASCII		EBCDIC	
	Decimal	Hex	Decimal	Hex
NUL	000	00	000	00
SOH	001	01	001	01
STX	002	02	002	02
ETX	003	03	003	03
EOT	004	04	055	37
ENQ	005	05	045	2D
ACK	006	06	046	2E

Character	ASCII		EBCDIC	
	Decimal	**Hex**	**Decimal**	**Hex**
BEL	007	07	047	2F
BS	008	08	022	16
HT	009	09	005	05
LF	010	0A	037	25
VT	011	0B	011	0B
FF	012	0C	012	0C
CR	013	0D	013	0D
SO	014	0E	014	0E
SI	015	0F	015	0F
DLE	016	10	016	10
DC1	017	11	017	11
DC2	018	12	018	12
DC3	019	13	019	13
DC4	020	14	060	3C
NAK	021	15	061	3D
SYN	022	16	050	32
ETB	023	17	038	26
CAN	024	18	024	18
EM	025	19	025	19
SUB	026	1A	063	3F
ESC	027	1B	039	27
FS	028	1C	028	1C
GS	029	1D	029	1D
RS	030	1E	030	1E
US	031	1F	031	1F
space	032	20	064	40
!	033	21	090	5A
"	034	22	127	7F

Character	ASCII		EBCDIC	
	Decimal	**Hex**	**Decimal**	**Hex**
	035	23	123	7B
$	036	24	091	5B
%	037	25	108	6C
&	038	26	080	50
’	039	27	125	7D
(	040	28	077	4D
)	041	29	093	5D
*	042	2A	092	5C
+	043	2B	078	4E
,	044	2C	107	6B
-	045	2D	096	60
.	046	2E	075	4B
/	047	2F	097	61
0	048	30	240	F0
1	049	31	241	F1
2	050	32	242	F2
3	051	33	243	F3
4	052	34	244	F4
5	053	35	245	F5
6	054	36	246	F6
7	055	37	247	F7
8	056	38	248	F8
9	057	39	249	F9
:	058	3A	122	7A
;	059	3B	094	5E
<	060	3C	076	4C
=	061	3D	126	7E
>	062	3E	110	6E

Character	ASCII		EBCDIC	
	Decimal	**Hex**	**Decimal**	**Hex**
?	063	3F	111	6F
@	064	40	124	7C
A	065	41	193	C1
B	066	42	194	C2
C	067	43	195	C3
D	068	44	196	C4
E	069	45	197	C5
F	070	46	198	C6
G	071	47	199	C7
H	072	48	200	C8
I	073	49	201	C9
J	074	4A	209	D1
K	075	4B	210	D2
L	076	4C	211	D3
M	077	4D	212	D4
N	078	4E	213	D5
O	079	4F	214	D6
P	080	50	215	D7
Q	081	51	216	D8
R	082	52	217	D9
S	083	53	226	E2
T	084	54	227	E3
U	085	55	228	E4
V	086	56	229	E5
W	087	57	230	E6
X	088	58	231	E7
Y	089	59	232	E8
Z	090	5A	233	E9

Character	ASCII		EBCDIC	
	Decimal	**Hex**	**Decimal**	**Hex**
[	091	5B		
\	092	5C	224	E0
]	093	5D		
^	094	5E	095	5F
_	095	5F	109	6D
‘	096	60	121	79
a	097	61	129	81
b	098	62	130	82
c	099	63	131	83
d	100	64	132	84
e	101	65	133	85
f	102	66	134	86
g	103	67	135	87
h	104	68	136	88
i	105	69	137	89
j	106	6A	145	91
k	107	6B	146	92
l	108	6C	147	93
m	109	6D	148	94
n	110	6E	149	95
o	111	6F	150	96
p	112	70	151	97
q	113	71	152	98
r	114	72	153	99
s	115	73	162	A2
t	116	74	163	A3
u	117	75	164	A4
v	118	76	165	A5

Character	ASCII		EBCDIC	
	Decimal	**Hex**	**Decimal**	**Hex**
w	119	77	166	A6
x	120	78	167	A7
y	121	79	168	A8
z	122	7A	169	A9
{	123	7B	192	C0
|	124	7C	106	6A
}	125	7D	208	D0
~	126	7E	161	A1
DEL	127	7F	007	07

Related Documentation

Documentation Sets

The documentation sets for our COBOL family include this book and the following:

DIGITAL VAX COBOL	*VAX COBOL User Manual* *VAX COBOL Reference Manual* *VAX COBOL Installation Guide*
DIGITAL COBOL for DIGITAL UNIX DIGITAL COBOL for OpenVMS Alpha	*DIGITAL COBOL User Manual* *DIGITAL COBOL Reference Manual* *DIGITAL COBOL Installation Guide* • For OpenVMS Alpha systems • For DIGITAL UNIX systems
Windows NT Alpha	*DIGITAL COBOL User Manual* *DIGITAL COBOL Online Reference*

Corresponding with Us

Documentation Comments

If you have comments or suggestions about this book, send them to the product team by one of the following:

FAX: 603–884–0120 Attn.: COBOL Documentation Project Leader

E-MAIL: cobol_docs@bookie.zko.dec.com

Online Services

To locate product-specific information, refer to the following online services:

- The Digital Equipment Corporation home page:

  http://www.digital.com

- The DIGITAL COBOL home page:

  http://www.openvms.digital.com/commercial/cobol/

Note that this home page covers four platforms.

How to Order Additional Documentation

To order additional documentation, use the following table:

U.S.A.	DECdirect 800–DIGITAL 800–344–4825 Fax: 800–234–2298	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	809–781–0505 Fax: 809–749–8300	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street, Suite 200 P.O. Box 11038 Metro Office Park San Juan, Puerto Rico 00910–2138
Canada	800–267–6215 Fax: 613–592–1946	Digital Equipment of Canada, Ltd. Box 1300 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International		Local DIGITAL subsidiary or approved distributor
Internal Orders	DTN: 264–4446 603–884–4446 Fax: 603–884–3960	U.S. Software Supply Business Digital Equipment Corporation Nashua, NH 03063–1260

Index

Symbols

A

C

D

F

T

U

V

W

X

Y