

INTRODUCTION

The role that Control Language (CL) plays on the AS/400 goes beyond that of other programming languages. No matter what other high-level languages are used, CL programs are a crucial component of every application. It is true that IBM Rochester has delivered increasing numbers of APIs as the mechanism for providing controlled, open access to OS/400. And, many APIs provide the same functions as CL commands and often work more quickly and use less system overhead than the CL commands. Yet, CL remains as the primary means of communicating with the operating system. Therefore, the first chapter of this book focuses not only on the syntax and structure of CL, but has a discussion of CL commands in general and the role they play on the IBM AS/400 computer as well. In successive chapters, readers will increase their vocabularies in CL by examining increasingly advanced examples of code.

PREREQUISITES

A basic knowledge of OS/400, the operating system of the IBM AS/400, and some prior exposure to SEU is essential to learning CL. Some exposure to DDS and any high-level language (HLL), such as RPG or COBOL, would be helpful but isn't required.

DISCLAIMERS

I leave choice of language to the programmer. I make no attempt to justify the use of CL where an API or other HLL might be available as an alternative option. I only intend to demonstrate how CL can be used to meet systems requirements.

Neither do I attempt to justify the application design criteria that might be implied by any of the examples. AS/400 application design is a topic that is well beyond the scope of this text. I do attempt to include examples that are representative of code that is likely to be found in a typical production environment, but without making judgments regarding the overall design of the application. It is impossible to evaluate design without being aware of all the design criteria.

COMPATIBILITY

This text is consistent with Version 4, Release 4.0 of OS/400.

IBM Terminology

Over the years, the terminology used by IBM in the AS/400 technical manuals and even on various screens has evolved. The terminology used by AS/400 customers, however, has, for the most part, not kept pace with that evolution. For example, most AS/400 programmers still refer to the “call stack” as the “program invocation stack.” I have attempted to consistently use the current IBM terminology in this edition of the text. In order to help bridge the gap for readers, at each point in the text that an evolved term is introduced, I make a reference to the older terminology.

Integrated Language Environment

The programming model, or environment that was part of OS/400 when the AS/400 was introduced, is now referred to as the original program model (OPM). The Integrated Language Environment (ILE) is an additional, optional programming environment. Many, if not most of the changes in terminology are related to the significant operating system changes required to support ILE. Recent versions of the AS/400e series CL Programming manual are very oriented, both in terminology and function, toward ILE. I have observed, however, that the vast majority of new CL programs are still developed in the OPM. Consequently, I discuss most topics primarily from an OPM viewpoint. Specifically, I generally refer to CL programs, even though most discussions are equally applicable to ILE procedures. Where there are special considerations for ILE, I point those out. And chapter 13 is dedicated to the use of CL in the Integrated Language Environment.

CONVENTIONS

The following conventions are used in this text:

- The symbol **b** denotes a blank character.
- The notation used to describe the size of a decimal variable is as follows:
xx,yy

The value *xx* is the total number of digits allocated to the variable, and *yy* represents the number of decimals positions. For example, the notation 15,5 describes a variable with 15 digits, 5 of which are to the right of the decimal point.

- Character literal values, or constants, are enclosed in apostrophes. For example: `'CL Programming'`
- Hexadecimal literal values are prefixed with the letter *X* and can include any valid hexadecimal digit, including 0 to 9 and A to F. For example:

`X'12345F'`

A hexadecimal literal must contain an even number of characters within the apostrophes. Each pair of characters represents the zone and digit portion of a single byte. For example, `X'F1'` represents the character “1” in EBCDIC.

RECOMMENDATIONS

The value of this text will be greatly enhanced if the reader has convenient access to an AS/400 workstation and Program Development Manager (PDM). I strongly recommend keying and compiling all CL program examples. Augmenting the reading with hands-on experience is sure to enhance the learning process.

